

STATA TIME-SERIES REFERENCE MANUAL RELEASE 11



A Stata Press Publication
StataCorp LP
College Station, Texas



Copyright © 1985–2009 by StataCorp LP
All rights reserved
Version 11

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845

Typeset in T_EX

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN-10: 1-59718-063-7

ISBN-13: 978-1-59718-063-4

This manual is protected by copyright. All rights are reserved. No part of this manual may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means—electronic, mechanical, photocopy, recording, or otherwise—without the prior written permission of StataCorp LP unless permitted by the license granted to you by StataCorp LP to use the software and documentation. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document.

StataCorp provides this manual “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. StataCorp may make improvements and/or changes in the product(s) and the program(s) described in this manual at any time and without notice.

The software described in this manual is furnished under a license agreement or nondisclosure agreement. The software may be copied only in accordance with the terms of the agreement. It is against the law to copy the software onto DVD, CD, disk, diskette, tape, or any other medium for any purpose other than backup or archival purposes.

The automobile dataset appearing on the accompanying media is Copyright © 1979 by Consumers Union of U.S., Inc., Yonkers, NY 10703-1057 and is reproduced by permission from CONSUMER REPORTS, April 1979.

Stata and Mata are registered trademarks and NetCourse is a trademark of StataCorp LP.

Other brand and product names are registered trademarks or trademarks of their respective companies.

For copyright information about the software, type `help copyright` within Stata.

The suggested citation for this software is

StataCorp. 2009. *Stata: Release 11*. Statistical Software. College Station, TX: StataCorp LP.

Table of contents

intro	Introduction to time-series manual	1
time series	Introduction to time-series commands	3
arch	Autoregressive conditional heteroskedasticity (ARCH) family of estimators	10
arch postestimation	Postestimation tools for arch	43
arima	ARIMA, ARMAX, and other dynamic regression models	48
arima postestimation	Postestimation tools for arima	71
corrgram	Tabulate and graph autocorrelations	77
cumsp	Cumulative spectral distribution	84
dfactor	Dynamic-factor models	87
dfactor postestimation	Postestimation tools for dfactor	106
dfgls	DF-GLS unit-root test	111
dfuller	Augmented Dickey–Fuller unit-root test	117
dvech	Diagonal vech multivariate GARCH models	122
dvech postestimation	Postestimation tools for dvech	133
fcast compute	Compute dynamic forecasts of dependent variables after var, svar, or vec	137
fcast graph	Graph forecasts of variables computed by fcast compute	145
haver	Load data from Haver Analytics database	148
irf	Create and analyze IRFs, dynamic-multiplier functions, and FEVDs	153
irf add	Add results from an IRF file to the active IRF file	157
irf cgraph	Combine graphs of IRFs, dynamic-multiplier functions, and FEVDs	159
irf create	Obtain IRFs, dynamic-multiplier functions, and FEVDs	164
irf ctable	Combine tables of IRFs, dynamic-multiplier functions, and FEVDs	186
irf describe	Describe an IRF file	191
irf drop	Drop IRF results from the active IRF file	194
irf graph	Graph IRFs, dynamic-multiplier functions, and FEVDs	196
irf ograph	Graph overlaid IRFs, dynamic-multiplier functions, and FEVDs	202
irf rename	Rename an IRF result in an IRF file	208
irf set	Set the active IRF file	210
irf table	Create tables of IRFs, dynamic-multiplier functions, and FEVDs	212
newey	Regression with Newey–West standard errors	217
newey postestimation	Postestimation tools for newey	222
pergram	Periodogram	224
pperron	Phillips–Perron unit-root test	232
prais	Prais–Winsten and Cochrane–Orcutt regression	237
prais postestimation	Postestimation tools for prais	248
rolling	Rolling-window and recursive estimation	250
sspace	State-space models	258
sspace postestimation	Postestimation tools for sspace	283
tsappend	Add observations to a time-series dataset	290
tsfill	Fill in gaps in time variable	296
tsline	Plot time-series data	300
tsreport	Report time-series aspects of a dataset or estimation sample	305
tsrevar	Time-series operator programming command	308

tsset	Declare data to be time-series data	311
tssmooth	Smooth and forecast univariate time-series data	327
tssmooth exponential	Double-exponential smoothing	329
tssmooth exponential	Single-exponential smoothing	335
tssmooth hwhinters	Holt–Winters nonseasonal smoothing	343
tssmooth ma	Moving-average filter	350
tssmooth nl	Nonlinear filter	355
tssmooth shwhinters	Holt–Winters seasonal smoothing	357
var intro	Introduction to vector autoregressive models	366
var	Vector autoregressive models	373
var postestimation	Postestimation tools for var	385
var svar	Structural vector autoregressive models	388
var svar postestimation	Postestimation tools for svar	408
varbasic	Fit a simple VAR and graph IRFS or FEVDs	411
varbasic postestimation	Postestimation tools for varbasic	416
vargranger	Perform pairwise Granger causality tests after var or svar	419
varlmar	Perform LM test for residual autocorrelation after var or svar	423
varnorm	Test for normally distributed disturbances after var or svar	426
varsoc	Obtain lag-order selection statistics for VARs and VECMs	432
varstable	Check the stability condition of VAR or SVAR estimates	438
varwle	Obtain Wald lag-exclusion statistics after var or svar	443
vec intro	Introduction to vector error-correction models	448
vec	Vector error-correction models	467
vec postestimation	Postestimation tools for vec	491
veclmar	Perform LM test for residual autocorrelation after vec	494
vecnorm	Test for normally distributed disturbances after vec	497
vecrank	Estimate the cointegrating rank of a VECM	501
vecstable	Check the stability condition of VECM estimates	509
wntestb	Bartlett’s periodogram-based test for white noise	513
wntestq	Portmanteau (Q) test for white noise	518
xcorr	Cross-correlogram for bivariate time series	521
Glossary		525
Subject and author index		531

Cross-referencing the documentation

When reading this manual, you will find references to other Stata manuals. For example,

[U] [26 Overview of Stata estimation commands](#)

[R] [regress](#)

[D] [reshape](#)

The first example is a reference to chapter 26, *Overview of Stata estimation commands*, in the *User's Guide*; the second is a reference to the `regress` entry in the *Base Reference Manual*; and the third is a reference to the `reshape` entry in the *Data-Management Reference Manual*.

All the manuals in the Stata Documentation have a shorthand notation:

[GSM]	<i>Getting Started with Stata for Mac</i>
[GSU]	<i>Getting Started with Stata for Unix</i>
[GSW]	<i>Getting Started with Stata for Windows</i>
[U]	<i>Stata User's Guide</i>
[R]	<i>Stata Base Reference Manual</i>
[D]	<i>Stata Data-Management Reference Manual</i>
[G]	<i>Stata Graphics Reference Manual</i>
[XT]	<i>Stata Longitudinal-Data/Panel-Data Reference Manual</i>
[MI]	<i>Stata Multiple-Imputation Reference Manual</i>
[MV]	<i>Stata Multivariate Statistics Reference Manual</i>
[P]	<i>Stata Programming Reference Manual</i>
[SVY]	<i>Stata Survey Data Reference Manual</i>
[ST]	<i>Stata Survival Analysis and Epidemiological Tables Reference Manual</i>
[TS]	<i>Stata Time-Series Reference Manual</i>
[I]	<i>Stata Quick Reference and Index</i>
[M]	<i>Mata Reference Manual</i>

Detailed information about each of these manuals may be found online at

<http://www.stata-press.com/manuals/>

Title

intro — Introduction to time-series manual

Description

This entry describes this manual and what has changed since Stata 10.

Remarks

This manual documents Stata's time-series commands and is referred to as [TS] in cross-references.

After this entry, [TS] [time series](#) provides an overview of the `ts` commands. The other parts of this manual are arranged alphabetically. If you are new to Stata's time-series features, we recommend that you read the following sections first:

[TS] time series	Introduction to time-series commands
[TS] tsset	Declare a dataset to be time-series data

Stata is continually being updated, and Stata users are always writing new commands. To ensure that you have the latest features, you should install the most recent official update; see [R] [update](#).

What's new

1. New estimation command `sspace` estimates linear state-space models by maximum likelihood. In state-space models, the dependent variables are linear functions of unobserved states and observed exogenous variables. A few of the many models are VARMA models, structural time-series models, some linear dynamic models, and some stochastic general-equilibrium models. `sspace` can estimate the parameters of most linear time-series models with time-invariant parameters because they can be cast as state-space models. `sspace` can estimate stationary and nonstationary models. For stationary models, `sspace` uses the Kalman filter to estimate the observed states. For nonstationary models, `sspace` uses the De Jong diffuse Kalman filter. See [TS] [sspace](#).
2. New estimation command `dvech` estimates diagonal vech multivariate GARCH models. These models allow the conditional variance matrix of the dependent variables to follow a flexible dynamic structure in which each element of the current conditional variance matrix depends on its own past and on past shocks. See [TS] [dvech](#).
3. New estimation command `dfactor` estimates dynamic-factor models. These multivariate time-series models allow the dependent variables and the unobserved factor variables to have vector autoregressive (VAR) structures and to be linear functions of exogenous variables. See [TS] [dfactor](#).
4. Estimation commands `newey`, `prais`, `sspace`, `dvech`, and `dfactor` allow Stata's new factor-variable varlist notation; see [U] [11.4.3 Factor variables](#). Also, these estimation commands allow the standard set of factor-variable–related reporting options; see [R] [estimation options](#).
5. New postestimation command `margins`, which calculates marginal means, predictive margins, marginal effects, and average marginal effects, is available after all time-series estimation commands, except `svar`. See [R] [margins](#).
6. New display option `vsquish` for estimation commands, which allows you to control the spacing in output containing time-series operators or factor variables, is available after all time-series estimation commands. See [R] [estimation options](#).

7. New display option `coeflegend` for estimation commands, which displays the coefficients' legend showing how to specify them in an expression, is available after all time-series estimation commands. See [\[R\] estimation options](#).
8. `predict` after `regress` now allows time-series operators in option `dfbeta()`; see [\[R\] regress postestimation](#). Also allowing time-series operators are `regress` postestimation commands `estat szroeter`, `estat hettest`, `avplot`, and `avplots`. See [\[R\] regress postestimation](#).
9. Existing estimation commands `mlogit`, `ologit`, and `oprobit` now allow time-series operators; see [\[R\] mlogit](#), [\[R\] ologit](#), and [\[R\] oprobit](#).
10. Existing estimation commands `arch` and `arma` now accept maximization option `showtolerance`; see [\[R\] maximize](#).
11. Existing estimation command `arch` now allows you to fit models assuming that the disturbances follow Student's t distribution or the generalized error distribution, as well as the Gaussian (normal) distribution. Specify which distribution to use with option `distribution()`. You can specify the shape or degree-of-freedom parameter, or you can let `arch` estimate it along with the other parameters of the model. See [\[TS\] arch](#).
12. Existing command `tsappend` is now faster. See [\[TS\] tsappend](#).

For a complete list of all the new features in Stata 11, see [\[U\] 1.3 What's new](#).

Also see

[\[U\] 1.3 What's new](#)

[\[R\] intro](#) — Introduction to base reference manual

Description

The *Time-Series Reference Manual* organizes the commands alphabetically, making it easy to find individual command entries if you know the name of the command. This overview organizes and presents the commands conceptually, that is, according to the similarities in the functions that they perform.

The commands listed under the heading **Data-management tools and time-series operators** help you prepare your data for further analysis. The commands listed under the heading **Univariate time series** are grouped together because they are either estimators or filters designed for univariate time series or preestimation or postestimation commands that are conceptually related to one or more univariate time-series estimators. The commands listed under the heading **Multivariate time series** are similarly grouped together because they are either estimators designed for use with multivariate time series or preestimation or postestimation commands conceptually related to one or more multivariate time-series estimators. Within these three broad categories, similar commands have been grouped together.

(Continued on next page)

Data-management tools and time-series operators

<code>tsset</code>	Declare data to be time-series data
<code>tsfill</code>	Fill in gaps in time variable
<code>tsappend</code>	Add observations to a time-series dataset
<code>tsreport</code>	Report time-series aspects of a dataset or estimation sample
<code>tsrevar</code>	Time-series operator programming command
<code>haver</code>	Load data from Haver Analytics database
<code>rolling</code>	Rolling-window and recursive estimation

Univariate time series**Estimators**

<code>arima</code>	ARIMA, ARMAX, and other dynamic regression models
<code>arima postestimation</code>	Postestimation tools for arima
<code>arch</code>	Autoregressive conditional heteroskedasticity (ARCH) family of estimators
<code>arch postestimation</code>	Postestimation tools for arch
<code>newey</code>	Regression with Newey–West standard errors
<code>newey postestimation</code>	Postestimation tools for newey
<code>prais</code>	Prais–Winsten and Cochrane–Orcutt regression
<code>prais postestimation</code>	Postestimation tools for prais

Time-series smoothers and filters

<code>tssmooth ma</code>	Moving-average filter
<code>tssmooth dexponential</code>	Double-exponential smoothing
<code>tssmooth exponential</code>	Single-exponential smoothing
<code>tssmooth hwinters</code>	Holt–Winters nonseasonal smoothing
<code>tssmooth shwinters</code>	Holt–Winters seasonal smoothing
<code>tssmooth nl</code>	Nonlinear filter

Diagnostic tools

<code>corrgram</code>	Tabulate and graph autocorrelations
<code>xcorr</code>	Cross-correlogram for bivariate time series
<code>cumsp</code>	Cumulative spectral distribution
<code>pergram</code>	Periodogram
<code>dfgls</code>	DF-GLS unit-root test
<code>dfuller</code>	Augmented Dickey–Fuller unit-root test
<code>pperron</code>	Phillips–Perron unit-root test
<code>estat dwatson</code>	Durbin–Watson d statistic
<code>estat durbinalt</code>	Durbin’s alternative test for serial correlation
<code>estat bgodfrey</code>	Breusch–Godfrey test for higher-order serial correlation
<code>estat archlm</code>	Engle’s LM test for the presence of autoregressive conditional heteroskedasticity
<code>wntestb</code>	Bartlett’s periodogram-based test for white noise
<code>wntestq</code>	Portmanteau (Q) test for white noise

Multivariate time series

Estimators

<code>dfactor</code>	Dynamic-factor models
<code>dfactor postestimation</code>	Postestimation tools for <code>dfactor</code>
<code>dvech</code>	Diagonal vech multivariate GARCH models
<code>dvech postestimation</code>	Postestimation tools for <code>dvech</code>
<code>sspace</code>	State-space models
<code>sspace postestimation</code>	Postestimation tools for <code>sspace</code>
<code>var</code>	Vector autoregressive models
<code>var postestimation</code>	Postestimation tools for <code>var</code>
<code>svar</code>	Structural vector autoregressive models
<code>svar postestimation</code>	Postestimation tools for <code>svar</code>
<code>varbasic</code>	Fit a simple VAR and graph IRFs or FEVDs
<code>varbasic postestimation</code>	Postestimation tools for <code>varbasic</code>
<code>vec</code>	Vector error-correction models
<code>vec postestimation</code>	Postestimation tools for <code>vec</code>

Diagnostic tools

<code>varlmar</code>	Perform LM test for residual autocorrelation after <code>var</code> or <code>svar</code>
<code>varnorm</code>	Test for normally distributed disturbances after <code>var</code> or <code>svar</code>
<code>varsoc</code>	Obtain lag-order selection statistics for VARs and VECMs
<code>varstable</code>	Check the stability condition of VAR or SVAR estimates
<code>varwle</code>	Obtain Wald lag-exclusion statistics after <code>var</code> or <code>svar</code>
<code>veclmar</code>	Perform LM test for residual autocorrelation after <code>vec</code>
<code>vecnorm</code>	Test for normally distributed disturbances after <code>vec</code>
<code>vecrank</code>	Estimate the cointegrating rank of a VECM
<code>vecstable</code>	Check the stability condition of VECM estimates

Forecasting, inference, and interpretation

<code>irf create</code>	Obtain IRFs, dynamic-multiplier functions, and FEVDs
<code>fcast compute</code>	Compute dynamic forecasts of dependent variables after <code>var</code> , <code>svar</code> , or <code>vec</code>
<code>vargranger</code>	Perform pairwise Granger causality tests after <code>var</code> or <code>svar</code>

Graphs and tables

<code>corrgram</code>	Tabulate and graph autocorrelations
<code>xcorr</code>	Cross-correlogram for bivariate time series
<code>pergram</code>	Periodogram
<code>irf graph</code>	Graph IRFs, dynamic-multiplier functions, and FEVDs
<code>irf cgraph</code>	Combine graphs of IRFs, dynamic-multiplier functions, and FEVDs
<code>irf ograph</code>	Graph overlaid IRFs, dynamic-multiplier functions, and FEVDs
<code>irf table</code>	Create tables of IRFs, dynamic-multiplier functions, and FEVDs
<code>irf ctable</code>	Combine tables of IRFs, dynamic-multiplier functions, and FEVDs
<code>fcast graph</code>	Graph forecasts of variables computed by <code>fcast compute</code>
<code>tsline</code>	Plot time-series data
<code>tsrline</code>	Plot time-series range plot data
<code>varstable</code>	Check the stability condition of VAR or SVAR estimates
<code>vecstable</code>	Check the stability condition of VECM estimates
<code>wntestb</code>	Bartlett's periodogram-based test for white noise

Results management tools

<code>irf add</code>	Add results from an IRF file to the active IRF file
<code>irf describe</code>	Describe an IRF file
<code>irf drop</code>	Drop IRF results from the active IRF file
<code>irf rename</code>	Rename an IRF result in an IRF file
<code>irf set</code>	Set the active IRF file

Remarks

Remarks are presented under the following headings:

Data-management tools and time-series operators

Univariate time series

Estimators

Time-series smoothers and filters

Diagnostic tools

Multivariate time series

Estimators

Diagnostic tools

We also offer a NetCourse on Stata's time-series capabilities; see <http://www.stata.com/netcourse/nc461.html>.

Data-management tools and time-series operators

Because time-series estimators are, by definition, a function of the temporal ordering of the observations in the estimation sample, Stata's time-series commands require the data to be sorted and indexed by time, using the `tsset` command, before they can be used. `tsset` is simply a way for you to tell Stata which variable in your dataset represents time; `tsset` then sorts and indexes the data appropriately for use with the time-series commands. Once your dataset has been `tsset`, you can use Stata's time-series operators in data manipulation or programming using that dataset and when specifying the syntax for most time-series commands. Stata has time-series operators for representing the lags, leads, differences, and seasonal differences of a variable. The time-series operators are documented in [TS] `tsset`.

`tsset` can also be used to declare that your dataset contains cross-sectional time-series data, often referred to as panel data. When you use `tsset` to declare your dataset to contain panel data, you specify a variable that identifies the panels, as well as identifying the time variable. Once your dataset has been `tsset` as panel data, the time-series operators work appropriately for the data.

`tsfill`, which is documented in [TS] `tsfill`, can be used after `tsset` to fill in missing times with missing observations. `tsset` will report any gaps in your data, and `tsreport` will provide more details about the gaps. `tsappend` adds observations to a time-series dataset by using the information set by `tsset`. This function can be particularly useful when you wish to predict out of sample after fitting a model with a time-series estimator. `tsrevar` is a programmer's command that provides a way to use *varlists* that contain time-series operators with commands that do not otherwise support time-series operators.

The `haver` commands documented in [TS] `haver` allow you to load and describe the contents of a Haver Analytics (<http://www.haver.com>) file.

`rolling` performs rolling regressions, recursive regressions, and reverse recursive regressions. Any command that saves results in `e()` or `r()` can be used with `rolling`.

Univariate time series

Estimators

The four univariate time-series estimators currently available in Stata are `arima`, `arch`, `newey`, and `prais`. The last two, `newey` and `prais`, are really just extensions to ordinary linear regression. When you fit a linear regression on time-series data via ordinary least squares (OLS), if the disturbances are autocorrelated, the parameter estimates are usually consistent, but the estimated standard errors tend to be biased downward. Several estimators have been developed to deal with this problem. One strategy is to use OLS for estimating the regression parameters and use a different estimator for the variances, one that is consistent in the presence of autocorrelated disturbances, such as the Newey–West estimator implemented in `newey`. Another strategy is to attempt to model the dynamics of the disturbances. The estimators found in `prais`, `arima`, and `arch` are based on such a strategy.

`prais` implements two such estimators: the Prais–Winsten and the Cochrane–Orcutt generalized least-squares (GLS) estimators. These estimators are GLS estimators, but they are fairly restrictive in that they permit only first-order autocorrelation in the disturbances. Although they have certain pedagogical and historical value, they are somewhat obsolete. Faster computers with more memory have made it possible to implement full information maximum likelihood (FIML) estimators, such as Stata’s `arima` command. These estimators permit much greater flexibility when modeling the disturbances and are more efficient estimators.

`arima` provides the means to fit linear models with autoregressive moving-average (ARMA) disturbances, or in the absence of linear predictors, autoregressive integrated moving-average (ARIMA) models. This means that, whether you think that your data are best represented as a distributed-lag model, a transfer-function model, or a stochastic difference equation, or you simply wish to apply a Box–Jenkins filter to your data, the model can be fit using `arima`. `arch`, a conditional maximum likelihood estimator, has similar modeling capabilities for the mean of the time series but can also model autoregressive conditional heteroskedasticity in the disturbances with a wide variety of specifications for the variance equation.

Time-series smoothers and filters

In addition to the estimators mentioned above, Stata also provides six time-series filters or smoothers. Included are a simple, uniformly weighted, moving-average filter with unit weights; a weighted moving-average filter in which you can specify the weights; single- and double-exponential smoothers; Holt–Winters seasonal and nonseasonal smoothers; and a nonlinear smoother.

Most of these smoothers were originally developed as ad hoc procedures and are used for reducing the noise in a time series (smoothing) or forecasting. Although they have limited application for signal extraction, these smoothers have all been found to be optimal for some underlying modern time-series model.

Diagnostic tools

Stata’s time-series commands also include several preestimation and postestimation diagnostic commands. `corrgram` estimates the autocorrelation function and partial autocorrelation function of a univariate time series, as well as Q statistics. These functions and statistics are often used to determine the appropriate model specification before fitting ARIMA models. `corrgram` can also be used with `wntestb` and `wntestq` to examine the residuals after fitting a model for evidence of model misspecification. Stata’s time-series commands also include the commands `pergram` and `cumsp`, which provide the log-standardized periodogram and the cumulative-sample spectral distribution, respectively, for time-series analysts who prefer to estimate in the frequency domain rather than the time domain.

`xcorr` estimates the cross-correlogram for bivariate time series and can similarly be used both for preestimation and postestimation. For example, the cross-correlogram can be used before fitting a transfer-function model to produce initial estimates of the IRF. This estimate can then be used to determine the optimal lag length of the input series to include in the model specification. It can also be used as a postestimation tool after fitting a transfer function. The cross-correlogram between the residual from a transfer-function model and the prewhitened input series of the model can be examined for evidence of model misspecification.

When you fit ARMA or ARIMA models, the dependent variable being modeled must be covariance stationary (ARMA models), or the order of integration must be known (ARIMA models). Stata has three commands that can test for the presence of a unit root in a time-series variable: `dfuller` performs the augmented Dickey–Fuller test, `pperron` performs the Phillips–Perron test, and `dfgls` performs a modified Dickey–Fuller test.

The remaining diagnostic tools for univariate time series are for use after fitting a linear model via OLS with Stata’s `regress` command. They are documented collectively in [R] [regress postestimation time series](#). They include `estat dwatson`, `estat durbinalt`, `estat bgodfrey`, and `estat archlm`. `estat dwatson` computes the Durbin–Watson d statistic to test for the presence of first-order autocorrelation in the OLS residuals. `estat durbinalt` likewise tests for the presence of autocorrelation in the residuals. By comparison, however, Durbin’s alternative test is more general and easier to use than the Durbin–Watson test. With `estat durbinalt`, you can test for higher orders of autocorrelation, the assumption that the covariates in the model are strictly exogenous is relaxed, and there is no need to consult tables to compute rejection regions, as you must with the Durbin–Watson test. `estat bgodfrey` computes the Breusch–Godfrey test for autocorrelation in the residuals, and although the computations are different, the test in `estat bgodfrey` is asymptotically equivalent to the test in `estat durbinalt`. Finally, `estat archlm` performs Engle’s LM test for the presence of autoregressive conditional heteroskedasticity.

Multivariate time series

Estimators

Stata provides commands for fitting the most widely applied multivariate time-series models. `var` and `svar` fit vector autoregressive and structural vector autoregressive models to stationary data. `vec` fits cointegrating vector error-correction models. `dfactor` fits dynamic-factor models. `dvech` fits diagonal vech multivariate GARCH models. `sspace` fits state-space models. Many linear time-series models, including vector autoregressive moving average (VARMA) models and structural time-series models, can be cast as state-space models and fit by `sspace`.

Diagnostic tools

Before fitting a multivariate time-series model, you must specify the number of lags to include. `varsoc` produces statistics for determining the order of a VAR or VECM.

Several postestimation commands perform the most common specification analysis on a previously fitted VAR or SVAR. You can use `varlmar` to check for serial correlation in the residuals, `varnorm` to test the null hypothesis that the disturbances come from a multivariate normal distribution, and `varstable` to see if the fitted VAR or SVAR is stable. Two common types of inference about VAR models are whether one variable Granger-causes another and whether a set of lags can be excluded from the model. `vargranger` reports Wald tests of Granger causation, and `varwle` reports Wald lag exclusion tests.

Similarly, several postestimation commands perform the most common specification analysis on a previously fitted VECM. You can use `vecldmar` to check for serial correlation in the residuals, `vecnorm` to test the null hypothesis that the disturbances come from a multivariate normal distribution, and `vecstable` to analyze the stability of the previously fitted VECM.

VARs and VECMs are often fit to produce baseline forecasts. `fcast` produces dynamic forecasts from previously fitted VARs and VECMs.

Many researchers fit VARs, SVARs, and VECMs because they want to analyze how unexpected shocks affect the dynamic paths of the variables. Stata has a suite of `irf` commands for estimating IRF functions and interpreting, presenting, and managing these estimates; see [TS] [irf](#).

References

- Baum, C. F. 2005. *Stata: The language of choice for time-series analysis?* *Stata Journal* 5: 46–63.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Pisati, M. 2001. [sg162: Tools for spatial data analysis](#). *Stata Technical Bulletin* 60: 21–37. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 277–298. College Station, TX: Stata Press.
- Stock, J. H., and M. W. Watson. 2001. Vector autoregressions. *Journal of Economic Perspectives* 15: 101–115.

Also see

[U] [1.3 What's new](#)

[R] [intro](#) — Introduction to base reference manual

Syntax

arch *depvar* [*indepvars*] [*if*] [*in*] [*weight*] [, *options*]

<i>options</i>	description
Model	
<u>no</u> constant	suppress constant term
arch(<i>numlist</i>)	ARCH terms
<u>g</u> arch(<i>numlist</i>)	GARCH terms
<u>s</u> arch(<i>numlist</i>)	simple asymmetric ARCH terms
<u>t</u> arch(<i>numlist</i>)	threshold ARCH terms
<u>a</u> arch(<i>numlist</i>)	asymmetric ARCH terms
<u>n</u> arch(<i>numlist</i>)	nonlinear ARCH terms
narchk(<i>numlist</i>)	nonlinear ARCH terms with single shift
<u>a</u> barch(<i>numlist</i>)	absolute value ARCH terms
<u>a</u> tarch(<i>numlist</i>)	absolute threshold ARCH terms
<u>s</u> d <u>g</u> arch(<i>numlist</i>)	lags of σ_t
<u>e</u> arch(<i>numlist</i>)	news terms in Nelson's (1991) EGARCH model
<u>e</u> garch(<i>numlist</i>)	lags of $\ln(\sigma_t^2)$
<u>p</u> arch(<i>numlist</i>)	power ARCH terms
<u>t</u> parch(<i>numlist</i>)	threshold power ARCH terms
<u>a</u> parch(<i>numlist</i>)	asymmetric power ARCH terms
<u>n</u> parch(<i>numlist</i>)	nonlinear power ARCH terms
nparchk(<i>numlist</i>)	nonlinear power ARCH terms with single shift
<u>p</u> garch(<i>numlist</i>)	power GARCH terms
<u>c</u> onstraints(<i>constraints</i>)	apply specified linear constraints
<u>c</u> ollinear	keep collinear variables
Model 2	
archm	include ARCH-in-mean term in the mean-equation specification
<u>arch</u> mlags(<i>numlist</i>)	include specified lags of conditional variance in mean equation
<u>arch</u> mexp(<i>exp</i>)	apply transformation in <i>exp</i> to any ARCH-in-mean terms
arima(<i>#_p</i> , <i>#_d</i> , <i>#_q</i>)	specify ARIMA(<i>p</i> , <i>d</i> , <i>q</i>) model for dependent variable
ar(<i>numlist</i>)	autoregressive terms of the structural model disturbance
ma(<i>numlist</i>)	moving-average terms of the structural model disturbances
Model 3	
<u>d</u> istribution(<i>dist</i> [#])	use <i>dist</i> distribution for errors (may be <u>g</u> aussian, <u>n</u> ormal, t, or <u>g</u> ed; default is <u>g</u> aussian)
het(<i>varlist</i>)	include <i>varlist</i> in the specification of the conditional variance
<u>s</u> avespace	conserve memory during estimation

Priming	
<code>arch0(xb)</code>	compute priming values on the basis of the expected unconditional variance; the default
<code>arch0(xb0)</code>	compute priming values on the basis of the estimated variance of the residuals from OLS
<code>arch0(xbwt)</code>	compute priming values on the basis of the weighted sum of squares from OLS residuals
<code>arch0(xb0wt)</code>	compute priming values on the basis of the weighted sum of squares from OLS residuals, with more weight at earlier times
<code>arch0(zero)</code>	set priming values of ARCH terms to zero
<code>arch0(#)</code>	set priming values of ARCH terms to #
<code>arma0(zero)</code>	set all priming values of ARMA terms to zero; the default
<code>arma0(p)</code>	begin estimation after observation p , where p is the maximum AR lag in model
<code>arma0(q)</code>	begin estimation after observation q , where q is the maximum MA lag in model
<code>arma0(pq)</code>	begin estimation after observation $(p + q)$
<code>arma0(#)</code>	set priming values of ARMA terms to #
<code>condobs(#)</code>	set conditioning observations at the start of the sample to #
SE/Robust	
<code>vce(vcetype)</code>	<i>vcetype</i> may be <code>opg</code> , <code>robust</code> , or <code>oim</code>
Reporting	
<code>level(#)</code>	set confidence level; default is <code>level(95)</code>
<code>detail</code>	report list of gaps in time series
<code>nocnsreport</code>	do not display constraints
<code>display_options</code>	control spacing
Maximization	
<code>maximize_options</code>	control the maximization process; seldom used
[†] <code>coeflegend</code>	display coefficients' legend instead of coefficient table

[†]`coeflegend` does not appear in the dialog box.

You must `tsset` your data before using `arch`; see [TS] `tsset`.

`depvar` and `varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.

`by`, `rolling`, `statsby`, and `xi` are allowed; see [U] 11.1.10 Prefix commands.

`iweights` are allowed; see [U] 11.1.6 weight.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

To fit an ARCH($\#_m$) model with Gaussian errors, type

```
. arch depvar ... , arch(1/#_m)
```

To fit a GARCH($\#_m, \#_k$) model assuming that the errors follow Student's t distribution with 7 degrees of freedom, type

```
. arch depvar ... , arch(1/#_m) garch(1/#_k) distribution(t 7)
```

You can also fit many other models.

Details of syntax

The basic model arch fits is

$$\begin{aligned} y_t &= \mathbf{x}_t \boldsymbol{\beta} + \epsilon_t \\ \text{Var}(\epsilon_t) &= \sigma_t^2 = \gamma_0 + A(\boldsymbol{\sigma}, \boldsymbol{\epsilon}) + B(\boldsymbol{\sigma}, \boldsymbol{\epsilon})^2 \end{aligned} \quad (1)$$

The y_t equation may optionally include ARCH-in-mean and ARMA terms:

$$y_t = \mathbf{x}_t \boldsymbol{\beta} + \sum_i \psi_i g(\sigma_{t-i}^2) + \text{ARMA}(p, q) + \epsilon_t$$

If no options are specified, $A() = B() = 0$, and the model collapses to linear regression. The following options add to $A()$ (α , γ , and κ represent parameters to be estimated):

Option	Terms added to $A()$
arch()	$A() = A() + \alpha_{1,1}\epsilon_{t-1}^2 + \alpha_{1,2}\epsilon_{t-2}^2 + \dots$
garch()	$A() = A() + \alpha_{2,1}\sigma_{t-1}^2 + \alpha_{2,2}\sigma_{t-2}^2 + \dots$
saarch()	$A() = A() + \alpha_{3,1}\epsilon_{t-1} + \alpha_{3,2}\epsilon_{t-2} + \dots$
tarch()	$A() = A() + \alpha_{4,1}\epsilon_{t-1}^2(\epsilon_{t-1} > 0) + \alpha_{4,2}\epsilon_{t-2}^2(\epsilon_{t-2} > 0) + \dots$
aarch()	$A() = A() + \alpha_{5,1}(\epsilon_{t-1} + \gamma_{5,1}\epsilon_{t-1})^2 + \alpha_{5,2}(\epsilon_{t-2} + \gamma_{5,2}\epsilon_{t-2})^2 + \dots$
narch()	$A() = A() + \alpha_{6,1}(\epsilon_{t-1} - \kappa_{6,1})^2 + \alpha_{6,2}(\epsilon_{t-2} - \kappa_{6,2})^2 + \dots$
narchk()	$A() = A() + \alpha_{7,1}(\epsilon_{t-1} - \kappa_7)^2 + \alpha_{7,2}(\epsilon_{t-2} - \kappa_7)^2 + \dots$

The following options add to $B()$:

Option	Terms added to $B()$
abarch()	$B() = B() + \alpha_{8,1} \epsilon_{t-1} + \alpha_{8,2} \epsilon_{t-2} + \dots$
atarch()	$B() = B() + \alpha_{9,1} \epsilon_{t-1} (\epsilon_{t-1} > 0) + \alpha_{9,2} \epsilon_{t-2} (\epsilon_{t-2} > 0) + \dots$
sdgarch()	$B() = B() + \alpha_{10,1}\sigma_{t-1} + \alpha_{10,2}\sigma_{t-2} + \dots$

Each option requires a *numlist* argument (see [U] 11.1.8 *numlist*), which determines the lagged terms included. **arch(1)** specifies $\alpha_{1,1}\epsilon_{t-1}^2$, **arch(2)** specifies $\alpha_{1,2}\epsilon_{t-2}^2$, **arch(1,2)** specifies $\alpha_{1,1}\epsilon_{t-1}^2 + \alpha_{1,2}\epsilon_{t-2}^2$, **arch(1/3)** specifies $\alpha_{1,1}\epsilon_{t-1}^2 + \alpha_{1,2}\epsilon_{t-2}^2 + \alpha_{1,3}\epsilon_{t-3}^2$, etc.

If the **earch()** or **egarch()** option is specified, the basic model fit is

$$\begin{aligned} y_t &= \mathbf{x}_t \boldsymbol{\beta} + \sum_i \psi_i g(\sigma_{t-i}^2) + \text{ARMA}(p, q) + \epsilon_t \\ \ln \text{Var}(\epsilon_t) &= \ln \sigma_t^2 = \gamma_0 + C(\ln \boldsymbol{\sigma}, \mathbf{z}) + A(\boldsymbol{\sigma}, \boldsymbol{\epsilon}) + B(\boldsymbol{\sigma}, \boldsymbol{\epsilon})^2 \end{aligned} \quad (2)$$

where $z_t = \epsilon_t / \sigma_t$. $A()$ and $B()$ are given as above, but $A()$ and $B()$ now add to $\ln \sigma_t^2$ rather than σ_t^2 . (The options corresponding to $A()$ and $B()$ are rarely specified here.) $C()$ is given by

Option	Terms added to $C()$
<code>earch()</code>	$C() = C() + \alpha_{11,1}z_{t-1} + \gamma_{11,1}(z_{t-1} - \sqrt{2/\pi})$ $+ \alpha_{11,2}z_{t-2} + \gamma_{11,2}(z_{t-2} - \sqrt{2/\pi}) + \dots$
<code>egarch()</code>	$C() = C() + \alpha_{12,1}\ln\sigma_{t-1}^2 + \alpha_{12,2}\ln\sigma_{t-2}^2 + \dots$

Instead, if the `parch()`, `tparch()`, `aparch()`, `nparch()`, `nparchk()`, or `pgarch()` options are specified, the basic model fit is

$$y_t = \mathbf{x}_t\boldsymbol{\beta} + \sum_i \psi_i g(\sigma_{t-i}^2) + \text{ARMA}(p, q) + \epsilon_t \quad (3)$$

$$\{\text{Var}(\epsilon_t)\}^{\varphi/2} = \sigma_t^\varphi = \gamma_0 + D(\boldsymbol{\sigma}, \boldsymbol{\epsilon}) + A(\boldsymbol{\sigma}, \boldsymbol{\epsilon}) + B(\boldsymbol{\sigma}, \boldsymbol{\epsilon})^2$$

where φ is a parameter to be estimated. $A()$ and $B()$ are given as above, but $A()$ and $B()$ now add to σ_t^φ . (The options corresponding to $A()$ and $B()$ are rarely specified here.) $D()$ is given by

Option	Terms added to $D()$
<code>parch()</code>	$D() = D() + \alpha_{13,1}\epsilon_{t-1}^\varphi + \alpha_{13,2}\epsilon_{t-2}^\varphi + \dots$
<code>tparch()</code>	$D() = D() + \alpha_{14,1}\epsilon_{t-1}^\varphi(\epsilon_{t-1} > 0) + \alpha_{14,2}\epsilon_{t-2}^\varphi(\epsilon_{t-2} > 0) + \dots$
<code>aparch()</code>	$D() = D() + \alpha_{15,1}(\epsilon_{t-1} + \gamma_{15,1}\epsilon_{t-1})^\varphi + \alpha_{15,2}(\epsilon_{t-2} + \gamma_{15,2}\epsilon_{t-2})^\varphi + \dots$
<code>nparch()</code>	$D() = D() + \alpha_{16,1} \epsilon_{t-1} - \kappa_{16,1} ^\varphi + \alpha_{16,2} \epsilon_{t-2} - \kappa_{16,2} ^\varphi + \dots$
<code>nparchk()</code>	$D() = D() + \alpha_{17,1} \epsilon_{t-1} - \kappa_{17} ^\varphi + \alpha_{17,2} \epsilon_{t-2} - \kappa_{17} ^\varphi + \dots$
<code>pgarch()</code>	$D() = D() + \alpha_{18,1}\sigma_{t-1}^\varphi + \alpha_{18,2}\sigma_{t-2}^\varphi + \dots$

Common models

Common term	Options to specify
ARCH (Engle 1982)	<code>arch()</code>
GARCH (Bollerslev 1986)	<code>arch()</code> <code>garch()</code>
ARCH-in-mean (Engle, Lilien, and Robins 1987)	<code>archm</code> <code>arch()</code> [<code>garch()</code>]
GARCH with ARMA terms	<code>arch()</code> <code>garch()</code> <code>ar()</code> <code>ma()</code>
EGARCH (Nelson 1991)	<code>earch()</code> <code>egarch()</code>
TARCH, threshold ARCH (Zakoian 1994)	<code>abarch()</code> <code>atarch()</code> <code>sdgarch()</code>
GJR, form of threshold ARCH (Glosten, Jagannathan, and Runkle 1993)	<code>arch()</code> <code>tarch()</code> [<code>garch()</code>]
SAARCH, simple asymmetric ARCH (Engle 1990)	<code>arch()</code> <code>saarch()</code> [<code>garch()</code>]
PARCH, power ARCH (Higgins and Bera 1992)	<code>parch()</code> [<code>pgarch()</code>]
NARCH, nonlinear ARCH	<code>narch()</code> [<code>garch()</code>]
NARCHK, nonlinear ARCH with one shift	<code>narchk()</code> [<code>garch()</code>]
A-PARCH, asymmetric power ARCH (Ding, Granger, and Engle 1993)	<code>aparch()</code> [<code>pgarch()</code>]
NPARCH, nonlinear power ARCH	<code>nparch()</code> [<code>pgarch()</code>]

In all cases, you type

```
arch depvar [indepvars], options
```

where *options* are chosen from the table above. Each option requires that you specify as its argument a *numlist* that specifies the lags to be included. For most ARCH models, that value will be 1. For instance, to fit the classic first-order GARCH model on `cpi`, you would type

```
. arch cpi, arch(1) garch(1)
```

If you wanted to fit a first-order GARCH model of `cpi` on `wage`, you would type

```
. arch cpi wage, arch(1) garch(1)
```

If, for any of the options, you want first- and second-order terms, specify *optionname*(1/2). Specifying `garch(1) arch(1/2)` would fit a GARCH model with first- and second-order ARCH terms. If you specified `arch(2)`, only the lag 2 term would be included.

Reading arch output

The regression table reported by `arch` when using the normal distribution for the errors will appear as

op.depvar	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]
depvar					
x1	# ...				
x2					
L1.	# ...				
L2.	# ...				
_cons	# ...				
ARCHM					
sigma2	# ...				
ARMA					
ar					
L1.	# ...				
ma					
L1.	# ...				
HET					
z1	# ...				
z2					
L1.	# ...				
L2.	# ...				
ARCH					
arch					
L1.	# ...				
garch					
L1.	# ...				
aparch					
L1.	# ...				
etc.					
_cons	# ...				
POWER					
power	# ...				

Dividing lines separate “equations”.

The first one, two, or three equations report the mean model:

$$y_t = \mathbf{x}_t\boldsymbol{\beta} + \sum_i \psi_i g(\sigma_{t-i}^2) + \text{ARMA}(p, q) + \epsilon_t$$

The first equation reports $\boldsymbol{\beta}$, and the equation will be named `[depvar]`; if you fit a model on `d.cpi`, the first equation would be named `[cpi]`. In Stata, the coefficient on `x1` in the above example could be referred to as `[depvar]_b[x1]`. The coefficient on the lag 2 value of `x2` would be referred to as `[depvar]_b[L2.x2]`. Such notation would be used, for instance, in a later `test` command; see [\[R\] test](#).

The [ARCHM] equation reports the ψ coefficients if your model includes ARCH-in-mean terms; see options discussed under the **Model 2** tab above. Most ARCH-in-mean models include only a contemporaneous variance term, so the term $\sum_i \psi_i g(\sigma_{t-i}^2)$ becomes $\psi \sigma_t^2$. The coefficient ψ will be [ARCHM]_b[sigma2]. If your model includes lags of σ_t^2 , the additional coefficients will be [ARCHM]_b[L1.sigma2], and so on. If you specify a transformation $g()$ (option archmexp()), the coefficients will be [ARCHM]_b[sigma2ex], [ARCHM]_b[L1.sigma2ex], and so on. sigma2ex refers to $g(\sigma_t^2)$, the transformed value of the conditional variance.

The [ARMA] equation reports the ARMA coefficients if your model includes them; see options discussed under the **Model 2** tab above. This equation includes one or two “variables” named ar and ma. In later test statements, you could refer to the coefficient on the first lag of the autoregressive term by typing [ARMA]_b[L1.ar] or simply [ARMA]_b[L.ar] (the L operator is assumed to be lag 1 if you do not specify otherwise). The second lag on the moving-average term, if there were one, could be referred to by typing [ARMA]_b[L2.ma].

The next one, two, or three equations report the variance model.

The [HET] equation reports the multiplicative heteroskedasticity if the model includes it; see *Other options affecting specification of variance*. When you fit such a model, you specify the variables (and their lags), determining the multiplicative heteroskedasticity; after estimation, their coefficients are simply [HET]_b[op.varname].

The [ARCH] equation reports the ARCH, GARCH, etc., terms by referring to “variables” arch, garch, and so on. For instance, if you specified arch(1) garch(1) when you fit the model, the conditional variance is given by $\sigma_t^2 = \gamma_0 + \alpha_{1,1}\epsilon_{t-1}^2 + \alpha_{2,1}\sigma_{t-1}^2$. The coefficients would be named [ARCH]_b[_cons] (γ_0), [ARCH]_b[L.arch] ($\alpha_{1,1}$), and [ARCH]_b[L.garch] ($\alpha_{2,1}$).

The [POWER] equation appears only if you are fitting a variance model in the form of (3) above; the estimated φ is the coefficient [POWER]_b[power].

Also, if you use the distribution() option and specify either Student’s t or the generalized error distribution but do not specify the degree-of-freedom or shape parameter, then you will see two additional rows in the table. The final row contains the estimated degree-of-freedom or shape parameter. Immediately preceding the final row is a transformed version of the parameter that arch used during estimation to ensure that the degree-of-freedom parameter is greater than two or that the shape parameter is positive.

The naming convention for estimated ARCH, GARCH, etc., parameters is as follows (definitions for parameters α_i , γ_i , and κ_i can be found in the tables for $A()$, $B()$, $C()$, and $D()$ above):

Option	1st parameter	2nd parameter	Common parameter
arch()	$\alpha_1 = [\text{ARCH}]\_b[\text{arch}]$		
garch()	$\alpha_2 = [\text{ARCH}]\_b[\text{garch}]$		
saarch()	$\alpha_3 = [\text{ARCH}]\_b[\text{saarch}]$		
tarch()	$\alpha_4 = [\text{ARCH}]\_b[\text{tarch}]$		
aarch()	$\alpha_5 = [\text{ARCH}]\_b[\text{aarch}]$	$\gamma_5 = [\text{ARCH}]\_b[\text{aarch\_e}]$	
narch()	$\alpha_6 = [\text{ARCH}]\_b[\text{narch}]$	$\kappa_6 = [\text{ARCH}]\_b[\text{narch\_k}]$	
narchk()	$\alpha_7 = [\text{ARCH}]\_b[\text{narch}]$	$\kappa_7 = [\text{ARCH}]\_b[\text{narch\_k}]$	
abarch()	$\alpha_8 = [\text{ARCH}]\_b[\text{abarch}]$		
atarch()	$\alpha_9 = [\text{ARCH}]\_b[\text{atarch}]$		
sdgarch()	$\alpha_{10} = [\text{ARCH}]\_b[\text{sdgarch}]$		
earch()	$\alpha_{11} = [\text{ARCH}]\_b[\text{earch}]$	$\gamma_{11} = [\text{ARCH}]\_b[\text{earch\_a}]$	
egarch()	$\alpha_{12} = [\text{ARCH}]\_b[\text{egarch}]$		
parch()	$\alpha_{13} = [\text{ARCH}]\_b[\text{parch}]$		$\varphi = [\text{POWER}]\_b[\text{power}]$
tparch()	$\alpha_{14} = [\text{ARCH}]\_b[\text{tparch}]$		$\varphi = [\text{POWER}]\_b[\text{power}]$
aparch()	$\alpha_{15} = [\text{ARCH}]\_b[\text{aparch}]$	$\gamma_{15} = [\text{ARCH}]\_b[\text{aparch\_e}]$	$\varphi = [\text{POWER}]\_b[\text{power}]$
nparch()	$\alpha_{16} = [\text{ARCH}]\_b[\text{nparch}]$	$\kappa_{16} = [\text{ARCH}]\_b[\text{nparch\_k}]$	$\varphi = [\text{POWER}]\_b[\text{power}]$
nparchk()	$\alpha_{17} = [\text{ARCH}]\_b[\text{nparch}]$	$\kappa_{17} = [\text{ARCH}]\_b[\text{nparch\_k}]$	$\varphi = [\text{POWER}]\_b[\text{power}]$
pgarch()	$\alpha_{18} = [\text{ARCH}]\_b[\text{pgarch}]$		$\varphi = [\text{POWER}]\_b[\text{power}]$

Menu

ARCH/GARCH

Statistics > Time series > ARCH/GARCH > ARCH and GARCH models

EARCH/EGARCH

Statistics > Time series > ARCH/GARCH > Nelson's EGARCH model

ABARCH/ATARCH/SDGARCH

Statistics > Time series > ARCH/GARCH > Threshold ARCH model

ARCH/TARCH/GARCH

Statistics > Time series > ARCH/GARCH > GJR form of threshold ARCH model

ARCH/SAARCH/GARCH

Statistics > Time series > ARCH/GARCH > Simple asymmetric ARCH model

PARCH/PGARCH

Statistics > Time series > ARCH/GARCH > Power ARCH model

NARCH/GARCH

Statistics > Time series > ARCH/GARCH > Nonlinear ARCH model

NARCHK/GARCH

Statistics > Time series > ARCH/GARCH > Nonlinear ARCH model with one shift

APARCH/PGARCH

Statistics > Time series > ARCH/GARCH > Asymmetric power ARCH model

NPARCH/PGARCH

Statistics > Time series > ARCH/GARCH > Nonlinear power ARCH model

Description

`arch` fits regression models in which the volatility of a series varies through time. Usually, periods of high and low volatility are grouped together. ARCH models estimate future volatility as a function of prior volatility. To accomplish this, `arch` fits models of autoregressive conditional heteroskedasticity (ARCH) by using conditional maximum likelihood. In addition to ARCH terms, models may include multiplicative heteroskedasticity. Gaussian (normal), Student's t , and generalized error distributions are supported.

Concerning the regression equation itself, models may also contain ARCH-in-mean and ARMA terms.

Options

Model

`noconstant`; see [R] [estimation options](#).

`arch(numlist)` specifies the ARCH terms (lags of ϵ_t^2).

Specify `arch(1)` to include first-order terms, `arch(1/2)` to specify first- and second-order terms, `arch(1/3)` to specify first-, second-, and third-order terms, etc. Terms may be omitted. Specify `arch(1/3 5)` to specify terms with lags 1, 2, 3, and 5. All the options work this way.

`arch()` may not be specified with `aarch()`, `narch()`, `narchk()`, `nparchk()`, or `nparch()`, as this would result in collinear terms.

`garch(numlist)` specifies the GARCH terms (lags of σ_t^2).

`saarch(numlist)` specifies the simple asymmetric ARCH terms. Adding these terms is one way to make the standard ARCH and GARCH models respond asymmetrically to positive and negative innovations. Specifying `saarch()` with `arch()` and `garch()` corresponds to the SAARCH model of Engle (1990).

`saarch()` may not be specified with `narch()`, `narchk()`, `nparchk()`, or `nparch()`, as this would result in collinear terms.

`tarch(numlist)` specifies the threshold ARCH terms. Adding these is another way to make the standard ARCH and GARCH models respond asymmetrically to positive and negative innovations. Specifying `tarch()` with `arch()` and `garch()` corresponds to one form of the GJR model (Glosten, Jagannathan, and Runkle 1993).

`tarch()` may not be specified with `tparch()` or `aarch()`, as this would result in collinear terms.

`aarch(numlist)` specifies the lags of the two-parameter term $\alpha_i(|\epsilon_t| + \gamma_i \epsilon_t)^2$. This term provides the same underlying form of asymmetry as including `arch()` and `tarch()`, but it is expressed in a different way.

`aarch()` may not be specified with `arch()` or `tarch()`, as this would result in collinear terms.

`narch(numlist)` specifies the lags of the two-parameter term $\alpha_i(\epsilon_t - \kappa_i)^2$. This term allows the minimum conditional variance to occur at a value of lagged innovations other than zero. For any term specified at lag L , the minimum contribution to conditional variance of that lag occurs when $\epsilon_{t-L}^2 = \kappa_L$ —the squared innovations at that lag are equal to the estimated constant κ_L .

`narch()` may not be specified with `arch()`, `saarch()`, `narchk()`, `nparchk()`, or `nparch()`, as this would result in collinear terms.

`narchk(numlist)` specifies the lags of the two-parameter term $\alpha_i(\epsilon_t - \kappa)^2$; this is a variation of `narch()` with κ held constant for all lags.

`narchk()` may not be specified with `arch()`, `saarch()`, `narch()`, `nparchk()`, or `nparch()`, as this would result in collinear terms.

`abarch(numlist)` specifies lags of the term $|\epsilon_t|$.

`atarch(numlist)` specifies lags of $|\epsilon_t|(\epsilon_t > 0)$, where $(\epsilon_t > 0)$ represents the indicator function returning 1 when true and 0 when false. Like the TARCH terms, these ATARCH terms allow the effect of unanticipated innovations to be asymmetric about zero.

`sdgarch(numlist)` specifies lags of σ_t . Combining `atarch()`, `abarch()`, and `sdgarch()` produces the model by [Zakoian \(1994\)](#) that the author called the TARCH model. The acronym TARCH, however, refers to any model using thresholding to obtain asymmetry.

`earch(numlist)` specifies lags of the two-parameter term $\alpha z_t + \gamma(|z_t| - \sqrt{2/\pi})$. These terms represent the influence of news—lagged innovations—in Nelson’s (1991) EGARCH model. For these terms, $z_t = \epsilon_t/\sigma_t$, and `arch` assumes $z_t \sim N(0, 1)$. Nelson derived the general form of an EGARCH model for any assumed distribution and performed estimation assuming a generalized error distribution (GED). See [Hamilton \(1994\)](#) for a derivation where z_t is assumed normal. The z_t terms can be parameterized in either of these two equivalent ways. `arch` uses Nelson’s original parameterization; see [Hamilton \(1994\)](#) for an equivalent alternative.

`egarch(numlist)` specifies lags of $\ln(\sigma_t^2)$.

For the following options, the model is parameterized in terms of $h(\epsilon_t)^\varphi$ and σ_t^φ . One φ is estimated, even when more than one option is specified.

`parch(numlist)` specifies lags of $|\epsilon_t|^\varphi$. `parch()` combined with `pgarch()` corresponds to the class of nonlinear models of conditional variance suggested by [Higgins and Bera \(1992\)](#).

`tparch(numlist)` specifies lags of $(\epsilon_t > 0)|\epsilon_t|^\varphi$, where $(\epsilon_t > 0)$ represents the indicator function returning 1 when true and 0 when false. As with `tarch()`, `tparch()` specifies terms that allow for a differential impact of “good” (positive innovations) and “bad” (negative innovations) news for lags specified by `numlist`.

`tparch()` may not be specified with `tarch()`, as this would result in collinear terms.

`aparch(numlist)` specifies lags of the two-parameter term $\alpha(|\epsilon_t| + \gamma\epsilon_t)^\varphi$. This asymmetric power ARCH model, A-PARCH, was proposed by [Ding, Granger, and Engle \(1993\)](#) and corresponds to a Box–Cox function in the lagged innovations. The authors fit the original A-PARCH model on more than 16,000 daily observations of the Standard and Poor’s 500, and for good reason. As the number of parameters and the flexibility of the specification increase, more data are required to estimate the parameters of the conditional heteroskedasticity. See [Ding, Granger, and Engle \(1993\)](#) for a discussion of how seven popular ARCH models nest within the A-PARCH model.

When γ goes to 1, the full term goes to zero for many observations and can then be numerically unstable.

`nparch(numlist)` specifies lags of the two-parameter term $\alpha|\epsilon_t - \kappa_i|^\varphi$.

`nparch()` may not be specified with `arch()`, `saarch()`, `narch()`, `narchk()`, or `nparchk()`, as this would result in collinear terms.

`nparchk(numlist)` specifies lags of the two-parameter term $\alpha|\epsilon_t - \kappa|^\varphi$; this is a variation of `nparch()` with κ held constant for all lags. This is the direct analog of `narchk()`, except for the power of φ . `nparchk()` corresponds to an extended form of the model of [Higgins and Bera \(1992\)](#) as

presented by Bollerslev, Engle, and Nelson (1994). `nparchk()` would typically be combined with the `pgarch()` option.

`nparchk()` may not be specified with `arch()`, `saarch()`, `narch()`, `narchk()`, or `nparch()`, as this would result in collinear terms.

`pgarch(numlist)` specifies lags of σ_t^φ .

`constraints(constraints)`, `collinear`; see [R] [estimation options](#).

Model 2

`archm` specifies that an ARCH-in-mean term be included in the specification of the mean equation. This term allows the expected value of *depvar* to depend on the conditional variance. ARCH-in-mean is most commonly used in evaluating financial time series when a theory supports a tradeoff between asset risk and return. By default, no ARCH-in-mean terms are included in the model.

`archm` specifies that the contemporaneous expected conditional variance be included in the mean equation. For example, typing

```
. arch y x, archm arch(1)
```

specifies the model

$$y_t = \beta_0 + \beta_1 x_t + \psi \sigma_t^2 + \epsilon_t$$

$$\sigma_t^2 = \gamma_0 + \gamma \epsilon_{t-1}^2$$

`archmlags(numlist)` is an expansion of `archm` that includes lags of the conditional variance σ_t^2 in the mean equation. To specify a contemporaneous and once-lagged variance, specify either `archm archmlags(1)` or `archmlags(0/1)`.

`archmexp(exp)` applies the transformation in *exp* to any ARCH-in-mean terms in the model. The expression should contain an *X* wherever a value of the conditional variance is to enter the expression. This option can be used to produce the commonly used ARCH-in-mean of the conditional standard deviation. With the example from `archm`, typing

```
. arch y x, archm arch(1) archmexp(sqrt(X))
```

specifies the mean equation $y_t = \beta_0 + \beta_1 x_t + \psi \sigma_t + \epsilon_t$. Alternatively, typing

```
. arch y x, archm arch(1) archmexp(1/sqrt(X))
```

specifies $y_t = \beta_0 + \beta_1 x_t + \psi / \sigma_t + \epsilon_t$.

`arima(#p, #d, #q)` is an alternative, shorthand notation for specifying autoregressive models in the dependent variable. The dependent variable and any independent variables are differenced #_d times, 1 through #_p lags of autocorrelations are included, and 1 through #_q lags of moving averages are included. For example, the specification

```
. arch y, arima(2,1,3)
```

is equivalent to

```
. arch D.y, ar(1/2) ma(1/3)
```

The former is easier to write for classic ARIMA models of the mean equation, but it is not nearly as expressive as the latter. If gaps in the AR or MA lags are to be modeled, or if different operators are to be applied to independent variables, the latter syntax is required.

`ar(numlist)` specifies the autoregressive terms of the structural model disturbance to be included in the model. For example, `ar(1/3)` specifies that lags 1, 2, and 3 of the structural disturbance be included in the model. `ar(1,4)` specifies that lags 1 and 4 be included, possibly to account for quarterly effects.

If the model does not contain regressors, these terms can also be considered autoregressive terms for the dependent variable; see [TS] [arima](#).

`ma(numlist)` specifies the moving-average terms to be included in the model. These are the terms for the lagged innovations or white-noise disturbances.

Model 3

`distribution(dist [#])` specifies the distribution to assume for the error term. *dist* may be `gaussian`, `normal`, `t`, or `ged`. `gaussian` and `normal` are synonyms, and `#` cannot be specified with them.

If `distribution(t)` is specified, `arch` assumes that the errors follow Student's *t* distribution, and the degree-of-freedom parameter is estimated along with the other parameters of the model. If `distribution(t #)` is specified, then `arch` uses Student's *t* distribution with `#` degrees of freedom. `#` must be greater than 2.

If `distribution(ged)` is specified, `arch` assumes that the errors have a generalized error distribution, and the shape parameter is estimated along with the other parameters of the model. If `distribution(ged #)` is specified, then `arch` uses the generalized error distribution with shape parameter `#`. `#` must be positive. The generalized error distribution is identical to the normal distribution when the shape parameter equals 2.

`het(varlist)` specifies that *varlist* be included in the specification of the conditional variance. *varlist* may contain time-series operators. This *varlist* enters the variance specification collectively as multiplicative heteroskedasticity; see Judge et al. (1985). If `het()` is not specified, the model will not contain multiplicative heteroskedasticity.

Assume that the conditional variance depends on variables *x* and *w* and has an ARCH(1) component. We request this specification by using the `het(x w) arch(1)` options, and this corresponds to the conditional-variance model

$$\sigma_t^2 = \exp(\lambda_0 + \lambda_1 \mathbf{x}_t + \lambda_2 \mathbf{w}_t) + \alpha \epsilon_{t-1}^2$$

Multiplicative heteroskedasticity enters differently with an EGARCH model because the variance is already specified in logs. For the `het(x w) earch(1) egarch(1)` options, the variance model is

$$\ln(\sigma_t^2) = \lambda_0 + \lambda_1 \mathbf{x}_t + \lambda_2 \mathbf{w}_t + \alpha z_{t-1} + \gamma(|z_{t-1}| - \sqrt{2/\pi}) + \delta \ln(\sigma_{t-1}^2)$$

`savespace` conserves memory by retaining only those variables required for estimation. The original dataset is restored after estimation. This option is rarely used and should be specified only if there is insufficient memory to fit a model without the option. `arch` requires considerably more temporary storage during estimation than most estimation commands in Stata.

Priming

`arch0(cond_method)` is a rarely used option that specifies how to compute the conditioning (presample or priming) values for σ_t^2 and ϵ_t^2 . In the presample period, it is assumed that $\sigma_t^2 = \epsilon_t^2$ and that this value is constant. If `arch0()` is not specified, the priming values are computed as the expected unconditional variance given the current estimates of the β coefficients and any ARMA parameters.

`arch0(xb)`, the default, specifies that the priming values are the expected unconditional variance of the model, which is $\sum_1^T \hat{\epsilon}_t^2 / T$, where $\hat{\epsilon}_t$ is computed from the mean equation and any ARMA terms.

`arch0(xb0)` specifies that the priming values are the estimated variance of the residuals from an OLS estimate of the mean equation.

`arch0(xbwt)` specifies that the priming values are the weighted sum of the $\hat{\epsilon}_t^2$ from the current conditional mean equation (and ARMA terms) that places more weight on estimates of ϵ_t^2 at the beginning of the sample.

`arch0(xb0wt)` specifies that the priming values are the weighted sum of the $\hat{\epsilon}_t^2$ from an OLS estimate of the mean equation (and ARMA terms) that places more weight on estimates of ϵ_t^2 at the beginning of the sample.

`arch0(zero)` specifies that the priming values are 0. Unlike the priming values for ARIMA models, 0 is generally not a consistent estimate of the presample conditional variance or squared innovations.

`arch0(#)` specifies that $\sigma_t^2 = \epsilon_t^2 = \#$ for any specified nonnegative $\#$. Thus `arch0(0)` is equivalent to `arch0(zero)`.

`arma0(cond_method)` is a rarely used option that specifies how the ϵ_t values are initialized at the beginning of the sample for the ARMA component, if the model has one. This option has an effect only when AR or MA terms are included in the model (the `ar()`, `ma()`, or `arma()` options specified).

`arma0(zero)`, the default, specifies that all priming values of ϵ_t be taken as 0. This fits the model over the entire requested sample and takes ϵ_t as its expected value of 0 for all lags required by the ARMA terms; see Judge et al. (1985).

`arma0(p)`, `arma0(q)`, and `arma0(pq)` specify that estimation begin after priming the recursions for a certain number of observations. `p` specifies that estimation begin after the p th observation in the sample, where p is the maximum AR lag in the model; `q` specifies that estimation begin after the q th observation in the sample, where q is the maximum MA lag in the model; and `pq` specifies that estimation begin after the $(p + q)$ th observation in the sample.

During the priming period, the recursions necessary to generate predicted disturbances are performed, but results are used only to initialize preestimation values of ϵ_t . To understand the definition of preestimation, say that you fit a model in 10/100. If the model is specified with `ar(1,2)`, preestimation refers to observations 10 and 11.

The ARCH terms σ_t^2 and ϵ_t^2 are also updated over these observations. Any required lags of ϵ_t before the priming period are taken to be their expected value of 0, and ϵ_t^2 and σ_t^2 take the values specified in `arch0()`.

`arma0(#)` specifies that the presample values of ϵ_t are to be taken as $\#$ for all lags required by the ARMA terms. Thus `arma0(0)` is equivalent to `arma0(zero)`.

`condobs(#)` is a rarely used option that specifies a fixed number of conditioning observations at the start of the sample. Over these priming observations, the recursions necessary to generate predicted disturbances are performed, but only to initialize preestimation values of ϵ_t , ϵ_t^2 , and σ_t^2 . Any required lags of ϵ_t before the initialization period are taken to be their expected value of 0 (or the value specified in `arma0()`), and required values of ϵ_t^2 and σ_t^2 assume the values specified by `arch0()`. `condobs()` can be used if conditioning observations are desired for the lags in the ARCH terms of the model. If `arma()` is also specified, the maximum number of conditioning observations required by `arma()` and `condobs(#)` is used.

SE/Robust

`vce(vcetype)` specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification and that are derived from asymptotic theory; see [R] [vce_option](#).

For ARCH models, the robust or quasi-maximum likelihood estimates (QMLE) of variance are robust to symmetric nonnormality in the disturbances. The robust variance estimates generally are not robust to functional misspecification of the mean equation; see [Bollerslev and Wooldridge \(1992\)](#).

The robust variance estimates computed by `arch` are based on the full Huber/White/sandwich formulation, as discussed in [P] [_robust](#). Many other software packages report robust estimates that set some terms to their expectations of zero ([Bollerslev and Wooldridge 1992](#)), which saves them from calculating second derivatives of the log-likelihood function.

Reporting

`level(#)`; see [R] [estimation options](#).

`detail` specifies that a detailed list of any gaps in the series be reported, including gaps due to missing observations or missing data for the dependent variable or independent variables.

`nocnsreport`; see [R] [estimation options](#).

`display_options`: `vsquish`; see [R] [estimation options](#).

Maximization

`maximize_options`: `difficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `gtolerance(#)`, `nrtolerance(#)`, `nonnrtolerance`, `from(init_specs)`; see [R] [maximize](#) for all options except `gtolerance()`, and see below for information on `gtolerance()`.

These options are often more important for ARCH models than for other maximum likelihood models because of convergence problems associated with ARCH models—ARCH model likelihoods are notoriously difficult to maximize.

Setting `technique()` to something other than the default or BHHH changes the `vcetype` to `vce(oim)`.

The following options are all related to maximization and are either particularly important in fitting ARCH models or not available for most other estimators.

`gtolerance(#)` specifies the tolerance for the gradient relative to the coefficients. When $|g_i b_i| \leq \text{gtolerance}()$ for all parameters b_i and the corresponding elements of the gradient g_i , the gradient tolerance criterion is met. The default gradient tolerance for `arch` is `gtolerance(.05)`.

`gtolerance(999)` may be specified to disable the gradient criterion. If the optimizer becomes stuck with repeated “(backed up)” messages, the gradient probably still contains substantial values, but an uphill direction cannot be found for the likelihood. With this option, results can often be obtained, but whether the global maximum likelihood has been found is unclear.

When the maximization is not going well, it is also possible to set the maximum number of iterations (see [R] [maximize](#)) to the point where the optimizer appears to be stuck and to inspect the estimation results at that point.

`from(init_specs)` specifies the initial values of the coefficients. ARCH models may be sensitive to initial values and may have coefficient values that correspond to local maximums. The default starting values are obtained via a series of regressions, producing results that, on the basis of asymptotic theory, are consistent for the β and ARMA parameters and generally

reasonable for the rest. Nevertheless, these values may not always be feasible in that the likelihood function cannot be evaluated at the initial values `arch` first chooses. In such cases, the estimation is restarted with ARCH and ARMA parameters initialized to zero. It is possible, but unlikely, that even these values will be infeasible and that you will have to supply initial values yourself.

The standard syntax for `from()` accepts a matrix, a list of values, or coefficient name value pairs; see [R] [maximize](#). `arch` also allows the following:

`from(archb0)` sets the starting value for all the ARCH/GARCH/... parameters in the conditional-variance equation to 0.

`from(armab0)` sets the starting value for all ARMA parameters in the model to 0.

`from(archb0 armab0)` sets the starting value for all ARCH/GARCH/... and ARMA parameters to 0.

The following option is available with `arch` but is not shown in the dialog box:

`coeflegend`; see [R] [estimation options](#).

Remarks

The volatility of a series is not constant through time; periods of relatively low volatility and periods of relatively high volatility tend to be grouped together. This is a commonly observed characteristic of economic time series and is even more pronounced in many frequently sampled financial series. ARCH models seek to estimate this time-dependent volatility as a function of observed prior volatility. Sometimes the model of volatility is of more interest than the model of the conditional mean. As implemented in `arch`, the volatility model may also include regressors to account for a structural component in the volatility—usually referred to as multiplicative heteroskedasticity.

ARCH models were introduced by [Engle \(1982\)](#) in a study of inflation rates, and there has since been a barrage of proposed parametric and nonparametric specifications of autoregressive conditional heteroskedasticity. Overviews of the literature can found in [Bollerslev, Engle, and Nelson \(1994\)](#) and [Bollerslev, Chou, and Kroner \(1992\)](#). Introductions to basic ARCH models appear in many general econometrics texts, including [Davidson and MacKinnon \(1993, 2004\)](#), [Greene \(2008\)](#), [Kmenta \(1997\)](#), [Stock and Watson \(2007\)](#), and [Wooldridge \(2009\)](#). [Harvey \(1989\)](#) and [Enders \(2004\)](#) provide introductions to ARCH in the larger context of econometric time-series modeling, and [Hamilton \(1994\)](#) gives considerably more detail in the same context.

`arch` fits models of autoregressive conditional heteroskedasticity (ARCH, GARCH, etc.) using conditional maximum likelihood. By “conditional”, we mean that the likelihood is computed based on an assumed or estimated set of priming values for the squared innovations ϵ_t^2 and variances σ_t^2 prior to the estimation sample; see [Hamilton \(1994\)](#) or [Bollerslev \(1986\)](#). Sometimes more conditioning is done on the first a , g , or $a + g$ observations in the sample, where a is the maximum ARCH term lag and g is the maximum GARCH term lag (or the maximum lags from the other ARCH family terms).

The original ARCH model proposed by [Engle \(1982\)](#) modeled the variance of a regression model’s disturbances as a linear function of lagged values of the squared regression disturbances. We can write an ARCH(m) model as

$$\begin{aligned} y_t &= \mathbf{x}_t\boldsymbol{\beta} + \epsilon_t && \text{(conditional mean)} \\ \sigma_t^2 &= \gamma_0 + \gamma_1\epsilon_{t-1}^2 + \gamma_2\epsilon_{t-2}^2 + \cdots + \gamma_m\epsilon_{t-m}^2 && \text{(conditional variance)} \end{aligned}$$

where

ϵ_t^2 is the squared residuals (or innovations)

γ_i are the ARCH parameters

The ARCH model has a specification for both the conditional mean and the conditional variance, and the variance is a function of the size of prior unanticipated innovations— ϵ_t^2 . This model was generalized by [Bollerslev \(1986\)](#) to include lagged values of the conditional variance—a GARCH model. The GARCH(m, k) model is written as

$$y_t = \mathbf{x}_t \boldsymbol{\beta} + \epsilon_t$$

$$\sigma_t^2 = \gamma_0 + \gamma_1 \epsilon_{t-1}^2 + \gamma_2 \epsilon_{t-2}^2 + \cdots + \gamma_m \epsilon_{t-m}^2 + \delta_1 \sigma_{t-1}^2 + \delta_2 \sigma_{t-2}^2 + \cdots + \delta_k \sigma_{t-k}^2$$

where

γ_i are the ARCH parameters

δ_i are the GARCH parameters

In his pioneering work, [Engle \(1982\)](#) assumed that the error term, ϵ_t , followed a Gaussian (normal) distribution: $\epsilon_t \sim N(0, \sigma_t^2)$. However, as [Mandelbrot \(1963\)](#) and many others have noted, the distribution of stock returns appears to be leptokurtotic, meaning that extreme stock returns are more frequent than would be expected if the returns were normally distributed. Researchers have therefore assumed other distributions that can have fatter tails than the normal distribution; `arch` allows you to fit models assuming the errors follow Student's t distribution or the generalized error distribution. The t distribution has fatter tails than the normal distribution; as the degree-of-freedom parameter approaches infinity, the t distribution converges to the normal distribution. The generalized error distribution's tails are fatter than the normal distribution's when the shape parameter is less than two and are thinner than the normal distribution's when the shape parameter is greater than two.

The GARCH model of conditional variance can be considered an ARMA process in the squared innovations, although not in the variances as the equations might seem to suggest; see [Hamilton \(1994\)](#). Specifically, the standard GARCH model implies that the squared innovations result from

$$\epsilon_t^2 = \gamma_0 + (\gamma_1 + \delta_1) \epsilon_{t-1}^2 + (\gamma_2 + \delta_2) \epsilon_{t-2}^2 + \cdots + (\gamma_k + \delta_k) \epsilon_{t-k}^2 + w_t - \delta_1 w_{t-1} - \delta_2 w_{t-2} - \delta_3 w_{t-3}$$

where

$$w_t = \epsilon_t^2 - \sigma_t^2$$

w_t is a white-noise process that is fundamental for ϵ_t^2

One of the primary benefits of the GARCH specification is its parsimony in identifying the conditional variance. As with ARIMA models, the ARMA specification in GARCH allows the conditional variance to be modeled with fewer parameters than with an ARCH specification alone. Empirically, many series with a conditionally heteroskedastic disturbance have been adequately modeled with a GARCH(1,1) specification.

An ARMA process in the disturbances can easily be added to the mean equation. For example, the mean equation can be written with an ARMA(1,1) disturbance as

$$y_t = \mathbf{x}_t \boldsymbol{\beta} + \rho(y_{t-1} - \mathbf{x}_{t-1} \boldsymbol{\beta}) + \theta \epsilon_{t-1} + \epsilon_t$$

with an obvious generalization to ARMA(p, q) by adding terms; see [\[TS\] arima](#) for more discussion of this specification. This change affects only the conditional-variance specification in that ϵ_t^2 now results from a different specification of the conditional mean.

Much of the literature on ARCH models focuses on alternative specifications of the variance equation. `arch` allows many of these specifications to be requested using the `saarch()` through `pgarch()` options, which imply that one or more terms may be changed or added to the specification of the variance equation.

These alternative specifications also address asymmetry. Both the ARCH and GARCH specifications imply a symmetric impact of innovations. Whether an innovation ϵ_t^2 is positive or negative makes no difference to the expected variance σ_t^2 in the ensuing periods; only the size of the innovation matters—good news and bad news have the same effect. Many theories, however, suggest that positive and negative innovations should vary in their impact. For risk-averse investors, a large unanticipated drop in the market is more likely to lead to higher volatility than a large unanticipated increase (see [Black \[1976\]](#), [Nelson \[1991\]](#)). `saarch()`, `tarch()`, `aarch()`, `abarch()`, `earch()`, `aparch()`, and `tparch()` allow various specifications of asymmetric effects.

`narch()`, `narchk()`, `nparch()`, and `nparchk()` imply an asymmetric impact of a specific form. All the models considered so far have a minimum conditional variance when the lagged innovations are all zero. “No news is good news” when it comes to keeping the conditional variance small. `narch()`, `narchk()`, `nparch()`, and `nparchk()` also have a symmetric response to innovations, but they are not centered at zero. The entire news-response function (response to innovations) is shifted horizontally so that minimum variance lies at some specific positive or negative value for prior innovations.

ARCH-in-mean models allow the conditional variance of the series to influence the conditional mean. This is particularly convenient for modeling the risk–return relationship in financial series; the riskier an investment, with all else equal, the lower its expected return. ARCH-in-mean models modify the specification of the conditional mean equation to be

$$y_t = \mathbf{x}_t\boldsymbol{\beta} + \psi\sigma_t^2 + \epsilon_t$$

(ARCH-in-mean)

Although this linear form in the current conditional variance has dominated the literature, `arch` allows the conditional variance to enter the mean equation through a nonlinear transformation $g(\cdot)$ and for this transformed term to be included contemporaneously or lagged.

$$y_t = \mathbf{x}_t\boldsymbol{\beta} + \psi_0g(\sigma_t^2) + \psi_1g(\sigma_{t-1}^2) + \psi_2g(\sigma_{t-2}^2) + \cdots + \epsilon_t$$

Square root is the most commonly used $g(\cdot)$ transformation because researchers want to include a linear term for the conditional standard deviation, but any transform $g(\cdot)$ is allowed.

► Example 1: ARCH model

Consider a simple model of the U.S. Wholesale Price Index (WPI) ([Enders 2004](#), 87–93), which we also consider in [\[TS\] arima](#). The data are quarterly over the period 1960q1 through 1990q4.

In [\[TS\] arima](#), we fit a model of the continuously compounded rate of change in the WPI, $\ln(\text{WPI}_t) - \ln(\text{WPI}_{t-1})$. The graph of the differenced series—see [\[TS\] arima](#)—clearly shows periods of high volatility and other periods of relative tranquility. This makes the series a good candidate for ARCH modeling. Indeed, price indices have been a common target of ARCH models. [Engle \(1982\)](#) presented the original ARCH formulation in an analysis of U.K. inflation rates.

First, we fit a constant-only model by OLS and test ARCH effects by using Engle’s Lagrange-multiplier test (`estat archlm`).


```
. use http://www.stata-press.com/data/r11/wpil
```

```
. regress D.ln_wpi
```

Source	SS	df	MS	Number of obs = 123	
Model	0	0	.	F(0, 122) =	0.00
Residual	.02521709	122	.000206697	Prob > F =	.
				R-squared =	0.0000
				Adj R-squared =	0.0000
Total	.02521709	122	.000206697	Root MSE =	.01438

D.ln_wpi	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
_cons	.0108215	.0012963	8.35	0.000	.0082553	.0133878

```
. estat archlm, lags(1)
```

LM test for autoregressive conditional heteroskedasticity (ARCH)

lags(p)	chi2	df	Prob > chi2
1	8.366	1	0.0038

H0: no ARCH effects vs. H1: ARCH(p) disturbance

Because the LM test shows a p -value of 0.0038, which is well below 0.05, we reject the null hypothesis of no ARCH(1) effects. Thus we can further estimate the ARCH(1) parameter by specifying `arch(1)`. See [\[R\] regress postestimation time series](#) for more information on Engle's LM test.

The first-order generalized ARCH model (GARCH, [Bollerslev 1986](#)) is the most commonly used specification for the conditional variance in empirical work and is typically written GARCH(1,1). We can estimate a GARCH(1,1) process for the log-differenced series by typing

```
. arch D.ln_wpi, arch(1) garch(1)
```

```
(setting optimization to BHHH)
```

```
Iteration 0: log likelihood = 355.2346
```

```
Iteration 1: log likelihood = 365.64589
```

```
(output omitted)
```

```
Iteration 10: log likelihood = 373.23397
```

ARCH family regression

Sample: 1960q2 - 1990q4

Distribution: Gaussian

Log likelihood = 373.234

Number of obs = 123

Wald chi2(.) = .

Prob > chi2 = .

D.ln_wpi	OPG		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
ln_wpi						
_cons	.0061167	.0010616	5.76	0.000	.0040361	.0081974
ARCH						
arch						
L1.	.4364123	.2437428	1.79	0.073	-.0413147	.9141394
garch						
L1.	.4544606	.1866605	2.43	0.015	.0886126	.8203085
_cons	.0000269	.0000122	2.20	0.028	2.97e-06	.0000508

We have estimated the ARCH(1) parameter to be 0.436 and the GARCH(1) parameter to be 0.454, so our fitted GARCH(1,1) model is

$$y_t = 0.0061 + \epsilon_t$$
$$\sigma_t^2 = 0.436 \epsilon_{t-1}^2 + 0.454 \sigma_{t-1}^2$$

where $y_t = \ln(\text{wpi}_t) - \ln(\text{wpi}_{t-1})$.

The model Wald test and probability are both reported as missing (.). By convention, Stata reports the model test for the mean equation. Here and fairly often for ARCH models, the mean equation consists only of a constant, and there is nothing to test.

➤ Example 2: ARCH model with ARMA process

We can retain the GARCH(1, 1) specification for the conditional variance and model the mean as an ARMA process with AR(1) and MA(1) terms as well as a fourth-lag MA term to control for quarterly seasonal effects by typing

```
. arch D.ln_wpi, ar(1) ma(1 4) arch(1) garch(1)
(setting optimization to BHHH)
Iteration 0:  log likelihood = 380.99952
Iteration 1:  log likelihood = 388.57801
Iteration 2:  log likelihood = 391.34179
Iteration 3:  log likelihood = 396.37029
Iteration 4:  log likelihood = 398.01112
(switching optimization to BFGS)
Iteration 5:  log likelihood = 398.23657
BFGS stepping has contracted, resetting BFGS Hessian (0)
Iteration 6:  log likelihood = 399.21491
Iteration 7:  log likelihood = 399.21531 (backed up)
(output omitted)
(switching optimization to BHHH)
Iteration 15: log likelihood = 399.51441
Iteration 16: log likelihood = 399.51443
Iteration 17: log likelihood = 399.51443
ARCH family regression -- ARMA disturbances
Sample: 1960q2 - 1990q4                Number of obs   =      123
Distribution: Gaussian                  Wald chi2(3)    =     153.56
Log likelihood = 399.5144               Prob > chi2     =      0.0000
```

D.ln_wpi	OPG		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
ln_wpi						
_cons	.0069541	.0039517	1.76	0.078	-.000791	.0146992
ARMA						
ar						
L1.	.7922673	.1072225	7.39	0.000	.582115	1.002419
ma						
L1.	-.3417738	.1499944	-2.28	0.023	-.6357573	-.0477902
L4.	.2451725	.1251131	1.96	0.050	-.0000446	.4903896
ARCH						
arch						
L1.	.2040451	.1244992	1.64	0.101	-.039969	.4480591
garch						
L1.	.694968	.189218	3.67	0.000	.3241075	1.065828
_cons	.0000119	.0000104	1.14	0.253	-8.52e-06	.0000324

To clarify exactly what we have estimated, we could write our model as

$$y_t = 0.007 + 0.792 (y_{t-1} - 0.007) - 0.342 \epsilon_{t-1} + 0.245 \epsilon_{t-4} + \epsilon_t$$

$$\sigma_t^2 = 0.204 \epsilon_{t-1}^2 + .695 \sigma_{t-1}^2$$

where $y_t = \ln(\text{wpi}_t) - \ln(\text{wpi}_{t-1})$.

The ARCH(1) coefficient, 0.204, is not significantly different from zero, but the ARCH(1) and GARCH(1) coefficients are significant collectively. If you doubt this, you can check with `test`.

```
. test [ARCH]L1.arch [ARCH]L1.garch
( 1)  [ARCH]L.arch = 0
( 2)  [ARCH]L.garch = 0
      chi2( 2) =    84.92
      Prob > chi2 =    0.0000
```

(For comparison, we fit the model over the same sample used in the example in [TS] [arima](#); Enders fits this GARCH model but over a slightly different sample.)

◀

□ Technical note

The rather ugly iteration log on the previous result is typical, as difficulty in converging is common in ARCH models. This is actually a fairly well-behaved likelihood for an ARCH model. The “switching optimization to . . .” messages are standard messages from the default optimization method for `arch`. The “backed up” messages are typical of BFGS stepping as the BFGS Hessian is often overoptimistic, particularly during early iterations. These messages are nothing to be concerned about.

Nevertheless, watch out for the messages “BFGS stepping has contracted, resetting BFGS Hessian” and “backed up”, which can flag problems that may result in an iteration log that goes on and on. Stata will never report convergence and will never report final results. The question is, when do you give up and press *Break*, and if you do, what then?

If the “BFGS stepping has contracted” message occurs repeatedly (more than, say, five times), it often indicates that convergence will never be achieved. Literally, it means that the BFGS algorithm was stuck and reset its Hessian and take a steepest-descent step.

The “backed up” message, if it occurs repeatedly, also indicates problems, but only if the likelihood value is simultaneously not changing. If the message occurs repeatedly but the likelihood value is changing, as it did above, all is going well; it is just going slowly.

If you have convergence problems, you can specify options to assist the current maximization method or try a different method. Or, your model specification and data may simply lead to a likelihood that is not concave in the allowable region and thus cannot be maximized.

If you see the “backed up” message with no change in the likelihood, you can reset the gradient tolerance to a larger value. Specifying the `gtolerance(999)` option disables gradient checking, allowing convergence to be declared more easily. This does not guarantee that convergence will be declared, and even if it is, the global maximum likelihood may not have been found.

You can also try to specify initial values.

Finally, you can try a different maximization method; see options discussed under the [Maximization](#) tab above.

ARCH models are notorious for having convergence difficulties. Unlike in most estimators in Stata, it is common for convergence to require many steps or even to fail. This is particularly true of the explicitly nonlinear terms such as `aarch()`, `narch()`, `aparch()`, or `archm` (ARCH-in-mean), and of any model with several lags in the ARCH terms. There is not always a solution. You can try other maximization methods or different starting values, but if your data do not support your assumed ARCH structure, convergence simply may not be possible.

ARCH models can be susceptible to irrelevant regressors or unnecessary lags, whether in the specification of the conditional mean or in the conditional variance. In these situations, `arch` will often continue to iterate, making little to no improvement in the likelihood. We view this conservative approach as better than declaring convergence prematurely when the likelihood has not been fully maximized. `arch` is estimating the conditional form of second sample moments, often with flexible functions, and that is asking much of the data.

□

□ Technical note

if *exp* and *in range* are interpreted differently with commands accepting time-series operators. The time-series operators are resolved *before* the conditions are tested, which may lead to some confusion. Note the results of the following `list` commands:

```
. use http://www.stata-press.com/data/r11/archxmpl
. list t y l.y in 5/10
```

	t	y	L.y
5.	1961q1	30.8	30.7
6.	1961q2	30.5	30.8
7.	1961q3	30.5	30.5
8.	1961q4	30.6	30.5
9.	1962q1	30.7	30.6
10.	1962q2	30.6	30.7

```
. keep in 5/10
(118 observations deleted)
. list t y l.y
```

	t	y	L.y
1.	1961q1	30.8	.
2.	1961q2	30.5	30.8
3.	1961q3	30.5	30.5
4.	1961q4	30.6	30.5
5.	1962q1	30.7	30.6
6.	1962q2	30.6	30.7

We have one more lagged observation for `y` in the first case: `l.y` was resolved before the `in` restriction was applied. In the second case, the dataset no longer contains the value of `y` to compute the first lag. This means that

```
. use http://www.stata-press.com/data/r11/archxmpl, clear
. arch y l.x if twwithin(1962q2, 1990q3), arch(1)
```

is not the same as

```
. keep if twwithin(1962q2, 1990q3)
. arch y l.x, arch(1)
```



► Example 3: Asymmetric effects—EGARCH model

Continuing with the WPI data, we might be concerned that the economy as a whole responds differently to unanticipated increases in wholesale prices than it does to unanticipated decreases. Perhaps unanticipated increases lead to cash flow issues that affect inventories and lead to more volatility. We can see if the data support this supposition by specifying an ARCH model that allows an asymmetric effect of “news”—innovations or unanticipated changes. One of the most popular such models is EGARCH (Nelson 1991). The full first-order EGARCH model for the WPI can be specified as follows:

```
. use http://www.stata-press.com/data/r11/wpi1, clear
. arch D.ln_wpi, ar(1) ma(1 4) earch(1) egarch(1)
(setting optimization to BHHH)
Iteration 0:   log likelihood =   227.5251
Iteration 1:   log likelihood =  381.68456
(output omitted)
Iteration 23:  log likelihood =  405.31453
ARCH family regression -- ARMA disturbances
Sample: 1960q2 - 1990q4                Number of obs   =       123
Distribution: Gaussian                  Wald chi2(3)     =      156.01
Log likelihood =  405.3145              Prob > chi2      =       0.0000
```

D.ln_wpi		OPG		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.					
ln_wpi							
_cons	.008734	.0034003	2.57	0.010	.0020695	.0153984	
ARMA							
ar							
L1.	.7692058	.0968413	7.94	0.000	.5794003	.9590112	
ma							
L1.	-.3554634	.1265743	-2.81	0.005	-.6035445	-.1073823	
L4.	.2414641	.0863834	2.80	0.005	.0721558	.4107724	
ARCH							
earch							
L1.	.4063981	.1163538	3.49	0.000	.1783489	.6344474	
earch_a							
L1.	.2467486	.1233407	2.00	0.045	.0050052	.488492	
egarch							
L1.	.8417231	.0704095	11.95	0.000	.703723	.9797232	
_cons	-1.488461	.6604538	-2.25	0.024	-2.782927	-.1939952	

Our result for the variance is

$$\ln(\sigma_t^2) = -1.49 + .406 z_{t-1} + .247 (|z_{t-1}| - \sqrt{2/\pi}) + .842 \ln(\sigma_{t-1}^2)$$

where $z_t = \epsilon_t / \sigma_t$, which is distributed as $N(0, 1)$.

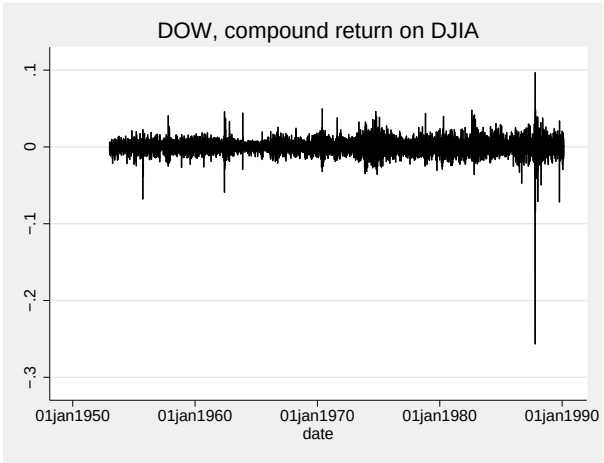
This is a strong indication for a leverage effect. The positive `L1.earch` coefficient implies that positive innovations (unanticipated price increases) are more destabilizing than negative innovations. The effect appears strong (0.406) and is substantially larger than the symmetric effect (0.247). In fact, the relative scales of the two coefficients imply that the positive leverage completely dominates the symmetric effect.

This can readily be seen if we plot what is often referred to as the news-response or news-impact function. This curve shows the resulting conditional variance as a function of unanticipated news, in the form of innovations, that is, the conditional variance σ_t^2 as a function of ϵ_t . Thus we must evaluate σ_t^2 for various values of ϵ_t —say, -4 to 4 —and then graph the result.

◀

➤ Example 4: Asymmetric power ARCH model

As an example of a frequently sampled, long-run series, consider the daily closing indices of the Dow Jones Industrial Average, variable `downclose`. To avoid the first half of the century, when the New York Stock Exchange was open for Saturday trading, only data after 1jan1953 are used. The compound return of the series is used as the dependent variable and is graphed below.



We formed this difference by referring to `D.ln_dow`, but only after playing a trick. The series is daily, and each observation represents the Dow closing index for the day. Our data included a time variable recorded as a daily date. We wanted, however, to model the log differences in the series, and we wanted the span from Friday to Monday to appear as a single-period difference. That is, the day before Monday is Friday. Because our dataset was `tsset` with `date`, the span from Friday to Monday was 3 days. The solution was to create a second variable that sequentially numbered the observations. By `tssetting` the data with this new variable, we obtained the desired differences.

```
. generate t = _n
. tsset t
```

Now our data look like this:

```
. use http://www.stata-press.com/data/r11/dow1
. generate dayofwk = dow(date)
. list date dayofwk t ln_dow D.ln_dow in 1/8
```

	date	dayofwk	t	ln_dow	D.ln_dow
1.	02jan1953	5	1	5.677096	.
2.	05jan1953	1	2	5.682899	.0058026
3.	06jan1953	2	3	5.677439	-.0054603
4.	07jan1953	3	4	5.672636	-.0048032
5.	08jan1953	4	5	5.671259	-.0013762
6.	09jan1953	5	6	5.661223	-.0100365
7.	12jan1953	1	7	5.653191	-.0080323
8.	13jan1953	2	8	5.659134	.0059433

```
. list date dayofwk t ln_dow D.ln_dow in -8/1
```

	date	dayofwk	t	ln_dow	D.ln_dow
9334.	08feb1990	4	9334	7.880188	.0016198
9335.	09feb1990	5	9335	7.881635	.0014472
9336.	12feb1990	1	9336	7.870601	-.011034
9337.	13feb1990	2	9337	7.872665	.0020638
9338.	14feb1990	3	9338	7.872577	-.0000877
9339.	15feb1990	4	9339	7.88213	.009553
9340.	16feb1990	5	9340	7.876863	-.0052676
9341.	20feb1990	2	9341	7.862054	-.0148082

The difference operator D spans weekends because the specified time variable, t , is not a true date and has a difference of 1 for all observations. We must leave this contrived time variable in place during estimation, or `arch` will be convinced that our dataset has gaps. If we were using calendar dates, we would indeed have gaps.

Ding, Granger, and Engle (1993) fit an A-PARCH model of daily returns of the Standard and Poor's 500 (S&P 500) for 3jan1928–30aug1991. We will fit the same model for the Dow data shown above. The model includes an AR(1) term as well as the A-PARCH specification of conditional variance.

(Continued on next page)

```
. arch D.ln_dow, ar(1) aparch(1) pgarch(1)
(setting optimization to BHHH)
Iteration 0:   log likelihood =   31139.553
Iteration 1:   log likelihood =   31350.751
(output omitted)
Iteration 60:  log likelihood =   32273.555
ARCH family regression -- AR disturbances
Sample: 2 - 9341                                Number of obs   =      9340
Distribution: Gaussian                          Wald chi2(1)    =      175.46
Log likelihood =   32273.56                      Prob > chi2     =       0.0000
```

D.ln_dow	OPG		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
ln_dow						
_cons	.0001786	.0000875	2.04	0.041	7.15e-06	.00035
ARMA						
ar						
L1.	.1410944	.0106519	13.25	0.000	.1202171	.1619717
ARCH						
aparch						
L1.	.0626324	.0034307	18.26	0.000	.0559083	.0693565
aparch_e						
L1.	-.3645107	.0378485	-9.63	0.000	-.4386924	-.2903289
pgarch						
L1.	.9299016	.0030998	299.99	0.000	.9238261	.9359771
_cons	7.19e-06	2.53e-06	2.84	0.004	2.23e-06	.0000121
POWER						
power	1.585178	.0629182	25.19	0.000	1.461861	1.708496

In the iteration log, the final iteration reports the message “backed up”. For most estimators, ending on a “backed up” message would be a cause for great concern, but not with `arch` or, for that matter, `arima`, as long as you do not specify the `gtolerance()` option. `arch` and `arima`, by default, monitor the gradient and declare convergence only if, in addition to everything else, the gradient is small enough.

The fitted model demonstrates substantial asymmetry, with the large negative `L1.aparch_e` coefficient indicating that the market responds with much more volatility to unexpected drops in returns (bad news) than it does to increases in returns (good news).



➤ Example 5: ARCH model with nonnormal errors

Stock returns tend to be leptokurtotic, meaning that large returns (either positive or negative) occur more frequently than one would expect if returns were in fact normally distributed. Here we refit the previous A-PARCH model assuming the errors follow the generalized error distribution, and we let `arch` estimate the shape parameter of the distribution.


```

. use http://www.stata-press.com/data/r11/dow1, clear
. arch D.ln_dow, ar(1) aparch(1) pgarch(1) distribution(ged)

(setting optimization to BHHH)
Iteration 0:   log likelihood =   31139.553
Iteration 1:   log likelihood =   31348.131
(output omitted)
Iteration 60:  log likelihood =   32486.461

ARCH family regression -- AR disturbances

Sample: 2 - 9341
Distribution: GED
Log likelihood =   32486.46
Number of obs   =       9340
Wald chi2(1)    =       178.21
Prob > chi2     =       0.0000

```

D.ln_dow	OPG		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
ln_dow _cons	.0002735	.0000078	3.51	0.000	.0001207	.0004264
ARMA ar L1.	.1337478	.0100188	13.35	0.000	.1141114	.1533842
ARCH aparch L1.	.0641769	.00494	12.99	0.000	.0544947	.073859
aparch_e L1.	-.4052149	.0573043	-7.07	0.000	-.5175293	-.2929005
pgarch L1.	.9341739	.0045667	204.56	0.000	.9252233	.9431245
_cons	.0000216	.0000117	1.84	0.066	-1.39e-06	.0000446
POWER power	1.325272	.1030696	12.86	0.000	1.123259	1.527285
/lnshape	.3527196	.0094816	37.20	0.000	.334136	.3713033
shape	1.422932	.0134917			1.396733	1.449623

The ARMA and ARCH coefficients are similar to those we obtained when we assumed normally distributed errors, though we do note that the power term is now closer to 1. The estimated shape parameter for the generalized error distribution is shown at the bottom of the output. Here the shape parameter is 1.42; because it is less than 2, the distribution of the errors has tails that are fatter than they would be if the errors were normally distributed.



► Example 6: ARCH model with constraints

Engle's (1982) original model, which sparked the interest in ARCH, provides an example requiring constraints. Most current ARCH specifications use GARCH terms to provide flexible dynamic properties without estimating an excessive number of parameters. The original model was limited to ARCH terms, and to help cope with the collinearity of the terms, a declining lag structure was imposed in the parameters. The conditional variance equation was specified as

$$\begin{aligned}\sigma_t^2 &= \alpha_0 + \alpha(.4 \epsilon_{t-1} + .3 \epsilon_{t-2} + .2 \epsilon_{t-3} + .1 \epsilon_{t-4}) \\ &= \alpha_0 + .4 \alpha \epsilon_{t-1} + .3 \alpha \epsilon_{t-2} + .2 \alpha \epsilon_{t-3} + .1 \alpha \epsilon_{t-4}\end{aligned}$$

From the earlier arch output, we know how the coefficients will be named. In Stata, the formula is

$$\begin{aligned}\sigma_t^2 &= [\text{ARCH}]\_cons + .4 [\text{ARCH}]L1.arch \epsilon_{t-1} + .3 [\text{ARCH}]L2.arch \epsilon_{t-2} \\ &\quad + .2 [\text{ARCH}]L3.arch \epsilon_{t-3} + .1 [\text{ARCH}]L4.arch \epsilon_{t-4}\end{aligned}$$

We could specify these linear constraints many ways, but the following seems fairly intuitive; see [\[R\] constraint](#) for syntax.

```
. use http://www.stata-press.com/data/r11/wpi1, clear
. constraint 1 (3/4)*[ARCH]l1.arch = [ARCH]l2.arch
. constraint 2 (2/4)*[ARCH]l1.arch = [ARCH]l3.arch
. constraint 3 (1/4)*[ARCH]l1.arch = [ARCH]l4.arch
```

The original model was fit on U.K. inflation; we will again use the WPI data and retain our earlier specification of the mean equation, which differs from Engle’s U.K. inflation model. With our constraints, we type

```
. arch D.ln_wpi, ar(1) ma(1 4) arch(1/4) constraints(1/3)
(setting optimization to BHHH)
Iteration 0:  log likelihood = 396.80192
Iteration 1:  log likelihood = 399.07808
(output omitted)
Iteration 9:  log likelihood = 399.46243
ARCH family regression -- ARMA disturbances
Sample: 1960q2 - 1990q4                Number of obs   =      123
Distribution: Gaussian                  Wald chi2(3)    =    123.32
Log likelihood = 399.4624               Prob > chi2     =     0.0000

( 1)  .75 [ARCH]L.arch - [ARCH]L2.arch = 0
( 2)  .5 [ARCH]L.arch - [ARCH]L3.arch = 0
( 3)  .25 [ARCH]L.arch - [ARCH]L4.arch = 0
```

D.ln_wpi	OPG		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
ln_wpi						
_cons	.0077204	.0034531	2.24	0.025	.0009525	.0144883
ARMA						
ar						
L1.	.7388168	.1126811	6.56	0.000	.517966	.9596676
ma						
L1.	-.2559691	.1442861	-1.77	0.076	-.5387646	.0268264
L4.	.2528922	.1140185	2.22	0.027	.02942	.4763644
ARCH						
arch						
L1.	.2180138	.0737787	2.95	0.003	.0734101	.3626174
L2.	.1635103	.055334	2.95	0.003	.0550576	.2719631
L3.	.1090069	.0368894	2.95	0.003	.0367051	.1813087
L4.	.0545034	.0184447	2.95	0.003	.0183525	.0906544
_cons	.0000483	7.66e-06	6.30	0.000	.0000333	.0000633

L1.arch, L2.arch, L3.arch, and L4.arch coefficients have the constrained relative sizes.

◀

Saved results

arch saves the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(N_gaps)</code>	number of gaps
<code>e(condobs)</code>	number of conditioning observations
<code>e(k)</code>	number of parameters
<code>e(k_eq)</code>	number of equations in <code>e(b)</code>
<code>e(k_eq_model)</code>	number of equations in model Wald test
<code>e(k_dv)</code>	number of dependent variables
<code>e(k_aux)</code>	number of auxiliary parameters
<code>e(df_m)</code>	model degrees of freedom
<code>e(ll)</code>	log likelihood
<code>e(chi2)</code>	χ^2 statistic
<code>e(p)</code>	significance
<code>e(archi)</code>	$\sigma_0^2 = \epsilon_0^2$, priming values
<code>e(archany)</code>	1 if model contains ARCH terms, 0 otherwise
<code>e(tdf)</code>	degrees of freedom for Student's t distribution
<code>e(shape)</code>	shape parameter for generalized error distribution
<code>e(tmin)</code>	minimum time
<code>e(tmax)</code>	maximum time
<code>e(power)</code>	φ for power ARCH terms
<code>e(rank)</code>	rank of <code>e(V)</code>
<code>e(ic)</code>	number of iterations
<code>e(rc)</code>	return code
<code>e(converged)</code>	1 if converged, 0 otherwise

Macros

e(cmd)	arch
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(covariates)	list of covariates
e(eqnames)	names of equations
e(wtype)	weight type
e(wexp)	weight expression
e(title)	title in estimation output
e(tmins)	formatted minimum time
e(tmaxs)	formatted maximum time
e(dist)	distribution for error term: <i>gaussian</i> , <i>t</i> , or <i>ged</i>
e(mhet)	1 if multiplicative heteroskedasticity
e(dfopt)	yes if degrees of freedom for <i>t</i> distribution or shape parameter for GED distribution was estimated; no otherwise
e(cond)	flag if called by <i>arima</i>
e(chi2type)	Wald; type of model χ^2 test
e(vce)	<i>vcetype</i> specified in <i>vce()</i>
e(vcetype)	title used to label Std. Err.
e(diparm#)	display transformed parameter #
e(ma)	lags for moving-average terms
e(ar)	lags for autoregressive terms
e(arch)	lags for ARCH terms
e(archm)	ARCH-in-mean lags
e(archmexp)	ARCH-in-mean exp
e(earch)	lags for EARCH terms
e(egarch)	lags for EGARCH terms
e(aarch)	lags for AARCH terms
e(narch)	lags for NARCH terms
e(aparch)	lags for A-PARCH terms
e(nparch)	lags for NPARCH terms
e(saarch)	lags for SAARCH terms
e(parch)	lags for PARCH terms
e(tparch)	lags for TPARCH terms
e(abarch)	lags for ABARCH terms
e(tarch)	lags for TARCH terms
e(sdgarch)	lags for SDGARCH terms
e(pgarch)	lags for PGARCH terms
e(garch)	lags for GARCH terms
e(opt)	type of optimization
e(ml_method)	type of ml method
e(user)	name of likelihood-evaluator program
e(technique)	maximization technique
e(tech)	maximization technique, including number of iterations
e(tech_steps)	number of iterations performed before switching techniques
e(crittype)	optimization criterion
e(properties)	b V
e(estat_cmd)	program used to implement <i>estat</i>
e(predict)	program used to implement <i>predict</i>
e(marginsok)	predictions allowed by <i>margins</i>
e(marginsnotok)	predictions disallowed by <i>margins</i>

Matrices	
<code>e(b)</code>	coefficient vector
<code>e(Cns)</code>	constraints matrix
<code>e(ilog)</code>	iteration log (up to 20 iterations)
<code>e(gradient)</code>	gradient vector
<code>e(V)</code>	variance–covariance matrix of the estimators
<code>e(V_modelbased)</code>	model-based variance
Functions	
<code>e(sample)</code>	marks estimation sample

Methods and formulas

`arch` is implemented as an ado-file using the `ml` commands; see [R] [ml](#).

The mean equation for the model fit by `arch` and with ARMA terms can be written as

$$y_t = \mathbf{x}_t\boldsymbol{\beta} + \sum_{i=1}^p \psi_i g(\sigma_{t-i}^2) + \sum_{j=1}^p \rho_j \left\{ y_{t-j} - x_{t-j}\boldsymbol{\beta} - \sum_{i=1}^p \psi_i g(\sigma_{t-j-i}^2) \right\} + \sum_{k=1}^q \theta_k \epsilon_{t-k} + \epsilon_t \quad (\text{conditional mean})$$

where

$\boldsymbol{\beta}$ are the regression parameters,

$\boldsymbol{\psi}$ are the ARCH-in-mean parameters,

$\boldsymbol{\rho}$ are the autoregression parameters,

$\boldsymbol{\theta}$ are the moving-average parameters, and

$g()$ is a general function, see the `archmexp()` option.

Any of the parameters in this full specification of the conditional mean may be zero. For example, the model need not have moving-average parameters ($\boldsymbol{\theta} = 0$) or ARCH-in-mean parameters ($\boldsymbol{\psi} = 0$).

The variance equation will be one of the following:

$$\sigma^2 = \gamma_0 + A(\boldsymbol{\sigma}, \boldsymbol{\epsilon}) + B(\boldsymbol{\sigma}, \boldsymbol{\epsilon})^2 \quad (1)$$

$$\ln \sigma_t^2 = \gamma_0 + C(\ln \boldsymbol{\sigma}, \mathbf{z}) + A(\boldsymbol{\sigma}, \boldsymbol{\epsilon}) + B(\boldsymbol{\sigma}, \boldsymbol{\epsilon})^2 \quad (2)$$

$$\sigma_t^\varphi = \gamma_0 + D(\boldsymbol{\sigma}, \boldsymbol{\epsilon}) + A(\boldsymbol{\sigma}, \boldsymbol{\epsilon}) + B(\boldsymbol{\sigma}, \boldsymbol{\epsilon})^2 \quad (3)$$

where $A(\boldsymbol{\sigma}, \boldsymbol{\epsilon})$, $B(\boldsymbol{\sigma}, \boldsymbol{\epsilon})$, $C(\ln \boldsymbol{\sigma}, \mathbf{z})$, and $D(\boldsymbol{\sigma}, \boldsymbol{\epsilon})$ are linear sums of the appropriate ARCH terms; see [Details of syntax](#) for more information. Equation (1) is used if no EGARCH or power ARCH terms are included in the model, (2) if EGARCH terms are included, and (3) if any power ARCH terms are included; see [Details of syntax](#).

Methods and formulas are presented under the following headings:

[Priming values](#)

[Likelihood from prediction error decomposition](#)

[Missing data](#)

Priming values

The above model is recursive with potentially long memory. It is necessary to assume preestimation sample values for ϵ_t , ϵ_t^2 , and σ_t^2 to begin the recursions, and the remaining computations are therefore conditioned on these priming values, which can be controlled using the `arch0()` and `arma0()` options. See options discussed under the **Priming** tab above.

The `arch0(xb0wt)` and `arch0(xbwt)` options compute a weighted sum of estimated disturbances with more weight on the early observations. With either of these options,

$$\sigma_{t_0-i}^2 = \epsilon_{t_0-i}^2 = (1 - .7) \sum_{t=0}^{T-1} .7^{T-t-1} \epsilon_{T-t}^2 \quad \forall i$$

where t_0 is the first observation for which the likelihood is computed; see options discussed under the **Priming** tab above. The ϵ_t^2 are all computed from the conditional mean equation. If `arch0(xb0wt)` is specified, β , ψ_i , ρ_j , and θ_k are taken from initial regression estimates and held constant during optimization. If `arch0(xbwt)` is specified, the current estimates of β , ψ_i , ρ_j , and θ_k are used to compute ϵ_t^2 on every iteration. If any ψ_i is in the mean equation (ARCH-in-mean is specified), the estimates of ϵ_t^2 from the initial regression estimates are not consistent.

Likelihood from prediction error decomposition

The likelihood function for ARCH has a particularly simple form. Given priming (or conditioning) values of ϵ_t , ϵ_t^2 , and σ_t^2 , the mean equation above can be solved recursively for every ϵ_t (prediction error decomposition). Likewise, the conditional variance can be computed recursively for each observation by using the variance equation. Using these predicted errors, their associated variances, and the assumption that $\epsilon_t \sim N(0, \sigma_t^2)$, we find that the log likelihood for each observation t is

$$\ln L_t = -\frac{1}{2} \left\{ \ln(2\pi\sigma_t^2) + \frac{\epsilon_t^2}{\sigma_t^2} \right\}$$

If we assume that $\epsilon_t \sim t(\text{df})$, then as given in [Hamilton \(1994, 662\)](#),

$$\ln L_t = \ln \Gamma\left(\frac{\text{df} + 1}{2}\right) - \ln \Gamma\left(\frac{\text{df}}{2}\right) - \frac{1}{2} \left[\ln \{(\text{df} - 2)\pi\sigma_t^2\} + (\text{df} + 1) \ln \left\{1 + \frac{\epsilon_t^2}{(\text{df} - 2)\sigma_t^2}\right\} \right]$$

The likelihood is not defined for $\text{df} \leq 2$, so instead of estimating df directly, we estimate $m = \ln(\text{df} - 2)$. Then $\text{df} = \exp(m) + 2 > 2$ for any m .

Following [Bollerslev, Engle, and Nelson \(1994, 2978\)](#), the log likelihood for the t th observation, assuming $\epsilon_t \sim \text{GED}(s)$, is

$$\ln L_t = \ln s - \ln \lambda - \frac{s+1}{s} \ln 2 - \ln \Gamma(s^{-1}) - \frac{1}{2} \left| \frac{\epsilon_t}{\lambda \sigma_t} \right|^s$$

where

$$\lambda = \left\{ \frac{\Gamma(s^{-1})}{2^{2/s} \Gamma(3s^{-1})} \right\}^{1/2}$$

To enforce the restriction that $s > 0$, we estimate $r = \ln s$.

This command supports the Huber/White/sandwich estimator of the variance using `vce(robust)`. See [P] [_robust](#), in particular, in [Maximum likelihood estimators](#) and [Methods and formulas](#).

Missing data

ARCH allows missing data or missing observations but does not attempt to condition on the surrounding data. If a dynamic component cannot be computed— ϵ_t , ϵ_t^2 , and/or σ_t^2 —its priming value is substituted. If a covariate, the dependent variable, or the entire observation is missing, the observation does not enter the likelihood, and its dynamic components are set to their priming values for that observation. This is acceptable only asymptotically and should not be used with a great deal of missing data.

Robert Fry Engle (1942–) was born in Syracuse, New York. He earned degrees in physics and economics at Williams College and Cornell and then worked at MIT and the University of California, San Diego, before moving to New York University Stern School of Business in 2000. He was awarded the 2003 Nobel Prize in Economics for research on autoregressive conditional heteroskedasticity and is a leading expert in time-series analysis, especially the analysis of financial markets.

References

- Adkins, L. C., and R. C. Hill. 2008. *Using Stata for Principles of Econometrics*. 3rd ed. Hoboken, NJ: Wiley.
- Baum, C. F. 2000. [sts15: Tests for stationarity of a time series](#). *Stata Technical Bulletin* 57: 36–39. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 356–360. College Station, TX: Stata Press.
- Baum, C. F., and R. Sperling. 2000. [sts15.1: Tests for stationarity of a time series: Update](#). *Stata Technical Bulletin* 58: 35–36. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 360–362. College Station, TX: Stata Press.
- Baum, C. F., and V. L. Wiggins. 2000. [sts16: Tests for long memory in a time series](#). *Stata Technical Bulletin* 57: 39–44. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 362–368. College Station, TX: Stata Press.
- Berndt, E. K., B. H. Hall, R. E. Hall, and J. A. Hausman. 1974. Estimation and inference in nonlinear structural models. *Annals of Economic and Social Measurement* 3/4: 653–665.
- Black, F. 1976. Studies of stock price volatility changes. *Proceedings of the American Statistical Association, Business and Economics Statistics* 177–181.
- Bollerslev, T. 1986. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics* 31: 307–327.
- Bollerslev, T., R. Y. Chou, and K. F. Kroner. 1992. ARCH modeling in finance. *Journal of Econometrics* 52: 5–59.
- Bollerslev, T., R. F. Engle, and D. B. Nelson. 1994. ARCH models. In *Handbook of Econometrics, Volume IV*, ed. R. F. Engle and D. L. McFadden. New York: Elsevier.
- Bollerslev, T., and J. M. Wooldridge. 1992. Quasi-maximum likelihood estimation and inference in dynamic models with time-varying covariances. *Econometric Reviews* 11: 143–172.
- Davidson, R., and J. G. MacKinnon. 1993. *Estimation and Inference in Econometrics*. New York: Oxford University Press.
- . 2004. *Econometric Theory and Methods*. New York: Oxford University Press.
- Diebold, F. X. 2003. The ET Interview: Professor Robert F. Engle. *Econometric Theory* 19: 1159–1193.
- Ding, Z., C. W. J. Granger, and R. F. Engle. 1993. A long memory property of stock market returns and a new model. *Journal of Empirical Finance* 1: 83–106.
- Enders, W. 2004. *Applied Econometric Time Series*. 2nd ed. New York: Wiley.
- Engle, R. F. 1982. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica* 50: 987–1007.

- . 1990. Discussion: Stock volatility and the crash of '87. *Review of Financial Studies* 3: 103–106.
- Engle, R. F., D. M. Lilien, and R. P. Robins. 1987. Estimating time varying risk premia in the term structure: The ARCH-M model. *Econometrica* 55: 391–407.
- Glosten, L. R., R. Jagannathan, and D. E. Runkle. 1993. On the relation between the expected value and the volatility of the nominal excess return on stocks. *Journal of Finance* 48: 1779–1801.
- Greene, W. H. 2008. *Econometric Analysis*. 6th ed. Upper Saddle River, NJ: Prentice–Hall.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Harvey, A. C. 1989. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge: Cambridge University Press.
- . 1990. *The Econometric Analysis of Time Series*. 2nd ed. Cambridge, MA: MIT Press.
- Higgins, M. L., and A. K. Bera. 1992. A class of nonlinear ARCH models. *International Economic Review* 33: 137–158.
- Hill, R. C., W. E. Griffiths, and G. C. Lim. 2008. *Principles of Econometrics*. 3rd ed. Hoboken, NJ: Wiley.
- Judge, G. G., W. E. Griffiths, R. C. Hill, H. Lütkepohl, and T.-C. Lee. 1985. *The Theory and Practice of Econometrics*. 2nd ed. New York: Wiley.
- Kmenta, J. 1997. *Elements of Econometrics*. 2nd ed. Ann Arbor: University of Michigan Press.
- Mandelbrot, B. 1963. The variation of certain speculative prices. *Journal of Business* 36: 394–419.
- Nelson, D. B. 1991. Conditional heteroskedasticity in asset returns: A new approach. *Econometrica* 59: 347–370.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. *Numerical Recipes in C: The Art of Scientific Computing*. 3rd ed. Cambridge: Cambridge University Press.
- Stock, J. H., and M. W. Watson. 2007. *Introduction to Econometrics*. 2nd ed. Boston: Addison–Wesley.
- Wooldridge, J. M. 2009. *Introductory Econometrics: A Modern Approach*. 4th ed. Cincinnati, OH: South-Western.
- Zakoian, J. M. 1994. Threshold heteroskedastic models. *Journal of Economic Dynamics and Control* 18: 931–955.

Also see

- [TS] **arch postestimation** — Postestimation tools for arch
- [TS] **tsset** — Declare data to be time-series data
- [TS] **arima** — ARIMA, ARMAX, and other dynamic regression models
- [TS] **dvech** — Diagonal vech multivariate GARCH models
- [R] **regress** — Linear regression
- [U] **20 Estimation and postestimation commands**

Description

The following postestimation commands are available for `arch`:

command	description
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	point estimates, standard errors, testing, and inference for generalized predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

`predict` [*type*] *newvar* [*if*] [*in*] [, *statistic options*]

<i>statistic</i>	description
Main	
<code>xb</code>	predicted values for mean equation—the differenced series; the default
<code>y</code>	predicted values for the mean equation in <i>y</i> —the undifferenced series
<code>variance</code>	predicted values for the conditional variance
<code>het</code>	predicted values of the variance, considering only the multiplicative heteroskedasticity
<code>residuals</code>	residuals or predicted innovations
<code>yresiduals</code>	residuals or predicted innovations in <i>y</i> —the undifferenced series

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

<i>options</i>	description
Options	
<code>dynamic(time_constant)</code>	how to handle the lags of y_t
<code>at(varname_ε #_ε varname_{σ²} #_{σ²})</code>	make static predictions
<code>t0(time_constant)</code>	set starting point for the recursions to <i>time_constant</i>
<code>structural</code>	calculate considering the structural component only

time_constant is a # or a time literal, such as `td(1jan1995)` or `tq(1995q1)`, etc.; see [D] [dates and times](#).

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

Six statistics can be computed by using `predict` after `arch`: the predictions of the mean equation (option `xb`, the default), the undifferenced predictions of the mean equation (option `y`), the predictions of the conditional variance (option `variance`), the predictions of the multiplicative heteroskedasticity component of variance (option `het`), the predictions of residuals or innovations (option `residuals`), and the predictions of residuals or innovations in terms of y (option `yresiduals`). Given the dynamic nature of ARCH models and because the dependent variable might be differenced, there are other ways of computing each statistic. We can use all the data on the dependent variable available right up to the time of each prediction (the default, which is often called a one-step prediction), or we can use the data up to a particular time, after which the predicted value of the dependent variable is used recursively to make later predictions (option `dynamic()`). Either way, we can consider or ignore the ARMA disturbance component, which is considered by default and is ignored if you specify the `structural` option. We might also be interested in predictions at certain fixed points where we specify the prior values of ϵ_t and σ_t^2 (option `at()`).

Main

`xb`, the default, calculates the predictions from the mean equation. If `D.depvar` is the dependent variable, these predictions are of `D.depvar` and not of `depvar` itself.

`y` specifies that predictions of `depvar` are to be made even if the model was specified for, say, `D.depvar`.

`variance` calculates predictions of the conditional variance $\hat{\sigma}_t^2$.

`het` calculates predictions of the multiplicative heteroskedasticity component of variance.

`residuals` calculates the residuals. If no other options are specified, these are the predicted innovations ϵ_t ; i.e., they include any ARMA component. If the `structural` option is specified, these are the residuals from the mean equation, ignoring any ARMA terms; see `structural` below. The residuals are always from the estimated equation, which may have a differenced dependent variable; if `depvar` is differenced, they are not the residuals of the undifferenced `depvar`.

`yresiduals` calculates the residuals for `depvar`, even if the model was specified for, say, `D.depvar`. As with `residuals`, the `yresiduals` are computed from the model, including any ARMA component. If the `structural` option is specified, any ARMA component is ignored and `yresiduals` are the residuals from the structural equation; see `structural` below.

Options

`dynamic(time_constant)` specifies how lags of y_t in the model are to be handled. If `dynamic()` is not specified, actual values are used everywhere lagged values of y_t appear in the model to produce one-step-ahead forecasts.

`dynamic(time_constant)` produces dynamic (also known as recursive) forecasts. `time_constant` specifies when the forecast is to switch from one step ahead to dynamic. In dynamic forecasts, references to y_t evaluate to the prediction of y_t for all periods at or after `time_constant`; they evaluate to the actual value of y_t for all prior periods.

`dynamic(10)`, for example, would calculate predictions where any reference to y_t with $t < 10$ evaluates to the actual value of y_t and any reference to y_t with $t \geq 10$ evaluates to the prediction of y_t . This means that one-step-ahead predictions would be calculated for $t < 10$ and dynamic

predictions would be calculated thereafter. Depending on the lag structure of the model, the dynamic predictions might still refer to some actual values of y_t .

You may also specify `dynamic(.)` to have `predict` automatically switch from one-step-ahead to dynamic predictions at $p + q$, where p is the maximum AR lag and q is the maximum MA lag.

`at(varname $\epsilon$  | # $\epsilon$  varname $\sigma^2$  | # $\sigma^2$ )` makes static predictions. `at()` and `dynamic()` may not be specified together.

Specifying `at()` allows static evaluation of results for a given set of disturbances. This is useful, for instance, in generating the news response function. `at()` specifies two sets of values to be used for ϵ_t and σ_t^2 , the dynamic components in the model. These specified values are treated as given. Also, any lagged values of *depvar* in the model are obtained from the real values of the dependent variable. All computations are based on actual data and the given values.

`at()` requires that you specify two arguments, which can be either a variable name or a number. The first argument supplies the values to be used for ϵ_t ; the second supplies the values to be used for σ_t^2 . If σ_t^2 plays no role in your model, the second argument may be specified as `'.'` to indicate missing.

`t0(time_constant)` specifies the starting point for the recursions to compute the predicted statistics; disturbances are assumed to be 0 for $t < t0()$. The default is to set `t0()` to the minimum t observed in the estimation sample, meaning that observations before that are assumed to have disturbances of 0.

`t0()` is irrelevant if `structural` is specified because then all observations are assumed to have disturbances of 0.

`t0(5)`, for example, would begin recursions at $t = 5$. If your data were quarterly, you might instead type `t0(tq(1961q2))` to obtain the same result.

Any ARMA component in the mean equation or GARCH term in the conditional-variance equation makes `arch` recursive and dependent on the starting point of the predictions. This includes one-step-ahead predictions.

`structural` makes the calculation considering the structural component only, ignoring any ARMA terms, and producing the steady-state equilibrium predictions.

Remarks

► Example 1

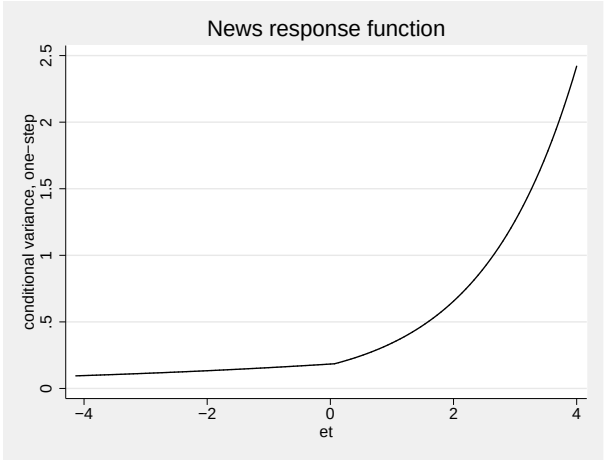
Continuing with our EGARCH model example (example 3) in [TS] `arch`, we can see that `predict`, `at()` calculates σ_t^2 given a set of specified innovations ($\epsilon_t, \epsilon_{t-1}, \dots$) and prior conditional variances ($\sigma_{t-1}^2, \sigma_{t-2}^2, \dots$). The syntax is

```
. predict newvar, variance at(epsilon sigma2)
```

epsilon and *sigma2* are either variables or numbers. Using *sigma2* is a little tricky because you specify values of σ_t^2 , which `predict` is supposed to predict. *predict* does not simply copy variable *sigma2* into *newvar* but uses the lagged values contained in *sigma2* to produce the predicted value of σ_t^2 . It does this for all t , and those results are stored in *newvar*. (If you are interested in dynamic predictions of σ_t^2 , see *Options for predict*.)

We will generate predictions for σ_t^2 , assuming that the lagged values of σ_t^2 are 1, and we will vary ϵ_t from -4 to 4 . First, we will create variable `et` containing ϵ_t , and then we will create and graph the predictions:

```
. generate et = (_n-64)/15
. predict sigma2, variance at(et 1)
. line sigma2 et in 2/l, m(i) c(l) title(News response function)
```



The positive asymmetry does indeed dominate the shape of the news response function. In fact, the response is a monotonically increasing function of news. The form of the response function shows that, for our simple model, only positive, unanticipated price increases have the destabilizing effect that we observe as larger conditional variances.



➤ Example 2

Continuing with our ARCH model with constraints example (example 5) in [TS] [arch](#), using `lincom` we can recover the α parameter from the original specification.

```
. lincom [ARCH]l1.arch/.4
( 1)  2.5 [ARCH]L.arch = 0
```

D.ln_wpi	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
(1)	.5450344	.1844468	2.95	0.003	.1835253	.9065436

Any `arch` parameter could be used to produce an identical estimate.



Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Also see

[TS] [arch](#) — Autoregressive conditional heteroskedasticity (ARCH) family of estimators

[U] [20 Estimation and postestimation commands](#)

Syntax

Basic syntax for a regression model with ARMA disturbances

```
arima devar [indepvars] , ar(numlist) ma(numlist)
```

Basic syntax for an ARIMA(p, d, q) model

```
arima devar , arima(#p,#d,#q)
```

Basic syntax for a multiplicative seasonal ARIMA(p, d, q) $\times$ (P, D, Q)_s model

```
arima devar , arima(#p,#d,#q) sarima(#P,#D,#Q,#s)
```

Full syntax

```
arima devar [indepvars] [if] [in] [weight] [, options]
```

<i>options</i>	description
Model	
<u>no</u> constant	suppress constant term
arima(<i>#p</i> , <i>#d</i> , <i>#q</i>)	specify ARIMA(p, d, q) model for dependent variable
ar(<i>numlist</i>)	autoregressive terms of the structural model disturbance
ma(<i>numlist</i>)	moving-average terms of the structural model disturbance
<u>constraints</u> (<i>constraints</i>)	apply specified linear constraints
<u>collinear</u>	keep collinear variables
Model 2	
sarima(<i>#P</i> , <i>#D</i> , <i>#Q</i> , <i>#s</i>)	specify period- <i>#s</i> multiplicative seasonal ARIMA term
mar(<i>numlist</i> , <i>#s</i>)	multiplicative seasonal autoregressive term; may be repeated
mma(<i>numlist</i> , <i>#s</i>)	multiplicative seasonal moving-average term; may be repeated
Model 3	
<u>condition</u>	use conditional MLE instead of full MLE
<u>savespace</u>	conserve memory during estimation
<u>diffuse</u>	use diffuse prior for starting Kalman filter recursions
p0(<i>#</i> <i>matname</i>)	use alternate prior for starting Kalman recursions; seldom used
state0(<i>#</i> <i>matname</i>)	use alternate state vector for starting Kalman filter recursions
SE/Robust	
vce(<i>vcetype</i>)	<i>vcetype</i> may be opg, <u>r</u> obust, or oim

Reporting

<code>level(#)</code>	set confidence level; default is <code>level(95)</code>
<code>detail</code>	report list of gaps in time series
<code>nocnsreport</code>	do not display constraints
<code>display_options</code>	control spacing

Maximization

<code>maximize_options</code>	control the maximization process; seldom used

[†] <code>coeflegend</code>	display coefficients' legend instead of coefficient table

[†]`coeflegend` does not appear in the dialog box.

You must `tsset` your data before using `arima`; see [TS] [tsset](#).

`devar` and `indepvars` may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

`by`, `rolling`, `statsby`, and `xi` are allowed; see [U] [11.1.10 Prefix commands](#).

`iweights` are allowed; see [U] [11.1.6 weight](#).

See [U] [20 Estimation and postestimation commands](#) for more capabilities of estimation commands.

Menu

Statistics > Time series > ARIMA and ARMAX models

Description

`arima` fits univariate models with time-dependent disturbances. `arima` fits a model of `devar` on `indepvars` where the disturbances are allowed to follow a linear autoregressive moving-average (ARMA) specification. The dependent and independent variables may be differenced or seasonally differenced to any degree. When independent variables are included in the specification, such models are often called ARMAX models; and when independent variables are not specified, they reduce to Box–Jenkins autoregressive integrated moving-average (ARIMA) models in the dependent variable. Multiplicative seasonal ARMAX and ARIMA models can also be fit. Missing data are allowed and are handled using the Kalman filter and methods suggested by [Harvey \(1989 and 1993\)](#); see [Methods and formulas](#).

In the full syntax, `devar` is the variable being modeled, and the structural or regression part of the model is specified in `indepvars`. `ar()` and `ma()` specify the lags of autoregressive and moving-average terms, respectively; and `mar()` and `mma()` specify the multiplicative seasonal autoregressive and moving-average terms, respectively.

`arima` allows time-series operators in the dependent variable and independent variable lists, and making extensive use of these operators is often convenient; see [U] [11.4.4 Time-series varlists](#) and [U] [13.9 Time-series operators](#) for an extended discussion of time-series operators.

`arima` typed without arguments redisplay the previous estimates.

Options

Model

`noconstant`; see [R] [estimation options](#).

`arima(#p,#d,#q)` is an alternative, shorthand notation for specifying models with ARMA disturbances. The dependent variable and any independent variables are differenced #_d times, and 1 through #_p lags of autocorrelations and 1 through #_q lags of moving averages are included in the model. For example, the specification

```
. arima D.y, ar(1/2) ma(1/3)
```

is equivalent to

```
. arima y, arima(2,1,3)
```

The latter is easier to write for simple ARMAX and ARIMA models, but if gaps in the AR or MA lags are to be modeled, or if different operators are to be applied to independent variables, the first syntax is required.

ar(*numlist*) specifies the autoregressive terms of the structural model disturbance to be included in the model. For example, **ar**(1/3) specifies that lags of 1, 2, and 3 of the structural disturbance be included in the model; **ar**(1 4) specifies that lags 1 and 4 be included, perhaps to account for additive quarterly effects.

If the model does not contain regressors, these terms can also be considered autoregressive terms for the dependent variable.

ma(*numlist*) specifies the moving-average terms to be included in the model. These are the terms for the lagged innovations (white-noise disturbances).

constraints(*constraints*), **collinear**; see [R] [estimation options](#).

If constraints are placed between structural model parameters and ARMA terms, the first few iterations may attempt steps into nonstationary areas. This process can be ignored if the final solution is well within the bounds of stationary solutions.

Model 2

sarima(*#_P*, *#_D*, *#_Q*, *#_s*) is an alternative, shorthand notation for specifying the multiplicative seasonal components of models with ARMA disturbances. The dependent variable and any independent variables are lag-*#_s* seasonally differenced *#_D* times, and 1 through *#_P* seasonal lags of autoregressive terms and 1 through *#_Q* seasonal lags of moving-average terms are included in the model. For example, the specification

```
. arima DS12.y, ar(1/2) ma(1/3) mar(1/2,12) mma(1/2,12)
```

is equivalent to

```
. arima y, arima(2,1,3) sarima(2,1,2,12)
```

mar(*numlist*, *#_s*) specifies the lag-*#_s* multiplicative seasonal autoregressive terms. For example, **mar**(1/2,12) requests that the first two lag-12 multiplicative seasonal autoregressive terms be included in the model.

mma(*numlist*, *#_s*) specifies the lag-*#_s* multiplicative seasonal moving-average terms. For example, **mma**(1 3,12) requests that the first and third (but not the second) lag-12 multiplicative seasonal moving-average terms be included in the model.

Model 3

condition specifies that conditional, rather than full, maximum likelihood estimates be produced.

The presample values for ϵ_t and μ_t are taken to be their expected value of zero, and the estimate of the variance of ϵ_t is taken to be constant over the entire sample; see [Hamilton \(1994, 132\)](#). This estimation method is not appropriate for nonstationary series but may be preferable for long series or for models that have one or more long AR or MA lags. **diffuse**, **p0()**, and **state0()** have no meaning for models fit from the conditional likelihood and may not be specified with **condition**.

If the series is long and stationary and the underlying data-generating process does not have a long memory, estimates will be similar, whether estimated by unconditional maximum likelihood (the default), conditional maximum likelihood (`condition`), or maximum likelihood from a diffuse prior (`diffuse`).

In small samples, however, results of conditional and unconditional maximum likelihood may differ substantially; see [Ansley and Newbold \(1980\)](#). Whereas the default unconditional maximum likelihood estimates make the most use of sample information when all the assumptions of the model are met, [Harvey \(1989\)](#) and [Ansley and Kohn \(1985\)](#) argue for diffuse priors often, particularly in ARIMA models corresponding to an underlying structural model.

The `condition` or `diffuse` options may also be preferred when the model contains one or more long AR or MA lags; this avoids inverting potentially large matrices (see `diffuse` below).

When `condition` is specified, estimation is performed by the `arch` command (see [\[TS\] arch](#)), and more control of the estimation process can be obtained using `arch` directly.

`condition` cannot be specified if the model contains any multiplicative seasonal terms.

`savespace` specifies that memory use be conserved by retaining only those variables required for estimation. The original dataset is restored after estimation. This option is rarely used and should be used only if there is not enough space to fit a model without the option. However, `arima` requires considerably more temporary storage during estimation than most estimation commands in Stata.

`diffuse` specifies that a diffuse prior (see [Harvey 1989](#) or [1993](#)) be used as a starting point for the Kalman filter recursions. Using `diffuse`, nonstationary models may be fit with `arima` (see the `p0()` option below; `diffuse` is equivalent to specifying `p0(1e9)`).

By default, `arima` uses the unconditional expected value of the state vector ξ_t (see [Methods and formulas](#)) and the mean squared error (MSE) of the state vector to initialize the filter. When the process is stationary, this corresponds to the expected value and expected variance of a random draw from the state vector and produces unconditional maximum likelihood estimates of the parameters. When the process is not stationary, however, this default is not appropriate, and the unconditional MSE cannot be computed. For a nonstationary process, another starting point must be used for the recursions.

In the absence of nonsample or presample information, `diffuse` may be specified to start the recursions from a state vector of zero and a state MSE matrix corresponding to an effectively infinite variance on this initial state. This method amounts to an uninformative and improper prior that is updated to a proper MSE as data from the sample become available; see [Harvey \(1989\)](#).

Nonstationary models may also correspond to models with infinite variance given a particular specification. This and other problems with nonstationary series make convergence difficult and sometimes impossible.

`diffuse` can also be useful if a model contains one or more long AR or MA lags. Computation of the unconditional MSE of the state vector (see [Methods and formulas](#)) requires construction and inversion of a square matrix that is of dimension $\{\max(p, q + 1)\}^2$, where p and q are the maximum AR and MA lags, respectively. If $q = 27$, for example, we would require a 784-by-784 matrix. Estimation with `diffuse` does not require this matrix.

For large samples, there is little difference between using the default starting point and the `diffuse` starting point. Unless the series has a long memory, the initial conditions affect the likelihood of only the first few observations.

`p0(#|matname)` is a rarely specified option that can be used for nonstationary series or when an alternate prior for starting the Kalman recursions is desired (see `diffuse` above for a discussion of the default starting point and [Methods and formulas](#) for background).

`matname` specifies a matrix to be used as the MSE of the state vector for starting the Kalman filter recursions— $\mathbf{P}_{1|0}$. Instead, one number, `#`, may be supplied, and the MSE of the initial state vector $\mathbf{P}_{1|0}$ will have this number on its diagonal and all off-diagonal values set to zero.

This option may be used with nonstationary series to specify a larger or smaller diagonal for $\mathbf{P}_{1|0}$ than that supplied by `diffuse`. It may also be used with `state0()` when you believe that you have a better prior for the initial state vector and its MSE.

`state0(#|matname)` is a rarely used option that specifies an alternate initial state vector, $\xi_{1|0}$ (see [Methods and formulas](#)), for starting the Kalman filter recursions. If `#` is specified, all elements of the vector are taken to be `#`. The default initial state vector is `state0(0)`.

SE/Robust

`vce(vctype)` specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification and that are derived from asymptotic theory; see [\[R\] vce_option](#).

For state-space models in general and ARMAX and ARIMA models in particular, the robust or quasi-maximum likelihood estimates (QMLEs) of variance are robust to symmetric nonnormality in the disturbances, including, as a special case, heteroskedasticity. The robust variance estimates are not generally robust to functional misspecification of the structural or ARMA components of the model; see [Hamilton \(1994, 389\)](#) for a brief discussion.

Reporting

`level(#)`; see [\[R\] estimation options](#).

`detail` specifies that a detailed list of any gaps in the series be reported, including gaps due to missing observations or missing data for the dependent variable or independent variables.

`nocnsreport`; see [\[R\] estimation options](#).

`display_options`: `vsquish`; see [\[R\] estimation options](#).

Maximization

`maximize_options`: `difficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, `gtolerance(#)`, `nonrntolerance(#)`, `from(init_specs)`; see [\[R\] maximize](#) for all options except `gtolerance()`, and see below for information on `gtolerance()`.

These options are sometimes more important for ARIMA models than most maximum likelihood models because of potential convergence problems with ARIMA models, particularly if the specified model and the sample data imply a nonstationary model.

Several alternate optimization methods, such as Berndt–Hall–Hall–Hausman (BHHH) and Broyden–Fletcher–Goldfarb–Shanno (BFGS), are provided for ARIMA models. Although ARIMA models are not as difficult to optimize as ARCH models, their likelihoods are nevertheless generally not quadratic and often pose optimization difficulties; this is particularly true if a model is nonstationary or nearly nonstationary. Because each method approaches optimization differently, some problems can be successfully optimized by an alternate method when one method fails.

Setting `technique()` to something other than the default or BHHH changes the `vctype` to `vce(oim)`.

The following options are all related to maximization and are either particularly important in fitting ARIMA models or not available for most other estimators.

`technique(algorithm_spec)` specifies the optimization technique to use to maximize the likelihood function.

`technique(bhhh)` specifies the Berndt–Hall–Hall–Hausman (BHHH) algorithm.

`technique(dfpr)` specifies the Davidon–Fletcher–Powell (DFP) algorithm.

`technique(bfgs)` specifies the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

`technique(nr)` specifies Stata’s modified Newton–Raphson (NR) algorithm.

You can specify multiple optimization methods. For example,

```
technique(bhhh 10 nr 20)
```

requests that the optimizer perform 10 BHHH iterations, switch to Newton–Raphson for 20 iterations, switch back to BHHH for 10 more iterations, and so on.

The default for `arima` is `technique(bhhh 5 bfgs 10)`.

`gtolerance(#)` specifies the tolerance for the gradient relative to the coefficients. When $|g_i b_i| \leq \text{gtolerance}()$ for all parameters b_i and the corresponding elements of the gradient g_i , the gradient tolerance criterion is met. The default gradient tolerance for `arima` is `gtolerance(.05)`.

`gtolerance(999)` may be specified to disable the gradient criterion. If the optimizer becomes stuck with repeated “(backed up)” messages, the gradient probably still contains substantial values, but an uphill direction cannot be found for the likelihood. With this option, results can often be obtained, but whether the global maximum likelihood has been found is unclear.

When the maximization is not going well, it is also possible to set the maximum number of iterations (see [R] [maximize](#)) to the point where the optimizer appears to be stuck and to inspect the estimation results at that point.

`from(init_specs)` allows you to set the starting values of the model coefficients; see [R] [maximize](#) for a general discussion and syntax options.

The standard syntax for `from()` accepts a matrix, a list of values, or coefficient name value pairs; see [R] [maximize](#). `arima` also accepts `from(armab0)`, which sets the starting value for all ARMA parameters in the model to zero prior to optimization.

ARIMA models may be sensitive to initial conditions and may have coefficient values that correspond to local maximums. The default starting values for `arima` are generally good, particularly in large samples for stationary series.

The following option is available with `arima` but is not shown in the dialog box:

`coeflegend`; see [R] [estimation options](#).

Remarks

Remarks are presented under the following headings:

[Introduction](#)

[ARIMA models](#)

[Multiplicative seasonal ARIMA models](#)

[ARMAX models](#)

[Dynamic forecasting](#)

Introduction

`arima` fits both standard ARIMA models that are autoregressive in the dependent variable and structural models with ARMA disturbances. Good introductions to the former models can be found in [Box, Jenkins, and Reinsel \(1994\)](#); [Hamilton \(1994\)](#); [Harvey \(1993\)](#); [Newton \(1988\)](#); [Diggle \(1990\)](#); and many others. The latter models are developed fully in [Hamilton \(1994\)](#) and [Harvey \(1989\)](#), both of which provide extensive treatment of the Kalman filter ([Kalman 1960](#)) and the state-space form used by `arima` to fit the models.

Consider a first-order autoregressive moving-average process. Then `arima` estimates all the parameters in the model

$$\begin{aligned} y_t &= \mathbf{x}_t \boldsymbol{\beta} + \mu_t && \text{structural equation} \\ \mu_t &= \rho \mu_{t-1} + \theta \epsilon_{t-1} + \epsilon_t && \text{disturbance, ARMA}(1, 1) \end{aligned}$$

where

- ρ is the first-order autocorrelation parameter
- θ is the first-order moving-average parameter
- $\epsilon_t \sim i.i.d. N(0, \sigma^2)$, meaning that ϵ_t is a white-noise disturbance

You can combine the two equations and write a general ARMA(p, q) in the disturbances process as

$$\begin{aligned} y_t &= \mathbf{x}_t \boldsymbol{\beta} + \rho_1(y_{t-1} - \mathbf{x}_{t-1} \boldsymbol{\beta}) + \rho_2(y_{t-2} - \mathbf{x}_{t-2} \boldsymbol{\beta}) + \cdots + \rho_p(y_{t-p} - \mathbf{x}_{t-p} \boldsymbol{\beta}) \\ &\quad + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t \end{aligned}$$

It is also common to write the general form of the ARMA model more succinctly using lag operator notation as

$$\rho(L^p)(y_t - \mathbf{x}_t \boldsymbol{\beta}) = \theta(L^q) \epsilon_t \quad \text{ARMA}(p, q)$$

where

$$\begin{aligned} \rho(L^p) &= 1 - \rho_1 L - \rho_2 L^2 - \cdots - \rho_p L^p \\ \theta(L^q) &= 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q \end{aligned}$$

and $L^j y_t = y_{t-j}$.

For stationary series, full or unconditional maximum likelihood estimates are obtained via the Kalman filter. For nonstationary series, if some prior information is available, you can specify initial values for the filter by using `state0()` and `p0()` as suggested by [Hamilton \(1994\)](#) or assume an uninformative prior by using the `diffuse` option as suggested by [Harvey \(1989\)](#).

ARIMA models

Pure ARIMA models without a structural component do not have regressors and are often written as autoregressions in the dependent variable, rather than autoregressions in the disturbances from a structural equation. For example, an ARMA(1, 1) model can be written as

$$y_t = \alpha + \rho y_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \quad (1a)$$

Other than a scale factor for the constant term α , these models are equivalent to the ARMA in the disturbances formulation estimated by `arima`, though the latter are more flexible and allow a wider class of models.

To see this effect, replace $\mathbf{x}_t\beta$ in the structural equation above with a constant term β_0 so that

$$\begin{aligned} y_t &= \beta_0 + \mu_t \\ &= \beta_0 + \rho\mu_{t-1} + \theta\epsilon_{t-1} + \epsilon_t \\ &= \beta_0 + \rho(y_{t-1} - \beta_0) + \theta\epsilon_{t-1} + \epsilon_t \\ &= (1 - \rho)\beta_0 + \rho y_{t-1} + \theta\epsilon_{t-1} + \epsilon_t \end{aligned} \quad (1b)$$

Equations (1a) and (1b) are equivalent, with $\alpha = (1 - \rho)\beta_0$, so whether we consider an ARIMA model as autoregressive in the dependent variable or disturbances is immaterial. Our illustration can easily be extended from the ARMA(1, 1) case to the general ARIMA(p, d, q) case.

➤ Example 1: ARIMA model

Enders (2004, 87–93) considers an ARIMA model of the U.S. Wholesale Price Index (WPI) using quarterly data over the period 1960q1 through 1990q4. The simplest ARIMA model that includes differencing and both autoregressive and moving-average components is the ARIMA(1,1,1) specification. We can fit this model with `arima` by typing

```
. use http://www.stata-press.com/data/r11/wpi1
. arima wpi, arima(1,1,1)
(setting optimization to BHHH)
Iteration 0:   log likelihood = -139.80133
Iteration 1:   log likelihood = -135.6278
Iteration 2:   log likelihood = -135.41838
Iteration 3:   log likelihood = -135.36691
Iteration 4:   log likelihood = -135.35892
(switiching optimization to BFGS)
Iteration 5:   log likelihood = -135.35471
Iteration 6:   log likelihood = -135.35135
Iteration 7:   log likelihood = -135.35132
Iteration 8:   log likelihood = -135.35131
ARIMA regression
Sample: 1960q2 - 1990q4                Number of obs   =       123
                                         Wald chi2(2)    =       310.64
Log likelihood = -135.3513              Prob > chi2     =       0.0000
```

D.wpi	OPG		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
wpi						
_cons	.7498197	.3340968	2.24	0.025	.0950019	1.404637
ARMA						
ar						
L1.	.8742288	.0545435	16.03	0.000	.7673256	.981132
ma						
L1.	-.4120458	.1000284	-4.12	0.000	-.6080979	-.2159938
/sigma	.7250436	.0368065	19.70	0.000	.6529042	.7971829

Examining the estimation results, we see that the AR(1) coefficient is 0.874, the MA(1) coefficient is -0.412 , and both are highly significant. The estimated standard deviation of the white-noise disturbance ϵ is 0.725.

This model also could have been fit by typing

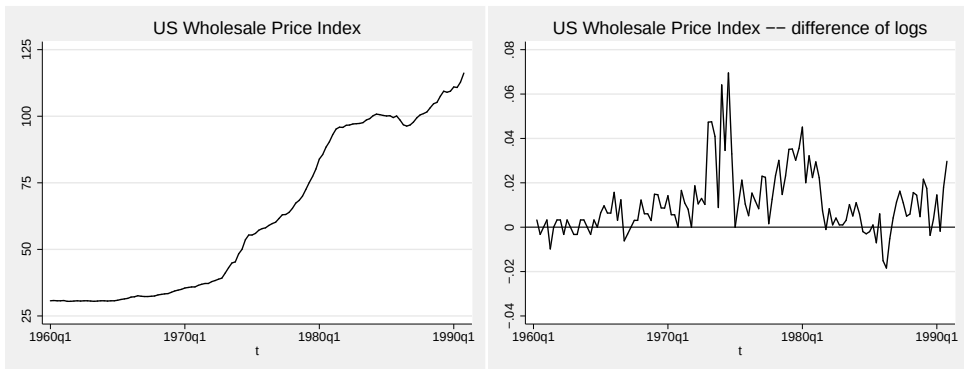
```
. arima D.wpi, ar(1) ma(1)
```

The `D.` placed in front of the dependent variable `wpi` is the Stata time-series operator for differencing. Thus we would be modeling the first difference in WPI from the second quarter of 1960 through the fourth quarter of 1990 because the first observation is lost because of differencing. This second syntax allows a richer choice of models.



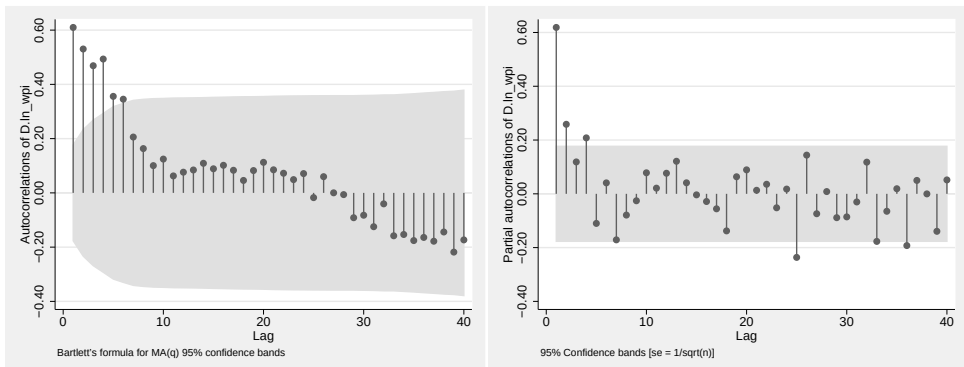
► Example 2: ARIMA model with additive seasonal effects

After examining first-differences of WPI, Enders chose a model of differences in the natural logarithms to stabilize the variance in the differenced series. The raw data and first-difference of the logarithms are graphed below.



On the basis of the autocorrelations, partial autocorrelations (see graphs below), and the results of preliminary estimations, Enders identified an ARMA model in the log-differenced series.

```
. ac D.ln_wpi, ylabel(-.4(.2).6)
. pac D.ln_wpi, ylabel(-.4(.2).6)
```



In addition to an autoregressive term and an MA(1) term, an MA(4) term is included to account for a remaining quarterly effect. Thus the model to be fit is

$$\Delta \ln(\text{wpi}_t) = \beta_0 + \rho_1 \{\Delta \ln(\text{wpi}_{t-1}) - \beta_0\} + \theta_1 \epsilon_{t-1} + \theta_4 \epsilon_{t-4} + \epsilon_t$$

We can fit this model with `arima` and Stata's standard difference operator:

```
. arima D.ln_wpi, ar(1) ma(1 4)
(setting optimization to BHHH)
Iteration 0:  log likelihood = 382.67447
Iteration 1:  log likelihood = 384.80754
Iteration 2:  log likelihood = 384.84749
Iteration 3:  log likelihood = 385.39213
Iteration 4:  log likelihood = 385.40983
(switching optimization to BFGS)
Iteration 5:  log likelihood = 385.9021
Iteration 6:  log likelihood = 385.95646
Iteration 7:  log likelihood = 386.02979
Iteration 8:  log likelihood = 386.03326
Iteration 9:  log likelihood = 386.03354
Iteration 10: log likelihood = 386.03357
ARIMA regression
Sample: 1960q2 - 1990q4                Number of obs   =      123
                                         Wald chi2(3)    =     333.60
Log likelihood = 386.0336                Prob > chi2     =     0.0000
```

D.ln_wpi	OPG				[95% Conf. Interval]	
	Coef.	Std. Err.	z	P> z		
ln_wpi						
_cons	.0110493	.0048349	2.29	0.022	.0015731	.0205255
ARMA						
ar						
L1.	.7806991	.0944946	8.26	0.000	.5954931	.965905
ma						
L1.	-.3990039	.1258753	-3.17	0.002	-.6457149	-.1522928
L4.	.3090813	.1200945	2.57	0.010	.0737003	.5444622
/sigma	.0104394	.0004702	22.20	0.000	.0095178	.0113609

In this final specification, the log-differenced series is still highly autocorrelated at a level of 0.781, though innovations have a negative impact in the ensuing quarter (−0.399) and a positive seasonal impact of 0.309 in the following year.

◀

□ Technical note

In one way, the results differ from most of Stata's estimation commands: the standard error of the coefficients is reported as OPG Std. Err. As noted in *Options*, the default standard errors and covariance matrix for `arima` estimates are derived from the outer product of gradients (OPG). This is one of three asymptotically equivalent methods of estimating the covariance matrix of the coefficients (only two of which are usually tractable to derive). Discussions and derivations of all three estimates can be found in Davidson and MacKinnon (1993), Greene (2008), and Hamilton (1994). Bollerslev, Engle, and Nelson (1994) suggest that the OPG estimates are more numerically stable in time-series regressions when the likelihood and its derivatives depend on recursive computations, which is certainly the case for the Kalman filter. To date, we have found no numerical instabilities in either estimate of the covariance matrix—subject to the stability and convergence of the overall model.

Most of Stata's estimation commands provide covariance estimates derived from the Hessian of the likelihood function. These alternate estimates can also be obtained from `arima` by specifying the `vce(oim)` option.

□

Multiplicative seasonal ARIMA models

Many time series exhibit a periodic seasonal component, and a seasonal ARIMA model, often abbreviated SARIMA, can then be used. For example, monthly sales data for air conditioners have a strong seasonal component, with sales high in the summer months and low in the winter months.

In the previous example, we accounted for quarterly effects by fitting the model

$$(1 - \rho_1 L) \{ \Delta \ln(\text{wpi}_t) - \beta_0 \} = (1 + \theta_1 L + \theta_4 L^4) \epsilon_t$$

This is an *additive* seasonal ARIMA model, in the sense that the first- and fourth-order MA terms work additively: $(1 + \theta_1 L + \theta_4 L^4)$.

Another way to handle the quarterly effect would be to fit a multiplicative seasonal ARIMA model. A multiplicative SARIMA model of order $(1, 1, 1) \times (0, 0, 1)_4$ for the $\ln(\text{wpi}_t)$ series is

$$(1 - \rho_1 L) \{ \Delta \ln(\text{wpi}_t) - \beta_0 \} = (1 + \theta_1 L)(1 + \theta_{4,1} L^4) \epsilon_t$$

or, upon expanding terms,

$$\Delta \ln(\text{wpi}_t) = \beta_0 + \rho_1 \{ \Delta \ln(\text{wpi}_t) - \beta_0 \} + \theta_1 \epsilon_{t-1} + \theta_{4,1} \epsilon_{t-4} + \theta_1 \theta_{4,1} \epsilon_{t-5} + \epsilon_t \quad (2)$$

In the notation $(1, 1, 1) \times (0, 0, 1)_4$, the $(1, 1, 1)$ means that there is one nonseasonal autoregressive term $(1 - \rho_1 L)$ and one nonseasonal moving-average term $(1 + \theta_1 L)$ and that the time series is first-differenced one time. The $(0, 0, 1)_4$ indicates that there is no lag-4 seasonal autoregressive term, that there is one lag-4 seasonal moving-average term $(1 + \theta_{4,1} L^4)$, and that the series is seasonally differenced zero times. This is known as a *multiplicative* SARIMA model because the nonseasonal and seasonal factors work multiplicatively: $(1 + \theta_1 L)(1 + \theta_{4,1} L^4)$. Multiplying the terms imposes nonlinear constraints on the parameters of the fifth-order lagged values; `arima` imposes these constraints automatically.

To further clarify the notation, consider a $(2, 1, 1) \times (1, 1, 2)_4$ multiplicative SARIMA model:

$$(1 - \rho_1 L - \rho_2 L^2)(1 - \rho_{4,1} L^4) \Delta \Delta_4 z_t = (1 + \theta_1 L)(1 + \theta_{4,1} L^4 + \theta_{4,2} L^8) \epsilon_t \quad (3)$$

where Δ denotes the difference operator $\Delta y_t = y_t - y_{t-1}$ and Δ_s denotes the lag- s seasonal difference operator $\Delta_s y_t = y_t - y_{t-s}$. Expanding (3), we have

$$\begin{aligned} \tilde{z}_t = & \rho_1 \tilde{z}_{t-1} + \rho_2 \tilde{z}_{t-2} + \rho_{4,1} \tilde{z}_{t-4} - \rho_1 \rho_{4,1} \tilde{z}_{t-5} - \rho_2 \rho_{4,1} \tilde{z}_{t-6} \\ & + \theta_1 \epsilon_{t-1} + \theta_{4,1} \epsilon_{t-4} + \theta_1 \theta_{4,1} \epsilon_{t-5} + \theta_{4,2} \epsilon_{t-8} + \theta_1 \theta_{4,2} \epsilon_{t-9} \end{aligned}$$

where

$$\tilde{z}_t = \Delta \Delta_4 z_t = \Delta(z_t - z_{t-4}) = z_t - z_{t-1} - (z_{t-4} - z_{t-5})$$

and $z_t = y_t - \mathbf{x}_t \beta$ if regressors are included in the model, $z_t = y_t - \beta_0$ if just a constant term is included, and $z_t = y_t$ otherwise.

More generally, a $(p, d, q) \times (P, D, Q)_s$ multiplicative SARIMA model is

$$\rho(L^p)\rho_s(L^P)\Delta^d\Delta_s^D z_t = \theta(L^q)\theta_s(L^Q)$$

where

$$\begin{aligned}\rho_s(L^P) &= (1 - \rho_{s,1}L^s - \rho_{s,2}L^{2s} - \cdots - \rho_{s,P}L^{Ps}) \\ \theta_s(L^Q) &= (1 + \theta_{s,1}L^s + \theta_{s,2}L^{2s} + \cdots + \theta_{s,Q}L^{Qs})\end{aligned}$$

$\rho(L^p)$ and $\theta(L^q)$ were defined previously, Δ^d means apply the Δ operator d times, and similarly for Δ_s^D . Typically, d and D will be 0 or 1; and p, q, P , and Q will seldom be more than 2 or 3. s will typically be 4 for quarterly data and 12 for monthly data. In fact, the model can be extended to include both monthly and quarterly seasonal factors, as we explain below.

If a plot of the data suggests that the seasonal effect is proportional to the mean of the series, then the seasonal effect is probably multiplicative and a multiplicative SARIMA model may be appropriate. [Box, Jenkins, and Reinsel \(1994, sec. 9.3.1\)](#) suggest starting with a multiplicative SARIMA model with any data that exhibit seasonal patterns and then exploring nonmultiplicative SARIMA models if the multiplicative models do not fit the data well. On the other hand, [Chatfield \(2004, 14\)](#) suggests that taking the logarithm of the series will make the seasonal effect additive, in which case an additive SARIMA model as fit in the previous example would be appropriate. In short, the analyst should probably try both additive and multiplicative SARIMA models to see which provides better fits and forecasts.

Unless the `diffuse` option is used, `arima` must create square matrices of dimension $\{\max(p, q + 1)\}^2$, where p and q are the maximum AR and MA lags, respectively; and the inclusion of long seasonal terms can make this dimension rather large. For example, with monthly data, you might fit a $(0, 1, 1) \times (0, 1, 2)_{12}$ SARIMA model. The maximum MA lag is $2 \times 12 + 1 = 25$, requiring a matrix with $26^2 = 676$ rows and columns.

► Example 3: Multiplicative SARIMA model

One of the most common multiplicative SARIMA specifications is the $(0, 1, 1) \times (0, 1, 1)_{12}$ “airline” model of [Box, Jenkins, and Reinsel \(1994, sec. 9.2\)](#). The dataset `airline.dta` contains monthly international airline passenger data from January 1949 through December 1960. After first- and seasonally differencing the data, we do not suspect the presence of a trend component, so we use the `noconstant` option with `arima`:

(Continued on next page)

```
. use http://www.stata-press.com/data/r11/air2
(TIMESLAB: Airline passengers)

. generate lnair = ln(air)

. arima lnair, arima(0,1,1) sarima(0,1,1,12) noconstant
(setting optimization to BHHH)
Iteration 0:   log likelihood =   223.8437
Iteration 1:   log likelihood =  239.80405
(output omitted)
Iteration 8:   log likelihood =  244.69651
ARIMA regression
Sample:   14 - 144                                Number of obs   =       131
                                                    Wald chi2(2)    =       84.53
Log likelihood = 244.6965                            Prob > chi2     =       0.0000
```

DS12.lnair	OPG		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
ARMA						
ma						
L1.	-.4018324	.0730307	-5.50	0.000	-.5449698	-.2586949
ARMA12						
ma						
L1.	-.5569342	.0963129	-5.78	0.000	-.745704	-.3681644
/sigma	.0367167	.0020132	18.24	0.000	.0327708	.0406625

Thus our model of the monthly number of international airline passengers is

$$\Delta\Delta_{12}\text{lnair}_t = -0.402\epsilon_{t-1} - 0.557\epsilon_{t-12} + 0.224\epsilon_{t-13} + \epsilon_t$$
$$\hat{\sigma} = 0.037$$

In (2), for example, the coefficient on ϵ_{t-13} is the product of the coefficients on the ϵ_{t-1} and ϵ_{t-12} terms ($0.224 \approx -0.402 \times -0.557$). `arima` labeled the dependent variable `DS12.lnair` to indicate that it has applied the difference operator Δ and the lag-12 seasonal difference operator Δ_{12} to `lnair`; see [U] 11.4.4 Time-series varlists for more information.

We could have fit this model by typing

```
. arima DS12.lnair, ma(1) mma(1, 12) noconstant
```

For simple multiplicative models, using the `sarima()` option is easier, though this second syntax allows us to incorporate more complicated seasonal terms.



The `mar()` and `mma()` options can be repeated, allowing us to control for multiple seasonal patterns. For example, we may have monthly sales data that exhibit a quarterly pattern as businesses purchase our product at the beginning of calendar quarters when new funds are budgeted, and our product is purchased more frequently in a few months of the year than in most others, even after we control for quarterly fluctuations. Thus we might choose to fit the model

$$(1-\rho L)(1-\rho_{4,1}L^4)(1-\rho_{12,1}L^{12})(\Delta\Delta_4\Delta_{12}\text{sales}_t-\beta_0) = (1+\theta L)(1+\theta_{4,1}L^4)(1+\theta_{12,1}L^{12})\epsilon_t$$

Although this model looks rather complicated, estimating it using `arima` is straightforward:

```
. arima DS4S12.sales, ar(1) mar(1, 4) mar(1, 12) ma(1) mma(1, 4) mma(1, 12)
```

If we instead wanted to include two lags in the lag-4 seasonal AR term and the first and third (but not the second) term in the lag-12 seasonal MA term, we would type

```
. arima DS4S12.sales, ar(1) mar(1 2, 4) mar(1, 12) ma(1) mma(1 3, 12)
```

However, models with multiple seasonal terms can be difficult to fit. Usually, one seasonal factor with just one or two AR or MA terms is adequate.

ARMAX models

Thus far all our examples have been pure ARIMA models in which the dependent variable was modeled solely as a function of its past values and disturbances. Also, `arima` can fit ARMAX models, which model the dependent variable in terms of a linear combination of independent variables, as well as an ARMA disturbance process. The `prais` command, for example, allows you to control for only AR(1) disturbances, whereas `arima` allows you to control for a much richer dynamic error structure. `arima` allows for both nonseasonal and seasonal ARMA components in the disturbances.

► Example 4: ARMAX model

For a simple example of a model including covariates, we can estimate an update of Friedman and Meiselman's (1963) equation representing the quantity theory of money. They postulate a straightforward relationship between personal-consumption expenditures (`consump`) and the money supply as measured by M2 (`m2`).

$$\text{consump}_t = \beta_0 + \beta_1 m2_t + \mu_t$$

Friedman and Meiselman fit the model over a period ending in 1956; we will refit the model over the period 1959q1 through 1981q4. We restrict our attention to the period prior to 1982 because the Federal Reserve manipulated the money supply extensively in the later 1980s to control inflation, and the relationship between consumption and the money supply becomes much more complex during the later part of the decade.

To demonstrate `arima`, we will include both an autoregressive term and a moving-average term for the disturbances in the model; the original estimates included neither. Thus we model the disturbance of the structural equation as

$$\mu_t = \rho\mu_{t-1} + \theta\epsilon_{t-1} + \epsilon_t$$

As per the original authors, the relationship is estimated on seasonally adjusted data, so there is no need to include seasonal effects explicitly. Obtaining seasonally unadjusted data and simultaneously modeling the structural and seasonal effects might be preferable.

We will restrict the estimation to the desired sample by using the `tin()` function in an `if` expression; see [D] [functions](#). By leaving the first argument of `tin()` blank, we are including all available data through the second date (1981q4). We fit the model by typing

```
. use http://www.stata-press.com/data/r11/friedman2, clear
. arima consump m2 if tin(, 1981q4), ar(1) ma(1)
(setting optimization to BHHH)
Iteration 0:   log likelihood = -344.67575
Iteration 1:   log likelihood = -341.57248
(output omitted)
Iteration 10:  log likelihood = -340.50774
ARIMA regression
Sample: 1959q1 - 1981q4                Number of obs      =          92
                                         Wald chi2(3)        =    4394.80
Log likelihood = -340.5077              Prob > chi2         =     0.0000
```

consump		OPG		z	P> z	[95% Conf. Interval]	
		Coef.	Std. Err.				
consump	m2	1.122029	.0363563	30.86	0.000	1.050772	1.193286
	_cons	-36.09872	56.56703	-0.64	0.523	-146.9681	74.77062
ARMA							
ar	L1.	.9348486	.0411323	22.73	0.000	.8542308	1.015467
	ma						
L1.		.3090592	.0885883	3.49	0.000	.1354293	.4826891
	/sigma	9.655308	.5635157	17.13	0.000	8.550837	10.75978

We find a relatively small money velocity with respect to consumption (1.122) over this period, although consumption is only one facet of the income velocity. We also note a very large first-order autocorrelation in the disturbances, as well as a statistically significant first-order moving average.

We might be concerned that our specification has led to disturbances that are heteroskedastic or non-Gaussian. We refit the model by using the `vce(robust)` option.

```
. arima consump m2 if tin(, 1981q4), ar(1) ma(1) vce(robust)
(setting optimization to BHHH)
Iteration 0:   log pseudolikelihood = -344.67575
Iteration 1:   log pseudolikelihood = -341.57248
(output omitted)
Iteration 10:  log pseudolikelihood = -340.50774
ARIMA regression
Sample: 1959q1 - 1981q4                Number of obs      =          92
                                         Wald chi2(3)        =       1176.26
Log pseudolikelihood = -340.5077       Prob > chi2         =        0.0000
```

consump	Semirobust		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
consump						
m2	1.122029	.0433303	25.89	0.000	1.037103	1.206954
_cons	-36.09872	28.10479	-1.28	0.199	-91.1831	18.98566
ARMA						
ar						
L1.	.9348486	.0493428	18.95	0.000	.8381385	1.031559
ma						
L1.	.3090592	.1605359	1.93	0.054	-.0055854	.6237038
/sigma	9.655308	1.082639	8.92	0.000	7.533375	11.77724

We note a substantial increase in the estimated standard errors, and our once clearly significant moving-average term is now only marginally significant.

◀

Dynamic forecasting

Another feature of the `arima` command is the ability to use `predict` afterward to make dynamic forecasts. Suppose that we wish to fit the regression model

$$y_t = \beta_0 + \beta_1 x_t + \rho y_{t-1} + \epsilon_t$$

by using a sample of data from $t = 1 \dots T$ and make forecasts beginning at time f .

If we use `regress` or `prais` to fit the model, then we can use `predict` to make one-step-ahead forecasts. That is, `predict` will compute

$$\hat{y}_f = \hat{\beta}_0 + \hat{\beta}_1 x_f + \hat{\rho} y_{f-1}$$

Most importantly, here `predict` will use the actual value of y at period $f - 1$ in computing the forecast for time f . Thus, if we use `regress` or `prais`, we cannot make forecasts for any periods beyond $f = T + 1$ unless we have observed values for y for those periods.

If we instead fit our model with `arima`, then `predict` can produce dynamic forecasts by using the Kalman filter. If we use the `dynamic(f)` option, then for period f `predict` will compute

$$\hat{y}_f = \hat{\beta}_0 + \hat{\beta}_1 x_f + \hat{\rho} y_{f-1}$$

by using the observed value of y_{f-1} just as `predict` after `regress` or `prais`. However, for period $f + 1$ `predict newvar, dynamic(f)` will compute

$$\widehat{y}_{f+1} = \widehat{\beta}_0 + \widehat{\beta}_1 x_{f+1} + \widehat{\rho} \widehat{y}_f$$

using the *predicted* value of y_f instead of the observed value. Similarly, the period $f + 2$ forecast will be

$$\widehat{y}_{f+2} = \widehat{\beta}_0 + \widehat{\beta}_1 x_{f+2} + \widehat{\rho} \widehat{y}_{f+1}$$

Of course, because our model includes the regressor x_t , we can make forecasts only through periods for which we have observations on x_t . However, for pure ARIMA models, we can compute dynamic forecasts as far beyond the final period of our dataset as desired.

For more information on `predict` after `arima`, see [\[TS\] arima postestimation](#).

Saved results

`arima` saves the following in `e()`:

Scalars	
<code>e(N)</code>	number of observations
<code>e(N_gaps)</code>	number of gaps
<code>e(k)</code>	number of parameters
<code>e(k_eq)</code>	number of equations in <code>e(b)</code>
<code>e(k_eq_model)</code>	number of equations in model Wald test
<code>e(k_dv)</code>	number of dependent variables
<code>e(k1)</code>	number of variables in first equation
<code>e(df_m)</code>	model degrees of freedom
<code>e(ll)</code>	log likelihood
<code>e(sigma)</code>	sigma
<code>e(chi2)</code>	χ^2 statistic
<code>e(p)</code>	significance
<code>e(tmin)</code>	minimum time
<code>e(tmax)</code>	maximum time
<code>e(ar_max)</code>	maximum AR lag
<code>e(ma_max)</code>	maximum MA lag
<code>e(rank)</code>	rank of <code>e(V)</code>
<code>e(ic)</code>	number of iterations
<code>e(rc)</code>	return code
<code>e(converged)</code>	1 if converged, 0 otherwise

Macros

<code>e(cmd)</code>	<code>arima</code>
<code>e(cmdline)</code>	command as typed
<code>e(depvar)</code>	name of dependent variable
<code>e(covariates)</code>	list of covariates
<code>e(eqnames)</code>	names of equations
<code>e(wtype)</code>	weight type
<code>e(wexp)</code>	weight expression
<code>e(title)</code>	title in estimation output
<code>e(tmins)</code>	formatted minimum time
<code>e(tmaxs)</code>	formatted maximum time
<code>e(chi2type)</code>	Wald; type of model χ^2 test
<code>e(vce)</code>	<code>vcetype</code> specified in <code>vce()</code>
<code>e(vcetype)</code>	title used to label Std. Err.
<code>e(ma)</code>	lags for moving-average terms
<code>e(ar)</code>	lags for autoregressive terms
<code>e(mari)</code>	multiplicative AR terms and lag $i=1\dots$ (# seasonal AR terms)
<code>e(mmai)</code>	multiplicative MA terms and lag $i=1\dots$ (# seasonal MA terms)
<code>e(seasons)</code>	seasonal lags in model
<code>e(unsta)</code>	unstationary or blank
<code>e(opt)</code>	type of optimization
<code>e(ml_method)</code>	type of ml method
<code>e(user)</code>	name of likelihood-evaluator program
<code>e(technique)</code>	maximization technique
<code>e(tech_steps)</code>	number of iterations performed before switching techniques
<code>e(crittype)</code>	optimization criterion
<code>e(properties)</code>	<code>b V</code>
<code>e(estat_cmd)</code>	program used to implement <code>estat</code>
<code>e(predict)</code>	program used to implement <code>predict</code>
<code>e(marginsok)</code>	predictions allowed by <code>margins</code>
<code>e(marginsnotok)</code>	predictions disallowed by <code>margins</code>

Matrices

<code>e(b)</code>	coefficient vector
<code>e(Cns)</code>	constraints matrix
<code>e(ilog)</code>	iteration log (up to 20 iterations)
<code>e(gradient)</code>	gradient vector
<code>e(V)</code>	variance–covariance matrix of the estimators
<code>e(V_modelbased)</code>	model-based variance

Functions

<code>e(sample)</code>	marks estimation sample

Methods and formulas

`arima` is implemented as an ado-file.

Estimation is by maximum likelihood using the Kalman filter via the prediction error decomposition; see [Hamilton \(1994\)](#), [Gourieroux and Monfort \(1997\)](#), or, in particular, [Harvey \(1989\)](#). Any of these sources will serve as excellent background for the fitting of these models with the state-space form; each source also provides considerable detail on the method outlined below.

Methods and formulas are presented under the following headings:

[ARIMA model](#)
[Kalman filter equations](#)
[Kalman filter or state-space representation of the ARIMA model](#)
[Kalman filter recursions](#)
[Kalman filter initial conditions](#)
[Likelihood from prediction error decomposition](#)
[Missing data](#)

ARIMA model

The model to be fit is

$$y_t = \mathbf{x}_t \boldsymbol{\beta} + \mu_t$$
$$\mu_t = \sum_{i=1}^p \rho_i \mu_{t-i} + \sum_{j=1}^q \theta_j \epsilon_{t-j} + \epsilon_t$$

which can be written as the single equation

$$y_t = \mathbf{x}_t \boldsymbol{\beta} + \sum_{i=1}^p \rho_i (y_{t-i} - x_{t-i} \boldsymbol{\beta}) + \sum_{j=1}^q \theta_j \epsilon_{t-j} + \epsilon_t$$

Some of the ρ s and θ s may be constrained to zero or, for multiplicative seasonal models, the products of other parameters.

Kalman filter equations

We will roughly follow Hamilton's (1994) notation and write the Kalman filter

$$\boldsymbol{\xi}_t = \mathbf{F} \boldsymbol{\xi}_{t-1} + \mathbf{v}_t \quad (\text{state equation})$$

$$\mathbf{y}_t = \mathbf{A}' \mathbf{x}_t + \mathbf{H}' \boldsymbol{\xi}_t + \mathbf{w}_t \quad (\text{observation equation})$$

and

$$\begin{pmatrix} \mathbf{v}_t \\ \mathbf{w}_t \end{pmatrix} \sim N \left\{ \mathbf{0}, \begin{pmatrix} \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{pmatrix} \right\}$$

We maintain the standard Kalman filter matrix and vector notation, although for univariate models $\mathbf{y}_t$, $\mathbf{w}_t$, and $\mathbf{R}$ are scalars.

Kalman filter or state-space representation of the ARIMA model

A univariate ARIMA model can be cast in state-space form by defining the Kalman filter matrices as follows (see [Hamilton \[1994\]](#), or [Gourieroux and Monfort \[1997\]](#), for details):

$$F = \begin{bmatrix} \rho_1 & \rho_2 & \cdots & \rho_{p-1} & \rho_p \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

$$\mathbf{v}_t = \begin{bmatrix} \epsilon_{t-1} \\ 0 \\ \cdots \\ \cdots \\ \cdots \\ 0 \end{bmatrix}$$

$$\mathbf{A}' = \beta$$

$$\mathbf{H}' = [1 \quad \theta_1 \quad \theta_2 \quad \cdots \quad \theta_q]$$

$$\mathbf{w}_t = 0$$

The Kalman filter representation does not require the moving-average terms to be invertible.

Kalman filter recursions

To demonstrate how missing data are handled, the updating recursions for the Kalman filter will be left in two steps. Writing the updating equations as one step using the gain matrix $\mathbf{K}$ is common. We will provide the updating equations with little justification; see the sources listed above for details.

As a linear combination of a vector of random variables, the state ξ_t can be updated to its expected value on the basis of the prior state as

$$\xi_{t|t-1} = \mathbf{F}\xi_{t-1} + \mathbf{v}_{t-1} \quad (1)$$

This state is a quadratic form that has the covariance matrix

$$\mathbf{P}_{t|t-1} = \mathbf{F}\mathbf{P}_{t-1}\mathbf{F}' + \mathbf{Q} \quad (2)$$

The estimator of $\mathbf{y}_t$ is

$$\hat{\mathbf{y}}_{t|t-1} = \mathbf{x}_t\beta + \mathbf{H}'\xi_{t|t-1}$$

which implies an innovation or prediction error

$$\hat{\mathbf{u}}_t = \mathbf{y}_t - \hat{\mathbf{y}}_{t|t-1}$$

This value or vector has mean squared error (MSE)

$$\mathbf{M}_t = \mathbf{H}'\mathbf{P}_{t|t-1}\mathbf{H} + \mathbf{R}$$

Now the expected value of ξ_t conditional on a realization of $\mathbf{y}_t$ is

$$\xi_t = \xi_{t|t-1} + \mathbf{P}_{t|t-1}\mathbf{H}\mathbf{M}_t^{-1}\hat{\mathbf{u}}_t \quad (3)$$

with MSE

$$\mathbf{P}_t = \mathbf{P}_{t|t-1} - \mathbf{P}_{t|t-1}\mathbf{H}\mathbf{M}_t^{-1}\mathbf{H}'\mathbf{P}_{t|t-1} \quad (4)$$

This expression gives the full set of Kalman filter recursions.

Kalman filter initial conditions

When the series is stationary, conditional on $\mathbf{x}_t\beta$, the initial conditions for the filter can be considered a random draw from the stationary distribution of the state equation. The initial values of the state and the state MSE are the expected values from this stationary distribution. For an ARIMA model, these can be written as

$$\xi_{1|0} = \mathbf{0}$$

and

$$\text{vec}(\mathbf{P}_{1|0}) = (\mathbf{I}_{r^2} - \mathbf{F} \otimes \mathbf{F})^{-1} \text{vec}(\mathbf{Q})$$

where $\text{vec}()$ is an operator representing the column matrix resulting from stacking each successive column of the target matrix.

If the series is not stationary, the initial state conditions do not constitute a random draw from a stationary distribution, and some other values must be chosen. [Hamilton \(1994\)](#) suggests that they be chosen based on prior expectations, whereas Harvey suggests a diffuse and improper prior having a state vector of $\mathbf{0}$ and an infinite variance. This method corresponds to $\mathbf{P}_{1|0}$ with diagonal elements of ∞ . Stata allows either approach to be taken for nonstationary series—initial priors may be specified with `state0()` and `p0()`, and a diffuse prior may be specified with `diffuse`.

Likelihood from prediction error decomposition

Given the outputs from the Kalman filter recursions and assuming that the state and observation vectors are Gaussian, the likelihood for the state-space model follows directly from the resulting multivariate normal in the predicted innovations. The log likelihood for observation t is

$$\ln L_t = -\frac{1}{2} \{ \ln(2\pi) + \ln(|\mathbf{M}_t|) - \hat{\boldsymbol{\epsilon}}_t' \mathbf{M}_t^{-1} \hat{\boldsymbol{\epsilon}}_t \}$$

This command supports the Huber/White/sandwich estimator of the variance using `vce(robust)`. See [\[P\] `\_robust`](#), in particular, in [Maximum likelihood estimators](#) and [Methods and formulas](#).

Missing data

Missing data, whether a missing dependent variable y_t , one or more missing covariates $\mathbf{x}_t$, or completely missing observations, are handled by continuing the state-updating equations without any contribution from the data; see [Harvey \(1989 and 1993\)](#). That is, (1) and (2) are iterated for every missing observation, whereas (3) and (4) are ignored. Thus, for observations with missing data, $\xi_t = \xi_{t|t-1}$ and $\mathbf{P}_t = \mathbf{P}_{t|t-1}$. Without any information from the sample, this effectively assumes that the prediction error for the missing observations is 0. Other methods of handling missing data on the basis of the EM algorithm have been suggested, e.g., [Shumway \(1984, 1988\)](#).

George Edward Pelham Box (1919–) was born in Kent, England, and earned degrees in statistics at the University of London. After work in the chemical industry, he taught and researched at Princeton and the University of Wisconsin. His many major contributions to statistics include papers and books in Bayesian inference, robustness (a term he introduced to statistics), modeling strategy, experimental design and response surfaces, time-series analysis, distribution theory, transformations, and nonlinear estimation.

Gwilym Meirion Jenkins (1933–1982) was a British mathematician and statistician who spent his career in industry and academia, working for extended periods at Imperial College London and the University of Lancaster before running his own company. His interests were centered on time series and he collaborated with G. E. P. Box on what are often called Box–Jenkins models. The last years of Jenkins’ life were marked by a slowly losing battle against Hodgkin’s disease.

References

- Ansley, C. F., and R. Kohn. 1985. Estimation, filtering, and smoothing in state space models with incompletely specified initial conditions. *Annals of Statistics* 13: 1286–1316.
- Ansley, C. F., and P. Newbold. 1980. Finite sample properties of estimators for autoregressive moving average models. *Journal of Econometrics* 13: 159–183.
- Baum, C. F. 2000. [sts15: Tests for stationarity of a time series](#). *Stata Technical Bulletin* 57: 36–39. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 356–360. College Station, TX: Stata Press.
- Baum, C. F., and T. Room. 2001. [sts18: A test for long-range dependence in a time series](#). *Stata Technical Bulletin* 60: 37–39. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 370–373. College Station, TX: Stata Press.
- Baum, C. F., and R. Sperling. 2000. [sts15.1: Tests for stationarity of a time series: Update](#). *Stata Technical Bulletin* 58: 35–36. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 360–362. College Station, TX: Stata Press.
- Baum, C. F., and V. L. Wiggins. 2000. [sts16: Tests for long memory in a time series](#). *Stata Technical Bulletin* 57: 39–44. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 362–368. College Station, TX: Stata Press.
- Berndt, E. K., B. H. Hall, R. E. Hall, and J. A. Hausman. 1974. Estimation and inference in nonlinear structural models. *Annals of Economic and Social Measurement* 3/4: 653–665.
- Bollerslev, T., R. F. Engle, and D. B. Nelson. 1994. ARCH models. In *Handbook of Econometrics, Volume IV*, ed. R. F. Engle and D. L. McFadden. New York: Elsevier.
- Box, G. E. P. 1983. Obituary: G. M. Jenkins, 1933–1982. *Journal of the Royal Statistical Society, Series A* 146: 205–206.
- Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control*. 3rd ed. Englewood Cliffs, NJ: Prentice–Hall.
- Chatfield, C. 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- David, J. S. 1999. [sts14: Bivariate Granger causality test](#). *Stata Technical Bulletin* 51: 40–41. Reprinted in *Stata Technical Bulletin Reprints*, vol. 9, pp. 350–351. College Station, TX: Stata Press.
- Davidson, R., and J. G. MacKinnon. 1993. *Estimation and Inference in Econometrics*. New York: Oxford University Press.
- DeGroot, M. H. 1987. A conversation with George Box. *Statistical Science* 2: 239–258.
- Diggle, P. J. 1990. *Time Series: A Biostatistical Introduction*. Oxford: Oxford University Press.
- Enders, W. 2004. *Applied Econometric Time Series*. 2nd ed. New York: Wiley.
- Friedman, M., and D. Meiselman. 1963. The relative stability of monetary velocity and the investment multiplier in the United States, 1897–1958. In *Stabilization Policies*, Commission on Money and Credit, 123–126. Englewood Cliffs, NJ: Prentice–Hall.
- Gourieroux, C., and A. Monfort. 1997. *Time Series and Dynamic Models*. Trans. ed. G. M. Gallo. Cambridge: Cambridge University Press.
- Greene, W. H. 2008. *Econometric Analysis*. 6th ed. Upper Saddle River, NJ: Prentice–Hall.

- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Harvey, A. C. 1989. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge: Cambridge University Press.
- . 1993. *Time Series Models*. 2nd ed. Cambridge, MA: MIT Press.
- Hipel, K. W., and A. I. McLeod. 1994. *Time Series Modelling of Water Resources and Environmental Systems*. Amsterdam: Elsevier.
- Kalman, R. E. 1960. A new approach to linear filtering and prediction problems. *Transactions of the ASME—Journal of Basic Engineering, Series D* 82: 35–45.
- McDowell, A. W. 2002. [From the help desk: Transfer functions](#). *Stata Journal* 2: 71–85.
- . 2004. [From the help desk: Polynomial distributed lag models](#). *Stata Journal* 4: 180–189.
- Newton, H. J. 1988. *TIMESLAB: A Time Series Analysis Laboratory*. Belmont, CA: Wadsworth.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. *Numerical Recipes in C: The Art of Scientific Computing*. 3rd ed. Cambridge: Cambridge University Press.
- Shumway, R. H. 1984. Some applications of the EM algorithm to analyzing incomplete time series data. In *Time Series Analysis of Irregularly Observed Data*, ed. E. Parzen, 290–324. New York: Springer.
- . 1988. *Applied Statistical Time Series Analysis*. Upper Saddle River, NJ: Prentice–Hall.

Also see

- [\[TS\] **arima postestimation**](#) — Postestimation tools for arima
- [\[TS\] **tsset**](#) — Declare data to be time-series data
- [\[TS\] **arch**](#) — Autoregressive conditional heteroskedasticity (ARCH) family of estimators
- [\[TS\] **dfactor**](#) — Dynamic-factor models
- [\[TS\] **dvech**](#) — Diagonal vech multivariate GARCH models
- [\[TS\] **prais**](#) — Prais–Winsten and Cochrane–Orcutt regression
- [\[TS\] **sspace**](#) — State-space models
- [\[R\] **regress**](#) — Linear regression
- [\[U\] **20 Estimation and postestimation commands**](#)

Description

The following postestimation commands are available for `arima`:

command	description
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	point estimates, standard errors, testing, and inference for generalized predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

<code>predict</code> [<i>type</i>] <i>newvar</i> [<i>if</i>] [<i>in</i>] [, <i>statistic options</i>]	
<i>statistic</i>	description
Main	
<code>xb</code>	predicted values for mean equation—the differenced series; the default
<code>stdp</code>	standard error of the linear prediction
<code>y</code>	predicted values for the mean equation in <i>y</i> —the undifferenced series
<code>mse</code>	mean squared error of the predicted values
<code><u>r</u>esiduals</code>	residuals or predicted innovations
<code><u>y</u>residuals</code>	residuals or predicted innovations in <i>y</i> , reversing any time-series operators

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.
Predictions are not available for conditional ARIMA models fit to panel data.

<i>options</i>	description
Options	
<code><u>d</u>ynamic(<i>time_constant</i>)</code>	how to handle the lags of y_t
<code><u>t</u>0(<i>time_constant</i>)</code>	set starting point for the recursions to <i>time_constant</i>
<code><u>s</u>tructural</code>	calculate considering the structural component only

time_constant is a # or a time literal, such as `td(1jan1995)` or `tq(1995q1)`; see [\[D\] dates and times](#).

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

Five statistics can be computed using `predict` after `arima`: the predictions from the model (the default also given by `xb`), the predictions after reversing any time-series operators applied to the dependent variable (`y`), the MSE of `xb` (`mse`), the predictions of residuals or innovations (`residual`), and the predicted residuals or innovations in terms of `y` (`yresiduals`). Given the dynamic nature of the ARMA component and because the dependent variable might be differenced, there are other ways of computing each. We can use all the data on the dependent variable that is available right up to the time of each prediction (the default, which is often called a one-step prediction), or we can use the data up to a particular time, after which the predicted value of the dependent variable is used recursively to make later predictions (`dynamic()`). Either way, we can consider or ignore the ARMA disturbance component (the component is considered by default and is ignored if you specify `structural`).

All calculations can be made in or out of sample.

Main

`xb`, the default, calculates the predictions from the model. If `D.depvar` is the dependent variable, these predictions are of `D.depvar` and not of `depvar` itself.

`stdp` calculates the standard error of the linear prediction `xb`. `stdp` does not include the variation arising from the disturbance equation; use `mse` to calculate standard errors and confidence bands around the predicted values.

`y` specifies that predictions of `depvar` be made, even if the model was specified in terms of, say, `D.depvar`.

`mse` calculates the MSE of the predictions.

`residuals` calculates the residuals. If no other options are specified, these are the predicted innovations ϵ_t ; i.e., they include the ARMA component. If `structural` is specified, these are the residuals μ_t from the structural equation; see `structural` below.

`yresiduals` calculates the residuals in terms of `depvar`, even if the model was specified in terms of, say, `D.depvar`. As with `residuals`, the `yresiduals` are computed from the model, including any ARMA component. If `structural` is specified, any ARMA component is ignored, and `yresiduals` are the residuals from the structural equation; see `structural` below.

Options

`dynamic(time_constant)` specifies how lags of y_t in the model are to be handled. If `dynamic()` is not specified, actual values are used everywhere that lagged values of y_t appear in the model to produce one-step-ahead forecasts.

`dynamic(time_constant)` produces dynamic (also known as recursive) forecasts. `time_constant` specifies when the forecast is to switch from one step ahead to dynamic. In dynamic forecasts, references to y_t evaluate to the prediction of y_t for all periods at or after `time_constant`; they evaluate to the actual value of y_t for all prior periods.

For example, `dynamic(10)` would calculate predictions in which any reference to y_t with $t < 10$ evaluates to the actual value of y_t and any reference to y_t with $t \geq 10$ evaluates to the prediction of y_t . This means that one-step-ahead predictions are calculated for $t < 10$ and dynamic predictions

thereafter. Depending on the lag structure of the model, the dynamic predictions might still refer some actual values of y_t .

You may also specify `dynamic(.)` to have `predict` automatically switch from one-step-ahead to dynamic predictions at $p + q$, where p is the maximum AR lag and q is the maximum MA lag.

`t0(time_constant)` specifies the starting point for the recursions to compute the predicted statistics; disturbances are assumed to be 0 for $t < t0()$. The default is to set `t0()` to the minimum t observed in the estimation sample, meaning that observations before that are assumed to have disturbances of 0.

`t0()` is irrelevant if `structural` is specified because then all observations are assumed to have disturbances of 0.

`t0(5)` would begin recursions at $t = 5$. If the data were quarterly, you might instead type `t0(tq(1961q2))` to obtain the same result.

The ARMA component of ARIMA models is recursive and depends on the starting point of the predictions. This includes one-step-ahead predictions.

`structural` specifies that the calculation be made considering the structural component only, ignoring the ARMA terms, producing the steady-state equilibrium predictions.

Remarks

We assume that you have already read [TS] [arima](#). In this entry, we illustrate some of the features of `predict` after fitting ARIMA, ARMAX, and other dynamic models by using `arima`. In [example 2](#) of [TS] [arima](#), we fit the model

$$\Delta \ln(wpi_t) = \beta_0 + \rho_1 \{\Delta \ln(wpi_{t-1}) - \beta_0\} + \theta_1 \epsilon_{t-1} + \theta_4 \epsilon_{t-4} + \epsilon_t$$

by typing

```
. use http://www.stata-press.com/data/r11/wpil
. arima D.ln_wpi, ar(1) ma(1 4)
(output omitted)
```

If we use the command

```
. predict xb, xb
```

then Stata computes xb_t as

$$xb_t = \hat{\beta}_0 + \hat{\rho}_1 \{\Delta \ln(wpi_{t-1}) - \hat{\beta}_0\} + \hat{\theta}_1 \hat{\epsilon}_{t-1} + \hat{\theta}_4 \hat{\epsilon}_{t-4}$$

where

$$\hat{\epsilon}_{t-j} = \begin{cases} \Delta \ln(wpi_{t-j}) - xb_{t-j} & t-j > 0 \\ 0 & \text{otherwise} \end{cases}$$

meaning that `predict newvar, xb` calculates predictions by using the metric of the dependent variable. In this example, the dependent variable represented *changes* in $\ln(wpi_t)$, and so the predictions are likewise for *changes* in that variable.

If we instead use

```
. predict y, y
```

Stata computes y_t as $y_t = xb_t + \ln(wpi_{t-1})$ so that y_t represents the predicted *levels* of $\ln(wpi_t)$. In general, `predict newvar, y` will reverse any time-series operators applied to the dependent variable during estimation.

If we want to ignore the ARMA error components when making predictions, we use the `structural` option,

```
. predict xbs, xb structural
```

which generates $xbs_t = \hat{\beta}_0$ because there are no regressors in this model, and

```
. predict ys, y structural
```

generates $ys_t = \hat{\beta}_0 + \ln(wpi_{t-1})$

► Example 1: Dynamic forecasts

An attractive feature of the `arima` command is the ability to make dynamic forecasts. In [example 4](#) of [\[TS\] arima](#), we fit the model

$$\begin{aligned} \text{consump}_t &= \beta_0 + \beta_1 m2_t + \mu_t \\ \mu_t &= \rho \mu_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \end{aligned}$$

First, we refit the model by using data up through the first quarter of 1978, and then we will evaluate the one-step-ahead and dynamic forecasts.

```
. use http://www.stata-press.com/data/r11/friedman2
. keep if time<=tq(1981q4)
(67 observations deleted)
. arima consump m2 if tin(, 1978q1), ar(1) ma(1)
(output omitted)
```

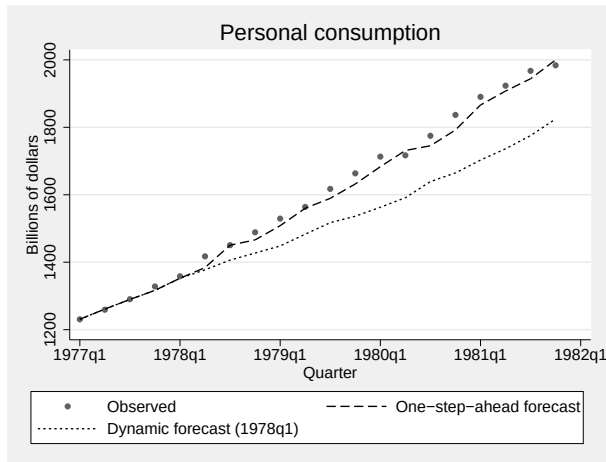
To make one-step-ahead forecasts, we type

```
. predict chat, y
(52 missing values generated)
```

(Because our dependent variable contained no time-series operators, we could have instead used `predict chat, xb` and accomplished the same thing.) We will also make dynamic forecasts, switching from observed values of `consump` to forecasted values at the first quarter of 1978:

```
. predict chatdy, dynamic(tq(1978q1)) y
(52 missing values generated)
```

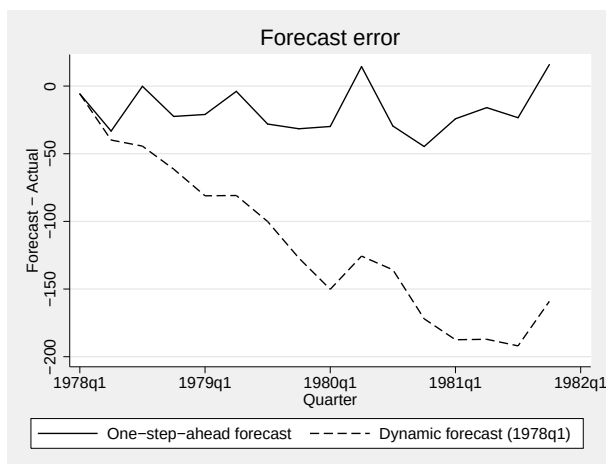
The following graph compares the forecasted values to the observed values for the first few years following the estimation sample:



The one-step-ahead forecasts never deviate far from the observed values, though over time the dynamic forecasts have larger errors. To understand why that is the case, rewrite the model as

$$\begin{aligned} \text{consump}_t &= \beta_0 + \beta_1 m2_t + \rho \mu_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \\ &= \beta_0 + \beta_1 m2_t + \rho (\text{consump}_{t-1} - \beta_0 - \beta_1 m2_{t-1}) + \theta \epsilon_{t-1} + \epsilon_t \end{aligned}$$

This form shows that the forecasted value of consumption at time t depends on the value of consumption at time $t - 1$. When making the one-step-ahead forecast for period t , we know the actual value of consumption at time $t - 1$. On the other hand, with the `dynamic(tq(1978q1))` option, the forecasted value of consumption for period 1978q1 is based on the observed value of consumption in period 1977q4, but the forecast for 1978q2 is based on the forecast value for 1978q1, the forecast for 1978q3 is based on the forecast value for 1978q2, and so on. Thus, with dynamic forecasts, prior forecast errors accumulate over time. The following graph illustrates this effect.



Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Also see

[TS] [arima](#) — ARIMA, ARMAX, and other dynamic regression models

[U] [20 Estimation and postestimation commands](#)

Syntax

Autocorrelations, partial autocorrelations, and Portmanteau (Q) statistics

```
corrgram varname [if] [in] [, corrgram_options]
```

Graph autocorrelations with confidence intervals

```
ac varname [if] [in] [, ac_options]
```

Graph partial autocorrelations with confidence intervals

```
pac varname [if] [in] [, pac_options]
```

corrgram_options	description
Main	
<u>lags</u> (#)	calculate # autocorrelations
<u>noplot</u>	suppress character-based plots
<u>yw</u>	calculate partial autocorrelations by using Yule–Walker equations

ac_options	description
Main	
<u>lags</u> (#)	calculate # autocorrelations
<u>generate</u> (<i>newvar</i>)	generate a variable to hold the autocorrelations
<u>level</u> (#)	set confidence level; default is <code>level(95)</code>
<u>fft</u>	calculate autocorrelation by using Fourier transforms

Plot	
<i>line_options</i>	change look of dropped lines
<i>marker_options</i>	change look of markers (color, size, etc.)
<i>marker_label_options</i>	add marker labels; change look or position

CI plot	
<u>ciopts</u> (<i>area_options</i>)	affect rendition of the confidence bands

Add plots	
<u>addplot</u> (<i>plot</i>)	add other plots to the generated graph

Y axis, X axis, Titles, Legend, Overall	
<i>twoway_options</i>	any options other than <code>by()</code> documented in [G] twoway_options

<i>pac_options</i>	description
Main	
<code>lags(#)</code>	calculate # partial autocorrelations
<code>generate(newvar)</code>	generate a variable to hold the partial autocorrelations
<code>yw</code>	calculate partial autocorrelations by using Yule–Walker equations
<code>level(#)</code>	set confidence level; default is <code>level(95)</code>
Plot	
<code>line_options</code>	change look of dropped lines
<code>marker_options</code>	change look of markers (color, size, etc.)
<code>marker_label_options</code>	add marker labels; change look or position
CI plot	
<code>ciopts(area_options)</code>	affect rendition of the confidence bands
SRV plot	
<code>srv</code>	include standardized residual variances in graph
<code>srvopts(marker_options)</code>	affect rendition of the plotted standardized residual variances (SRVs)
Add plots	
<code>addplot(plot)</code>	add other plots to the generated graph
Y axis, X axis, Titles, Legend, Overall	
<code>twoway_options</code>	any options other than <code>by()</code> documented in [G] twoway_options

You must `tsset` your data before using `corrgram`, `ac`, or `pac`; see [\[TS\] tsset](#). Also, the time series must be dense (nonmissing and no gaps in the time variable) in the sample if you specify the `fft` option. `varname` may contain time-series operators; see [\[U\] 11.4.4 Time-series varlists](#).

Menu

corrgram

Statistics > Time series > Graphs > Autocorrelations & partial autocorrelations

ac

Statistics > Time series > Graphs > Correlogram (ac)

pac

Statistics > Time series > Graphs > Partial correlogram (pac)

Description

`corrgram` produces a table of the autocorrelations, partial autocorrelations, and Portmanteau (Q) statistics. It also displays a character-based plot of the autocorrelations and partial autocorrelations. See [\[TS\] wntestq](#) for more information on the Q statistic.

`ac` produces a correlogram (a graph of autocorrelations) with pointwise confidence intervals that is based on Bartlett’s formula for MA(q) processes.

`pac` produces a partial correlogram (a graph of partial autocorrelations) with confidence intervals calculated using a standard error of $1/\sqrt{n}$. The residual variances for each lag may optionally be included on the graph.

Options for corrgram

Main

`lags(#)` specifies the number of autocorrelations to calculate. The default is to use $\min(\lfloor n/2 \rfloor - 2, 40)$, where $\lfloor n/2 \rfloor$ is the greatest integer less than or equal to $n/2$.

`noplots` prevents the character-based plots from being in the listed table of autocorrelations and partial autocorrelations.

`yw` specifies that the partial autocorrelations be calculated using the Yule–Walker equations instead of using the default regression-based technique. `yw` cannot be used if `srv` is used.

Options for ac and pac

Main

`lags(#)` specifies the number of autocorrelations to calculate. The default is to use $\min(\lfloor n/2 \rfloor - 2, 40)$, where $\lfloor n/2 \rfloor$ is the greatest integer less than or equal to $n/2$.

`generate(newvar)` specifies a new variable to contain the autocorrelation (`ac` command) or partial autocorrelation (`pac` command) values. This option is required if the `nograph` option is used.

`nograph` (implied when using `generate()` in the dialog box) prevents `ac` and `pac` from constructing a graph. This option requires the `generate()` option.

`yw` (`pac` only) specifies that the partial autocorrelations be calculated using the Yule–Walker equations instead of using the default regression-based technique. `yw` cannot be used if `srv` is used.

`level(#)` specifies the confidence level, as a percentage, for the confidence bands in the `ac` or `pac` graph. The default is `level(95)` or as set by `set level`; see [\[R\] level](#).

`fft` (`ac` only) specifies that the autocorrelations be calculated using two Fourier transforms. This technique can be faster than simply iterating over the requested number of lags.

Plot

`line_options`, `marker_options`, and `marker_label_options` affect the rendition of the plotted autocorrelations (with `ac`) or partial autocorrelations (with `pac`).

`line_options` specify the look of the dropped lines, including pattern, width, and color; see [\[G\] line_options](#).

`marker_options` specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [\[G\] marker_options](#).

`marker_label_options` specify if and how the markers are to be labeled; see [\[G\] marker_label_options](#).

CI plot

`ciopts(area_options)` affects the rendition of the confidence bands; see [\[G\] area_options](#).

SRV plot

`srv` (`pac` only) specifies that the standardized residual variances be plotted with the partial autocorrelations. `srv` cannot be used if `yw` is used.

`srvopts(marker_options)` (`pac` only) affects the rendition of the plotted standardized residual variances; see [G] [marker_options](#). This option implies the `srv` option.

Add plots

`addplot(plot)` adds specified plots to the generated graph; see [G] [addplot_option](#).

Y axis, X axis, Titles, Legend, Overall

`twoway_options` are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

Remarks

`corrgram` tabulates autocorrelations, partial autocorrelations, and Portmanteau (Q) statistics and plots the autocorrelations and partial autocorrelations. The Q statistics are the same as those produced by [TS] `wntestq`. `ac` produces graphs of the autocorrelations, and `pac` produces graphs of the partial autocorrelations.

► Example 1

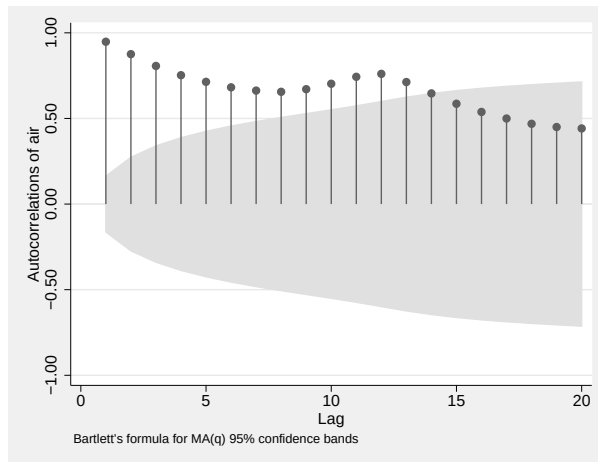
Here we use the international airline passengers dataset (Box, Jenkins, and Reinsel 1994, Series G). This dataset has 144 observations on the monthly number of international airline passengers from 1949 through 1960. We can list the autocorrelations and partial autocorrelations by using `corrgram`.

```
. use http://www.stata-press.com/data/r11/air2
(TIMESLAB: Airline passengers)
. corrgram air, lags(20)
```

LAG	AC	PAC	Q	Prob>Q	-1 0 1 [Autocorrelation]	-1 0 1 [Partial Autocor]
1	0.9480	0.9589	132.14	0.0000		
2	0.8756	-0.3298	245.65	0.0000		
3	0.8067	0.2018	342.67	0.0000		
4	0.7526	0.1450	427.74	0.0000		
5	0.7138	0.2585	504.8	0.0000		
6	0.6817	-0.0269	575.6	0.0000		
7	0.6629	0.2043	643.04	0.0000		
8	0.6556	0.1561	709.48	0.0000		
9	0.6709	0.5686	779.59	0.0000		
10	0.7027	0.2926	857.07	0.0000		
11	0.7432	0.8402	944.39	0.0000		
12	0.7604	0.6127	1036.5	0.0000		
13	0.7127	-0.6660	1118	0.0000		
14	0.6463	-0.3846	1185.6	0.0000		
15	0.5859	0.0787	1241.5	0.0000		
16	0.5380	-0.0266	1289	0.0000		
17	0.4997	-0.0581	1330.4	0.0000		
18	0.4687	-0.0435	1367	0.0000		
19	0.4499	0.2773	1401.1	0.0000		
20	0.4416	-0.0405	1434.1	0.0000		

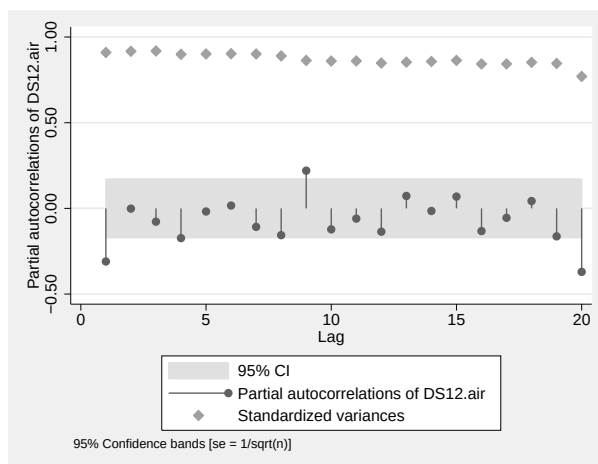
We can use `ac` to produce a graph of the autocorrelations.

```
. ac air, lags(20)
```



The data probably have a trend component as well as a seasonal component. First-differencing will mitigate the effects of the trend, and seasonal differencing will help control for seasonality. To accomplish this goal, we can use Stata's time-series operators. Here we graph the partial autocorrelations after controlling for trends and seasonality. We also use `srv` to include the standardized residual variances.

```
. pac DS12.air, lags(20) srv
```



See [\[U\] 11.4.4 Time-series varlists](#) for more information about time-series operators.

Saved results

`corrgram` saves the following in `r()`:

Scalars

`r(lags)` number of lags

Matrices

`r(AC)` vector of autocorrelations

`r(PAC)` vector of partial autocorrelations

`r(Q)` vector of Q statistics

`r(ac#)` AC for lag #

`r(pac#)` PAC for lag #

`r(q#)` Q for lag #

Methods and formulas

`corrgram`, `ac`, and `pac` are implemented as ado-files.

[Box, Jenkins, and Reinsel \(1994\)](#); [Newton \(1988\)](#); [Chatfield \(2004\)](#); and [Hamilton \(1994\)](#) provide excellent descriptions of correlograms. [Newton \(1988\)](#) also discusses the calculation of the various quantities.

The autocovariance function for a time series $x_1, x_2, \dots, x_n$ is defined for $|v| < n$ as

$$\hat{R}(v) = \frac{1}{n} \sum_{i=1}^{n-|v|} (x_i - \bar{x})(x_{i+v} - \bar{x})$$

where $\bar{x}$ is the sample mean, and the autocorrelation function is then defined as

$$\hat{\rho}_v = \frac{\hat{R}(v)}{\hat{R}(0)}$$

The variance of $\hat{\rho}_v$ is given by Bartlett's formula for MA(q) processes. From [Brockwell and Davis \(2002, 94\)](#), we have

$$\text{Var}(\hat{\rho}_v) = \begin{cases} 1/n & v = 1 \\ \frac{1}{n} \left\{ 1 + 2 \sum_{i=1}^{v-1} \hat{\rho}^2(i) \right\} & v > 1 \end{cases}$$

The partial autocorrelation at lag v measures the correlation between x_t and x_{t+v} after the effects of $x_{t+1}, \dots, x_{t+v-1}$ have been removed. By default, `corrgram` and `pac` use a regression-based method to estimate it. We run an OLS regression of x_t on $x_{t-1}, \dots, x_{t-v}$ and a constant term. The estimated coefficient on x_{t-v} is our estimate of the v th partial autocorrelation. The residual variance is the estimated variance of that regression, which we then standardize by dividing by $\hat{R}(0)$.

If the `yw` option is specified, `corrgram` and `pac` use the Yule–Walker equations to estimate the partial autocorrelations. Per [Enders \(2004, 65\)](#), let ϕ_{vv} denote the v th partial autocorrelation coefficient. We then have

$$\hat{\phi}_{11} = \hat{\rho}_1$$

and for $v > 1$

$$\hat{\phi}_{vv} = \frac{\hat{\rho}_v - \sum_{j=1}^{v-1} \hat{\phi}_{v-1,j} \hat{\rho}_{v-j}}{1 - \sum_{j=1}^{v-1} \hat{\phi}_{v-1,j} \hat{\rho}_j}$$

and

$$\hat{\phi}_{vj} = \hat{\phi}_{v-1,j} - \hat{\phi}_{vv} \hat{\phi}_{v-1,v-j} \quad j = 1, 2, \dots, v-1$$

Unlike the regression-based method, the Yule–Walker equations-based method ensures that the first-sample partial autocorrelation equal the first-sample autocorrelation coefficient, as must be true in the population; see [Greene \(2008, 725\)](#).

[McCullough \(1998\)](#) discusses other methods of estimating ϕ_{vv} ; he finds that relative to other methods, such as linear regression, the Yule–Walker equations-based method performs poorly, in part because it is susceptible to numerical error. [Box, Jenkins, and Reinsel \(1994, 68\)](#) also caution against using the Yule–Walker equations-based method, especially with data that are nearly nonstationary.

Acknowledgment

The `ac` and `pac` commands are based on the `ac` and `pac` commands written by Sean [Beckett \(1992\)](#), a past editor of the *Stata Technical Bulletin*.

References

- Beckett, S. 1992. [stsl: Autocorrelation and partial autocorrelation graphs](#). *Stata Technical Bulletin* 5: 27–28. Reprinted in *Stata Technical Bulletin Reprints*, vol. 1, pp. 221–223. College Station, TX: Stata Press.
- Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control*. 3rd ed. Englewood Cliffs, NJ: Prentice–Hall.
- Brockwell, P. J., and R. A. Davis. 2002. *Introduction to Times Series and Forecasting*. 2nd ed. New York: Springer.
- Chatfield, C. 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- Enders, W. 2004. *Applied Econometric Time Series*. 2nd ed. New York: Wiley.
- Greene, W. H. 2008. *Econometric Analysis*. 6th ed. Upper Saddle River, NJ: Prentice–Hall.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- McCullough, B. D. 1998. Algorithm choice for (partial) autocorrelation functions. *Journal of Economic and Social Measurement* 24: 265–278.
- Newton, H. J. 1988. *TIMESLAB: A Time Series Analysis Laboratory*. Belmont, CA: Wadsworth.

Also see

- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [wntestq](#) — Portmanteau (Q) test for white noise
- [TS] [pergram](#) — Periodogram

Title

cumsp — Cumulative spectral distribution

Syntax

cumsp *varname* [*if*] [*in*] [, *options*]

<i>options</i>	description
Main	
<u>generate</u> (<i>newvar</i>)	create <i>newvar</i> holding distribution values
Plot	
<i>cline_options</i>	affect rendition of the plotted points connected by lines
<i>marker_options</i>	change look of markers (color, size, etc.)
<i>marker_label_options</i>	add marker labels; change look or position
Add plots	
<u>addplot</u> (<i>plot</i>)	add other plots to the generated graph
Y axis, X axis, Titles, Legend, Overall	
<i>twoway_options</i>	any options other than by() documented in [G] <i>twoway_options</i>

You must `tsset` your data before using `cumsp`; see [TS] `tsset`. Also, the time series must be dense (nonmissing with no gaps in the time variable) in the sample specified.

varname may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Menu

Statistics > Time series > Graphs > Cumulative spectral distribution

Description

`cumsp` plots the cumulative sample spectral-distribution function evaluated at the natural frequencies for a (dense) time series.

Options

Main

`generate`(*newvar*) specifies a new variable to contain the estimated cumulative spectral-distribution values.

Plot

cline_options affect the rendition of the plotted points connected by lines; see [G] *cline_options*.

marker_options specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G] *marker_options*.

marker_label_options specify if and how the markers are to be labeled; see [G] *marker_label_options*.

Add plots

`addplot(plot)` provides a way to add other plots to the generated graph; see [G] [addplot_option](#).

Y axis, X axis, Titles, Legend, Overall

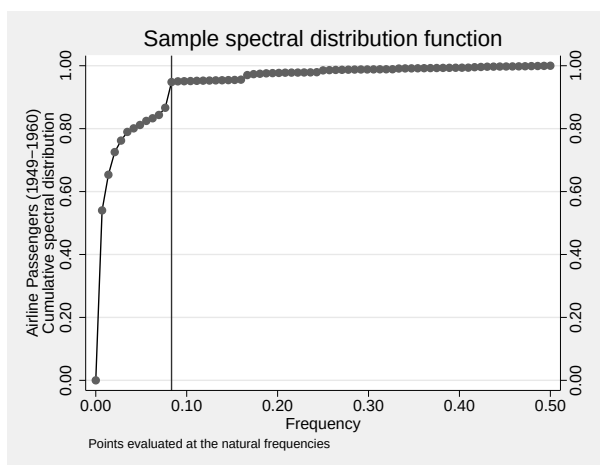
`twoway_options` are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

Remarks

► Example 1

Here we use the international airline passengers dataset (Box, Jenkins, and Reinsel 1994, Series G). This dataset has 144 observations on the monthly number of international airline passengers from 1949 through 1960. In the cumulative sample spectral distribution function for these data, we also request a vertical line at frequency $1/12$. Because the data are monthly, there will be a pronounced jump in the cumulative sample spectral-distribution plot at the $1/12$ value if there is an annual cycle in the data.

```
. use http://www.stata-press.com/data/r11/air2
(TIMESLAB: Airline passengers)
. cumsp air, xline(.08333333)
```



The cumulative sample spectral-distribution function clearly illustrates the annual cycle.

4

Methods and formulas

`cumsp` is implemented as an ado-file.

A time series of interest is decomposed into a unique set of sinusoids of various frequencies and amplitudes.

A plot of the sinusoidal amplitudes versus the frequencies for the sinusoidal decomposition of a time series gives us the spectral density of the time series. If we calculate the sinusoidal amplitudes for a discrete set of “natural” frequencies $(1/n, 2/n, \dots, q/n)$, we obtain the periodogram.

Let $x(1), \dots, x(n)$ be a time series, and let $\omega_k = (k-1)/n$ denote the natural frequencies for $k = 1, \dots, \lfloor n/2 \rfloor + 1$ where $\lfloor \cdot \rfloor$ indicates the greatest integer function. Define

$$C_k^2 = \frac{1}{n^2} \left| \sum_{t=1}^n x(t) e^{2\pi i(t-1)\omega_k} \right|^2$$

A plot of nC_k^2 versus ω_k is then called the periodogram.

The sample spectral density may then be defined as $\hat{f}(\omega_k) = nC_k^2$.

If we let $\hat{f}(\omega_1), \dots, \hat{f}(\omega_Q)$ be the sample spectral density function of the time series evaluated at the frequencies $\omega_j = (j-1)/Q$ for $j = 1, \dots, Q$ and we let $q = \lfloor Q/2 \rfloor + 1$, then

$$\hat{F}(\omega_k) = \frac{\sum_{j=1}^q \hat{f}(\omega_j)}{\sum_{j=1}^q \hat{f}(\omega_j)}$$

is the sample spectral-distribution function of the time series.

References

- Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall.
- Newton, H. J. 1988. *TIMESLAB: A Time Series Analysis Laboratory*. Belmont, CA: Wadsworth.

Also see

- [TS] **tsset** — Declare data to be time-series data
- [TS] **corrgram** — Tabulate and graph autocorrelations
- [TS] **pergram** — Periodogram

Syntax

```
dfactor obs_eq [fac_eq] [if] [in] [, options]
```

obs_eq specifies the equation for the observed dependent variables, and it has the form
(*depvars* = [*exog_d*] [, *sopts*])

fac_eq specifies the equation for the unobserved factors, and it has the form
(*facvars* = [*exog_f*] [, *sopts*])

depvars are the observed dependent variables. *exog_d* are the exogenous variables that enter into the equations for the observed dependent variables. (All factors are automatically entered into the equations for the observed dependent variables.) *facvars* are the names for the unobserved factors in the model. You may specify the names of existing variables in *facvars*, but **dfactor** treats them only as names and takes no notice that they are also variables. *exog_f* are the exogenous variables that enter into the equations for the factors.

<i>options</i>	description
Model	
<u>constraints</u> (<i>constraints</i>)	apply specified linear constraints
SE/Robust	
<u>vce</u> (<i>vcetype</i>)	<i>vcetype</i> may be <i>oim</i> or <u>robust</u>
Reporting	
<u>level</u> (#)	set confidence level; default is <code>level(95)</code>
<u>nocnsreport</u>	do not display constraints
<i>display_options</i>	control spacing and display of omitted variables and base and empty cells
Maximization	
<i>maximize_options</i>	control the maximization process; seldom used
<u>from</u> (<i>matname</i>)	specify initial values for the maximization process; seldom used
Advanced	
<u>method</u> (<i>method</i>)	specify the method for calculating the log likelihood; seldom used
[†] <u>coeflegend</u>	display coefficients' legend instead of coefficient table

<i>sopts</i>	description
Model	
<u>noconstant</u>	suppress constant term from the equation; allowed only in <i>obs_eq</i>
<u>ar</u> (<i>numlist</i>)	autoregressive terms
<u>arstructure</u> (<i>arstructure</i>)	structure of autoregressive coefficient matrices
<u>covstructure</u> (<i>covstructure</i>)	covariance structure
<i>arstructure</i>	description
<u>diagonal</u>	diagonal matrix; the default
<u>ltriangular</u>	lower triangular matrix
<u>general</u>	general matrix
<i>covstructure</i>	description
<u>identity</u>	identity matrix
<u>dscalar</u>	diagonal scalar matrix
<u>diagonal</u>	diagonal matrix
<u>unstructured</u>	symmetric, positive-definite matrix
<i>method</i>	description
<u>hybrid</u>	use the stationary Kalman filter and the De Jong diffuse Kalman filter; the default
<u>dejong</u>	use the stationary De Jong method and the De Jong diffuse Kalman filter

[†] `coeflegend` does not appear in the dialog box.
You must `tsset` your data before using `dfactor`; see [TS] [tsset](#).
`exog_d` and `exog_f` may contain factor variables; see [U] [11.4.3 Factor variables](#).
`depvars`, `exog_d`, and `exog_f` may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).
`by`, `statsby`, and `rolling` are allowed; see [U] [11.1.10 Prefix commands](#).
See [U] [20 Estimation and postestimation commands](#) for more capabilities of estimation commands.

Menu

Statistics > Multivariate time series > Dynamic-factor models

Description

`dfactor` estimates the parameters of dynamic-factor models by maximum likelihood. Dynamic-factor models are flexible models for multivariate time series in which unobserved factors have a vector autoregressive structure, exogenous covariates are permitted in both the equations for the latent factors and the equations for observable dependent variables, and the disturbances in the equations for the dependent variables may be autocorrelated.

Options

Model

`constraints`(*constraints*) apply linear constraints. Some specifications require linear constraints for parameter identification.

`noconstant` suppresses the constant term.

`ar`(*numlist*) specifies the vector autoregressive lag structure in the equation. By default, no lags are included in either the observable or the factor equations.

`arstructure`(*diagonal*|*ltriangular*|*general*) specifies the structure of the matrices in the vector autoregressive lag structure.

`arstructure`(*diagonal*) specifies the matrices to be diagonal—separate parameters for each lag, but no cross-equation autocorrelations. `arstructure`(*diagonal*) is the default for both the observable and the factor equations.

`arstructure`(*ltriangular*) specifies the matrices to be lower triangular—parameterizes a recursive, or Wold causal, structure.

`arstructure`(*general*) specifies the matrices to be general matrices—separate parameters for each possible autocorrelation and cross-correlation.

`covstructure`(*identity*|*dscalar*|*diagonal*|*unstructured*) specifies the covariance structure of the errors.

`covstructure`(*identity*) specifies a covariance matrix equal to an identity matrix, and it is the default for the errors in the factor equations.

`covstructure`(*dscalar*) specifies a covariance matrix equal to σ^2 times an identity matrix.

`covstructure`(*diagonal*) specifies a diagonal covariance matrix, and it is the default for the errors in the observable variables.

`covstructure`(*unstructured*) specifies a symmetric, positive-definite covariance matrix with parameters for all variances and covariances.

SE/Robust

`vce`(*vcetype*) specifies the estimator for the variance–covariance matrix of the estimator.

`vce`(*oim*), the default, causes `dfactor` to use the observed information matrix estimator.

`vce`(*robust*) causes `dfactor` to use the Huber/White/sandwich estimator.

Reporting

`level`(#); see [\[R\] estimation options](#).

`nocnsreport`; see [\[R\] estimation options](#).

`display_options`: `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`; see [\[R\] estimation options](#).

Maximization

`maximize_options`: `difficult`, `technique`(*algorithm_spec*), `iterate`(#), `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance`(#), `ltolerance`(#), `nrtolerance`(#), `from`(*matname*); see [\[R\] maximize](#) for all options except `from`(), and see below for information on `from`(). These options are seldom used.

`from(matname)` specifies initial values for the maximization process. `from(b0)` causes `dfactor` to begin the maximization algorithm with the values in `b0`. `b0` must be a row vector; the number of columns must equal the number of parameters in the model; and the values in `b0` must be in the same order as the parameters in `e(b)`. This option is seldom used.

Advanced

`method(method)` specifies how to compute the log likelihood. `dfactor` writes the model in state-space form and uses `sspace` to estimate the parameters. `method()` offers two methods for dealing with some of the technical aspects of the state-space likelihood. This option is seldom used.

`method(hybrid)`, the default, uses the Kalman filter with model-based initial values when the model is stationary and uses the [De Jong \(1988, 1991\)](#) diffuse Kalman filter when the model is nonstationary.

`method(dejong)` uses the [De Jong \(1988\)](#) method for estimating the initial values for the Kalman filter when the model is stationary and uses the [De Jong \(1988, 1991\)](#) diffuse Kalman filter when the model is nonstationary.

The following option is available with `dfactor` but is not shown in the dialog box:

`coeflegend`; see [\[R\] estimation options](#).

Remarks

Remarks are presented under the following headings:

[An introduction to dynamic-factor models](#)
[Some examples](#)

An introduction to dynamic-factor models

`dfactor` estimates the parameters of dynamic-factor models by maximum likelihood (ML). Dynamic-factor models represent a vector of k endogenous variables as linear functions of $n_f < k$ unobserved factors and some exogenous covariates. The unobserved factors and the disturbances in the equations for the observed variables may follow vector autoregressive structures.

Dynamic-factor models have been developed and applied in macroeconomics; see [Geweke \(1977\)](#), [Sargent and Sims \(1977\)](#), [Stock and Watson \(1989, 1991\)](#), and [Watson and Engle \(1983\)](#).

Dynamic-factor models are very flexible; in a sense, they are too flexible. Constraints must be imposed to identify the parameters of dynamic-factor and static-factor models. The parameters in the default specifications in `dfactor` are identified, but other specifications require additional restrictions. The factors are identified only up to a sign, which means that the coefficients on the unobserved factors can flip signs and still produce the same predictions and the same log likelihood. The flexibility of the model sometimes produces convergence problems.

`dfactor` is designed to handle cases in which the number of modeled endogenous variables, k , is small. The ML estimator is implemented by writing the model in state-space form and by using the Kalman filter to derive and implement the log likelihood. As k grows, the number of parameters quickly exceeds the number that can be estimated.

A dynamic-factor model has the form

$$\begin{aligned} \mathbf{y}_t &= \mathbf{P}\mathbf{f}_t + \mathbf{Q}\mathbf{x}_t + \mathbf{u}_t \\ \mathbf{f}_t &= \mathbf{R}\mathbf{w}_t + \mathbf{A}_1\mathbf{f}_{t-1} + \mathbf{A}_2\mathbf{f}_{t-2} + \cdots + \mathbf{A}_{t-p}\mathbf{f}_{t-p} + \boldsymbol{\nu}_t \\ \mathbf{u}_t &= \mathbf{C}_1\mathbf{u}_{t-1} + \mathbf{C}_2\mathbf{u}_{t-2} + \cdots + \mathbf{C}_{t-q}\mathbf{u}_{t-q} + \boldsymbol{\epsilon}_t \end{aligned}$$

where the definitions are given in the following table:

Item	dimension	definition
$\mathbf{y}_t$	$k \times 1$	vector of dependent variables
$\mathbf{P}$	$k \times n_f$	matrix of parameters
$\mathbf{f}_t$	$n_f \times 1$	vector of unobservable factors
$\mathbf{Q}$	$k \times n_x$	matrix of parameters
$\mathbf{x}_t$	$n_x \times 1$	vector of exogenous variables
$\mathbf{u}_t$	$k \times 1$	vector of disturbances
$\mathbf{R}$	$n_f \times n_w$	matrix of parameters
$\mathbf{w}_t$	$n_w \times 1$	vector of exogenous variables
$\mathbf{A}_i$	$n_f \times n_f$	matrix of autocorrelation parameters for $i \in \{1, 2, \dots, p\}$
$\boldsymbol{\nu}_t$	$n_f \times 1$	vector of disturbances
$\mathbf{C}_i$	$k \times k$	matrix of autocorrelation parameters for $i \in \{1, 2, \dots, q\}$
$\boldsymbol{\epsilon}_t$	$k \times 1$	vector of disturbances

By selecting different numbers of factors and lags, the dynamic-factor model encompasses the six models in the table below:

Dynamic factors with vector autoregressive errors	(DFAR)	$n_f > 0$	$p > 0$	$q > 0$
Dynamic factors	(DF)	$n_f > 0$	$p > 0$	$q = 0$
Static factors with vector autoregressive errors	(SFAR)	$n_f > 0$	$p = 0$	$q > 0$
Static factors	(SF)	$n_f > 0$	$p = 0$	$q = 0$
Vector autoregressive errors	(VAR)	$n_f = 0$	$p = 0$	$q > 0$
Seemingly unrelated regression	(SUR)	$n_f = 0$	$p = 0$	$q = 0$

In addition to the time-series models, **dfactor** can estimate the parameters of SF models and SUR models. **dfactor** can place equality constraints on the disturbance covariances, which **sureg** and **var** do not allow.

Some examples

► Example 1

Stock and Watson (1989, 1991) wrote a simple macroeconomic model as a DF model, estimated the parameters by ML, and extracted an economic indicator. In this example, we estimate the parameters of a DF model. In [TS] **dfactor postestimation**, we extend this example and extract an economic indicator for the differenced series.

We have data on an industrial-production index, **ipman**; real disposable income, **income**; an aggregate weekly hours index, **hours**; and aggregate unemployment, **unemp**. We believe that these variables are first-difference stationary. We model their first-differences as linear functions of an unobserved factor that follows a second-order autoregressive process.

```
. use http://www.stata-press.com/data/r11/dfex
(St. Louis Fed (FRED) macro data)

. dfactor (D.(ipman income hours unemp) = , noconstant) (f = , ar(1/2))
searching for initial values .....
(setting technique to bhhh)
Iteration 0:   log likelihood = -719.79299
Iteration 1:   log likelihood = -675.96316
(output omitted)
Refining estimates:
Iteration 0:   log likelihood = -662.09507
Iteration 1:   log likelihood = -662.09507

Dynamic-factor model
Sample: 1972m2 - 2008m11
Log likelihood = -662.09507
Number of obs   =      442
Wald chi2(6)    =      751.95
Prob > chi2     =      0.0000
```

		OIM		z	P> z	[95% Conf. Interval]	
		Coef.	Std. Err.				
f	f						
	L1.	.2651932	.0568663	4.66	0.000	.1537372	.3766491
	L2.	.4820398	.0624635	7.72	0.000	.3596136	.604466
D.ipman	f	.3502249	.0287389	12.19	0.000	.2938976	.4065522
D.income	f	.0746338	.0217319	3.43	0.001	.0320401	.1172276
D.hours	f	.2177469	.0186769	11.66	0.000	.1811407	.254353
D.unemp	f	-.0676016	.0071022	-9.52	0.000	-.0815217	-.0536816
var(De.ipman)		.1383158	.0167086	8.28	0.000	.1055675	.1710641
var(De.inc~e)		.2773808	.0188302	14.73	0.000	.2404743	.3142873
var(De.hours)		.0911446	.0080847	11.27	0.000	.0752988	.1069903
var(De.unemp)		.0237232	.0017932	13.23	0.000	.0202086	.0272378

Note: Tests of variances against zero are conservative and are provided only for reference.

For a discussion of the atypical iteration log, see [example 1](#) in [\[TS\] sspace](#).

The header in the output describes the estimation sample, reports the log-likelihood function at the maximum, and gives the results of a Wald test against the null hypothesis that the coefficients on the independent variables, the factors, and the autoregressive components are all zero. In this example, the null hypothesis that all parameters except for the variance parameters are zero is rejected at all conventional levels.

The results in the estimation table indicate that the unobserved factor is quite persistent and that it is a significant predictor for each of the observed variables.

The footer informs us that the reported tests of the variance parameters against zero are conservative. These tests are conservative because the variance parameters must be greater than or equal to zero.

`dfactor` writes the DF model as a state-space model and uses the same methods as `sspace` to estimate the parameters. [Example 5](#) in [TS] `sspace` writes the model considered here in state-space form and uses `sspace` to estimate the parameters.



□ Technical note

The signs of the coefficients on the unobserved factors are not identified. They are not identified because we can multiply the unobserved factors and the coefficients on the unobserved factors by negative one without changing the log likelihood or any of the model predictions.

Altering either the starting values for the maximization process, the maximization `technique()` used, or the platform on which the command is run can cause the signs of the estimated coefficients on the unobserved factors to change.

Changes in the signs of the estimated coefficients on the unobserved factors do not alter the implications of the model or the model predictions.



▷ Example 2

Here we extend the previous example by allowing the errors in the equations for the observables to be autocorrelated. This extension yields a constrained VAR model with an unobserved autocorrelated factor.

We estimate the parameters by typing

(Continued on next page)

```
. dfactor (D.(ipman income hours unemp) = , noconstant ar(1)) (f = , ar(1/2))
searching for initial values .....
(setting technique to bhhh)
Iteration 0:   log likelihood = -647.99719
Iteration 1:   log likelihood = -618.43253
(output omitted)
Refining estimates:
Iteration 0:   log likelihood = -610.28846
Iteration 1:   log likelihood = -610.28846

Dynamic-factor model
Sample: 1972m2 - 2008m11
Log likelihood = -610.28846
Number of obs   =      442
Wald chi2(10)   =     990.91
Prob > chi2     =      0.0000
```

		OIM				
		Coef.	Std. Err.	z	P> z	[95% Conf. Interval]
f						
	f					
	L1.	.4058457	.0906183	4.48	0.000	.2282371 .5834543
	L2.	.3663499	.0849584	4.31	0.000	.1998344 .5328654
De.ipman						
	e.ipman					
	LD.	-.2772149	.068808	-4.03	0.000	-.4120761 -.1423538
De.income						
	e.income					
	LD.	-.2213824	.0470578	-4.70	0.000	-.3136141 -.1291508
De.hours						
	e.hours					
	LD.	-.3969317	.0504256	-7.87	0.000	-.495764 -.2980994
De.unemp						
	e.unemp					
	LD.	-.1736835	.0532071	-3.26	0.001	-.2779675 -.0693995
D.ipman						
	f	.3214972	.027982	11.49	0.000	.2666535 .3763408
D.income						
	f	.0760412	.0173844	4.37	0.000	.0419684 .110114
D.hours						
	f	.1933165	.0172969	11.18	0.000	.1594151 .2272179
D.unemp						
	f	-.0711994	.0066553	-10.70	0.000	-.0842435 -.0581553
var(De.ipman)		.1387909	.0154558	8.98	0.000	.1084981 .1690837
var(De.inc~e)		.2636239	.0179043	14.72	0.000	.2285322 .2987157
var(De.hours)		.0822919	.0071096	11.57	0.000	.0683574 .0962265
var(De.unemp)		.0218056	.0016658	13.09	0.000	.0185407 .0250704

Note: Tests of variances against zero are conservative and are provided only for reference.

The autoregressive (AR) terms are displayed in error notation. `e.varname` stands for the error in the equation for *varname*. The estimate of the *p*th AR term from *y1* on *y2* is reported as `Lpe.y1` in equation `e.y2`. In the above output, the estimated first-order AR term of `D.ipman` on `D.ipman` is `-0.277` and is labeled as `LDe.ipman` in equation `De.ipman`.



The previous two examples illustrate how to use `dfactor` to estimate the parameters of DF models. Although the previous example indicates that the more general DFAR model fits the data well, we use these data to illustrate how to estimate the parameters of more restrictive models.

➤ Example 3

In this example, we use `dfactor` to estimate the parameters of a SUR model with constraints on the error-covariance matrix. The model is also a constrained VAR with constraints on the error-covariance matrix, because we include the lags of two dependent variables as exogenous variables to model the dynamic structure of the data. Previous exploratory work suggested that we should drop the lag of `D.unemp` from the model.

(Continued on next page)

```
. constraint 1 [cov(De.unemp,De.income)]_cons = 0
. dfactor (D.(ipman income unemp) = LD.(ipman income), noconstant
> covstructure(unstructured)), constraints(1)
searching for initial values ..
(setting technique to bhhh)
Iteration 0:   log likelihood = -535.15602
Iteration 1:   log likelihood = -535.14302
(output omitted)
Refining estimates:
Iteration 0:   log likelihood = -535.12973
Iteration 1:   log likelihood = -535.12973
Dynamic-factor model
Sample: 1972m3 - 2008m11
Log likelihood = -535.12973
(1) [cov(De.unemp,De.income)]_cons = 0
```

Number of obs = 441
Wald chi2(6) = 88.32
Prob > chi2 = 0.0000

		OIM		z	P> z	[95% Conf. Interval]	
		Coef.	Std. Err.				
D.ipman							
	ipman						
	LD.	.206276	.0471654	4.37	0.000	.1138335	.2987185
income							
	LD.	.1867384	.0512139	3.65	0.000	.086361	.2871158
D.income							
	ipman						
	LD.	.1043733	.0434048	2.40	0.016	.0193015	.1894451
income							
	LD.	-.1957893	.0471305	-4.15	0.000	-.2881634	-.1034153
D.unemp							
	ipman						
	LD.	-.0865823	.0140747	-6.15	0.000	-.1141681	-.0589964
income							
	LD.	-.0200749	.0152828	-1.31	0.189	-.0500285	.0098788
var(De.ipman)		.3243902	.0218533	14.84	0.000	.2815584	.3672219
cov(De.inc~e,							
De.ipman)		.0445794	.013696	3.25	0.001	.0177358	.071423
cov(De.unemp,							
De.ipman)		-.0298076	.0047755	-6.24	0.000	-.0391674	-.0204478
var(De.inc~e)		.2747234	.0185008	14.85	0.000	.2384624	.3109844
cov(De.unemp,							
De.inc~e)		(omitted)					
var(De.unemp)		.0288866	.0019453	14.85	0.000	.0250738	.0326994

Note: Tests of variances against zero are conservative and are provided only for reference.

The output indicates that the model fits well, except that the lag of first-differenced income is not a significant predictor of first-differenced unemployment.

□ Technical note

The [previous example](#) shows how to use `dfactor` to estimate the parameters of a SUR model with constraints on the error-covariance matrix. Neither `sureg` nor `var` allow for constraints on the error-covariance matrix. Without the constraints on the error-covariance matrix and including the lag of `D.unemp`,

```
. dfactor (D.(ipman income unemp) = LD.(ipman income unemp),
> noconstant covstructure(unstructured))
(output omitted)

. var D.(ipman income unemp), lags(1) noconstant
(output omitted)
```

and

```
. sureg (D.ipman LD.(ipman income unemp), noconstant)
>        (D.income LD.(ipman income unemp), noconstant)
>        (D.unemp LD.(ipman income unemp), noconstant)
(output omitted)
```

produce the same estimates after allowing for small numerical differences.

□

▷ Example 4

The [previous example](#) estimated the parameters of a constrained VAR model with a constraint on the error-covariance matrix. This example makes two refinements on the previous one: we use an unconditional estimator instead of a conditional estimator, and we constrain the AR parameters to have a lower triangular structure. (See the next [technical note](#) for a discussion of conditional and unconditional estimators.) The results are

(Continued on next page)

```
. constraint 1 [cov(De.unemp,De.income)]_cons = 0
. dfactor (D.(ipman income unemp) = , ar(1) arstructure(ltriangular) noconstant
> covstructure(unstructured)), constraints(1)
searching for initial values ....
(setting technique to bhhh)
Iteration 0:   log likelihood = -544.1134
Iteration 1:   log likelihood = -541.44043
(output omitted)
Refining estimates:
Iteration 0:   log likelihood = -540.36159
Iteration 1:   log likelihood = -540.36159
Dynamic-factor model
Sample: 1972m2 - 2008m11
Log likelihood = -540.36159
(1) [cov(De.unemp,De.income)]_cons = 0
```

Number of obs = 442
Wald chi2(6) = 75.48
Prob > chi2 = 0.0000

	OIM				[95% Conf. Interval]	
	Coef.	Std. Err.	z	P> z		
De.ipman e.ipman LD.	.2297308	.0473147	4.86	0.000	.1369957	.3224659
De.income e.ipman LD.	.1075441	.0433357	2.48	0.013	.0226077	.1924805
e.income LD.	-.2209485	.0471116	-4.69	0.000	-.3132943	-.1286028
De.unemp e.ipman LD.	-.0975759	.0151301	-6.45	0.000	-.1272304	-.0679215
e.income LD.	-.0000467	.0147848	-0.00	0.997	-.0290244	.0289309
e.unemp LD.	-.0795348	.0482213	-1.65	0.099	-.1740469	.0149773
var(De.ipman)	.3335286	.0224282	14.87	0.000	.2895702	.377487
cov(De.inc~e, De.ipman)	.0457804	.0139123	3.29	0.001	.0185127	.0730481
cov(De.unemp, De.ipman)	-.0329438	.0051423	-6.41	0.000	-.0430226	-.022865
var(De.inc~e)	.2743375	.0184657	14.86	0.000	.2381454	.3105296
cov(De.unemp, De.inc~e)	(omitted)					
var(De.unemp)	.0292088	.00199	14.68	0.000	.0253083	.0331092

Note: Tests of variances against zero are conservative and are provided only for reference.

The estimated AR terms of D.income and D.unemp on D.unemp are -0.000047 and -0.079535 , and they are not significant at the 1% or 5% levels. The estimated AR term of D.ipman on D.income is 0.107544 and is significant at the 5% level but not at the 1% level.

□ Technical note

We obtained the unconditional estimator in [example 4](#) by specifying the `ar()` option instead of including the lags of the endogenous variables as exogenous variables, as we did in [example 3](#). The unconditional estimator has an additional observation and is more efficient. This change is analogous to estimating an AR coefficient by `arma` instead of using `regress` on the lagged endogenous variable. For example, to obtain the unconditional estimator in a univariate model, typing

```
. arima D.ipman, ar(1) noconstant technique(nr)
(output omitted)
```

will produce the same estimated AR coefficient as

```
. dfactor (D.ipman, ar(1) noconstant)
(output omitted)
```

We obtain the conditional estimator by typing either

```
. regress D.ipman LD.ipman, noconstant
(output omitted)
```

or

```
. dfactor (D.ipman LD.ipman, noconstant)
(output omitted)
```

□

▷ Example 5

In this example, we fit regional unemployment data to an SF model. We have data on the unemployment levels for the four regions in the U.S. census: `west` for the West, `south` for the South, `ne` for the Northeast, and `midwest` for the Midwest. We treat the variables as first-difference stationary and model the first-differences of these variables. Using `dfactor` yields

(Continued on next page)

```
. use http://www.stata-press.com/data/r11/urate
(Monthly unemployment rates in US Census regions)

. dfactor (D.(west south ne midwest) = , noconstant) (z = )
searching for initial values
(setting technique to bhhh)
Iteration 0:   log likelihood =   679.24575
Iteration 1:   log likelihood =   714.35563
(output omitted)
Refining estimates:
Iteration 0:   log likelihood =   873.0755
Iteration 1:   log likelihood =   873.0755

Dynamic-factor model
Sample: 1990m2 - 2008m12
                                Number of obs   =          227
                                Wald chi2(4)      =          342.56
Log likelihood =   873.0755      Prob > chi2    =          0.0000
```

		OIM		z	P> z	[95% Conf. Interval]	
		Coef.	Std. Err.				
D.west	z	.0978324	.0065644	14.90	0.000	.0849664	.1106983
D.south	z	.0859494	.0061762	13.92	0.000	.0738442	.0980546
D.ne	z	.0918607	.0072814	12.62	0.000	.0775893	.106132
D.midwest	z	.0861102	.0074652	11.53	0.000	.0714787	.1007417
var(De.west)		.0036887	.0005834	6.32	0.000	.0025453	.0048322
var(De.south)		.0038902	.0005228	7.44	0.000	.0028656	.0049149
var(De.ne)		.0064074	.0007558	8.48	0.000	.0049261	.0078887
var(De.mid~t)		.0074749	.0008271	9.04	0.000	.0058538	.009096

Note: Tests of variances against zero are conservative and are provided only for reference.

The estimates indicate that we could reasonably suppose that the unobserved factor has the same effect on the changes in unemployment in all four regions. The output below shows that we cannot reject the null hypothesis that these coefficients are the same.

```
. test [D.west]z = [D.south]z = [D.ne]z = [D.midwest]z
( 1)  [D.west]z - [D.south]z = 0
( 2)  [D.west]z - [D.ne]z = 0
( 3)  [D.west]z - [D.midwest]z = 0

      chi2( 3) =      3.58
      Prob > chi2 =     0.3109
```



➤ Example 6

In this example, we impose the constraint that the unobserved factor has the same impact on changes in unemployment in all four regions. This constraint was suggested by the results of the previous example. The previous example did not allow for any dynamics in the variables, a problem we alleviate by allowing the disturbances in the equation for each observable to follow an AR(1) process.

```

. constraint 2 [D.west]z = [D.south]z
. constraint 3 [D.west]z = [D.ne]z
. constraint 4 [D.west]z = [D.midwest]z
. dfactor (D.(west south ne midwest) = , noconstant ar(1)) (z = ),
> constraints(2/4)
searching for initial values
(setting technique to bhhh)
Iteration 0:   log likelihood =   683.32697
Iteration 1:   log likelihood =   708.41454
(output omitted)
Refining estimates:
Iteration 0:   log likelihood =   880.97488
Iteration 1:   log likelihood =   880.97488
Dynamic-factor model
Sample: 1990m2 - 2008m12                                Number of obs   =          227
                                                         Wald chi2(5)    =          363.34
Log likelihood =   880.97488                               Prob > chi2     =          0.0000
( 1)  [D.west]z - [D.south]z = 0
( 2)  [D.west]z - [D.ne]z = 0
( 3)  [D.west]z - [D.midwest]z = 0

```

	OIM					
	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
De.west						
e.west						
LD.	.1297198	.0992663	1.31	0.191	-.0648386	.3242781
De.south						
e.south						
LD.	-.2829014	.0909205	-3.11	0.002	-.4611023	-.1047004
De.ne						
e.ne						
LD.	.2866958	.0847851	3.38	0.001	.12052	.4528715
De.midwest						
e.midwest						
LD.	.0049427	.0782188	0.06	0.950	-.1483634	.1582488
D.west						
z	.0904724	.0049326	18.34	0.000	.0808047	.1001401
D.south						
z	.0904724	.0049326	18.34	0.000	.0808047	.1001401
D.ne						
z	.0904724	.0049326	18.34	0.000	.0808047	.1001401
D.midwest						
z	.0904724	.0049326	18.34	0.000	.0808047	.1001401
var(De.west)	.0038959	.0005111	7.62	0.000	.0028941	.0048977
var(De.south)	.0035518	.0005097	6.97	0.000	.0025528	.0045507
var(De.ne)	.0058173	.0006983	8.33	0.000	.0044488	.0071859
var(De.mid~t)	.0075444	.0008268	9.12	0.000	.0059239	.009165

Note: Tests of variances against zero are conservative and are provided only for reference.

The results indicate that the model might not fit well. Two of the four AR coefficients are statistically insignificant, while the two significant coefficients have opposite signs and sum to about zero. We suspect that a DF model might fit these data better than an SF model with autocorrelated disturbances.



Saved results

`dfactor` saves the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(k)</code>	number of parameters
<code>e(k_aux)</code>	number of auxiliary parameters
<code>e(k_eq)</code>	number of equations in <code>e(b)</code>
<code>e(k_autoCns)</code>	number of base, empty, and omitted constraints
<code>e(k_dv)</code>	number of dependent variables
<code>e(k_obser)</code>	number of observation equations
<code>e(k_state)</code>	number of state equations
<code>e(k_obser_err)</code>	number of observation-error terms
<code>e(k_state_err)</code>	number of state-error terms
<code>e(k_factor)</code>	number of factors specified
<code>e(o_ar_max)</code>	number of AR terms for the disturbances
<code>e(f_ar_max)</code>	number of AR terms for the factors
<code>e(df_m)</code>	model degrees of freedom
<code>e(ll)</code>	log likelihood
<code>e(chi2)</code>	χ^2 statistic
<code>e(p)</code>	significance
<code>e(tmin)</code>	minimum time in sample
<code>e(tmax)</code>	maximum time in sample
<code>e(stationary)</code>	1 if the estimated parameters indicate a stationary model, 0 otherwise
<code>e(rank)</code>	rank of VCE
<code>e(ic)</code>	number of iterations
<code>e(rc)</code>	return code
<code>e(converged)</code>	1 if converged, 0 otherwise

Macros

<code>e(cmd)</code>	<code>dfactor</code>
<code>e(cmdline)</code>	command as typed
<code>e(depvar)</code>	unoperated names of dependent variables in observation equations
<code>e(ober_deps)</code>	names of dependent variables in observation equations
<code>e(covariates)</code>	list of covariates
<code>e(indeps)</code>	independent variables
<code>e(factor_deps)</code>	names of unobserved factors in model
<code>e(tvar)</code>	time variable
<code>e(eqnames)</code>	names of equations
<code>e(model)</code>	type of dynamic-factor model specified
<code>e(title)</code>	title in estimation output
<code>e(tmins)</code>	formatted minimum time
<code>e(tmaxs)</code>	formatted maximum time
<code>e(o_ar)</code>	list of AR terms for disturbances
<code>e(f_ar)</code>	list of AR terms for factors
<code>e(R_structure)</code>	structure of observation-error covariance matrix
<code>e(Q_structure)</code>	structure of factor-error covariance matrix
<code>e(Q1_structure)</code>	covariance structure of factor errors
<code>e(Q2_structure)</code>	covariance structure of observable-variable errors
<code>e(chi2type)</code>	Wald; type of model χ^2 test
<code>e(vce)</code>	<i>vcetype</i> specified in <code>vce()</code>
<code>e(vcetype)</code>	title used to label Std. Err.
<code>e(opt)</code>	type of optimization
<code>e(method)</code>	likelihood method
<code>e(initial_values)</code>	type of initial values
<code>e(technique)</code>	maximization technique
<code>e(tech_steps)</code>	iterations taken in maximization technique(s)
<code>e(crittype)</code>	optimization criterion
<code>e(datasignature)</code>	the checksum
<code>e(datasignaturevars)</code>	variables used in calculation of checksum
<code>e(properties)</code>	<code>b V</code>
<code>e(estat_cmd)</code>	program used to implement <code>estat</code>
<code>e(predict)</code>	program used to implement <code>predict</code>
<code>e(marginsok)</code>	predictions allowed by <code>margins</code>
<code>e(marginsnotok)</code>	predictions disallowed by <code>margins</code>

(Continued on next page)

Matrices

<code>e(b)</code>	coefficient vector
<code>e(Cns)</code>	constraints matrix
<code>e(ilog)</code>	iteration log (up to 20 iterations)
<code>e(gradient)</code>	gradient vector
<code>e(gamma)</code>	mapping from parameter vector to state-space matrices
<code>e(A)</code>	estimated A matrix
<code>e(B)</code>	estimated B matrix
<code>e(C)</code>	estimated C matrix
<code>e(D)</code>	estimated D matrix
<code>e(F)</code>	estimated F matrix
<code>e(G)</code>	estimated G matrix
<code>e(chol_R)</code>	Cholesky factor of estimated R matrix
<code>e(chol_Q)</code>	Cholesky factor of estimated Q matrix
<code>e(chol_Q1)</code>	Cholesky factor of estimated Q1 matrix
<code>e(chol_Q2)</code>	Cholesky factor of estimated Q2 matrix
<code>e(chol_Sz0)</code>	Cholesky factor of initial state covariance matrix
<code>e(z0)</code>	initial state vector
<code>e(d)</code>	additional term in diffuse initial state vector, if nonstationary model
<code>e(T)</code>	inner part of quadratic form for initial state covariance in a partially nonstationary model
<code>e(M)</code>	outer part of quadratic form for initial state covariance in a partially nonstationary model
<code>e(Z)</code>	select nonstationary components in a partially nonstationary model
<code>e(V)</code>	variance-covariance matrix of the estimators
<code>e(V_modelbased)</code>	model-based variance

Functions

<code>e(sample)</code>	marks estimation sample

Methods and formulas

`dfactor` is implemented as an ado-file.

`dfactor` writes the specified model as a state-space model and uses `sspace` to estimate the parameters by maximum likelihood. See [Lütkepohl \(2005, 619–621\)](#) for how to write the DF model in state-space form. See [\[TS\] `sspace`](#) for the technical details.

References

- De Jong, P. 1988. The likelihood for a state space model. *Biometrika* 75: 165–169.
- . 1991. The diffuse Kalman filter. *Annals of Statistics* 19: 1073–1083.
- Geweke, J. 1977. The dynamic factor analysis of economic time series models. In *Latent Variables in Socioeconomic Models*, ed. D. J. Aigner and A. S. Goldberger, 365–383. Amsterdam: North-Holland.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Sargent, T. J., and C. A. Sims. 1977. Business cycle modeling without pretending to have too much a priori economic theory. In *New Methods in Business Cycle Research: Proceedings from a Conference*, ed. C. A. Sims, 45–109. Minneapolis: Federal Reserve Bank of Minneapolis.
- Stock, J. H., and M. W. Watson. 1989. New indexes of coincident and leading economic indicators. In *NBER Macroeconomics Annual 1989*, ed. O. J. Blanchard and S. Fischer, vol. 4, 351–394. Cambridge, MA: MIT Press.

- . 1991. A probability model of the coincident economic indicators. In *Leading Economic Indicators: New Approaches and Forecasting Records*, ed. K. Lahiri and G. H. Moore, 63–89. Cambridge: Cambridge University Press.
- Watson, M. W., and R. F. Engle. 1983. Alternative algorithms for the estimation of dynamic factor, MIMIC and varying coefficient regression models. *Journal of Econometrics* 23: 385–400.

Also see

- [TS] **dfactor postestimation** — Postestimation tools for dfactor
- [TS] **sspace** — State-space models
- [TS] **sspace postestimation** — Postestimation tools for sspace
- [TS] **var** — Vector autoregressive models
- [TS] **arima** — ARIMA, ARMAX, and other dynamic regression models
- [R] **regress** — Linear regression
- [R] **sureg** — Zellner’s seemingly unrelated regression
- [U] **20 Estimation and postestimation commands**

Description

The following standard postestimation commands are available for `dfactor`:

<i>command</i>	<i>description</i>
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	nonlinear predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

```
predict [type] { stub* | newvarlist } [if] [in] [, statistic options]
```

<i>statistic</i>	<i>description</i>
Main	
<code>y</code>	dependent variable, which is <code>xbf + residuals</code>
<code>xb</code>	linear predictions using the observable independent variables
<code>xbf</code>	linear predictions using the observable independent variables plus the factor contributions
<code>factors</code>	unobserved factor variables
<code>residuals</code>	autocorrelated disturbances
<code>innovations</code>	innovations, the observed dependent variable minus the predicted y

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

options	description
Options	
<code>equation(eqnames)</code>	specify name(s) of equation(s) for which predictions are to be made
<code>rmse(stub* newvarlist)</code>	put estimated root mean squared errors of predicted objects in new variables
<code>dynamic(time_constant)</code>	begin dynamic forecast at specified time
Advanced	
<code>smethod(method)</code>	method for predicting unobserved states
method	description
<code>onestep</code>	predict using past information
<code>smooth</code>	predict using all sample information
<code>filter</code>	predict using past and contemporaneous information

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

The mathematical notation used in this section is defined in [Description](#) of [TS] **dfactor**.

Main
<code>y</code>, <code>xb</code>, <code>xbf</code>, <code>factors</code>, <code>residuals</code>, and <code>innovations</code> specify the statistic to be predicted. <code>y</code>, the default, predicts the dependent variables. The predictions include the contributions of the unobserved factors, the linear predictions by using the observable independent variables, and any autocorrelation, $\hat{\mathbf{P}}\mathbf{f}_t + \hat{\mathbf{Q}}\mathbf{x}_t + \hat{\mathbf{u}}_t$. <code>xb</code> calculates the linear prediction by using the observable independent variables, $\hat{\mathbf{Q}}\mathbf{x}_t$. <code>xbf</code> calculates the contributions of the unobserved factors plus the linear prediction by using the observable independent variables, $\hat{\mathbf{P}}\mathbf{f}_t + \hat{\mathbf{Q}}\mathbf{x}_t$. <code>factors</code> estimates the unobserved factors, $\hat{\mathbf{f}}_t = \hat{\mathbf{R}}\mathbf{w}_t + \hat{\mathbf{A}}_1\hat{\mathbf{f}}_{t-1} + \hat{\mathbf{A}}_2\hat{\mathbf{f}}_{t-2} + \cdots + \hat{\mathbf{A}}_{t-p}\hat{\mathbf{f}}_{t-p}$. <code>residuals</code> calculates the autocorrelated residuals, $\hat{\mathbf{u}}_t = \hat{\mathbf{C}}_1\hat{\mathbf{u}}_{t-1} + \hat{\mathbf{C}}_2\hat{\mathbf{u}}_{t-2} + \cdots + \hat{\mathbf{C}}_{t-q}\hat{\mathbf{u}}_{t-q}$. <code>innovations</code> calculates the innovations, $\hat{\boldsymbol{\epsilon}}_t = \mathbf{y}_t - \hat{\mathbf{P}}\mathbf{f}_t + \hat{\mathbf{Q}}\mathbf{x}_t - \hat{\mathbf{u}}_t$.
Options

`equation(eqnames)` specifies the equation(s) for which the predictions are to be calculated.

You specify equation names, such as `equation(income consumption)` or `equation(factor1 factor2)`, to identify the equations. For the `factors` statistic, you must specify names of equations for factors; for all other statistics, you must specify names of equations for observable variables.

If you do not specify `equation()` and do not specify `stub*`, the results are the same as if you had specified the name of the first equation for the predicted statistic.

`equation()` may not be specified with `stub*`.

`rmse(stub*|newvarlist)` puts the root mean squared errors of the predicted objects into the specified new variables. The root mean squared errors measure the variances due to the disturbances but do not account for estimation error.

`dynamic(time_constant)` specifies when `predict` starts producing dynamic forecasts. The specified `time_constant` must be in the scale of the time variable specified in `tsset`, and the `time_constant` must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2008q4))` causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly, see [D] [dates and times](#). If the model contains exogenous variables, they must be present for the whole predicted sample. `dynamic()` may not be specified with `xb`, `xbf`, `innovations`, `smethod(filter)`, or `smethod(smooth)`.

Advanced

`smethod(method)` specifies the method used to predict the unobserved states in the model. `smethod()` may not be specified with `xb`.

`smethod(onestep)`, the default, causes `predict` to use previous information on the dependent variables. The full Kalman filter is performed on previous periods, but only the one-step predictions are made for the current period.

`smethod(smooth)` causes `predict` to estimate the states at each time period using all the sample data by the Kalman smoother.

`smethod(filter)` causes `predict` to estimate the states at each time period using previous and contemporaneous data by the Kalman filter. The full Kalman filter is performed on previous periods and the current period. `smethod(filter)` may be specified only with `factors` and `residuals`.

Remarks

We assume that you have already read [TS] [dfactor](#). In this entry, we illustrate some of the features of `predict` after using `dfactor`.

`dfactor` writes the specified model as a state-space model and estimates the parameters by maximum likelihood. The unobserved factors and the residuals are states in the state-space form of the model, and they are estimated by the Kalman filter or the Kalman smoother. The `smethod()` option controls how these states are estimated.

The Kalman filter or Kalman smoother is run over the specified sample. Changing the sample can alter the predicted value for a given observation, because the Kalman filter and Kalman smoother are recursive algorithms.

After estimating the parameters of a dynamic-factor model, there are many quantities of potential interest. Here we will discuss several of these statistics and illustrate how to use `predict` to compute them.

► Example 1

Let's begin by estimating the parameters of the dynamic-factor model considered in [example 2](#) in [TS] [dfactor](#).

```
. use http://www.stata-press.com/data/r11/dfex
(St. Louis Fed (FRED) macro data)

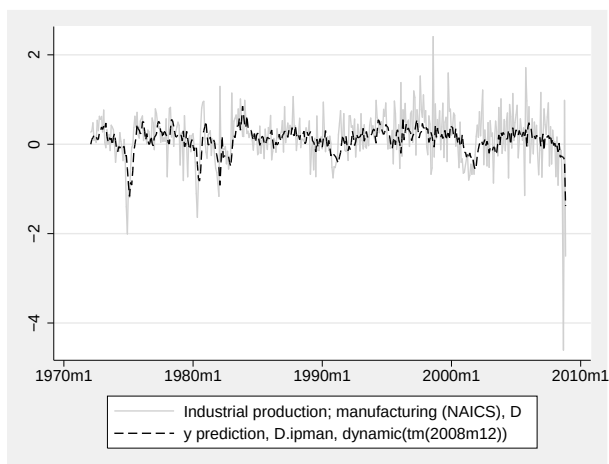
. dfactor (D.(ipman income hours unemp) = , noconstant ar(1)) (f = , ar(1/2))
(output omitted)
```

While several of the six statistics computed by `predict` might be of interest, we will look only at a few of these statistics for `D.ipman`. We begin by obtaining one-step predictions in the estimation sample and a six-month dynamic forecast for `D.ipman`. The graph of the in-sample predictions indicates that our model accounts only for a small fraction of the variability in `D.ipman`.

```
. tsappend, add(6)

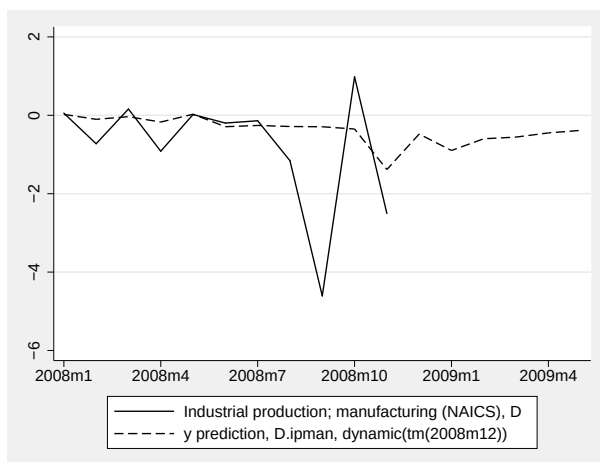
. predict Dipman_f, dynamic(tm(2008m12)) equation(D.ipman)
(option y assumed; fitted values)

. tsline D.ipman Dipman_f if month<=tm(2008m11), lcolor(gs13) xtitle("")
> legend(rows(2))
```



Graphing the last year of the sample and the six-month out-of-sample forecast yields

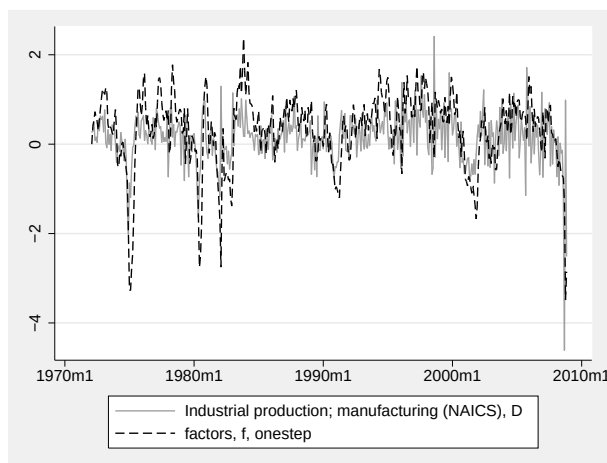
```
. tsline D.ipman Dipman_f if month>=tm(2008m1), xtitle("") legend(rows(2))
```



► Example 2

Another common task is to estimate an unobserved factor. We can estimate the unobserved factor at each time period by using only previous information (the `smethod(onestep)` option), previous and contemporaneous information (the `smethod(filter)` option), or all the sample information (the `smethod(smooth)` option). We are interested in the one-step predictive power of the unobserved factor, so we use the default, `smethod(onestep)`.

```
. predict fac if e(sample), factor
. tsline D.ipman fac, lcolor(gs10) xtitle("") legend(rows(2))
```



◀

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

`dfactor` estimates the parameters by writing the model in state-space form and using `sspace`. Analogously, `predict` after `dfactor` uses the methods described in [TS] [sspace postestimation](#). The unobserved factors and the residuals are states in the state-space form of the model.

See [Methods and formulas](#) of [TS] [sspace postestimation](#) for how predictions are made after estimating the parameters of a state-space model.

Also see

[TS] [dfactor](#) — Dynamic-factor models

[TS] [sspace](#) — State-space models

[TS] [sspace postestimation](#) — Postestimation tools for `sspace`

[U] [20 Estimation and postestimation commands](#)

Title

dfgls — DF-GLS unit-root test

Syntax

`dfgls` *varname* [*if*] [*in*] [, *options*]

<i>options</i>	description

Main

<code>maxlag(#)</code>	use # as the highest lag order for Dickey–Fuller GLS regressions
<code>notrend</code>	series is stationary around a mean instead of around a linear time trend
<code>ers</code>	present interpolated critical values from Elliott, Rothenberg, and Stock (1996)

You must `tsset` your data before using `dfgls`; see [TS] [tsset](#).

varname may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Tests > DF-GLS test for a unit root

Description

`dfgls` performs a modified Dickey–Fuller t test for a unit root in which the series has been transformed by a generalized least-squares regression.

Options

Main

`maxlag(#)` sets the value of k , the highest lag order for the first-differenced, detrended variable in the Dickey–Fuller regression. By default, `dfgls` sets k according to the method proposed by [Schwert \(1989\)](#); i.e., `dfgls` sets $k_{\max} = \text{int}[12\{(T + 1)/100\}^{0.25}]$.

`notrend` specifies that the alternative hypothesis be that the series is stationary around a mean instead of around a linear time trend. By default, a trend is included.

`ers` specifies that `dfgls` should present interpolated critical values from tables presented by [Elliott, Rothenberg, and Stock \(1996\)](#), which they obtained from simulations. See [Critical values](#) under *Methods and formulas* for details.

Remarks

`dfgls` tests for a unit root in a time series. It performs the modified Dickey–Fuller t test (known as the DF-GLS test) proposed by [Elliott, Rothenberg, and Stock \(1996\)](#). Essentially, the test is an augmented Dickey–Fuller test, similar to the test performed by Stata’s `dfuller` command, except that the time series is transformed via a generalized least squares (GLS) regression before performing the test. Elliott, Rothenberg, and Stock and later studies have shown that this test has significantly greater power than the previous versions of the augmented Dickey–Fuller test.

`dfgls` performs the DF-GLS test for the series of models that include 1 to k lags of the first-differenced, detrended variable, where k can be set by the user or by the method described in [Schwert \(1989\)](#). [Stock and Watson \(2007, 650–655\)](#) provide an excellent discussion of the approach.

As discussed in [\[TS\]](#) `dfuller`, the augmented Dickey–Fuller test involves fitting a regression of the form

$$\Delta y_t = \alpha + \beta y_{t-1} + \delta t + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \cdots + \zeta_k \Delta y_{t-k} + \epsilon_t$$

and then testing the null hypothesis $H_0: \beta = 0$. The DG-GLS test is performed analogously but on GLS-detrended data. The null hypothesis of the test is that y_t is a random walk, possibly with drift. There are two possible alternative hypotheses: y_t is stationary about a linear time trend or y_t is stationary with a possibly nonzero mean but with no linear time trend. The default is to use the former. To specify the latter alternative, use the `notrend` option.

► **Example 1**

Here we use the German macroeconomic dataset and test whether the natural log of income exhibits a unit root. We use the default options with `dfgls`.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)

. dfgls ln_inv
DF-GLS for ln_inv
Maxlag = 11 chosen by Schwert criterion
Number of obs =      80
```

[lags]	DF-GLS tau Test Statistic	1% Critical Value	5% Critical Value	10% Critical Value
11	-2.925	-3.610	-2.763	-2.489
10	-2.671	-3.610	-2.798	-2.523
9	-2.766	-3.610	-2.832	-2.555
8	-3.259	-3.610	-2.865	-2.587
7	-3.536	-3.610	-2.898	-2.617
6	-3.115	-3.610	-2.929	-2.646
5	-3.054	-3.610	-2.958	-2.674
4	-3.016	-3.610	-2.986	-2.699
3	-2.071	-3.610	-3.012	-2.723
2	-1.675	-3.610	-3.035	-2.744
1	-1.752	-3.610	-3.055	-2.762

```
Opt Lag (Ng-Perron seq t) = 7 with RMSE .0388771
Min SC = -6.169137 at lag 4 with RMSE .0398949
Min MAIC = -6.136371 at lag 1 with RMSE .0440319
```

The null of a unit root is rejected at the 5% level for lags 4–8 and 11 and at the 10% level for lags 9 and 10. For comparison, we also test for a unit root in log income by using `dfuller` with two different lag specifications. We need to use the `trend` option with `dfuller` because it is not included by default.

```
. dfuller ln_inv, lag(4) trend
Augmented Dickey-Fuller test for unit root
Number of obs =      87
```

	Test Statistic	1% Critical Value	5% Critical Value	10% Critical Value
Z(t)	-3.133	-4.069	-3.463	-3.158

MacKinnon approximate p-value for Z(t) = 0.0987

```
. dfuller ln_inv, lag(7) trend
Augmented Dickey-Fuller test for unit root      Number of obs   =      84

              Test              Interpolated Dickey-Fuller
              Statistic          1% Critical      5% Critical      10% Critical
                                Value             Value             Value
-----
Z(t)          -3.994            -4.075            -3.466            -3.160

MacKinnon approximate p-value for Z(t) = 0.0090
```

The results from `dfuller` also support rejecting the null hypothesis, though the p -value with four lags is close to 0.10. That the `dfuller` results are not as strong as those produced by `dfgls` is not surprising since the DF-GLS test with a trend has been shown to be more powerful than the standard augmented Dickey–Fuller test.



Saved results

If `maxlag(0)` is specified, `dfgls` saves the following in `r()`:

```
Scalars
      r(rmse0)  RMSE
      r(dft0)   DF-GLS statistic
```

Otherwise, `dfgls` saves the following in `r()`:

```
Scalars
      r(maxlag)  highest lag order  $k$ 
      r(N)       number of observations
      r(sclag)   lag chosen by Schwarz criterion
      r(maiclag) lag chosen by modified AIC method
      r(optlag)  lag chosen by sequential- $t$  method
Matrices
      r(results)  $k$ , MAIC, SIC, RMSE, and DF-GLS statistics
```

Methods and formulas

`dfgls` is implemented as an ado-file.

`dfgls` tests for a unit root. There are two possible alternative hypotheses: y_t is stationary around a linear trend or y_t is stationary with no linear time trend. Under the first alternative hypothesis, the DF-GLS test is performed by first estimating the intercept and trend via GLS. The GLS estimation is performed by generating the new variables, $\tilde{y}_t$, x_t , and z_t , where

$$\begin{aligned}
\tilde{y}_1 &= y_1 \\
\tilde{y}_t &= y_t - \alpha^* y_{t-1}, \quad t = 2, \dots, T \\
x_1 &= 1 \\
x_t &= 1 - \alpha^*, \quad t = 2, \dots, T \\
z_1 &= 1 \\
z_t &= t - \alpha^*(t - 1)
\end{aligned}$$

and $\alpha^* = 1 - (13.5/T)$. An OLS regression is then estimated for the equation

$$\tilde{y}_t = \delta_0 x_t + \delta_1 z_t + \epsilon_t$$

The OLS estimators $\hat{\delta}_0$ and $\hat{\delta}_1$ are then used to remove the trend from y_t ; i.e., we generate

$$y^* = y_t - (\hat{\delta}_0 + \hat{\delta}_1 t)$$

Finally, we perform an augmented Dickey–Fuller test on the transformed variable by fitting the OLS regression

$$\Delta y_t^* = \alpha + \beta y_{t-1}^* + \sum_{j=1}^k \zeta_j \Delta y_{t-j}^* + \epsilon_t$$

and then test the null hypothesis $H_0: \beta = 0$ by using tabulated critical values.

To perform the DF-GLS test under the second alternative hypothesis, we proceed as before but define $\alpha^* = 1 - (7/T)$, eliminate z from the GLS regression, compute $y^* = y_t - \delta_0$, fit the augmented Dickey–Fuller regression by using the newly transformed variable, and perform a test of the null hypothesis that $\beta = 0$ by using the tabulated critical values.

`dfgls` reports the DF-GLS statistic and its critical values obtained from the regression in (1) for $k \in \{1, 2, \dots, k_{\max}\}$. By default, `dfgls` sets $k_{\max} = \text{int}[12\{(T+1)/100\}^{0.25}]$ as proposed by [Schwert \(1989\)](#), although you can override this choice with another value. The sample size available with $k_{\max}$ lags is used in all the regressions. Because there are $k_{\max}$ lags of the first-differenced series, $k_{\max} + 1$ observations are lost, leaving $T - k_{\max}$ observations. `dfgls` requires that the sample of $T + 1$ observations on $y_t = (y_0, y_1, \dots, y_T)$ have no gaps.

`dfgls` reports the results of three different methods for choosing which value of k to use. These are method 1 the Ng–Perron sequential t , method 2 the minimum Schwarz information criterion (SIC), and method 3 the Ng–Perron modified Akaike information criterion (MAIC). Although the SIC has a long history in time-series modeling, the Ng–Perron sequential t was developed by [Ng and Perron \(1995\)](#), and the MAIC was developed by [Ng and Perron \(2000\)](#).

The SIC can be calculated using either the log likelihood or the sum-of-squared errors from a regression; `dfgls` uses the latter definition. Specifically, for each k

$$\text{SIC} = \ln(\widehat{\text{rmse}}^2) + (k + 1) \frac{\ln(T - k_{\max})}{(T - k_{\max})}$$

where

$$\widehat{\text{rmse}} = \frac{1}{(T - k_{\max})} \sum_{t=k_{\max}+1}^T \widehat{e}_t^2$$

`dfgls` reports the value of the smallest SIC and the k that produced it.

Ng and Perron (1995) derived a sequential- t algorithm for choosing k :

- i. Set $n = 0$ and run the regression in method 2 with all $k_{\max} - n$ lags. If the coefficient on $\beta_{k_{\max}}$ is significantly different from zero at level α , choose k to $k_{\max}$. Otherwise, continue to ii.
- ii. If $n < k_{\max}$, set $n = n + 1$ and continue to iii. Otherwise, set $k = 0$ and stop.
- iii. Run the regression in method 2 with $k_{\max} - n$ lags. If the coefficient on $\beta_{k_{\max}-n}$ is significantly different from zero at level α , choose k to $k_{\max} - n$. Otherwise, return to ii.

Per Ng and Perron (1995), `dfgls` uses $\alpha = 10\%$. `dfgls` reports the k selected by this sequential- t algorithm and the `rmse` from the regression.

Method (3) is based on choosing k to minimize the MAIC. The MAIC is calculated as

$$\text{MAIC}(k) = \ln(\widehat{\text{rmse}}^2) + \frac{2\{\tau(k) + k\}}{T - k_{\max}}$$

where

$$\tau(k) = \frac{1}{\widehat{\text{rmse}}^2} \widehat{\beta}_0^2 \sum_{t=k_{\max}+1}^T \widetilde{y}_t^2$$

and $\widetilde{y}$ was defined previously.

Critical values

By default, `dfgls` uses the 5% and 10% critical values computed from the response surface analysis of Cheung and Lai (1995). Because Cheung and Lai (1995) did not present results for the 1% case, the 1% critical values are always interpolated from the critical values presented by ERS.

ERS presented critical values, obtained from simulations, for the DF-GLS test with a linear trend and showed that the critical values for the mean-only DF-GLS test were the same as those for the ADF test. If `dfgls` is run with the `ers` option, `dfgls` will present interpolated critical values from these tables. The method of interpolation is standard. For the trend case, below 50 observations and above 200 there is no interpolation; the values for 50 and ∞ are reported from the tables. For a value N that lies between two values in the table, say, N_1 and N_2 , with corresponding critical values CV_1 and CV_2 , the critical value

$$\text{cv} = \text{CV}_1 + \frac{N - N_1}{N_2 - N_1} (\text{CV}_2 - \text{CV}_1)$$

is presented. The same method is used for the mean-only case, except that interpolation is possible for values between 50 and 500.

Acknowledgments

We thank Christopher F. Baum of Boston College and Richard Sperling for a previous version of `dfgls`.

References

- Cheung, Y.-W., and K. S. Lai. 1995. Lag order and critical values of a modified Dickey–Fuller test. *Oxford Bulletin of Economics and Statistics* 57: 411–419.
- Elliott, G., T. J. Rothenberg, and J. H. Stock. 1996. Efficient tests for an autoregressive unit root. *Econometrica* 64: 813–836.
- Ng, S., and P. Perron. 1995. Unit root tests in ARMA models with data-dependent methods for the selection of the truncation lag. *Journal of the American Statistical Association* 90: 268–281.
- . 2000. Lag length selection and the construction of unit root tests with good size and power. *Econometrica* 69: 1519–1554.
- Schwert, G. W. 1989. Tests for unit roots: A Monte Carlo investigation. *Journal of Business and Economic Statistics* 2: 147–159.
- Stock, J. H., and M. W. Watson. 2007. *Introduction to Econometrics*. 2nd ed. Boston: Addison–Wesley.

Also see

- [TS] **dfuller** — Augmented Dickey–Fuller unit-root test
- [TS] **pperron** — Phillips–Perron unit-root test
- [TS] **tsset** — Declare data to be time-series data
- [XT] **xtunitroot** — Panel-data unit-root tests

Title

dfuller — Augmented Dickey–Fuller unit-root test

Syntax

```
dfuller varname [if] [in] [, options]
```

options	description
Main	
<code>noconstant</code>	suppress constant term in regression
<code>trend</code>	include trend term in regression
<code>drift</code>	include drift term in regression
<code>regress</code>	display regression table
<code>lags(#)</code>	include # lagged differences

You must `tsset` your data before using `dfuller`; see [TS] [tsset](#).
`varname` may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Tests > Augmented Dickey-Fuller unit-root test

Description

`dfuller` performs the augmented Dickey–Fuller test that a variable follows a unit-root process. The null hypothesis is that the variable contains a unit root, and the alternative is that the variable was generated by a stationary process. You may optionally exclude the constant, include a trend term, and include lagged values of the difference of the variable in the regression.

Options

Main
<code>noconstant</code> suppresses the constant term (intercept) in the model and indicates that the process under the null hypothesis is a random walk without drift. <code>noconstant</code> cannot be used with the <code>trend</code> or <code>drift</code> option.
<code>trend</code> specifies that a trend term be included in the associated regression and that the process under the null hypothesis is a random walk, perhaps with drift. This option may not be used with the <code>noconstant</code> or <code>drift</code> option.
<code>drift</code> indicates that the process under the null hypothesis is a random walk with nonzero drift. This option may not be used with the <code>noconstant</code> or <code>trend</code> option.
<code>regress</code> specifies that the associated regression table appear in the output. By default, the regression table is not produced.
<code>lags(#)</code> specifies the number of lagged difference terms to include in the covariate list.

Remarks

Dickey and Fuller (1979) developed a procedure for testing whether a variable has a unit root or, equivalently, that the variable follows a random walk. Hamilton (1994, 528–529) describes the four different cases to which the augmented Dickey–Fuller test can be applied. The null hypothesis is always that the variable has a unit root. They differ in whether the null hypothesis includes a drift term and whether the regression used to obtain the test statistic includes a constant term and time trend.

The true model is assumed to be

$$y_t = \alpha + y_{t-1} + u_t$$

where u_t is an independently and identically distributed zero-mean error term. In cases one and two, presumably $\alpha = 0$, which is a random walk without drift. In cases three and four, we allow for a drift term by letting α be unrestricted.

The Dickey–Fuller test involves fitting the model

$$y_t = \alpha + \rho y_{t-1} + \delta t + u_t$$

by ordinary least squares (OLS), perhaps setting $\alpha = 0$ or $\delta = 0$. However, such a regression is likely to be plagued by serial correlation. To control for that, the augmented Dickey–Fuller test instead fits a model of the form

$$\Delta y_t = \alpha + \beta y_{t-1} + \delta t + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \cdots + \zeta_k \Delta y_{t-k} + \epsilon_t \tag{1}$$

where k is the number of lags specified in the `lags()` option. The `noconstant` option removes the constant term α from this regression, and the `trend` option includes the time trend δt , which by default is not included. Testing $\beta = 0$ is equivalent to testing $\rho = 1$, or, equivalently, that y_t follows a unit root process.

In the first case, the null hypothesis is that y_t follows a random walk without drift, and (1) is fit without the constant term α and the time trend δt . The second case has the same null hypothesis as the first, except that we include α in the regression. In both cases, the population value of α is zero under the null hypothesis. In the third case, we hypothesize that y_t follows a unit root with drift, so that the population value of α is nonzero; we do not include the time trend in the regression. Finally, in the fourth case, the null hypothesis is that y_t follows a unit root with or without drift so that α is unrestricted, and we include a time trend in the regression.

The following table summarizes the four cases.

Case	Process under null hypothesis	Regression restrictions	dfuller option
1	Random walk without drift	$\alpha = 0, \delta = 0$	<code>noconstant</code>
2	Random walk without drift	$\delta = 0$	(default)
3	Random walk with drift	$\delta = 0$	<code>drift</code>
4	Random walk with or without drift	(none)	<code>trend</code>

Except in the third case, the t -statistic used to test $H_0: \beta = 0$ does not have a standard distribution. Hamilton (1994, chap. 17) derives the limiting distributions, which are different for each of the three other cases. The critical values reported by `dfuller` are interpolated based on the tables in Fuller (1996). MacKinnon (1994) shows how to approximate the p -values on the basis of a regression surface, and `dfuller` also reports that p -value. In the third case, where the regression includes a constant term and under the null hypothesis the series has a nonzero drift parameter α , the t statistic has the usual t distribution; `dfuller` reports the one-sided critical values and p -value for the test of H_0 against the alternative $H_a: \beta < 0$, which is equivalent to $\rho < 1$.

Deciding which case to use involves a combination of theory and visual inspection of the data. If economic theory favors a particular null hypothesis, the appropriate case can be chosen based on that. If a graph of the data shows an upward trend over time, then case four may be preferred. If the data do not show a trend but do have a nonzero mean, then case two would be a valid alternative.

► Example 1

In this example, we examine the international airline passengers dataset from [Box, Jenkins, and Reinsel \(1994, Series G\)](#). This dataset has 144 observations on the monthly number of international airline passengers from 1949 through 1960. Because the data show a clear upward trend, we use the `trend` option with `dfuller` to include a constant and time trend in the augmented Dickey–Fuller regression.

```
. use http://www.stata-press.com/data/r11/air2
(TIMESLAB: Airline passengers)
. dfuller air, lags(3) trend regress
```

Augmented Dickey-Fuller test for unit root		Number of obs = 140		
Test Statistic	1% Critical Value	5% Critical Value	10% Critical Value	
Z(t)	-6.936	-4.027	-3.445	-3.145

MacKinnon approximate p-value for Z(t) = 0.0000

D.air	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
air					
L1.	-.5217089	.0752195	-6.94	0.000	-.67048 -.3729379
LD.	.5572871	.0799894	6.97	0.000	.399082 .7154923
L2D.	.095912	.0876692	1.09	0.276	-.0774825 .2693065
L3D.	.14511	.0879922	1.65	0.101	-.0289232 .3191433
_trend	1.407534	.2098378	6.71	0.000	.9925118 1.822557
_cons	44.49164	7.78335	5.72	0.000	29.09753 59.88575

Here we can overwhelmingly reject the null hypothesis of a unit root at all common significance levels. From the regression output, the estimated β of -0.522 implies that $\rho = (1 - 0.522) = 0.478$. Experiments with fewer or more lags in the augmented regression yield the same conclusion.

◀

► Example 2

In this example, we use the German macroeconomic dataset to determine whether the log of consumption follows a unit root. We will again use the `trend` option, because consumption grows over time.

(Continued on next page)

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. tsset qtr
      time variable:  qtr, 1960q1 to 1982q4
              delta:  1 quarter
```

```
. dfuller ln_consump, lags(4) trend
```

Augmented Dickey-Fuller test for unit root		Number of obs =		87
		Interpolated Dickey-Fuller		
	Test	1% Critical	5% Critical	10% Critical
	Statistic	Value	Value	Value
Z(t)	-1.318	-4.069	-3.463	-3.158

```
MacKinnon approximate p-value for Z(t) = 0.8834
```

As we might expect from economic theory, here we cannot reject the null hypothesis that log consumption exhibits a unit root. Again using different numbers of lag terms yield the same conclusion. ◀

Saved results

dfuller saves the following in `r()`:

Scalars			
<code>r(N)</code>	number of observations	<code>r(Zt)</code>	Dickey–Fuller test statistic
<code>r(lags)</code>	number of lagged differences	<code>r(p)</code>	MacKinnon approximate <i>p</i> -value (if there is a constant or trend in associated regression)

Methods and formulas

dfuller is implemented as an ado-file.
In the OLS estimation of an AR(1) process with Gaussian errors,

$$y_t = \rho y_{t-1} + \epsilon_t$$

where ϵ_t are independently and identically distributed as $N(0, \sigma^2)$ and $y_0 = 0$, the OLS estimate (based on an n -observation time series) of the autocorrelation parameter ρ is given by

$$\hat{\rho}_n = \frac{\sum_{t=1}^n y_{t-1} y_t}{\sum_{t=1}^n y_t^2}$$

If $|\rho| < 1$, then

$$\sqrt{n}(\hat{\rho}_n - \rho) \rightarrow N(0, 1 - \rho^2)$$

If this result were valid when $\rho = 1$, the resulting distribution would have a variance of zero. When $\rho = 1$, the OLS estimate $\hat{\rho}$ still converges in probability to one, though we need to find a suitable nondegenerate distribution so that we can perform hypothesis tests of $H_0: \rho = 1$. [Hamilton \(1994, chap. 17\)](#) provides a superb exposition of the requisite theory.

To compute the test statistics, we fit the augmented Dickey–Fuller regression

$$\Delta y_t = \alpha + \beta y_{t-1} + \delta t + \sum_{j=1}^k \zeta_j \Delta y_{t-j} + e_t$$

via OLS where, depending on the options specified, the constant term α or time trend δt is omitted and k is the number of lags specified in the `lags()` option. The test statistic for $H_0: \beta = 0$ is $Z_t = \hat{\beta} / \hat{\sigma}_\beta$, where $\hat{\sigma}_\beta$ is the standard error of $\hat{\beta}$.

The critical values included in the output are linearly interpolated from the table of values that appears in [Fuller \(1996\)](#), and the MacKinnon approximate p -values use the regression surface published in [MacKinnon \(1994\)](#).

David Alan Dickey (1945–) was born in Ohio and obtained degrees in mathematics at Miami University and a PhD in statistics at Iowa State University in 1976 as a student of Wayne Fuller. He works at North Carolina State University and specializes in time-series analysis.

Wayne Arthur Fuller (1931–) was born in Iowa, obtained three degrees at Iowa State University and then served on the faculty between 1959 and 2001. He has made many distinguished contributions to time series, measurement-error models, survey sampling, and econometrics.

References

- Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control*. 3rd ed. Englewood Cliffs, NJ: Prentice–Hall.
- Dickey, D. A., and W. A. Fuller. 1979. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association* 74: 427–431.
- Fuller, W. A. 1996. *Introduction to Statistical Time Series*. 2nd ed. New York: Wiley.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- MacKinnon, J. G. 1994. Approximate asymptotic distribution functions for unit-root and cointegration tests. *Journal of Business and Economic Statistics* 12: 167–176.
- Newton, H. J. 1988. *TIMESLAB: A Time Series Analysis Laboratory*. Belmont, CA: Wadsworth.

Also see

- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [dfgls](#) — DF-GLS unit-root test
- [TS] [pperron](#) — Phillips–Perron unit-root test
- [XT] [xtunitroot](#) — Panel-data unit-root tests

Syntax

```
dvech eq [eq ... eq] [if] [in] [, options]
```

where each *eq* has the form

```
(depvars = [indepvars], [noconstant])
```

options	description
Model	
<u>arch</u> (<i>numlist</i>)	ARCH terms
<u>garch</u> (<i>numlist</i>)	GARCH terms
<u>constraints</u> (<i>numlist</i>)	apply linear constraints
SE/Robust	
<u>vce</u> (<i>vcetype</i>)	<i>vcetype</i> may be <code>oim</code> or <code>robust</code>
Reporting	
<u>level</u> (#)	set confidence level; default is <code>level(95)</code>
<u>nocnsreport</u>	do not display constraints
<u>display_options</u>	control spacing and display of omitted variables and base and empty cells
Maximization	
<u>maximize_options</u>	control the maximization process; seldom used
<u>from</u> (<i>matname</i>)	initial values for the coefficients; seldom used
<u>svtechnique</u> (<i>algorithm_spec</i>)	starting-value maximization algorithm
<u>sviterate</u> (#)	number of starting-value iterations; default is <code>sviterate(25)</code>
[†] <u>coeflegend</u>	display coefficients' legend instead of coefficient table

[†]`coeflegend` does not appear in the dialog box.
You must `tsset` your data before using `dvech`; see [TS] `tsset`.
indepvars may contain factor variables; see [U] 11.4.3 Factor variables.
depvars and *indepvars* may contain time-series operators; see [U] 11.4.4 Time-series varlists.
`by`, `statsby`, and `rolling` are allowed; see [U] 11.1.10 Prefix commands.
See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

Menu

Statistics > Multivariate time series > Multivariate GARCH

Description

`dvech` estimates the parameters of a class of multivariate generalized autoregressive conditional-heteroskedasticity (GARCH) models. Multivariate GARCH models allow the conditional covariance matrix of the dependent variables to follow a flexible dynamic structure. `dvech` estimates the parameters of diagonal vech GARCH models in which each element of the current conditional covariance matrix of the dependent variables depends only on its own past and on past shocks.

Options

Model

`noconstant` suppresses the constant term(s).

`arch(numlist)` specifies the ARCH terms in the model. By default, no ARCH terms are specified.

`garch(numlist)` specifies the GARCH terms in the model. By default, no GARCH terms are specified.

`constraints(numlist)` specifies linear constraints to apply to the parameter estimates.

SE/Robust

`vce(vcetype)` specifies the estimator for the variance–covariance matrix of the estimator. `vce(oim)`, the default, specifies to use the observed information matrix (OIM) estimator. `vce(robust)` specifies to use the Huber/White/sandwich estimator.

Reporting

`level(#)`; see [\[R\] estimation options](#).

`nocnsreport`; see [\[R\] estimation options](#).

`display_options`: `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`; see [\[R\] estimation options](#).

Maximization

`maximize_options`: `difficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, and `from(matname)`; see [\[R\] maximize](#) for all options except `from()`, and see below for information on `from()`. These options are seldom used.

`from(matname)` specifies initial values for the coefficients. `from(b0)` causes `dvech` to begin the optimization algorithm with the values in `b0`. `b0` must be a row vector, and the number of columns must equal the number of parameters in the model.

`svtechnique(algorithm_spec)` and `sviterate(#)` specify options for the starting-value search process.

`svtechnique(algorithm_spec)` specifies the algorithm used to search for initial values. The syntax for `algorithm_spec` is the same as for the `technique()` option; see [\[R\] maximize](#). `svtechnique(bhhh 5 nr 16000)` is the default, and this option may not be specified with `from()`.

`sviterate(#)` specifies the maximum number of iterations that the search algorithm may perform. The default is `sviterate(25)`, and this option may not be specified with `from()`.

The following option is available with `dvech` but is not shown in the dialog box:

`coeflegend`; see [\[R\] estimation options](#).

Remarks

Remarks are presented under the following headings:

An introduction to diagonal-vech GARCH models
Some examples

An introduction to diagonal-vech GARCH models

Multivariate GARCH models allow the conditional covariance matrix of the dependent variables to follow a flexible dynamic structure. [Bollerslev, Engle, and Wooldridge \(1988\)](#); [Bollerslev, Engle, and Nelson \(1994\)](#); [Bauwens, Laurent, and Rombouts \(2006\)](#); and [Silvennoinen and Teräsvirta \(2009\)](#) discuss the general multivariate GARCH model and many special cases. There is a plethora of multivariate GARCH models because the unrestricted multivariate GARCH model is too general to fit to data and there are trade-offs in creating restricted models.

The general multivariate GARCH model developed by [Bollerslev, Engle, and Wooldridge \(1988\)](#) is

$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \boldsymbol{\epsilon}_t \quad (1)$$

$$\boldsymbol{\epsilon}_t = \mathbf{H}_t^{1/2} \boldsymbol{\nu}_t \quad (2)$$

$$\mathbf{h}_t = \mathbf{s} + \sum_{i=1}^p \mathbf{A}_i \text{vech}(\boldsymbol{\epsilon}_{t-i} \boldsymbol{\epsilon}_{t-i}') + \sum_{j=1}^q \mathbf{B}_j \mathbf{h}_{t-j} \quad (3)$$

where

$\mathbf{y}_t$ is an $m \times 1$ vector of dependent variables;

$\mathbf{C}$ is an $m \times k$ matrix of parameters;

$\mathbf{x}_t$ is a $k \times 1$ vector of independent variables, which may contain lags of $\mathbf{y}_t$;

$\mathbf{H}_t^{1/2}$ is the Cholesky factor of the time-varying conditional covariance matrix $\mathbf{H}_t$;

$\boldsymbol{\nu}_t$ is an $m \times 1$ vector of normal, independent, and identically distributed innovations;

$\mathbf{h}_t = \text{vech}(\mathbf{H}_t)$;

the $\text{vech}()$ function stacks the lower diagonal elements of a symmetric matrix into a column vector,

$$\text{vech} \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} = (1, 2, 3)'$$

$\mathbf{s}$ is an $m(m+1)/2 \times 1$ vector of parameters;

each $\mathbf{A}_i$ is an $\{m(m+1)/2\} \times \{m(m+1)/2\}$ matrix of parameters; and

each $\mathbf{B}_j$ is an $\{m(m+1)/2\} \times \{m(m+1)/2\}$ matrix of parameters.

This model is known as the general multivariate GARCH model and as the general-vec GARCH model. (Because the model uses the $\text{vech}()$ instead of the $\text{vec}()$ function, we will refer to it as the general-vech GARCH model.)

Bollerslev, Engle, and Wooldridge (1988) argued that the general-vech GARCH model in (1)–(3) was too flexible to fit to data, so they proposed restricting the matrices $\mathbf{A}_i$ and $\mathbf{B}_j$ to be diagonal matrices. It is for this restriction that the model is known as a diagonal vech GARCH model. The diagonal vech GARCH model can also be expressed by replacing (3) with

$$\mathbf{H}_t = \mathbf{S} + \sum_{i=1}^p \mathbf{A}_i \odot \epsilon_{t-i} \epsilon'_{t-i} + \sum_{j=1}^q \mathbf{B}_j \odot \mathbf{H}_{t-j} \quad (3')$$

where $\mathbf{S}$ is an $m \times m$ symmetric parameter matrix; each $\mathbf{A}_i$ is an $m \times m$ symmetric parameter matrix; $\odot$ is the elementwise or Hadamard product; and each $\mathbf{B}_j$ is an $m \times m$ symmetric parameter matrix.

The representation in (3') clarifies that each element in $\mathbf{H}_t$ depends on its past values and the past values of the corresponding autoregressive conditional-heteroskedasticity (ARCH) terms. While this representation does not allow cross-covariance effects, it is still quite flexible. The rapid rate at which the number of parameters grows with m , p , or q is one aspect of the model's flexibility.

`dvech` estimates the parameters of the diagonal vech GARCH model in (1), (2), and (3') by maximum likelihood (ML) assuming that the $\nu \sim N(\mathbf{0}, \mathbf{I}_m)$. The quasi-maximum likelihood (QML) estimator, which drops the normality assumption, is also consistent and normally distributed in large samples; see Berkes and Horváth (2003) and Comte and Lieberman (2003). Specify `vce(robust)` to estimate the parameters by QML. The QML parameter estimates are the same as the ML estimates, but the VCEs are different.

Some examples

► Example 1

We have data on a secondary market rate of a six-month U.S. Treasury bill, `tbill`, and on Moody's seasoned AAA corporate bond yield, `bond`. We model the first-differences of `tbill` and the first-differences of `bond` as a VAR(1) with an ARCH(1) term.

(Continued on next page)

```
. use http://www.stata-press.com/data/r11/irates4
(St. Louis Fed (FRED) financial data)

. dvech (D.bond D.tbill = LD.bond LD.tbill), arch(1)

Getting starting values
(setting technique to bhhh)
Iteration 0:   log likelihood =   3569.2723
Iteration 1:   log likelihood =   3708.4561
(output omitted)

Estimating parameters
(setting technique to bhhh)
Iteration 0:   log likelihood =   4184.5975
Iteration 1:   log likelihood =   4200.6303
(output omitted)

Diagonal vech multivariate GARCH model

Sample: 3 - 2456                                Number of obs   =       2454
                                                Wald chi2(4)    =       1183.52
Log likelihood =  4221.6577                    Prob > chi2     =        0.0000
```

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
D.bond						
bond						
LD.	.2967674	.0247149	12.01	0.000	.2483271	.3452077
tbill						
LD.	.0947949	.0098683	9.61	0.000	.0754533	.1141364
_cons	.0003991	.00143	0.28	0.780	-.0024036	.0032019
D.tbill						
bond						
LD.	.0108373	.0301501	0.36	0.719	-.0482558	.0699304
tbill						
LD.	.4344747	.0176497	24.62	0.000	.3998819	.4690675
_cons	.0011611	.0021033	0.55	0.581	-.0029612	.0052835
Sigma0						
1_1	.004894	.0002006	24.40	0.000	.0045008	.0052871
2_1	.0040986	.0002396	17.10	0.000	.0036289	.0045683
2_2	.0115149	.0005227	22.03	0.000	.0104904	.0125395
L.ARCH						
1_1	.4514942	.0456835	9.88	0.000	.3619562	.5410323
2_1	.2518879	.036736	6.86	0.000	.1798866	.3238893
2_2	.843368	.0608055	13.87	0.000	.7241914	.9625446

The output has three parts: an iteration log, a header, and an output table. The iteration log has two parts: the first part reports the iterations from the process of searching for starting values, and the second part reports the iterations from maximizing the log-likelihood function.

The header describes the estimation sample and reports a Wald test against the null hypothesis that all the coefficients on the independent variables in each equation are zero. Here the null hypothesis is rejected at all conventional levels.

The output table reports point estimates, standard errors, tests against zero, and confidence intervals for the estimated coefficients, the estimated elements of **S**, and any estimated elements of **A** or **B**. Here the output indicates that in the equation for **D.tbill**, neither the coefficient on **LD.bond** nor

the constant are statistically significant. The elements of $\mathbf{S}$ are reported in the Sigma0 equation. The estimate of $\mathbf{S}[1, 1]$ is 0.005, and the estimate of $\mathbf{S}[2, 1]$ is 0.004. The ARCH term results are reported in the L.ARCH equation. In the L.ARCH equation, 1_1 is the coefficient on the ARCH term for the conditional variance of the first dependent variable, 2_1 is the coefficient on the ARCH term for the conditional covariance between the first and second dependent variables, and 2_2 is the coefficient on the ARCH term for the conditional variance of the second dependent variable.

◀

► Example 2

We improve the previous example by removing the insignificant parameters from the model:

```
. dvech (D.bond = LD.bond LD.tbill, noconstant)
>      (D.tbill = LD.tbill, noconstant), arch(1)
```

Getting starting values

(setting technique to bhhh)

Iteration 0: log likelihood = 3566.8824

Iteration 1: log likelihood = 3701.6181

(output omitted)

Estimating parameters

(setting technique to bhhh)

Iteration 0: log likelihood = 4183.1113

Iteration 1: log likelihood = 4202.0304

(output omitted)

Diagonal vech multivariate GARCH model

Sample: 3 - 2456

Number of obs = 2454

Wald chi2(3) = 1197.76

Prob > chi2 = 0.0000

Log likelihood = 4221.433

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
D.bond						
bond						
LD.	.2941649	.0234734	12.53	0.000	.2481579	.3401718
tbill						
LD.	.0953158	.0098077	9.72	0.000	.076093	.1145386
D.tbill						
tbill						
LD.	.4385945	.0136672	32.09	0.000	.4118072	.4653817
Sigma0						
1_1	.0048922	.0002005	24.40	0.000	.0044993	.0052851
2_1	.0040949	.0002394	17.10	0.000	.0036256	.0045641
2_2	.0115043	.0005184	22.19	0.000	.0104883	.0125203
L.ARCH						
1_1	.4519233	.045671	9.90	0.000	.3624099	.5414368
2_1	.2515474	.0366701	6.86	0.000	.1796752	.3234195
2_2	.8437212	.0600839	14.04	0.000	.7259589	.9614836

We specified each equation separately to remove the insignificant parameters. None of the parameters is statistically insignificant.

◀

➤ Example 3

Here we analyze some fictional weekly data on the percentages of bad widgets found in the factories of Acme Inc. and Anvil Inc. We model the levels as a first-order autoregressive process. We believe that the adaptive management style in these companies causes the variances to follow a diagonal vech GARCH process with one ARCH term and one GARCH term. Furthermore, these close competitors follow essentially the same process, so we impose the constraints that the ARCH coefficients are the same for the two companies and that the GARCH coefficients are also the same.

Imposing these constraints yields

```
. use http://www.stata-press.com/data/r11/acme
. constraint 1 [L.ARCH]1_1 = [L.ARCH]2_2
. constraint 2 [L.GARCH]1_1 = [L.GARCH]2_2
. dvech (acme = L.acme) (anvil = L.anvil), arch(1) garch(1) constraints(1 2)
Getting starting values
(setting technique to bhhh)
Iteration 0: log likelihood = -6087.0665 (not concave)
(output omitted)
Estimating parameters
(setting technique to bhhh)
Iteration 0: log likelihood = -5974.4035
Iteration 1: log likelihood = -5973.812
(output omitted)
Diagonal vech multivariate GARCH model
Sample: 1969w35 - 1998w25
Log likelihood = -5973.7999
( 1) [L.ARCH]1_1 - [L.ARCH]2_2 = 0
( 2) [L.GARCH]1_1 - [L.GARCH]2_2 = 0
```

		Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
acme							
	acme						
	L1.	.3365278	.0255134	13.19	0.000	.2865225	.3865331
	_cons	1.124611	.060085	18.72	0.000	1.006847	1.242376
anvil							
	anvil						
	L1.	.3151955	.0263287	11.97	0.000	.2635922	.3667988
	_cons	1.215786	.0642052	18.94	0.000	1.089947	1.341626
Sigma0							
	1_1	1.889237	.2168733	8.71	0.000	1.464173	2.314301
	2_1	.4599576	.1139843	4.04	0.000	.2365525	.6833626
	2_2	2.063113	.2454633	8.40	0.000	1.582014	2.544213
L.ARCH							
	1_1	.2813443	.0299124	9.41	0.000	.222717	.3399716
	2_1	.181877	.0335393	5.42	0.000	.1161412	.2476128
	2_2	.2813443	.0299124	9.41	0.000	.222717	.3399716
L.GARCH							
	1_1	.1487581	.0697531	2.13	0.033	.0120445	.2854716
	2_1	.085404	.1446524	0.59	0.555	-.1981094	.3689175
	2_2	.1487581	.0697531	2.13	0.033	.0120445	.2854716

The results indicate that the model fits well. We could test our constraints by fitting the unconstrained model and performing either a Wald or a likelihood-ratio test. The results indicate that we might further restrict the time-invariant components of the conditional variances to be the same across companies.



► Example 4

Some models of financial data include no covariates or constant terms. For example, in modeling fictional data on the stock returns of Acme Inc. and Anvil Inc., we found it best not to include any covariates or constant terms. We include two ARCH terms and one GARCH term to model the conditional variances.

```
. use http://www.stata-press.com/data/r11/aacmer
. dvech (acme anvil = , noconstant), arch(1/2) garch(1)
Getting starting values
(setting technique to bhhh)
Iteration 0:   log likelihood = -18417.243   (not concave)
Iteration 1:   log likelihood = -18215.005
              (output omitted)
Estimating parameters
(setting technique to bhhh)
Iteration 0:   log likelihood = -17544.937
Iteration 1:   log likelihood = -17544.937
Diagonal vech multivariate GARCH model
Sample: 1 - 5000                                Number of obs   =           5000
Log likelihood = -17544.937
```

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
Sigma0						
1_1	1.026283	.0823348	12.46	0.000	.8649096	1.187656
2_1	.4300997	.0590294	7.29	0.000	.3144042	.5457952
2_2	1.019753	.0837146	12.18	0.000	.8556751	1.18383
L.ARCH						
1_1	.2878739	.02157	13.35	0.000	.2455975	.3301504
2_1	.1036685	.0161446	6.42	0.000	.0720256	.1353114
2_2	.2034196	.019855	10.25	0.000	.1645044	.2423347
L2.ARCH						
1_1	.1837825	.0274555	6.69	0.000	.1299706	.2375943
2_1	.0884425	.02208	4.01	0.000	.0451665	.1317185
2_2	.2025718	.0272639	7.43	0.000	.1491355	.256008
L.GARCH						
1_1	.0782467	.053944	1.45	0.147	-.0274816	.183975
2_1	.2888104	.0818303	3.53	0.000	.1284261	.4491948
2_2	.201618	.0470584	4.28	0.000	.1093853	.2938508

The model test is omitted from the output, because there are no covariates in the model. The univariate tests indicate that the included parameters fit the data well. In [TS] [dvech postestimation](#), we discuss prediction from models without covariates.



Saved results

dvech saves the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(k)</code>	number of parameters
<code>e(k_extra)</code>	number of auxiliary parameters
<code>e(k_eq)</code>	number of equations in <code>e(b)</code>
<code>e(k_dv)</code>	number of dependent variables
<code>e(df_m)</code>	model degrees of freedom
<code>e(ll)</code>	log likelihood
<code>e(chi2)</code>	χ^2 statistic
<code>e(p)</code>	significance
<code>e(tmin)</code>	minimum time in sample
<code>e(tmax)</code>	maximum time in sample
<code>e(rank)</code>	rank of VCE
<code>e(ic)</code>	number of iterations
<code>e(converged)</code>	1 if converged, 0 otherwise

Macros

<code>e(cmd)</code>	dvech
<code>e(cmdline)</code>	command as typed
<code>e(depvars)</code>	names of dependent variables
<code>e(covariates)</code>	list of covariates
<code>e(dv_eqs)</code>	dependent variables with mean equations
<code>e(indeps)</code>	independent variables in each equation
<code>e(tvar)</code>	time variable
<code>e(title)</code>	title in estimation output
<code>e(chi2type)</code>	Wald; type of model χ^2 test
<code>e(vce)</code>	<i>vcetype</i> specified in <code>vce()</code>
<code>e(vcetype)</code>	title used to label Std. Err.
<code>e(tmins)</code>	formatted minimum time
<code>e(tmaxs)</code>	formatted maximum time
<code>e(arch)</code>	specified ARCH terms
<code>e(garch)</code>	specified GARCH terms
<code>e(svtechnique)</code>	maximization technique(s) for starting values
<code>e(technique)</code>	maximization technique
<code>e(crittype)</code>	optimization criterion
<code>e(properties)</code>	b V
<code>e(estat_cmd)</code>	program used to implement <code>estat</code>
<code>e(predict)</code>	program used to implement <code>predict</code>
<code>e(marginsok)</code>	predictions allowed by <code>margins</code>
<code>e(marginsnotok)</code>	predictions disallowed by <code>margins</code>

Matrices

<code>e(b)</code>	coefficient vector
<code>e(Cns)</code>	constraints matrix
<code>e(ilog)</code>	iteration log (up to 20 iterations)
<code>e(gradient)</code>	gradient vector
<code>e(hessian)</code>	Hessian matrix
<code>e(A)</code>	estimates of A matrices
<code>e(B)</code>	estimates of B matrices
<code>e(S)</code>	estimates of Sigma0 matrix
<code>e(Sigma)</code>	Sigma hat
<code>e(pinfo)</code>	parameter information, used by <code>predict</code>
<code>e(V)</code>	variance–covariance matrix of the estimators
<code>e(V_modelbased)</code>	model-based variance

Functions

<code>e(sample)</code>	marks estimation sample

Methods and formulas

`dvech` is implemented as an ado-file.

Recall that the diagonal vech GARCH model can be written as

$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \boldsymbol{\epsilon}_t \quad (1)$$

$$\boldsymbol{\epsilon}_t = \mathbf{H}_t^{1/2} \boldsymbol{\nu}_t \quad (2)$$

$$\mathbf{H}_t = \mathbf{S} + \sum_{i=1}^p \mathbf{A}_i \odot \boldsymbol{\epsilon}_{t-i} \boldsymbol{\epsilon}_{t-i}' + \sum_{j=1}^q \mathbf{B}_j \odot \mathbf{H}_{t-j} \quad (3')$$

where

$\mathbf{y}_t$ is an $m \times 1$ vector of dependent variables;

$\mathbf{C}$ is an $m \times k$ matrix of parameters;

$\mathbf{x}_t$ is a $k \times 1$ vector of independent variables, which may contain lags of $\mathbf{y}_t$;

$\mathbf{H}_t^{1/2}$ is the Cholesky factor of the time-varying conditional covariance matrix $\mathbf{H}_t$;

$\boldsymbol{\nu}_t$ is an $m \times 1$ vector of normal, independent, and identically distributed innovations;

$\mathbf{S}$ is an $m \times m$ symmetric matrix of parameters;

each $\mathbf{A}_i$ is an $m \times m$ symmetric matrix of parameters;

$\odot$ is the elementwise or Hadamard product;

and each $\mathbf{B}_j$ is an $m \times m$ symmetric matrix of parameters.

`dvech` estimates the parameters by maximum likelihood. The log-likelihood function for observation t is

$$l_t = -0.5m \ln(2\pi) - 0.5 \ln \{\det(\mathbf{H}_t)\} - 0.5 \mathbf{u}_t' \mathbf{H}_t^{-1} \mathbf{u}_t$$

where $\mathbf{u}_t = \mathbf{y}_t - \mathbf{C}\mathbf{x}_t$. The log-likelihood function is $\sum_t^T l_t$. `dvech` ensures that $\mathbf{H}_t$ is positive definite for each t .

By default, `dvech` performs an iterative search for starting values. `dvech` estimates starting values for $\mathbf{C}$ by seemingly unrelated regression, uses these estimates to compute residuals $\hat{\mathbf{u}}_t$, plugs $\hat{\mathbf{u}}_t$ into the above log-likelihood function, and optimizes this log-likelihood function over the parameters in $\mathbf{H}_t$. This starting-value method plugs in consistent estimates of the parameters for the conditional means of the dependent variables and then iteratively searches for the variance parameters that maximize the log-likelihood function. Lütkepohl (2005, chap. 16) discusses this method as an estimator for the variance parameters.

GARCH estimators require initial values that can be plugged in for $\epsilon_{t-i}\epsilon'_{t-i}$ and $\mathbf{H}_{t-j}$ when $t-i < 1$ and $t-j < 1$. `dvech` substitutes an estimator of the unconditional covariance of the disturbances

$$\hat{\Sigma} = (1/T) \sum_{t=1}^T \mathbf{u}_t \mathbf{u}'_t \quad (4)$$

for $\epsilon_{t-i}\epsilon'_{t-i}$ when $t-i < 1$ and for $\mathbf{H}_{t-j}$ when $t-j < 1$.

`dvech` uses analytic first and second derivatives in maximizing the log-likelihood function.

References

- Bauwens, L., S. Laurent, and J. V. K. Rombouts. 2006. Multivariate GARCH models: A survey. *Journal of Applied Econometrics* 21: 79–109.
- Berkes, I., and L. Horváth. 2003. The rate of consistency of the quasi-maximum likelihood estimator. *Statistics and Probability Letters* 61: 133–143.
- Bollerslev, T., R. F. Engle, and D. B. Nelson. 1994. ARCH models. In *Handbook of Econometrics, Volume IV*, ed. R. F. Engle and D. L. McFadden. New York: Elsevier.
- Bollerslev, T., R. F. Engle, and J. M. Wooldridge. 1988. A capital asset pricing model with time-varying covariances. *Journal of Political Economy* 96: 116–131.
- Comte, F., and O. Lieberman. 2003. Asymptotic theory for multivariate GARCH processes. *Journal of Multivariate Analysis* 84: 61–84.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Silvennoinen, A., and T. Teräsvirta. 2009. Multivariate GARCH models. In *Handbook of Financial Time Series*, ed. T. G. Andersen, R. A. Davis, J.-P. Kreiß, and T. Mikosch, 201–229. New York: Springer.

Also see

- [TS] [arch](#) — Autoregressive conditional heteroskedasticity (ARCH) family of estimators
- [TS] [var](#) — Vector autoregressive models
- [U] [13.5 Accessing coefficients and standard errors](#)
- [U] [13.6 Accessing results from Stata commands](#)
- [U] [20 Estimation and postestimation commands](#)

Description

The following postestimation commands are available for `dvech`:

command	description
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions and residuals
<code>predictnl</code>	nonlinear predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

```
predict [type] { stub*|newvarlist } [if] [in] [, statistic options]
```

statistic	description
Main	
<code>xb</code>	dependent variables
<code>_residuals</code>	residuals
<code>_variance</code>	conditional variances and covariances

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

options	description
Options	
<code>equation(eqnames)</code>	names of equations for which predictions are made
<code>dynamic(time_constant)</code>	begin dynamic forecast at specified time

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

Main

`xb`, the default, calculates the predictions of the dependent variables.

`residuals` calculates the residuals.

`variance` predicts the conditional variances and conditional covariances.

Options

`equation(eqnames)` specifies the equation for which the predictions are calculated. Use this option to predict a statistic for a particular equation. Equation names, such as `equation(income)`, are used to identify equations.

One equation name may be specified when predicting the dependent variable, the residuals, or the conditional variance. For example, specifying `equation(income)` causes `predict` to predict `income`, and specifying `variance equation(income)` causes `predict` to predict the conditional variance of `income`.

Two equations may be specified when predicting a conditional variance or covariance. For example, specifying `equation(income, consumption) variance` causes `predict` to predict the conditional covariance of `income` and `consumption`.

`dynamic(time_constant)` specifies when `predict` starts producing dynamic forecasts. The specified *time_constant* must be in the scale of the time variable specified in `tsset`, and the *time_constant* must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2008q4))` causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] [dates and times](#). If the model contains exogenous variables, they must be present for the whole predicted sample. `dynamic()` may not be specified with `residuals`.

Remarks

We assume that you have already read [TS] [dvech](#). In this entry, we illustrate some of the features of `predict` after using `dvech` to estimate the parameters of diagonal vech GARCH models.

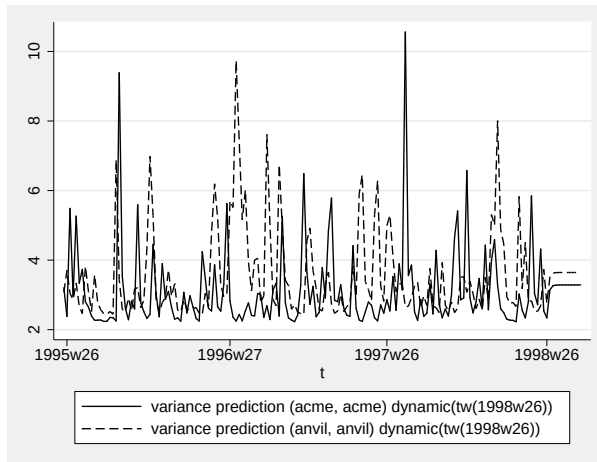
► Example 1

In this example, we obtain dynamic predictions for the Acme Inc. and Anvil Inc. fictional widget data modeled in [example 3](#) of [TS] [dvech](#). We begin by reestimating the parameters of the model.

```
. use http://www.stata-press.com/data/r11/acme
. constraint 1 [L.ARCH]1_1 = [L.ARCH]2_2
. constraint 2 [L.GARCH]1_1 = [L.GARCH]2_2
. dvech (acme = L.acme) (anvil = L.anvil), arch(1) garch(1) constraints(1 2)
(output omitted)
```

Now we use `tsappend` (see [TS] [tsappend](#)) to extend the data, use `predict` to obtain the dynamic predictions, and graph the predictions.

```
. tsappend, add(12)
. predict H*, variance dynamic(tw(1998w26))
. tsline H_acme_acme H_anvil_anvil if t>=tw(1995w25), legend(rows(2))
```



The graph shows that the in-sample predictions are similar for the conditional variances of Acme Inc. and Anvil Inc. and that the dynamic forecasts converge to similar levels. It also shows that the ARCH and GARCH parameters cause substantial time-varying volatility. The predicted conditional variance of `acme` ranges from lows of just over 2 to highs above 10.

◀

► Example 2

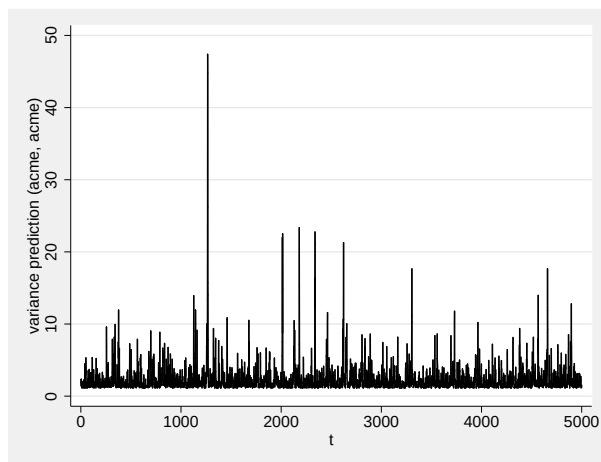
In this example, we obtain the in-sample predicted conditional variances of the returns for the fictional Acme Inc., which we modeled in [example 4](#) of [TS] `dvech`. First, we reestimate the parameters of the model.

```
. use http://www.stata-press.com/data/r11/aacmer, clear
. dvech (acme anvil = , noconstant), arch(1/2) garch(1)
(output omitted)
```

Now we use `predict` to obtain the in-sample conditional variances of `acme` and use `tsline` (see [TS] [tsline](#)) to graph the results.

(Continued on next page)

```
. predict h_acme, variance eq(acme, acme)
. tsline h_acme
```



The graph shows that the predicted conditional variances vary substantially over time, as the parameter estimates indicated.

Because there are no covariates in the model for `acme`, specifying `xb` puts a prediction of 0 in each observation, and specifying `residuals` puts the value of the dependent variable into the prediction.

◀

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

All one-step predictions are obtained by substituting the parameter estimates into the model. The estimated unconditional variance matrix of the disturbances, $\hat{\Sigma}$ is the initial value for the ARCH and GARCH terms. The postestimation routines recompute $\hat{\Sigma}$ using the prediction sample, the parameter estimates stored in `e(b)`, and (4) in *Methods and formulas* of [TS] [dvech](#).

For observations in which the residuals are missing, the estimated unconditional variance matrix of the disturbances is used in place of the outer product of the residuals.

Dynamic predictions of the dependent variables use previously predicted values beginning in the period specified by `dynamic()`.

Dynamic variance predictions are implemented by substituting $\hat{\Sigma}$ for the outer product of the residuals beginning in the period specified in `dynamic()`.

Also see

[TS] [dvech](#) — Diagonal vech multivariate GARCH models

[U] [20 Estimation and postestimation commands](#)

Title

fcast compute — Compute dynamic forecasts of dependent variables after var, svar, or vec

Syntax

After `var` and `svar`

```
fcast compute prefix [ , options1 ]
```

After `vec`

```
fcast compute prefix [ , options2 ]
```

prefix is the prefix appended to the names of the dependent variables to create the names of the variables holding the dynamic forecasts.

<i>options</i> ₁	description
Main	
<u>step</u> (#)	set # periods to forecast; default is <code>step(1)</code>
<u>dynamic</u> (<i>time_constant</i>)	begin dynamic forecasts at <i>time_constant</i>
<u>estimates</u> (<i>estname</i>)	use previously saved results <i>estname</i> ; default is to use active results
<u>replace</u>	replace existing forecast variables that have the same prefix
Std. Errors	
<u>nose</u>	suppress asymptotic standard errors
<u>bs</u>	obtain standard errors from bootstrapped residuals
<u>bsp</u>	obtain standard errors from parametric bootstrap
<u>bscentile</u>	estimate bounds by using centiles of bootstrapped dataset
<u>reps</u> (#)	perform # bootstrap replications; default is <code>reps(200)</code>
<u>nodots</u>	suppress the usual dot after each bootstrap replication
<u>saving</u> (<i>filename</i> [, <code>replace</code>])	save bootstrap results as <i>filename</i> ; use <code>replace</code> to overwrite existing <i>filename</i>
Reporting	
<u>level</u> (#)	set confidence level; default is <code>level(95)</code>

<i>options</i> ₂	description
Main	
<code>step(#)</code>	set # periods to forecast; default is <code>step(1)</code>
<code>dynamic(<i>time_constant</i>)</code>	begin dynamic forecasts at <i>time_constant</i>
<code>estimates(<i>estname</i>)</code>	use previously saved results <i>estname</i> ; default is to use active results
<code>replace</code>	replace existing forecast variables that have the same prefix
<code>differences</code>	save dynamic predictions of the first-differenced variables
Std. Errors	
<code>nose</code>	suppress asymptotic standard errors
Reporting	
<code>level(#)</code>	set confidence level; default is <code>level(95)</code>

Default is to use asymptotic standard errors if no options are specified.
fcast compute can be used only after var, svar, and vec; see [\[TS\] var](#), [\[TS\] var svar](#), and [\[TS\] vec](#).
You must `tsset` your data before using `fcast compute`; see [\[TS\] tsset](#).

Menu

Statistics > Multivariate time series > Dynamic forecasts > Compute forecasts (required for graph)

Description

`fcast compute` produces dynamic forecasts of the dependent variables in a model previously fit by `var`, `svar`, or `vec`. `fcast compute` creates new variables and, if necessary, extends the time frame of the dataset to contain the prediction horizon.

Options

Main

- `step(#)` specifies the number of periods to be forecast. The default is 1.
- `dynamic(time_constant)` specifies the period to begin the dynamic forecasts. The default is the period after the last observation in the estimation sample. The `dynamic()` option accepts either a Stata date function that returns an integer or an integer that corresponds to a date using the current `tsset` format. `dynamic()` must specify a date in the range of two or more periods into the estimation sample to one period after the estimation sample.
- `estimates(estname)` specifies that `fcast compute` use the estimation results stored as *estname*. By default, `fcast compute` uses the active estimation results. See [\[R\] estimates](#) for more information about saving and restoring previously obtained estimates.
- `replace` causes `fcast compute` to replace the variables in memory with the specified predictions.
- `differences` specifies that `fcast compute` also save dynamic predictions of the first-differenced variables. `differences` can be specified only with `vec` estimation results.

Std. Errors

nose specifies that the asymptotic standard errors of the forecasted levels and, thus the asymptotic confidence intervals for the levels, not be calculated. By default, the asymptotic standard errors and the asymptotic confidence intervals of the forecasted levels are calculated.

bs specifies that **fcast compute** use confidence bounds estimated by a simulation method based on bootstrapping the residuals.

bsp specifies that **fcast compute** use confidence bounds estimated via simulation in which the innovations are drawn from a multivariate normal distribution.

bscentile specifies that **fcast compute** use centiles of the bootstrapped dataset to estimate the bounds of the confidence intervals. By default, **fcast compute** uses the estimated standard errors and the quantiles of the standard normal distribution determined by **level()**.

reps(#) gives the number of repetitions used in the simulations. The default is 200.

nodots specifies that no dots be displayed while obtaining the simulation-based standard errors. By default, for each replication, a dot is displayed.

saving(filename[, replace]) specifies the name of the file to hold the dataset that contains the bootstrap replications. The **replace** option overwrites any file with this name.

replace indicates that *filename* be overwritten, if it exists. This option is not shown in the dialog box.

Reporting

level(#) specifies the confidence level, as a percentage, for confidence intervals. The default is **level(95)** or as set by **set level**; see [U] 20.7 Specifying the width of confidence intervals.

Remarks

Researchers often use VARs and VECMs to construct dynamic forecasts. **fcast compute** computes dynamic forecasts of the dependent variables in a VAR or VECM previously fit by **var**, **svar**, or **vec**. If you are interested in conditional, one-step-ahead predictions, use **predict** (see [TS] **var**, [TS] **var svar**, and [TS] **vec**).

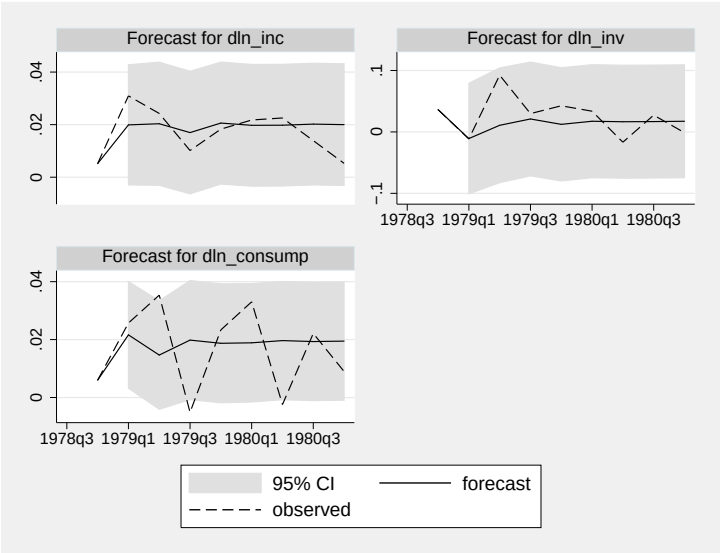
To obtain and analyze dynamic forecasts, you fit a model, use **fcast compute** to compute the dynamic forecasts, and use **fcast graph** to graph the results.

► Example 1

Typing

```
. use http://www.stata-press.com/data/r11/lutkepohl2
. var dln_inc dln_consump dln_inv if qtr<tq(1979q1)
. fcast compute m2_, step(8)
. fcast graph m2_dln_inc m2_dln_inv m2_dln_consump, observed
```

fits a VAR with two lags, computes eight-step dynamic predictions for each endogenous variable, and produces the graph



The graph shows that the model is better at predicting changes in income and investment than in consumption. The graph also shows how quickly the predictions from the two-lag model settle down to their mean values.

◀

`fcast compute` creates new variables in the dataset. If there are K dependent variables in the previously fitted model, `fcast compute` generates $4K$ new variables:

- K new variables that hold the forecasted levels, named by appending the specified prefix to the name of the original variable
- K estimated lower bounds for the forecast interval, named by appending the specified prefix and the suffix “_LB” to the name of the original variable
- K estimated upper bounds for the forecast interval, named by appending the specified prefix and the suffix “_UB” to the name of the original variable
- K estimated standard errors of the forecast, named by appending the specified prefix and the suffix “_SE” to the name of the original variable

If you specify options so that `fcast compute` does not calculate standard errors, the $3K$ variables that hold them and the bounds of the confidence intervals are not generated.

If the model previously fit is a VECM, specifying `differences` generates another K variables that hold the forecasts of the first differences of the dependent variables, named by appending the prefix “*prefixD_*” to the name of the original variable.

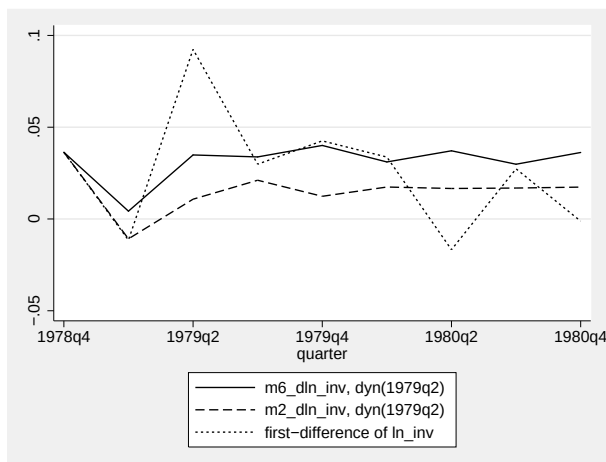
► Example 2

Plots of the forecasts from different models along with the observations from a holdout sample can provide insights to their relative forecasting performance. Continuing the previous example,

```

. var dln_inc dln_consump dln_inv if qtr< tq(1979q1), lags(1/6)
  (output omitted)
. fcast compute m6_, step(8)
. graph twoway line m6_dln_inv m2_dln_inv dln_inv qtr
> if m6_dln_inv < ., legend(cols(1))

```



The model with six lags predicts changes in investment better than the two-lag model in some periods but markedly worse in other periods.

◀

Methods and formulas

fcast compute is implemented as an ado-file.

Predictions after var and svar

A VAR with endogenous variables $\mathbf{y}_t$ and exogenous variables $\mathbf{x}_t$ can be written as

$$\mathbf{y}_t = \mathbf{v} + \mathbf{A}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{A}_p \mathbf{y}_{t-p} + \mathbf{B} \mathbf{x}_t + \mathbf{u}_t$$

where

$t = 1, \dots, T$
 $\mathbf{y}_t = (y_{1t}, \dots, y_{Kt})'$ is a $K \times 1$ random vector,
 the $\mathbf{A}_i$ are fixed $(K \times K)$ matrices of parameters,
 $\mathbf{x}_t$ is an $(M \times 1)$ vector of exogenous variables,
 $\mathbf{B}$ is a $(K \times M)$ matrix of coefficients,
 $\mathbf{v}$ is a $(K \times 1)$ vector of fixed parameters, and
 $\mathbf{u}_t$ is assumed to be white noise; that is,

$$\begin{aligned}
 E(\mathbf{u}_t) &= \mathbf{0}_K \\
 E(\mathbf{u}_t \mathbf{u}_t') &= \Sigma \\
 E(\mathbf{u}_t \mathbf{u}_s') &= \mathbf{0}_K \text{ for } t \neq s
 \end{aligned}$$

`fcast compute` will dynamically predict the variables in the vector $\mathbf{y}_t$ conditional on p initial values of the endogenous variables and any exogenous $\mathbf{x}_t$. Adopting the notation from Lütkepohl (2005, 402) to fit the case at hand, the optimal h -step-ahead forecast of $\mathbf{y}_{t+h}$ conditional on $\mathbf{x}_t$ is

$$\mathbf{y}_t(h) = \hat{\mathbf{v}} + \hat{\mathbf{A}}_1 \mathbf{y}_t(h-1) + \cdots + \hat{\mathbf{A}}_p \mathbf{y}_t(h-p) + \hat{\mathbf{B}} \mathbf{x}_t \quad (1)$$

If there are no exogenous variables, (1) becomes

$$\mathbf{y}_t(h) = \hat{\mathbf{v}} + \hat{\mathbf{A}}_1 \mathbf{y}_t(h-1) + \cdots + \hat{\mathbf{A}}_p \mathbf{y}_t(h-p)$$

When there are no exogenous variables, `fcast compute` can compute the asymptotic confidence bounds.

As shown by Lütkepohl (2005, 204–205), the asymptotic estimator of the covariance matrix of the prediction error is given by

$$\hat{\Sigma}_y(h) = \hat{\Sigma}_y(h) + \frac{1}{T} \hat{\Omega}(h) \quad (2)$$

where

$$\begin{aligned} \hat{\Sigma}_y(h) &= \sum_{i=0}^{h-1} \hat{\Phi}_i \hat{\Sigma} \hat{\Phi}_i' \\ \hat{\Omega}(h) &= \frac{1}{T} \sum_{t=0}^T \left\{ \sum_{i=0}^{h-1} \mathbf{z}_t' (\hat{\mathbf{B}}')^{h-1-i} \otimes \hat{\Phi}_i \right\} \hat{\Sigma}_\beta \left\{ \sum_{i=0}^{h-1} \mathbf{z}_t' (\hat{\mathbf{B}}')^{h-1-i} \otimes \hat{\Phi}_i \right\}' \\ \hat{\mathbf{B}} &= \begin{bmatrix} 1 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \hat{\mathbf{v}} & \hat{\mathbf{A}}_1 & \hat{\mathbf{A}}_2 & \cdots & \hat{\mathbf{A}}_{p-1} & \hat{\mathbf{A}}_p \\ \mathbf{0} & \mathbf{I}_K & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_K & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & & \ddots & & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I}_K & \mathbf{0} \end{bmatrix} \\ \mathbf{z}_t &= (1, \mathbf{y}_t', \dots, \mathbf{y}_{t-p-1}')' \\ \hat{\Phi}_0 &= \mathbf{I}_K \\ \hat{\Phi}_i &= \sum_{j=1}^i \hat{\Phi}_{i-j} \hat{\mathbf{A}}_j \quad i = 1, 2, \dots \\ \hat{\mathbf{A}}_j &= \mathbf{0} \quad \text{for } j > p \end{aligned} \quad (4)$$

$\hat{\Sigma}$ is the estimate of the covariance matrix of the innovations, and $\hat{\Sigma}_\beta$ is the estimated VCE of the coefficients in the VAR. The formula in (3) is general enough to handle the case in which constraints are placed on the coefficients in the VAR(p).

Equation (2) is made up of two terms. $\hat{\Sigma}_y(h)$ is the estimated mean squared error (MSE) of the forecast. $\hat{\Sigma}_y(h)$ estimates the error in the forecast arising from the unseen innovations. $T^{-1} \hat{\Omega}(h)$ estimates the error in the forecast that is due to using estimated coefficients instead of the true coefficients. As the sample size grows, uncertainty with respect to the coefficient estimates decreases, and $T^{-1} \hat{\Omega}(h)$ goes to zero.

If $\mathbf{y}_t$ is normally distributed, the bounds for the asymptotic $(1 - \alpha)100\%$ interval around the forecast for the k th component of $\mathbf{y}_t$, h periods ahead, are

$$\hat{\mathbf{y}}_{k,t}(h) \pm z_{(\frac{\alpha}{2})} \hat{\sigma}_k(h) \quad (3)$$

where $\hat{\sigma}_k(h)$ is the k th diagonal element of $\hat{\Sigma}_{\hat{\mathbf{y}}}(h)$.

Specifying the `bs` option causes the standard errors to be computed via simulation, using bootstrapped residuals. Both `var` and `svar` contain estimators for the coefficients of a VAR that are conditional on the first p observations on the endogenous variables in the data. Similarly, these algorithms are conditional on the first p observations of the endogenous variables in the data. However, the simulation-based estimates of the standard errors are also conditional on the estimated coefficients. The asymptotic standard errors are not conditional on the coefficient estimates because the second term on the right-hand side of (2) accounts for the uncertainty arising from using estimated parameters.

For a simulation with R repetitions, this method uses the following algorithm:

1. Fit the model and save the estimated coefficients.
2. Use the estimated coefficients to calculate the residuals.
3. Repeat steps 3a–3c R times.
 - 3a. Draw a simple random sample with replacement of size $T + h$ from the residuals. When the t th observation is drawn, all K residuals are selected, preserving any contemporaneous correlation among the residuals.
 - 3b. Use the sampled residuals, p initial values of the endogenous variables, any exogenous variables, and the estimated coefficients to construct a new sample dataset.
 - 3c. Save the simulated endogenous variables for the h forecast periods in the bootstrapped dataset.
4. For each endogenous variable and each forecast period, the simulated standard error is the estimated standard error of the R simulated forecasts. By default, the upper and lower bounds of the $(1 - \alpha)100\%$ are estimated using the simulation-based estimates of the standard errors and the normality assumption, as in (3). If the `bscentile` option is specified, the sample centiles for the upper and lower bounds of the R simulated forecasts are used for the upper and lower bounds of the confidence intervals.

If the `bsp` option is specified, a parametric simulation algorithm is used. Specifically, everything is as above except that 3a is replaced by 3a(bsp) as follows:

- 3a(bsp). Draw $T + h$ observations from a multivariate normal distribution with covariance matrix $\hat{\Sigma}$.

The algorithm above assumes that h forecast periods come after the original sample of T observations. If the h forecast periods lie within the original sample, smaller simulated datasets are sufficient.

Dynamic forecasts after vec

Methods and formulas of [TS] `vec` discusses how to obtain the one-step predicted differences and levels. `fcast compute` uses the previous dynamic predictions as inputs for later dynamic predictions.

Per [Lütkepohl \(2005, sec. 6.5\)](#), `fcast compute` uses

$$\widehat{\Sigma}_y(h) = \left(\frac{T}{T-d} \right) \sum_{i=0}^{h-1} \widehat{\Phi}_i \widehat{\Omega} \widehat{\Phi}_i'$$

where the $\widehat{\Phi}_i$ are the estimated matrices of impulse–response functions, T is the number of observations in the sample, d is the number of degrees of freedom, and $\widehat{\Omega}$ is the estimated cross-equation variance matrix. The formulas for d and $\widehat{\Omega}$ are given in [Methods and formulas](#) of [TS] `vec`.

The estimated standard errors at step h are the square roots of the diagonal elements of $\widehat{\Sigma}_y(h)$.

Per [Lütkepohl \(2005\)](#), the estimated forecast-error variance does not consider parameter uncertainty. As the sample size gets infinitely large, the importance of parameter uncertainty diminishes to zero.

References

- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Also see

- [TS] `fcast graph` — Graph forecasts of variables computed by `fcast compute`
- [TS] `var intro` — Introduction to vector autoregressive models
- [TS] `vec intro` — Introduction to vector error-correction models

Syntax

```
fcast graph varlist [if] [in] [, options]
```

where *varlist* contains one or more forecasted variables generated by `fcast compute`.

options	description
Main	
<u>d</u> ifferences	graph forecasts of the first-differenced variables (<code>vec</code> only)
<u>n</u> oci	suppress confidence bands
<u>o</u> bserved	include observed values of the predicted variables
Forecast plot	
<u>c</u> line_options	affect rendition of the forecast lines
CI plot	
<u>c</u> iopts(<i>area_options</i>)	affect rendition of the confidence bands
Observed plot	
<u>o</u> bopts(<i>cline_options</i>)	affect rendition of the observed values
Y axis, Time axis, Titles, Legend, Overall	
<u>t</u> woway_options	any options other than <code>by()</code> documented in [G] twoway_options
<u>b</u> yopts(<i>by_option</i>)	affect appearance of the combined graph; see [G] by_option

Menu

Statistics > Multivariate time series > Dynamic forecasts > Graph forecasts

Description

`fcast graph` graphs dynamic forecasts of the endogenous variables from a VAR(p) or VECM that has already been obtained from `fcast compute`; see [TS] [fcast compute](#).

Options

Main

- `differences` specifies that the forecasts of the first-differenced variables be graphed. This option is available only with forecasts computed by `fcast compute` after `vec`. The `differences` option implies `noci`.
- `noci` specifies that the confidence intervals be suppressed. By default, the confidence intervals are included.
- `observed` specifies that observed values of the predicted variables be included in the graph. By default, observed values are not graphed.

Forecast plot

cline_options affect the rendition of the plotted lines corresponding to the forecast; see [G] [cline_options](#).

CI plot

ciopts(area_options) affects the rendition of the confidence bands for the forecasts; see [G] [area_options](#).

Observed plot

obopts(cline_options) affects the rendition of the observed values of the predicted variables; see [G] [cline_options](#). This option implies the observed option.

Y axis, Time axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G] [twoway_options](#), excluding *by()*.

byopts(by_option) are documented in [G] [by_option](#). These options affect the appearance of the combined graph.

Remarks

`fcast graph` graphs dynamic forecasts created by `fcast compute`.

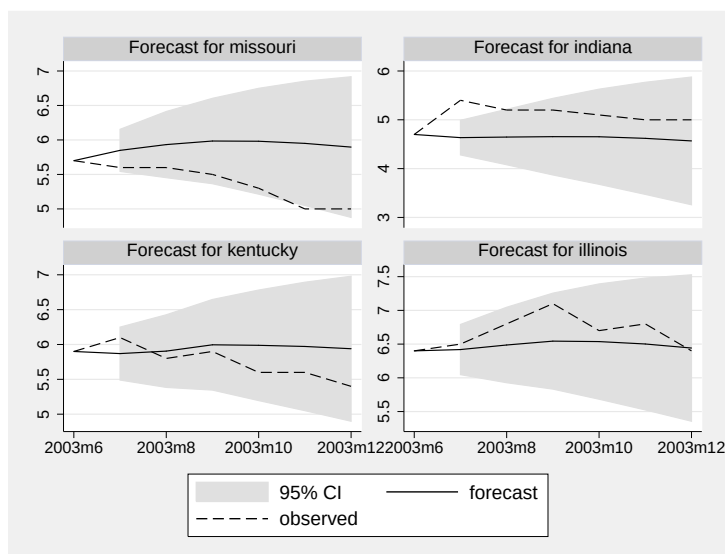
► Example 1

In this example, we use a cointegrating VECM to model the state-level unemployment rates in Missouri, Indiana, Kentucky, and Illinois, and we graph the forecasts against a 6-month holdout sample.

```
. use http://www.stata-press.com/data/r11/urates
. vec missouri indiana kentucky illinois if t < tm(2003m7), trend(rconstant)
> rank(2) lags(4)
(output omitted)
. fcast compute m1_, step(6)
```



```
. fcast graph m1_missouri m1_indiana m1_kentucky m1_illinois, observed
```



Because the 95% confidence bands for the predicted unemployment rates in Missouri and Indiana do not contain all their observed values, the model does not reliably predict these unemployment rates.

◀

Methods and formulas

`fcast graph` is implemented as an ado-file.

Also see

[TS] [fcast compute](#) — Compute dynamic forecasts of dependent variables after `var`, `svar`, or `vec`

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Title

haver — Load data from Haver Analytics database

Syntax

Describe contents of a Haver dataset

```
haver describe filename [, detail]
```

Describe specified variables in a Haver dataset

```
haver describe varlist using filename [, detail]
```

Load Haver dataset

```
haver use filename [, use_options]
```

Load specified variables from a Haver dataset

```
haver use varlist using filename [, use_options]
```

use_options	description
tin([constant], [constant])	load data within specified date range
twwithin([constant], [constant])	same as tin(), except exclude the endpoints of range
tvar(varname)	create time variable varname
hmissing(misval)	record missing values as misval
fill	include observations with missing data in resulting dataset and record missing values for them
clear	clear data in memory before loading the Haver dataset

Menu

File > Import > Haver Analytics database

Description

Haver Analytics (<http://www.haver.com>) provides economic and financial databases in the form of .dat files to which you can purchase access. The `haver` command allows you to use those datasets with Stata. The `haver` command is provided only with Stata for Windows.

- `haver describe` describes the contents of a Haver file.
- `haver use` loads the specified variables from a Haver file into Stata’s memory.
- If *filename* is specified without a suffix, .dat is assumed.

Option for use with `haver describe`

`detail` specifies that a detailed report of all information available on the variables be presented. By default, the Haver concepts data type, number of observations, aggregate type, difference type, magnitude, and date modified are not reported.

Options for use with `haver use`

`tin([constant], [constant])` specifies the date range of the data to be loaded. *constant* refers to a date constant specified in the usual way, e.g., `tin(1jan1999, 31dec1999)`, which would mean from and including 1 January 1999 through 31 December 1999.

`twithin([constant], [constant])` functions the same as `tin()`, except that the endpoints of the range will be excluded in the data loaded.

`tvar(varname)` specifies the name of the time variable Stata will create; the default is `tvar(time)`. The `tvar()` variable is the name of the variable that you would use to `tsset` the data after loading, although doing is unnecessary because `haver use` automatically `tssets` the data for you.

`hmissing(misval)` specifies which of Stata's 27 missing values (`.`, `.a`, ..., `.z`) is to be recorded when there are missing values in the Haver dataset.

Two kinds of missing values can occur when loading a Haver dataset. One is created when a variable's data does not span the entire time series, and these missing values are always recorded as `.` by Stata. The other corresponds to an actual missing value recorded in the Haver format. `hmissing()` sets what is recorded when this second kind of missing value is encountered. You could specify `hmissing(.h)` to cause Stata's `.h` code to be recorded here.

The default is to store `.` for both kinds of missing values. See [U] 12.2.1 Missing values.

`fill` specifies that observations with no data be left in the resulting dataset formed in memory and that missing values be recorded for them. The default is to exclude such observations, which can result in the loaded time-series dataset having gaps.

Specifying `fill` has the same effect as issuing the `tsfill` command after loading; see [TS] `tsfill`.

`clear` clears the data in memory before loading the Haver dataset.

Remarks

Remarks are presented under the following headings:

- [Installation](#)
- [Determining the contents of a Haver dataset](#)
- [Loading a Haver dataset](#)
- [Combining variables](#)
- [Using subsets](#)

Installation

Haver Analytics (<http://www.haver.com>) provides more than 100 economic and financial databases in the form of `.dat` files to which you can purchase access. The `haver` command allows you to use those datasets with Stata.

`haver` is provided only with Stata for Windows, and it uses the `DLXAPI32.DLL` file, which Haver provides. That file is probably already installed. If it is not, Stata will complain that “`DLXAPI32.DLL` could not be found”.

[illegible]

The entire dataset contains 10,849 observations. The variables FDB6 through FFED5 contain missing because there is no information about them for these first few dates; later in the dataset, nonmissing values for them will be found.

haver use requires that the variables loaded all be of the same time frequency (e.g., annual, quarterly). Haver's weekly data are treated by Stata as daily data with a 7-day interval, so weekly data is loaded as daily data with gaps.

Haver datasets include the following information about each variable:

- Source — the Haver code associated with the source for the data
- Group — the Haver group to which the variable belongs
- DataType — whether the variable is a ratio, index, etc.
- AggType — the Haver code for the level of aggregation
- DifType — the Haver code for the type of difference
- DecPrecision — the number of decimals to which the variable is recorded
- Magnitude — the Haver magnitude code
- DateTimeMod — the date the series was last modified by Haver

When a variable is loaded, this information is stored in variable characteristics (see [\[P\] char](#)). Those characteristics can be viewed using `char list`:

```
. char list FDB6[]
FDB6[Source]:          FRB
FDB6[Group]:          F01
FDB6[DataType]:       %
FDB6[AggType]:        1
FDB6[DifType]:        1
FDB6[DecPrecision]:   2
FDB6[Magnitude]:      0
FDB6[DateTimeMod]:    04aug2003
```

Combining variables

In fact, the variables FFED through FFED6 all contain information about the same thing (the Federal Funds rate) but over different times:

```
. haver describe dailyst
Dataset:      dailyst (use)
```

Variable	Description	Time span	Format	Group	Source
FDB6	6-Month Eurod..	02jan1996-01aug2003	Daily	F01	FRB
FFED	Federal Funds..	01jan1996-01aug2003	Daily	F01	FRB
FFED2	Federal Funds..	01jan1988-29dec1995	Daily	F70	FRB
FFED3	Federal Funds..	01jan1980-31dec1987	Daily	F70	FRB
FFED4	Federal Funds..	03jan1972-31dec1979	Daily	F70	FRB
FFED5	Federal Funds..	01jan1964-31dec1971	Daily	F70	FRB
FFED6	Federal Funds..	02jan1962-31dec1963	Daily	F70	FRB

In Haver datasets, variables may contain up to 2,100 data points, and if more are needed, Haver Analytics breaks the variables into multiple variables. You must put them back together.

To determine when variables need to be combined, look at the description and time span. To combine variables, use `egen's rowfirst()` function and then drop the originals:

```
. egen fedfunds = rowfirst(FFED FFED2 FFED3 FFED4 FFED5 FFED6)
. drop FFED*
```

You can also combine variables by hand:

```
. gen fedfunds = FFED
(8869 missing values generated)
. replace fedfunds = FFED2 if fedfunds >= .
(2086 real changes made)
. replace fedfunds = FFED3 if fedfunds >= .
(2088 real changes made)
. replace fedfunds = FFED4 if fedfunds >= .
(2086 real changes made)
. replace fedfunds = FFED5 if fedfunds >= .
(2088 real changes made)
. replace fedfunds = FFED6 if fedfunds >= .
(521 real changes made)
. drop FFED*
```

Using subsets

You can use a subset of the Haver dataset by specifying the variables to be loaded or the observations to be included:

```
. haver use FFED FFED2 using dailytst, tin(1jan1990,31dec1999)
(data from dailytst.dat loaded)
```

Above, we load only the variables FFED and FFED2 and the observations from 1jan1990 to 31dec1999.

Also see

- [\[TS\] **tsset**](#) — Declare data to be time-series data
- [\[D\] **infile**](#) — Overview of reading data into Stata
- [\[D\] **insheet**](#) — Read ASCII (text) data created by a spreadsheet
- [\[D\] **odbc**](#) — Load, write, or view data from ODBC sources

Syntax

`irf subcommand ... [ , ... ]`

<i>subcommand</i>	description
<code>create</code>	create IRF file containing IRFs, dynamic-multiplier functions, and FEVDs
<code>set</code>	set the active IRF file
<code>graph</code>	graph results from active file
<code>cgraph</code>	combine graphs of IRFs, dynamic-multiplier functions, and FEVDs
<code>ograph</code>	graph overlaid IRFs, dynamic-multiplier functions, and FEVDs
<code>table</code>	create tables of IRFs, dynamic-multiplier functions, and FEVDs from active file
<code>ctable</code>	combine tables of IRFs, dynamic-multiplier functions, and FEVDs
<code>describe</code>	describe contents of active file
<code>add</code>	add results from an IRF file to the active IRF file
<code>drop</code>	drop IRF results from active file
<code>rename</code>	rename IRF results within a file

IRF stands for impulse–response function; FEVD stands for forecast-error variance decomposition. `irf` can be used only after `var`, `svar`, or `vec`; see [TS] [var](#), [TS] [var svar](#), and [TS] [vec](#). See [TS] [irf create](#), [TS] [irf set](#), [TS] [irf graph](#), [TS] [irf cgraph](#), [TS] [irf ograph](#), [TS] [irf table](#), [TS] [irf ctable](#), [TS] [irf describe](#), [TS] [irf add](#), [TS] [irf drop](#), and [TS] [irf rename](#) for details about subcommands.

Description

`irf` creates and manipulates IRF files that contain estimates of the IRFs, dynamic-multiplier functions, and forecast-error variance decompositions (FEVDs) created after estimation by `var`, `svar`, or `vec`.

IRFs and FEVDs are described below, and the process of analyzing them is outlined. After reading this entry, please see [TS] [irf create](#).

Remarks

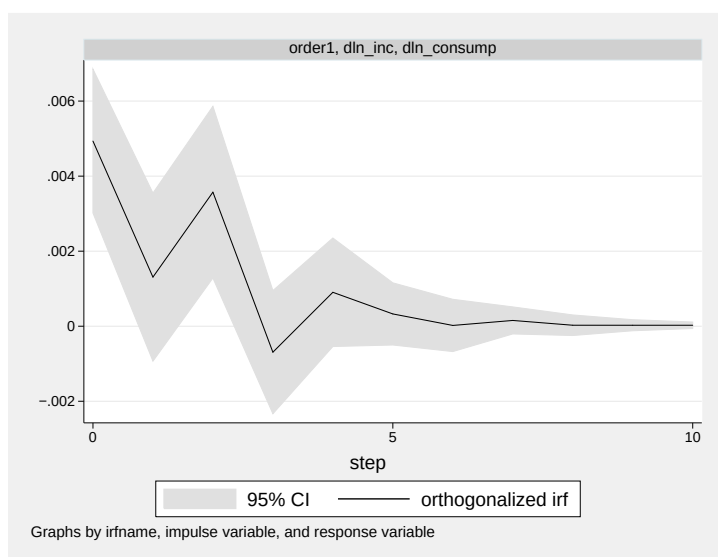
An IRF measures the effect of a shock to an endogenous variable on itself or on another endogenous variable; see [Lütkepohl \(2005, 51–63\)](#) and [Hamilton \(1994, 318–323\)](#) for formal definitions. Of the many types of IRFs, `irf create` estimates the five most important: simple IRFs, orthogonalized IRFs, cumulative IRFs, cumulative orthogonalized IRFs, and structural IRFs.

A dynamic-multiplier function, or transfer function, measures the impact of a unit increase in an exogenous variable on the endogenous variables over time; see [Lütkepohl \(2005, chap. 10\)](#) for formal definitions. `irf create` estimates simple and cumulative dynamic-multiplier functions after `var`.

The forecast-error variance decomposition (FEVD) measures the fraction of the forecast-error variance of an endogenous variable that can be attributed to orthogonalized shocks to itself or to another endogenous variable; see [Lütkepohl \(2005, 63–66\)](#) and [Hamilton \(1994, 323–324\)](#) for formal definitions. Of the many types of FEVDs, `irf create` estimates the two most important: Cholesky and structural.

To analyze IRFs and FEVDs in Stata, you first fit a model, then use `irf create` to estimate the IRFs and FEVDs and store them in a file, and finally use `irf graph` or any of the other `irf` analysis commands to examine results:

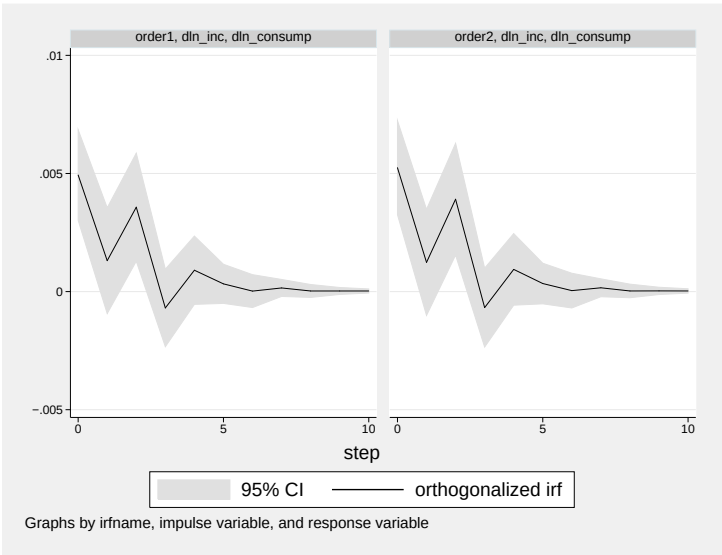
```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
(output omitted)
. irf create order1, step(10) set(myirf1)
(file myirf1.irf created)
(file myirf1.irf now active)
(file myirf1.irf updated)
. irf graph oirf, impulse(dln_inc) response(dln_consump)
```



Multiple sets of IRFs and FEVDs can be placed in the same file, with each set of results in a file bearing a distinct name. The `irf create` command above created file `myirf1.irf` and put one set of results in it, named `order1`. The `order1` results include estimates of the simple IRFs, orthogonalized IRFs, cumulative IRFs, cumulative orthogonalized IRFs, and Cholesky FEVDs.

Below we use the same estimated `var` but use a different Cholesky ordering to create a second set of IRF results, which we will store as `order2` in the same file, and then we will graph both results:


```
. irf create order2, step(10) order(dln_inc dln_inv dln_consump)
(file myirf1.irf updated)
. irf graph oirf, irf(order1 order2) impulse(dln_inc) response(dln_consump)
```



We have compared results for one model under two different identification schemes. We could just as well have compared results of two different models. We now use `irf table` to display the results tabularly:

```
. irf table oirf, irf(order1 order2) impulse(dln_inc) response(dln_consump)
Results from order1 order2
```

step	(1) oirf	(1) Lower	(1) Upper	(2) oirf	(2) Lower	(2) Upper
0	.004934	.003016	.006852	.005244	.003252	.007237
1	.001309	-.000931	.003549	.001235	-.001011	.003482
2	.003573	.001285	.005862	.00391	.001542	.006278
3	-.000692	-.002333	.00095	-.000677	-.002347	.000993
4	.000905	-.000541	.002351	.00094	-.000576	.002456
5	.000328	-.0005	.001156	.000341	-.000518	.001201
6	.000021	-.000675	.000717	.000042	-.000693	.000777
7	.000154	-.000206	.000515	.000161	-.000218	.00054
8	.000026	-.000248	.0003	.000027	-.000261	.000315
9	.000026	-.000121	.000174	.00003	-.000125	.000184
10	.000026	-.000061	.000113	.000027	-.000065	.00012

95% lower and upper bounds reported
(1) irfname = order1, impulse = dln_inc, and response = dln_consump
(2) irfname = order2, impulse = dln_inc, and response = dln_consump

Both the table and the graph show that the two orthogonalized IRFs are essentially the same. In both functions, an increase in the orthogonalized shock to `dln_inc` causes a short series of increases in `dln_consump` that dies out after four or five periods.

References

- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Also see

- [TS] **var** — Vector autoregressive models
- [TS] **var svar** — Structural vector autoregressive models
- [TS] **varbasic** — Fit a simple VAR and graph IRFS or FEVDs
- [TS] **vec** — Vector error-correction models
- [TS] **var intro** — Introduction to vector autoregressive models
- [TS] **vec intro** — Introduction to vector error-correction models

Title

irf add — Add results from an IRF file to the active IRF file

Syntax

```
irf add { _all | [newname=]oldname ... }, using(irf_filename)
```

Menu

Statistics > Multivariate time series > Manage IRF results and files > Add IRF results

Description

`irf add` copies results from one IRF file to another—from the specified `using()` file to the active IRF file, set by `irf set`.

Option

`using(irf_filename)` specifies the file from which results are to be obtained and is required. If *irf_filename* is specified without an extension, `.irf` is assumed.

Remarks

If you have not read [\[TS\] irf](#), please do so.

► Example 1

After fitting a VAR model, we create two separate IRF files:

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
(output omitted)
. irf create original, set(irf1, replace)
(file irf1.irf created)
(file irf1.irf now active)
(file irf1.irf updated)
. irf create order2, order(dln_inc dln_inv dln_consump) set(irf2, replace)
(file irf2.irf created)
(file irf2.irf now active)
(file irf2.irf updated)
```

We copy IRF results `original` to the active file giving them the name `order1`.

```
. irf add order1 = original, using(irf1)
(file irf2.irf updated)
```

Here we create new IRF results and save them in the new file `irf3`.

```
. irf create order3, order(dln_inc dln_consump dln_inv) set(irf3, replace)
(file irf3.irf created)
(file irf3.irf now active)
(file irf3.irf updated)
```

Now we copy all the IRF results in file `irf2` into the active file.

```
. irf add _all, using(irf2)
(file irf3.irf updated)
```



Methods and formulas

`irf add` is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Syntax

```
irf cgraph (spec1) [ (spec2) ... [ (specN) ] ] [ , options ]
```

where (*spec*_{*k*}) is

```
(irfname impulsevar responsevar stat [ , spec_options ])
```

irfname is the name of a set of IRF results in the active IRF file. *impulsevar* should be specified as an endogenous variable for all statistics except *dm* and *cdm*; for those, specify as an exogenous variable. *responsevar* is an endogenous variable name. *stat* is one or more statistics from the list below:

<i>stat</i>	description
Main	
<i>irf</i>	impulse–response function
<i>oirf</i>	orthogonalized impulse–response function
<i>dm</i>	dynamic-multiplier function
<i>cirf</i>	cumulative impulse–response function
<i>coirf</i>	cumulative orthogonalized impulse–response function
<i>cdm</i>	cumulative dynamic-multiplier function
<i>fevd</i>	Cholesky forecast-error variance decomposition
<i>sirf</i>	structural impulse–response function
<i>sfevd</i>	structural forecast-error variance decomposition

- Notes: 1. No statistic may appear more than once.
2. If confidence intervals are included (the default), only two statistics may be included.
3. If confidence intervals are suppressed (option *nocl*), up to four statistics may be included.

<i>options</i>	description
Main	
<i>set</i> (<i>filename</i>)	make <i>filename</i> active
Options	
<i>combine_options</i>	affect appearance of combined graph
Y axis, X axis, Titles, Legend, Overall	
<i>twoway_options</i>	any options other than <i>by()</i> documented in [G] twoway_options
* <i>spec_options</i>	level, steps, and rendition of plots and their CIs
† <i>individual</i>	graph each combination individually

* *spec_options* appear on multiple tabs in the dialog box.
† *individual* is not shown in the dialog box.

<i>spec_options</i>	description
Main	
<code>noci</code>	suppress confidence bands
Options	
<code>level(#)</code>	set confidence level; default is <code>level(95)</code>
<code>lstep(#)</code>	use <code>#</code> for first step
<code>ustep(#)</code>	use <code>#</code> for maximum step
Plots	
<code>plot#opts(line_options)</code>	affect rendition of the line plotting the <code># stat</code>
CI plots	
<code>ci#opts(area_options)</code>	affect rendition of the confidence interval for the <code># stat</code>

spec_options may be specified within a graph specification, globally, or in both. When specified in a graph specification, the *spec_options* affect only the specification in which they are used. When supplied globally, the *spec_options* affect all graph specifications. When supplied in both places, options in the graph specification take precedence.

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Combined graphs

Description

`irf cgraph` makes a graph or a combined graph of IRF results. Each block within a pair of matching parentheses—each (*spec_k*)—specifies the information for a specific graph. `irf cgraph` combines these graphs into one image, unless the `individual` option is also specified, in which case separate graphs for each block are created.

To become familiar with this command, we recommend that you type `db irf cgraph`.

Options

Main	
<code>noci</code>	suppresses graphing the confidence interval for each statistic. <code>noci</code> is assumed when the model was fit by <code>vec</code> because no confidence intervals were estimated.
<code>set(filename)</code>	specifies the file to be made active; see [TS] irf set . If <code>set()</code> is not specified, the active file is used.

Options	
<code>level(#)</code>	specifies the default confidence level, as a percentage, for confidence intervals, when they are reported. The default is <code>level(95)</code> or as set by <code>set level</code> ; see [U] 20.7 Specifying the width of confidence intervals . The value set of an overall <code>level()</code> can be overridden by the <code>level()</code> inside a (<i>spec_k</i>).
<code>lstep(#)</code>	specifies the first step, or period, to be included in the graph. <code>lstep(0)</code> is the default.
<code>ustep(#)</code>	<code># ≥ 1</code> , specifies the maximum step, or period, to be included in the graph.
<code>combine_options</code>	affect the appearance of the combined graph; see [G] graph combine .

Plots

`plot1opts(cline_options), ..., plot4opts(cline_options)` affect the rendition of the plotted statistics. `plot1opts()` affects the rendition of the first statistic; `plot2opts()`, the second; and so on. *cline_options* are as described in [G] [cline_options](#).

CI plots

`ci1opts1(area_options)` and `ci2opts2(area_options)` affect the rendition of the confidence intervals for the first (`ci1opts()`) and second (`ci2opts()`) statistics. See [TS] [irf graph](#) for a description of this option and [G] [area_options](#) for the suboptions that change the look of the CI.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

The following option is available with `irf cgraph` but is not shown in the dialog box:

`individual` specifies that each graph be displayed individually. By default, `irf cgraph` combines the subgraphs into one image.

Remarks

If you have not read [TS] [irf](#), please do so.

The relationship between `irf cgraph` and `irf graph` is syntactically and conceptually the same as that between `irf ctable` and `irf table`; see [TS] [irf ctable](#) for a description of the syntax.

`irf cgraph` is much the same as using `irf graph` to make individual graphs and then using `graph combine` to put them together. If you cannot use `irf cgraph` to do what you want, consider the other approach.

► Example 1

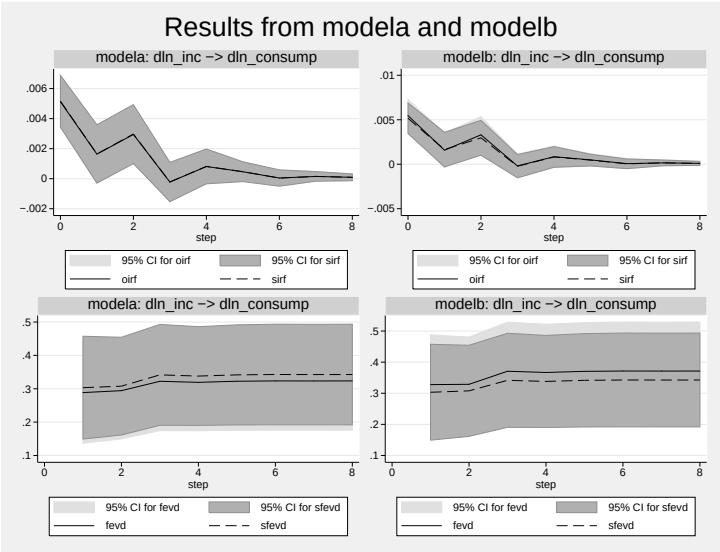
You have previously issued the commands

```
. use http://www.stata-press.com/data/r11/lutkepohl2
. mat a = (., 0, 0\0,.,0\.,.,.)
. mat b = I(3)
. svar dln_inv dln_inc dln_consump, aeq(a) beq(b)
. irf create modela, set(results3) step(8)
. svar dln_inc dln_inv dln_consump, aeq(a) beq(b)
. irf create modelb, step(8)
```

(Continued on next page)

You now type

```
. irf cgraph (modela dln_inc dln_consump oirf sirf)
> (modelb dln_inc dln_consump oirf sirf)
> (modela dln_inc dln_consump fevd sfevd, lstep(1))
> (modelb dln_inc dln_consump fevd sfevd, lstep(1)),
> title("Results from modela and modelb")
```



Saved results

irf cgraph saves the following in `r()`:

Scalars	
<code>r(k)</code>	number of specific graph commands
Macros	
<code>r(individual)</code>	individual, if specified
<code>r(save)</code>	<i>filename</i> , replace from <code>saving()</code> option for combined graph
<code>r(name)</code>	<i>name</i> , replace from <code>name()</code> option for combined graph
<code>r(title)</code>	title of the combined graph
<code>r(save#)</code>	<i>filename</i> , replace from <code>saving()</code> option for individual graphs
<code>r(name#)</code>	<i>name</i> , replace from <code>name()</code> option for individual graphs
<code>r(title#)</code>	title for the <i>#</i> th graph
<code>r(ci#)</code>	level applied to the <i>#</i> th confidence interval or <code>nocl</code>
<code>r(response#)</code>	response specified in the <i>#</i> th command
<code>r(impulse#)</code>	impulse specified in the <i>#</i> th command
<code>r(irfname#)</code>	IRF name specified in the <i>#</i> th command
<code>r(stats#)</code>	statistics specified in the <i>#</i> th command

Methods and formulas

`irf cgraph` is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

irf create — Obtain IRFs, dynamic-multiplier functions, and FEVDs

Syntax

After var

```
irf create irfname [ , var_options ]
```

After svar

```
irf create irfname [ , svar_options ]
```

After vec

```
irf create irfname [ , vec_options ]
```

irfname is any valid name that does not exceed 15 characters.

var_options	description
Main	
set(<i>filename</i> [, replace])	make <i>filename</i> active
replace	replace <i>irfname</i> if it already exists
step(#)	set forecast horizon to #; default is step(8)
order(<i>varlist</i>)	specify Cholesky ordering of endogenous variables
estimates(<i>estname</i>)	use previously saved results <i>estname</i> ; default is to use active results
Std. errors	
nose	do not calculate standard errors
bs	obtain standard errors from bootstrapped residuals
bsp	obtain standard errors from parametric bootstrap
nodots	do not display “.” for each bootstrap replication
reps(#)	use # bootstrap replications; default is reps(200)
bsaving(<i>filename</i> [, replace])	save bootstrap results in <i>filename</i>

svar_options	description
Main	
set(<i>filename</i> [, replace])	make <i>filename</i> active
replace	replace <i>irfname</i> if it already exists
step(#)	set forecast horizon to #; default is step(8)
estimates(<i>estname</i>)	use previously saved results <i>estname</i> ; default is to use active results

Std. errors

<code>nose</code>	do not calculate standard errors
<code>bs</code>	obtain standard errors from bootstrapped residual
<code>bsp</code>	obtain standard errors from parametric bootstrap
<code>nodots</code>	do not display “.” for each bootstrap replication
<code>reps(#)</code>	use # bootstrap replications; default is <code>reps(200)</code>
<code>bsaving(filename[, replace])</code>	save bootstrap results in <i>filename</i>

<i>vec_options</i>	description

Main

<code>set(filename[, replace])</code>	make <i>filename</i> active
<code>replace</code>	replace <i>irfname</i> if it already exists
<code>step(#)</code>	set forecast horizon to #; default is <code>step(8)</code>
<code>estimates(estname)</code>	use previously saved results <i>estname</i> ; default is to use active results

The default is to use asymptotic standard errors if no options are specified.

`irf create` is for use after fitting a model with the `var`, `svar`, or `vec` commands; see [TS] `var`, [TS] `var svar`, and [TS] `vec`.

You must `tsset` your data before using `var`, `svar`, or `vec` and, hence, before using `irf create`; see [TS] `tsset`.

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Obtain IRFs, dynamic-multiplier functions, and FEVDs

Description

`irf create` estimates multiple sets of impulse–response functions (IRFs), dynamic-multiplier functions, and forecast-error variance decompositions (FEVDs) after estimation by `var`, `svar`, or `vec`. These estimates and their standard errors are known collectively as IRF results and are stored in an IRF file under the specified *irfname*.

The following types of IRFs and dynamic-multiplier functions are stored:

simple IRFs	after <code>svar</code> , <code>var</code> , or <code>vec</code>
orthogonalized IRFs	after <code>svar</code> , <code>var</code> , or <code>vec</code>
dynamic multipliers	after <code>var</code>
cumulative IRFs	after <code>svar</code> , <code>var</code> , or <code>vec</code>
cumulative orthogonalized IRFs	after <code>svar</code> , <code>var</code> , or <code>vec</code>
cumulative dynamic multipliers	after <code>var</code>
structural IRFs	after <code>svar</code> only

The following types of FEVDs are stored:

Cholesky FEVDs	after <code>svar</code> , <code>var</code> , or <code>vec</code>
structural FEVDs	after <code>svar</code> only

Once you have created a set of IRF results, use the other `irf` commands to analyze them.

Options

Main

`set(filename[, replace])` specifies the IRF file to be used. If `set()` is not specified, the active IRF file is used; see [TS] [irf set](#).

If `set()` is specified, the specified file becomes the active file, just as if you had issued an `irf set` command.

`replace` specifies that the results stored under *irfname* may be replaced, if they already exist. IRF results are saved in files, and one file may contain multiple IRF results.

`step(#)` specifies the step (forecast) horizon; the default is eight periods.

`order(varlist)` is allowed only after estimation by `var`; it specifies the Cholesky ordering of the endogenous variables to be used when estimating the orthogonalized IRFs. By default, the order in which the variables were originally specified on the `var` command is used.

`estimates(estname)` specifies that estimation results previously estimated by `svar`, `var`, or `vec`, and stored by `estimates`, be used. This option is rarely specified; see [R] [estimates](#).

Std. errors

`nose`, `bs`, and `bsp` are alternatives that specify how (whether) standard errors are to be calculated. If none of these options is specified, asymptotic standard errors are calculated, except in two cases: after estimation by `vec` and after estimation by `svar` in which long-run constraints were applied. In those two cases, the default is as if `nose` were specified, although in the second case, you could specify `bs` or `bsp`. After estimation by `vec`, standard errors are simply not available.

`nose` specifies that no standard errors be calculated.

`bs` specifies that standard errors be calculated by bootstrapping the residuals. `bs` may not be specified if there are gaps in the data.

`bsp` specifies that standard errors be calculated via a multivariate-normal parametric bootstrap. `bsp` may not be specified if there are gaps in the data.

`nodots`, `reps(#)`, and `bsaving(filename[, replace])` are relevant only if `bs` or `bsp` is specified.

`nodots` specifies that dots not be displayed each time `irf create` performs a bootstrap replication.

`reps(#)`, `# > 50`, specifies the number of bootstrap replications to be performed. `reps(200)` is the default.

`bsaving(filename[, replace])` specifies that file *filename* be created and that the bootstrap replications be saved in it. New file *filename* is just a `.dta` dataset than can be loaded later using `use`. If *filename* is specified without an extension, `.dta` is assumed.

Remarks

If you have not read [TS] [irf](#), please do so. An introductory example using IRFs is presented there.

Remarks are presented under the following headings:

Introductory examples

Technical aspects of IRF files

IRFs and FEVDs

IRF results for VARs

An introduction to impulse-response functions for VARs

An introduction to dynamic-multiplier functions for VARs

An introduction to forecast-error variance decompositions for VARs

IRF results for VECMs

An introduction to impulse-response functions for VECMs

An introduction to forecast-error variance decompositions for VECMs

Introductory examples

► Example 1: After var

Below we compare bootstrap and asymptotic standard errors for a specific FEVD. We begin by fitting a VAR(2) model to the Lütkepohl data (we use the `var` command). We next use the `irf create` command twice, first to create results with asymptotic standard errors (saved under the name `asympt`) and then to recreate the same results, this time with bootstrap standard errors (saved under the name `bs`). Because bootstrapping is a random process, we set the random-number seed (`set seed 123456`) before using `irf create` the second time; this makes our results reproducible. Finally, we compare results by using the IRF analysis command `irf ctable`.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr>=tq(1961q2) & qtr<=tq(1978q4), lags(1/2)
(output omitted)
. irf create asymp, step(8) set(results1)
(file results1.irf created)
(file results1.irf now active)
(file results1.irf updated)
. set seed 123456
. irf create bs, step(8) bs reps(250) nodots
(file results1.irf updated)
. irf ctable (asympt dln_inc dln_consump fevd)
> (bs dln_inc dln_consump fevd), noci stderr
```

step	(1) fevd	(1) S.E.	(2) fevd	(2) S.E.
0	0	0	0	0
1	.282135	.087373	.282135	.104073
2	.278777	.083782	.278777	.096954
3	.33855	.090006	.33855	.100452
4	.339942	.089207	.339942	.099085
5	.342813	.090494	.342813	.099326
6	.343119	.090517	.343119	.09934
7	.343079	.090499	.343079	.099325
8	.34315	.090569	.34315	.099368

(1) `irfname = asymp`, `impulse = dln_inc`, and `response = dln_consump`

(2) `irfname = bs`, `impulse = dln_inc`, and `response = dln_consump`

Point estimates are, of course, the same. The bootstrap estimates of the standard errors, however, are larger than the asymptotic estimates, which suggests that the sample size of 71 is not large enough for the distribution of the estimator of the FEVD to be well approximated by the asymptotic distribution. Here we would expect the bootstrap confidence interval to be more reliable than the confidence interval that is based on the asymptotic standard error.

◀

□ Technical note

The details of the bootstrap algorithms are given in [Methods and formulas](#). These algorithms are conditional on the first p observations, where p is the order of the fitted VAR. (In an SVAR model, p is the order of the VAR that underlies the SVAR.) The bootstrapped estimates are conditional on the first p observations, just as the estimators of the coefficients in VAR models are conditional on the first p observations. With bootstrap standard errors (option `bs`), the p initial observations are used with resampling the residuals to produce the bootstrap samples used for estimation. With the more parametric bootstrap (option `bsp`), the p initial observations are used with draws from a multivariate-normal distribution with variance–covariance matrix $\hat{\Sigma}$ to generate the bootstrap samples.

□

□ Technical note

For `var` and `svar` `e()` results, `irf` uses $\hat{\Sigma}$, the estimated variance matrix of the disturbances, in computing the asymptotic standard errors of all the functions. The point estimates of the orthogonalized impulse–response functions, the structural impulse–response functions, and all the variance decompositions also depend on $\hat{\Sigma}$. As discussed in [\[TS\]](#) `var`, `var` and `svar` use the ML estimator of this matrix by default, but they have option `dfk`, which will instead use an estimator that includes a small-sample correction. Specifying `dfk` when the model is fit—when the `var` or `svar` command is given—changes the estimate of $\hat{\Sigma}$ and will change the IRF results that depend on it.

□

▷ Example 2: After var with exogenous variables

After fitting a VAR, `irf create` computes estimates of the dynamic multipliers, which describe the impact of a unit change in an exogenous variable on each endogenous variable. For instance, below we estimate and report the cumulative dynamic multipliers from a model in which changes in investment are exogenous. The results indicate that both of the cumulative dynamic multipliers are significant.

```
. var dln_inc dln_consump if qtr>=tq(1961q2) & qtr<=tq(1978q4), lags(1/2)
> exog(L(0/2).dln_inv)
(output omitted)
. irf create dm, step(8)
(file results1.irf updated)
```

```
. irf table cdm, impulse(dln_inv) irf(dm)
Results from dm
```

step	(1) cdm	(1) Lower	(1) Upper
0	.032164	-.027215	.091544
1	.096568	.003479	.189656
2	.140107	.022897	.257317
3	.150527	.032116	.268938
4	.148979	.031939	.26602
5	.151247	.033011	.269482
6	.150267	.033202	.267331
7	.150336	.032858	.267813
8	.150525	.033103	.267948

step	(2) cdm	(2) Lower	(2) Upper
0	.058681	.012529	.104832
1	.062723	-.005058	.130504
2	.126167	.032497	.219837
3	.136583	.038691	.234476
4	.146482	.04442	.248543
5	.146075	.045201	.24695
6	.145542	.044988	.246096
7	.146309	.045315	.247304
8	.145786	.045206	.246365

95% lower and upper bounds reported

(1) irfname = dm, impulse = dln_inv, and response = dln_inc

(2) irfname = dm, impulse = dln_inv, and response = dln_consump



► Example 3: After vec

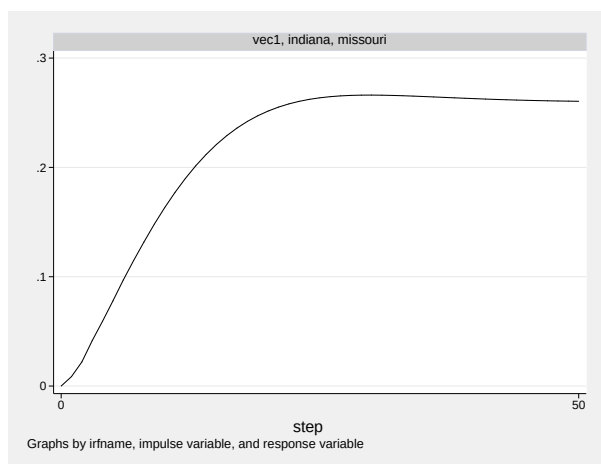
Although all IRFs and orthogonalized IRFs (OIRFs) from models with stationary variables will taper off to zero, some of the IRFs and OIRFs from models with first-difference stationary variables will not. This is the key difference between IRFs and OIRFs from systems of stationary variables fit by `var` or `svar` and those obtained from systems of first-difference stationary variables fit by `vec`. When the effect of the innovations dies out over time, the shocks are said to be transitory. In contrast, when the effect does not taper off, shocks are said to be permanent.

In this example, we look at the OIRF from one of the VECMs fit to the unemployment-rate data analyzed in [example 2](#) of [\[TS\]](#) `vec`. We see that an orthogonalized shock to Indiana has a permanent effect on the unemployment rate in Missouri:

```
. use http://www.stata-press.com/data/r11/urates
. vec missouri indiana kentucky illinois, trend(rconstant) rank(2) lags(4)
(output omitted)
. irf create vec1, set(vecirfs) step(50)
(file vecirfs.irf created)
(file vecirfs.irf now active)
(file vecirfs.irf updated)
```

Now we can use `irf graph` to graph the OIRF of interest:

```
. irf graph oirf, impulse(indiana) response(missouri)
```



The graph shows that the estimated OIRF converges to a positive asymptote, which indicates that an orthogonalized innovation to the unemployment rate in Indiana has a permanent effect on the unemployment rate in Missouri.

◀

Technical aspects of IRF files

This section is included for programmers wishing to extend the `irf` system.

`irf create` estimates a series of impulse–response functions and their standard errors. Although these estimates are stored in an IRF file, most users will never need to look at the contents of this file. The IRF commands `fill in`, `analyze`, `present`, and `manage` IRF results.

IRF files are just Stata datasets that have names ending in `.irf` instead of `.dta`. The dataset in the file has a nested panel structure.

Variable `irfname` contains the *irfname* specified by the user. Variable `impulse` records the name of the endogenous variable whose innovations are the impulse. Variable `response` records the name of the endogenous variable that is responding to the innovations. In a model with K endogenous variables, there are K^2 combinations of `impulse` and `response`. Variable `step` records the periods for which these estimates were computed.

Below is a catalog of the statistics that `irf create` estimates and the variable names under which they are stored in the IRF file.

Statistic	Name
impulse–response functions	<code>irf</code>
orthogonalized impulse–response functions	<code>oirf</code>
dynamic-multiplier functions	<code>dm</code>
cumulative impulse–response functions	<code>cirf</code>
cumulative orthogonalized impulse–response functions	<code>coirf</code>
cumulative dynamic-multiplier functions	<code>cdm</code>
Cholesky forecast-error decomposition	<code>fevd</code>
structural impulse–response functions	<code>sirf</code>
structural forecast-error decomposition	<code>sfevd</code>
standard error of the impulse–response functions	<code>stdirf</code>
standard error of the orthogonalized impulse–response functions	<code>stdoirf</code>
standard error of the cumulative impulse–response functions	<code>stdcirf</code>
standard error of the cumulative orthogonalized impulse–response functions	<code>stdcoirf</code>
standard error of the Cholesky forecast-error decomposition	<code>stdfevd</code>
standard error of the structural impulse–response functions	<code>stdsirf</code>
standard error of the structural forecast-error decomposition	<code>stdsfevd</code>

In addition to the variables, information is stored in `_dta` characteristics. Much of the following information is also available in `r()` after `irf describe`, where it is often more convenient to obtain the information. Characteristic `_dta[version]` contains the version number of the IRF file, which is currently 1.1. Characteristic `_dta[irfnames]` contains a list of all the *irfnames* in the IRF file. For each *irfname*, there are a series of additional characteristics:

Name	Contents
<code>_dta[irfname_model]</code>	<code>var</code> , <code>sr var</code> , <code>lr var</code> , or <code>vec</code>
<code>_dta[irfname_order]</code>	Cholesky order used in IRF estimates
<code>_dta[irfname_exog]</code>	exogenous variables, and their lags, in VAR
<code>_dta[irfname_exogvars]</code>	exogenous variables in VAR
<code>_dta[irfname_constant]</code>	<code>constant</code> or <code>noconstant</code> , depending on whether <code>noconstant</code> was specified in <code>var</code> or <code>svar</code>
<code>_dta[irfname_lags]</code>	lags in model
<code>_dta[irfname_exlags]</code>	lags of exogenous variables in model
<code>_dta[irfname_tmin]</code>	minimum value of <code>timevar</code> in the estimation sample
<code>_dta[irfname_tmax]</code>	maximum value of <code>timevar</code> in the estimation sample
<code>_dta[irfname_timevar]</code>	name of <code>tsset</code> <code>timevar</code>
<code>_dta[irfname_tsfmt]</code>	format of <code>timevar</code>
<code>_dta[irfname_varcns]</code>	<code>constrained</code> or colon-separated list of constraints placed on VAR coefficients
<code>_dta[irfname_svarcns]</code>	<code>constrained</code> or colon-separated list of constraints placed on VAR coefficients
<code>_dta[irfname_step]</code>	maximum step in IRF estimates
<code>_dta[irfname_stderror]</code>	<code>asymptotic</code> , <code>bs</code> , <code>bsp</code> , or <code>none</code> , depending on the type of standard errors requested
<code>_dta[irfname_reps]</code>	number of bootstrap replications performed
<code>_dta[irfname_version]</code>	version of the IRF file that originally held <i>irfname</i> IRF results
<code>_dta[irfname_rank]</code>	number of cointegrating equations
<code>_dta[irfname_trend]</code>	<code>trend()</code> specified in <code>vec</code>
<code>_dta[irfname_veccns]</code>	constraints placed on VECM parameters
<code>_dta[irfname_sind]</code>	normalized seasonal indicators included in <code>vec</code>

IRFs and FEVDs

`irf create` can estimate several types of IRFs and FEVDs for VARs and VECMs. We first discuss IRF results for VAR and SVAR models, and then we discuss them in the context of VECMs. Because the cointegrating VECM is an extension of the stationary VAR framework, the section that discusses the IRF results for VECMs draws on the earlier VAR material.

IRF results for VARs

An introduction to impulse–response functions for VARs

A p th-order vector autoregressive model (VAR) with exogenous variables is given by

$$\mathbf{y}_t = \mathbf{v} + \mathbf{A}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{A}_p \mathbf{y}_{t-p} + \mathbf{B} \mathbf{x}_t + \mathbf{u}_t$$

where

$\mathbf{y}_t = (y_{1t}, \dots, y_{Kt})'$ is a $K \times 1$ random vector,
the $\mathbf{A}_i$ are fixed $K \times K$ matrices of parameters,
 $\mathbf{x}_t$ is an $R_0 \times 1$ vector of exogenous variables,
 $\mathbf{B}$ is a $K \times R_0$ matrix of coefficients,
 $\mathbf{v}$ is a $K \times 1$ vector of fixed parameters, and
 $\mathbf{u}_t$ is assumed to be white noise; that is,
 $E(\mathbf{u}_t) = \mathbf{0}$
 $E(\mathbf{u}_t \mathbf{u}_t') = \Sigma$
 $E(\mathbf{u}_t \mathbf{u}_s') = \mathbf{0}$ for $t \neq s$

As discussed in [TS] [varstable](#), a VAR can be rewritten in moving-average form only if it is stable. Any exogenous variables are assumed to be covariance stationary. Because the functions of interest in this section depend only on the exogenous variables through their effect on the estimated $\mathbf{A}_i$, we can simplify the notation by dropping them from the analysis. All the formulas given below still apply, although the $\mathbf{A}_i$ are estimated jointly with $\mathbf{B}$ on the exogenous variables.

Below we discuss conditions under which the IRFs and forecast-error variance decompositions have a causal interpretation. Although estimation requires only that the exogenous variables be predetermined, i.e., that $E[\mathbf{x}_{jt} u_{it}] = 0$ for all i, j , and t , assigning a causal interpretation to IRFs and FEVDs requires that the exogenous variables be strictly exogenous, i.e., that $E[\mathbf{x}_{js} u_{it}] = 0$ for all i, j, s , and t .

IRFs describe how the innovations to one variable affect another variable after a given number of periods. For an example of how IRFs are interpreted, see [Stock and Watson \(2001\)](#). They use IRFs to investigate the effect of surprise shocks to the Federal Funds rate on inflation and unemployment. In another example, [Christiano, Eichenbaum, and Evans \(1999\)](#) use IRFs to investigate how shocks to monetary policy affect other macroeconomic variables.

Consider a VAR without exogenous variables:

$$\mathbf{y}_t = \mathbf{v} + \mathbf{A}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{A}_p \mathbf{y}_{t-p} + \mathbf{u}_t \tag{1}$$

The VAR represents the variables in $\mathbf{y}_t$ as functions of its own lags and serially uncorrelated innovations $\mathbf{u}_t$. All the information about contemporaneous correlations among the K variables in $\mathbf{y}_t$ is contained in Σ . In fact, as discussed in [TS] [var svar](#), a VAR can be viewed as the reduced form of a dynamic simultaneous-equation model.

To see how the innovations affect the variables in $\mathbf{y}_t$ after, say, i periods, rewrite the model in its moving-average form

$$\mathbf{y}_t = \boldsymbol{\mu} + \sum_{i=0}^{\infty} \boldsymbol{\Phi}_i \mathbf{u}_{t-i} \quad (2)$$

where $\boldsymbol{\mu}$ is the $K \times 1$ time-invariant mean of $\mathbf{y}_t$, and

$$\boldsymbol{\Phi}_i = \begin{cases} \mathbf{I}_K & \text{if } i = 0 \\ \sum_{j=1}^i \boldsymbol{\Phi}_{i-j} \mathbf{A}_j & \text{if } i = 1, 2, \dots \end{cases}$$

We can rewrite a VAR in the moving-average form only if it is stable. Essentially, a VAR is stable if the variables are covariance stationary and none of the autocorrelations are too high (the issue of stability is discussed in greater detail in [TS] [varstable](#)).

The $\boldsymbol{\Phi}_i$ are the simple IRFs. The j, k element of $\boldsymbol{\Phi}_i$ gives the effect of a 1-time unit increase in the k th element of $\mathbf{u}_t$ on the j th element of $\mathbf{y}_t$ after i periods, holding everything else constant. Unfortunately, these effects have no causal interpretation, which would require us to be able to answer the question, “How does an innovation to variable k , holding everything else constant, affect variable j after i periods?” Because the $\mathbf{u}_t$ are contemporaneously correlated, we cannot assume that everything else is held constant. Contemporaneous correlation among the $\mathbf{u}_t$ implies that a shock to one variable is likely to be accompanied by shocks to some of the other variables, so it does not make sense to shock one variable and hold everything else constant. For this reason, (2) cannot provide a causal interpretation.

This shortcoming may be overcome by rewriting (2) in terms of mutually uncorrelated innovations. Suppose that we had a matrix $\mathbf{P}$, such that $\boldsymbol{\Sigma} = \mathbf{P}\mathbf{P}'$. If we had such a $\mathbf{P}$, then $\mathbf{P}^{-1}\boldsymbol{\Sigma}\mathbf{P}'^{-1} = \mathbf{I}_K$, and

$$E\{\mathbf{P}^{-1}\mathbf{u}_t(\mathbf{P}^{-1}\mathbf{u}_t)'\} = \mathbf{P}^{-1}E\{(\mathbf{u}_t\mathbf{u}_t')\mathbf{P}'^{-1}\} = \mathbf{P}^{-1}\boldsymbol{\Sigma}\mathbf{P}'^{-1} = \mathbf{I}_K$$

We can thus use $\mathbf{P}^{-1}$ to orthogonalize the $\mathbf{u}_t$ and rewrite (2) as

$$\begin{aligned} \mathbf{y}_t &= \boldsymbol{\mu} + \sum_{i=0}^{\infty} \boldsymbol{\Phi}_i \mathbf{P}\mathbf{P}^{-1}\mathbf{u}_{t-i} \\ &= \boldsymbol{\mu} + \sum_{i=0}^{\infty} \boldsymbol{\Theta}_i \mathbf{P}^{-1}\mathbf{u}_{t-i} \\ &= \boldsymbol{\mu} + \sum_{i=0}^{\infty} \boldsymbol{\Theta}_i \mathbf{w}_{t-i} \end{aligned}$$

where $\boldsymbol{\Theta}_i = \boldsymbol{\Phi}_i \mathbf{P}$ and $\mathbf{w}_t = \mathbf{P}^{-1}\mathbf{u}_t$. If we had such a $\mathbf{P}$, the $\mathbf{w}_k$ would be mutually orthogonal, and no information would be lost in the holding-everything-else-constant assumption, implying that the $\boldsymbol{\Theta}_i$ would have the causal interpretation that we seek.

Choosing a $\mathbf{P}$ is similar to placing identification restrictions on a system of dynamic simultaneous equations. The simple IRFs do not identify the causal relationships that we wish to analyze. Thus we seek at least as many identification restrictions as necessary to identify the causal IRFs.

So, where do we get such a $\mathbf{P}$? Sims (1980) popularized the method of choosing $\mathbf{P}$ to be the Cholesky decomposition of $\widehat{\boldsymbol{\Sigma}}$. The IRFs based on this choice of $\mathbf{P}$ are known as the *orthogonalized IRFs*. Choosing $\mathbf{P}$ to be the Cholesky decomposition of $\widehat{\boldsymbol{\Sigma}}$ is equivalent to imposing a recursive structure for the corresponding dynamic structural equation model. The ordering of the recursive structure is the same as the ordering imposed in the Cholesky decomposition. Because this choice is arbitrary, some researchers will look at the OIRFs with different orderings assumed in the Cholesky decomposition. The `order()` option available with `irf create` facilitates this type of analysis.

The SVAR approach integrates the need to identify the causal IRFs into the model specification and estimation process. Sufficient identification restrictions can be obtained by placing either short-run or long-run restrictions on the model. The VAR in (1) can be rewritten as

$$\mathbf{y}_t - \mathbf{v} - \mathbf{A}_1 \mathbf{y}_{t-1} - \cdots - \mathbf{A}_p \mathbf{y}_{t-p} = \mathbf{u}_t \quad (3)$$

Similarly, a short-run SVAR model can be written as

$$\mathbf{A}(\mathbf{y}_t - \mathbf{v} - \mathbf{A}_1 \mathbf{y}_{t-1} - \cdots - \mathbf{A}_p \mathbf{y}_{t-p}) = \mathbf{A} \mathbf{u}_t = \mathbf{B} \mathbf{e}_t \quad (4)$$

where $\mathbf{A}$ and $\mathbf{B}$ are $K \times K$ nonsingular matrices of parameters to be estimated, $\mathbf{e}_t$ is a $K \times 1$ vector of disturbances with $\mathbf{e}_t \sim N(\mathbf{0}, \mathbf{I}_K)$, and $E[\mathbf{e}_t \mathbf{e}_s'] = \mathbf{0}_K$ for all $s \neq t$. Sufficient constraints must be placed on $\mathbf{A}$ and $\mathbf{B}$ so that $\mathbf{P}$ is identified. One way to see the connection is to draw out the implications of the latter equality in (4). From (4) it can be shown that

$$\boldsymbol{\Sigma} = \mathbf{A}^{-1} \mathbf{B} (\mathbf{A}^{-1} \mathbf{B})'$$

As discussed in [TS] [var svar](#), the estimates $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are obtained by maximizing the concentrated log-likelihood function on the basis of the $\hat{\boldsymbol{\Sigma}}$ obtained from the underlying VAR. The short-run SVAR approach chooses $\mathbf{P} = \hat{\mathbf{A}}^{-1} \hat{\mathbf{B}}$ to identify the causal IRFs. The long-run SVAR approach works similarly, with $\mathbf{P} = \hat{\mathbf{C}} = \hat{\mathbf{A}}^{-1} \hat{\mathbf{B}}$, where $\hat{\mathbf{A}}^{-1}$ is the matrix of estimated long-run or accumulated effects of the reduced-form VAR shocks.

There is one important difference between long-run and short-run SVAR models. As discussed by [Amisano and Giannini \(1997, chap. 6\)](#), in the short-run model the constraints are applied directly to the parameters in $\mathbf{A}$ and $\mathbf{B}$. Then $\mathbf{A}$ and $\mathbf{B}$ interact with the estimated parameters of the underlying VAR. In contrast, in a long-run model, the constraints are placed on functions of the estimated VAR parameters. Although estimation and inference of the parameters in $\mathbf{C}$ is straightforward, obtaining the asymptotic standard errors of the structural IRFs requires untenable assumptions. For this reason, `irf create` does not estimate the asymptotic standard errors of the structural IRFs generated by long-run SVAR models. However, bootstrap standard errors are still available.

An introduction to dynamic-multiplier functions for VARs

A dynamic-multiplier function measures the effect of a unit change in an exogenous variable on the endogenous variables over time. Per [Lütkepohl \(2005, chap. 10\)](#), if the VAR with exogenous variables is stable, it can be rewritten as

$$\mathbf{y}_t = \sum_{i=0}^{\infty} \mathbf{D}_i x_{t-i} + \sum_{i=0}^{\infty} \boldsymbol{\Phi}_i u_{t-i}$$

where the $\mathbf{D}_i$ are the dynamic-multiplier functions. (See [Methods and formulas](#) for details.) Some authors refer to the dynamic-multiplier functions as transfer functions because they specify how a unit change in an exogenous variable is “transferred” to the endogenous variables.

□ Technical note

`irf create` computes dynamic-multiplier functions only after `var`. After short-run SVAR models, the dynamic multipliers from the VAR are the same as those from the SVAR. The dynamic multipliers for long-run SVARs have not yet been worked out.

□

An introduction to forecast-error variance decompositions for VARs

Another measure of the effect of the innovations in variable k on variable j is the FEVD. This method, which is also known as *innovation accounting*, measures the fraction of the error in forecasting variable j after h periods that is attributable to the orthogonalized innovations in variable k . Because deriving the FEVD requires orthogonalizing the $\mathbf{u}_t$ innovations, the FEVD is always predicated upon a choice of $\mathbf{P}$.

Lütkepohl (2005, sec. 2.2.2) shows that the h -step forecast error can be written as

$$\mathbf{y}_{t+h} - \hat{\mathbf{y}}_t(h) = \sum_{i=0}^{h-1} \Phi_i \mathbf{u}_{t+h-i} \quad (6)$$

where y_{t+h} is the value observed at time $t+h$ and $\hat{y}_t(h)$ is the h -step-ahead predicted value for y_{t+h} that was made at time t .

Because the $\mathbf{u}_t$ are contemporaneously correlated, their distinct contributions to the forecast error cannot be ascertained. However, if we choose a $\mathbf{P}$ such that $\Sigma = \mathbf{P}\mathbf{P}'$, as above, we can orthogonalize the $\mathbf{u}_t$ into $\mathbf{w}_t = \mathbf{P}^{-1}\mathbf{u}_t$. We can then ascertain the relative contribution of the distinct elements of $\mathbf{w}_t$. Thus we can rewrite (6) as

$$\begin{aligned} \mathbf{y}_{t+h} - \hat{\mathbf{y}}_t(h) &= \sum_{i=0}^{h-1} \Phi_i \mathbf{P}\mathbf{P}^{-1} \mathbf{u}_{t+h-i} \\ &= \sum_{i=0}^{h-1} \Theta_i \mathbf{w}_{t+h-i} \end{aligned} \quad (7)$$

Because the forecast errors can be written in terms of the orthogonalized errors, the forecast-error variance can be written in terms of the orthogonalized error variances. Forecast-error variance decompositions measure the fraction of the total forecast-error variance that is attributable to each orthogonalized shock.

□ Technical note

The details in this note are not critical to the discussion that follows. A forecast-error variance decomposition is derived for a given $\mathbf{P}$. Per Lütkepohl (2005, sec. 2.3.3), letting $\theta_{mn,i}$ be the m , n th element of Θ_i , we can express the h -step forecast error of the j th component of $\mathbf{y}_t$ as

$$\begin{aligned} y_{j,t+h} - \hat{y}_j(h) &= \sum_{i=0}^{h-1} \theta_{j1,1} \mathbf{w}_{1,t+h-i} + \cdots + \theta_{jK,i} \mathbf{w}_{K,t+h-i} \\ &= \sum_{k=1}^K \theta_{jk,0} \mathbf{w}_{k,t+h} + \cdots + \theta_{jk,h-1} \mathbf{w}_{k,t+1} \end{aligned}$$

The $\mathbf{w}_t$, which were constructed using $\mathbf{P}$, are mutually orthogonal with unit variance. This allows us to compute easily the mean squared error (MSE) of the forecast of variable j at horizon h in terms of the contributions of the components of $\mathbf{w}_t$. Specifically,

$$E[\{y_{j,t+h} - y_{j,t}(h)\}^2] = \sum_{k=1}^K (\theta_{jk,0}^2 + \cdots + \theta_{jk,h-1}^2)$$

The k th term in the sum above is interpreted as the contribution of the orthogonalized innovations in variable k to the h -step forecast error of variable j . Note that the k th element in the sum above can be rewritten as

$$(\theta_{jk,0}^2 + \cdots + \theta_{jk,h-1}^2) = \sum_{i=0}^{h-1} (\mathbf{e}_j' \boldsymbol{\Theta}_i \mathbf{e}_k)^2$$

where $\mathbf{e}_i$ is the i th column of $\mathbf{I}_K$. Normalizing by the forecast error for variable j at horizon h yields

$$\omega_{jk,h} = \frac{\sum_{i=0}^{h-1} (\mathbf{e}_j' \boldsymbol{\Theta}_i \mathbf{e}_k)^2}{\text{MSE}[y_{j,t}(h)]}$$

where $\text{MSE}[y_{j,t}(h)] = \sum_{i=0}^{h-1} \sum_{k=1}^K \theta_{jk,i}^2$.

□

Because the FEVD depends on the choice of $\mathbf{P}$, there are different forecast-error variance decompositions associated with each distinct $\mathbf{P}$. `irf create` can estimate the FEVD for a VAR or an SVAR. For a VAR, $\mathbf{P}$ is the Cholesky decomposition of $\hat{\boldsymbol{\Sigma}}$. For an SVAR, $\mathbf{P}$ is the estimated structural decomposition, $\mathbf{P} = \hat{\mathbf{A}}^{-1} \hat{\mathbf{B}}$ for short-run models and $\mathbf{P} = \hat{\mathbf{C}}$ for long-run SVAR models. Due to the same complications that arose with the structural impulse–response functions, the asymptotic standard errors of the structural FEVD are not available after long-run SVAR models, but bootstrap standard errors are still available.

IRF results for VECMs

An introduction to impulse–response functions for VECMs

As discussed in [TS] [vec intro](#), the VECM is a reparameterization of the VAR that is especially useful for fitting VARs with cointegrating variables. This implies that the estimated parameters for the corresponding VAR model can be backed out from the estimated parameters of the VECM model. This relationship means we can use the VAR form of the cointegrating VECM to discuss the IRFs for VECMs.

Consider a cointegrating VAR with one lag with no constant or trend,

$$\mathbf{y}_t = \mathbf{A} \mathbf{y}_{t-1} + \mathbf{u}_t \quad (8)$$

where $\mathbf{y}_t$ is a $K \times 1$ vector of endogenous, first-difference stationary variables among which there are $1 \leq r < K$ cointegration equations; $\mathbf{A}$ is $K \times K$ matrix of parameters; and $\mathbf{u}_t$ is a $K \times 1$ vector of i.i.d. disturbances.

We developed intuition for the IRFs from a stationary VAR by rewriting the VAR as an infinite-order vector moving-average (VMA) process. While the Granger representation theorem establishes the existence of a VMA formulation of this model, because the cointegrating VAR is not stable, the inversion is not nearly so intuitive. (See [Johansen \[1995, chapters 3 and 4\]](#) for more details.) For this reason, we use (8) to develop intuition for the IRFs from a cointegrating VAR.

Suppose that K is 3, that $\mathbf{u}_1 = (1, 0, 0)$, and that we want to analyze the time paths of the variables in $\mathbf{y}$ conditional on the initial values $\mathbf{y}_0 = \mathbf{0}$, $\mathbf{A}$, and the condition that there are no more shocks to the system, i.e., $\mathbf{0} = \mathbf{u}_2 = \mathbf{u}_3 = \dots$. These assumptions and (8) imply that

$$\mathbf{y}_1 = \mathbf{u}_1$$

$$\mathbf{y}_2 = \mathbf{A}\mathbf{y}_1 = \mathbf{A}\mathbf{u}_1$$

$$\mathbf{y}_3 = \mathbf{A}\mathbf{y}_2 = \mathbf{A}^2\mathbf{u}_1$$

and so on. The i th-row element of the first column of $\mathbf{A}^s$ contains the effect of the unit shock to the first variable after s periods. The first column of $\mathbf{A}^s$ contains the IRF of a unit impulse to the first variable after s periods. We could deduce the IRFs of a unit impulse to any of the other variables by administering the unit shock to one of them instead of to the first variable. Thus we can see that the (i, j) th element of $\mathbf{A}^s$ contains the unit IRF from variable j to variable i after s periods. By starting with orthogonalized shocks of the form $\mathbf{P}^{-1}\mathbf{u}_t$, we can use the same logic to derive the OIRFs to be $\mathbf{A}^s\mathbf{P}$.

For the stationary VAR, stability implies that all the eigenvalues of $\mathbf{A}$ have moduli strictly less than one, which in turn implies that all the elements of $\mathbf{A}^s \rightarrow \mathbf{0}$ as $s \rightarrow \infty$. This implies that all the IRFs from a stationary VAR taper off to zero as $s \rightarrow \infty$. In contrast, in a cointegrating VAR, some of the eigenvalues of $\mathbf{A}$ are 1, while the remaining eigenvalues have moduli strictly less than 1. This implies that in cointegrating VARs some of the elements of $\mathbf{A}^s$ are not going to zero as $s \rightarrow \infty$, which in turn implies that some of the IRFs and OIRFs are not going to zero as $s \rightarrow \infty$. The fact that the IRFs and OIRFs taper off to zero for stationary VARs but not for cointegrating VARs is one of the key differences between the two models.

When the IRF or OIRF from the innovation in one variable to another tapers off to zero as time goes on, the innovation to the first variable is said to have a transitory effect on the second variable. When the IRF or OIRF does not go to zero, the effect is said to be permanent.

Note that, because some of the IRFs and OIRFs do not taper off to zero, some of the cumulative IRFs and OIRFs diverge over time.

An introduction to forecast-error variance decompositions for VECMs

The results from [An introduction to impulse–response functions for VECMs](#) can be used to show that the interpretation of FEVDs for a finite number of steps in cointegrating VARs is essentially the same as in the stationary case. Because the MSE of the forecast is diverging, this interpretation is valid only for a finite number of steps. (See [\[TS\] vec intro](#) and [\[TS\] fcast compute](#) for more information on this point.)

Methods and formulas

`irf create` is implemented as an ado-file.

Methods and formulas are presented under the following headings:

Impulse–response function formulas for VARs
Dynamic-multiplier function formulas for VARs
Forecast-error variance decomposition formulas for VARs
Impulse–response function formulas for VECMs
Algorithms for bootstrapping the VAR IRF and FEVD standard errors

Impulse–response function formulas for VARs

The previous discussion implies that there are three different choices of $\mathbf{P}$ that can be used to obtain distinct Θ_i . $\mathbf{P}$ is the Cholesky decomposition of Σ for the OIRFs. For the structural IRFs, $\mathbf{P} = \mathbf{A}^{-1}\mathbf{B}$ for short-run models, and $\mathbf{P} = \mathbf{C}$ for long-run models. We will distinguish between the three by defining Θ_i^o to be the OIRFs, Θ_i^{sr} to be the short-run structural IRFs, and Θ_i^{lr} to be the long-run structural IRFs.

We also define $\hat{\mathbf{P}}_c$ to be the Cholesky decomposition of $\hat{\Sigma}$, $\hat{\mathbf{P}}_{\text{sr}} = \hat{\mathbf{A}}^{-1}\hat{\mathbf{B}}$ to be the short-run structural decomposition, and $\hat{\mathbf{P}}_{\text{lr}} = \hat{\mathbf{C}}$ to be the long-run structural decomposition.

Given estimates of the $\hat{\mathbf{A}}_i$ and $\hat{\Sigma}$ from `var` or `svar`, the estimates of the simple IRFs and the OIRFs are, respectively,

$$\hat{\Phi}_i = \sum_{j=1}^i \hat{\Phi}_{i-j} \hat{\mathbf{A}}_j$$

and

$$\hat{\Theta}_i^o = \hat{\Phi}_i \hat{\mathbf{P}}_c$$

where $\hat{\mathbf{A}}_j = \mathbf{0}_K$ for $j > p$.

Given the estimates $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$, or $\hat{\mathbf{C}}$, from `svar`, the estimates of the structural IRFs are either

$$\hat{\Theta}_i^{\text{sr}} = \hat{\Phi}_i \hat{\mathbf{P}}_{\text{sr}}$$

or

$$\hat{\Theta}_i^{\text{lr}} = \hat{\Phi}_i \hat{\mathbf{P}}_{\text{lr}}$$

The estimated structural IRFs stored in an IRF file with the variable name `sirf` may be from either a short-run model or a long-run model, depending on the estimation results used to create the IRFs. As discussed in [TS] [irf describe](#), you can easily determine whether the structural IRFs were generated from a short-run or a long-run SVAR model using `irf describe`.

Following Lütkepohl (2005, sec. 3.7), estimates of the cumulative IRFs and the cumulative orthogonalized impulse–response functions (COIRFs) at period n are, respectively,

$$\hat{\Psi}_n = \sum_{i=0}^n \hat{\Phi}_i$$

and

$$\hat{\Xi}_n = \sum_{i=0}^n \hat{\Theta}_i$$

The asymptotic standard errors of the different impulse–response functions are obtained by applications of the delta method. See [Lütkepohl \(2005, sec. 3.7\)](#) and [Amisano and Giannini \(1997, chap. 4\)](#) for the derivations. See [Serfling \(1980, sec. 3.3\)](#) for a discussion of the delta method. In presenting the variance–covariance matrix estimators, we make extensive use of the `vec()` operator, where $\text{vec}(\mathbf{X})$ is the vector obtained by stacking the columns of $\mathbf{X}$.

[Lütkepohl \(2005, sec. 3.7\)](#) derives the asymptotic VCEs of $\text{vec}(\hat{\Phi}_i)$, $\text{vec}(\hat{\Theta}_i^o)$, $\text{vec}(\hat{\Psi}_n)$, and $\text{vec}(\hat{\Xi}_n)$. Because $\text{vec}(\hat{\Phi}_i)$ is $K^2 \times 1$, the asymptotic VCE of $\text{vec}(\hat{\Phi}_i)$ is $K^2 \times K^2$, and it is given by

$$\mathbf{G}_i \hat{\Sigma}_{\hat{\alpha}} \mathbf{G}_i'$$

where

$$\begin{aligned} \mathbf{G}_i &= \sum_{m=0}^{i-1} \mathbf{J}(\hat{\mathbf{M}}')^{(i-1-m)} \otimes \hat{\Phi}_m & \mathbf{G}_i \text{ is } K^2 \times K^2 p \\ \mathbf{J} &= (\mathbf{I}_K, \mathbf{0}_K, \dots, \mathbf{0}_K) & \mathbf{J} \text{ is } K \times Kp \\ \hat{\mathbf{M}} &= \begin{bmatrix} \hat{\mathbf{A}}_1 & \hat{\mathbf{A}}_2 & \dots & \hat{\mathbf{A}}_{p-1} & \hat{\mathbf{A}}_p \\ \mathbf{I}_K & \mathbf{0}_K & \dots & \mathbf{0}_K & \mathbf{0}_K \\ \mathbf{0}_K & \mathbf{I}_K & & \mathbf{0}_K & \mathbf{0}_K \\ \vdots & & \ddots & \vdots & \vdots \\ \mathbf{0}_K & \mathbf{0}_K & \dots & \mathbf{I}_K & \mathbf{0}_K \end{bmatrix} & \hat{\mathbf{M}} \text{ is } Kp \times Kp \end{aligned}$$

The $\hat{\mathbf{A}}_i$ are the estimates of the coefficients on the lagged variables in the VAR, and $\hat{\Sigma}_{\hat{\alpha}}$ is the VCE matrix of $\hat{\alpha} = \text{vec}(\hat{\mathbf{A}}_1, \dots, \hat{\mathbf{A}}_p)$. $\hat{\Sigma}_{\hat{\alpha}}$ is a $K^2 p \times K^2 p$ matrix whose elements come from the VCE of the VAR coefficient estimator. As such, this VCE is the VCE of the constrained estimator if there are any constraints placed on the VAR coefficients.

The $K^2 \times K^2$ asymptotic VCE matrix for $\text{vec}(\hat{\Psi}_n)$ after n periods is given by

$$\mathbf{F}_n \hat{\Sigma}_{\hat{\alpha}} \mathbf{F}_n'$$

where

$$\mathbf{F}_n = \sum_{i=1}^n \mathbf{G}_i$$

The $K^2 \times K^2$ asymptotic VCE matrix of the vectorized, orthogonalized, IRFs at horizon i , $\text{vec}(\hat{\Theta}_i^o)$, is

$$\mathbf{C}_i \hat{\Sigma}_{\hat{\alpha}} \mathbf{C}_i' + \overline{\mathbf{C}}_i \hat{\Sigma}_{\hat{\sigma}} \overline{\mathbf{C}}_i'$$

(Continued on next page)

where

$$\begin{array}{ll}
 \mathbf{C}_0 = \mathbf{0} & \mathbf{C}_0 \text{ is } K^2 \times K^2 p \\
 \mathbf{C}_i = (\hat{\mathbf{P}}'_c \otimes \mathbf{I}_K) \mathbf{G}_i, \quad i = 1, 2, \dots & \mathbf{C}_i \text{ is } K^2 \times K^2 p \\
 \overline{\mathbf{C}}_i = (\mathbf{I}_K \otimes \Phi_i) \mathbf{H}, \quad i = 0, 1, \dots & \overline{\mathbf{C}}_i \text{ is } K^2 \times K^2 \\
 \mathbf{H} = \mathbf{L}'_K \left\{ \mathbf{L}_K \mathbf{N}_K (\hat{\mathbf{P}}_c \otimes \mathbf{I}_K) \mathbf{L}'_K \right\}^{-1} & \mathbf{H} \text{ is } K^2 \times K \frac{(K+1)}{2} \\
 \mathbf{L}_K \text{ solves } \text{vech}(\mathbf{F}) = \mathbf{L}_K \text{vec}(\mathbf{F}) & \mathbf{L}_K \text{ is } K \frac{(K+1)}{2} \times K^2 \\
 & \text{for } \mathbf{F} \text{ } K \times K \text{ and symmetric} \\
 \mathbf{K}_K \text{ solves } \mathbf{K}_K \text{vec}(\mathbf{G}) = \text{vec}(\mathbf{G}') \text{ for any } K \times K \text{ matrix } \mathbf{G} & \mathbf{K}_K \text{ is } K^2 \times K^2 \\
 \mathbf{N}_K = \frac{1}{2} (\mathbf{I}_{K^2} + \mathbf{K}_K) & \mathbf{N}_K \text{ is } K^2 \times K^2 \\
 \hat{\Sigma}_{\hat{\sigma}} = 2\mathbf{D}_K^+ (\hat{\Sigma} \otimes \hat{\Sigma}) \mathbf{D}_K^+ & \hat{\Sigma}_{\hat{\sigma}} \text{ is } K \frac{(K+1)}{2} \times K \frac{(K+1)}{2} \\
 \mathbf{D}_K^+ = (\mathbf{D}'_K \mathbf{D}_K)^{-1} \mathbf{D}'_K & \mathbf{D}_K^+ \text{ is } K \frac{(K+1)}{2} \times K^2 \\
 \mathbf{D}_K \text{ solves } \mathbf{D}_K \text{vech}(\mathbf{F}) = \text{vec}(\mathbf{F}) \quad \text{for } \mathbf{F} \text{ } K \times K \text{ and symmetric} & \mathbf{D}_K \text{ is } K^2 \times K \frac{(K+1)}{2}
 \end{array}$$

$$\text{vech}(\mathbf{X}) = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{K1} \\ x_{22} \\ \vdots \\ x_{K2} \\ \vdots \\ x_{KK} \end{bmatrix} \quad \text{for } \mathbf{X} \text{ } K \times K \quad \text{vech}(\mathbf{X}) \text{ is } K \frac{(K+1)}{2} \times 1$$

Note that $\hat{\Sigma}_{\hat{\sigma}}$ is the VCE of $\text{vech}(\hat{\Sigma})$. More details about $\mathbf{L}_K$, $\mathbf{K}_K$, $\mathbf{D}_K$ and $\text{vech}()$ are available in [Lütkepohl \(2005, sec. A.12\)](#). Finally, as [Lütkepohl \(2005, 113–114\)](#) discusses, $\mathbf{D}_K^+$ is the Moore–Penrose inverse of $\mathbf{D}_K$.

As discussed in [Amisano and Giannini \(1997, chap. 6\)](#), the asymptotic standard errors of the structural IRFs are available for short-run SVAR models but not for long-run SVAR models. Following [Amisano and Giannini \(1997, chap. 5\)](#), the asymptotic $K^2 \times K^2$ VCE of the short-run structural IRFs after i periods, when a maximum of h periods are estimated, is the i, i block of

$$\hat{\Sigma}(h)_{ij} = \tilde{\mathbf{G}}_i \hat{\Sigma}_{\hat{\alpha}} \tilde{\mathbf{G}}'_j + \left\{ \mathbf{I}_K \otimes (\mathbf{J} \hat{\mathbf{M}}^i \mathbf{J}') \right\} \Sigma(0) \left\{ \mathbf{I}_K \otimes (\mathbf{J} \hat{\mathbf{M}}^j \mathbf{J}') \right\}'$$

where

$$\begin{aligned}
 \tilde{\mathbf{G}}_0 &= \mathbf{0}_K & \mathbf{G}_0 & \text{is } K^2 \times K^2 p \\
 \tilde{\mathbf{G}}_i &= \sum_{k=0}^{i-1} \left\{ \hat{\mathbf{P}}'_{\text{sr}} \mathbf{J} (\widehat{\mathbf{M}}')^{i-1-k} \otimes \left(\mathbf{J} \widehat{\mathbf{M}}^k \mathbf{J}' \right) \right\} & \mathbf{G}_i & \text{is } K^2 \times K^2 p \\
 \widehat{\Sigma}(0) &= \mathbf{Q}_2 \widehat{\Sigma}_W \mathbf{Q}_2' & \widehat{\Sigma}(0) & \text{is } K^2 \times K^2 \\
 \widehat{\Sigma}_W &= \mathbf{Q}_1 \widehat{\Sigma}_{AB} \mathbf{Q}_1' & \widehat{\Sigma}_W & \text{is } K^2 \times K^2 \\
 \mathbf{Q}_2 &= \hat{\mathbf{P}}'_{\text{sr}} \otimes \hat{\mathbf{P}}_{\text{sr}} & \mathbf{Q}_2 & \text{is } K^2 \times K^2 \\
 \mathbf{Q}_1 &= \left\{ (\mathbf{I}_K \otimes \widehat{\mathbf{B}}^{-1}), (-\hat{\mathbf{P}}'^{-1}_{\text{sr}} \otimes \mathbf{B}^{-1}) \right\} & \mathbf{Q}_1 & \text{is } K^2 \times 2K^2
 \end{aligned}$$

and $\widehat{\Sigma}_{AB}$ is the $2K^2 \times 2K^2$ VCE of the estimator of $\text{vec}(\mathbf{A}, \mathbf{B})$.

Dynamic-multiplier function formulas for VARs

This section provides the details of how `irf create` estimates the dynamic-multiplier functions and their asymptotic standard errors.

A p th order vector autoregressive model (VAR) with exogenous variables may be written as

$$\mathbf{y}_t = \mathbf{v} + \mathbf{A}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{A}_p \mathbf{y}_{t-p} + \mathbf{B}_0 \mathbf{x}_t + \mathbf{B}_1 \mathbf{x}_{t-1} + \cdots + \mathbf{B}_s \mathbf{x}_{t-s} + \mathbf{u}_t$$

where all the notation is the same as above except that the s $K \times R$ matrices $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_s$ are explicitly included and s is the number of lags of the R exogenous variables in the model.

Lütkepohl (2005) shows that the dynamic-multipliers $\mathbf{D}_i$ are consistently estimated by

$$\widehat{\mathbf{D}}_i = \mathbf{J}_x \tilde{\mathbf{A}}_x^i \widehat{\mathbf{B}}_x \quad i \in \{0, 1, \dots\}$$

where

$$\begin{aligned}
 \mathbf{J}_x &= (\mathbf{I}_K, \mathbf{0}_K, \dots, \mathbf{0}_K) & \mathbf{J} & \text{is } K \times (Kp + Rs) \\
 \tilde{\mathbf{A}}_x &= \begin{bmatrix} \widehat{\mathbf{M}} & \widehat{\mathbf{B}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} & \tilde{\mathbf{A}}_x & \text{is } (Kp + Rs) \times (Kp + Rs) \\
 \widehat{\mathbf{B}} &= \begin{bmatrix} \widehat{\mathbf{B}}_1 & \widehat{\mathbf{B}}_2 & \cdots & \widehat{\mathbf{B}}_s \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} & \widehat{\mathbf{B}} & \text{is } Kp \times Rs \\
 \tilde{\mathbf{I}} &= \begin{bmatrix} \mathbf{0}_R & \mathbf{0}_R & \cdots & \mathbf{0}_R & \mathbf{0}_R \\ \mathbf{I}_R & \mathbf{0}_R & \cdots & \mathbf{0}_R & \mathbf{0}_R \\ \mathbf{0}_R & \mathbf{I}_R & & \mathbf{0}_R & \mathbf{0}_R \\ \vdots & & \ddots & \vdots & \vdots \\ \mathbf{0}_R & \mathbf{0}_R & \cdots & \mathbf{I}_R & \mathbf{0}_R \end{bmatrix} & \tilde{\mathbf{I}} & \text{is } Rs \times Rs \\
 \widehat{\mathbf{B}}'_x &= [\tilde{\mathbf{B}}' \quad \ddot{\mathbf{I}}'] & \widehat{\mathbf{B}}'_x & \text{is } R \times (Kp + Rs) \\
 \tilde{\mathbf{B}}' &= [\widehat{\mathbf{B}}'_0 \quad \ddot{\mathbf{0}}' \cdots \ddot{\mathbf{0}}'] & \tilde{\mathbf{B}} & \text{is } R \times Kp \\
 \ddot{\mathbf{I}}' &= [\mathbf{I}_R \quad \mathbf{0}_R \cdots \mathbf{0}_R] & \ddot{\mathbf{I}} & \text{is } R \times Rs
 \end{aligned}$$

and $\ddot{\mathbf{0}}$ is a $K \times R$ matrix of 0s and $\tilde{\mathbf{0}}$ is a $Rs \times Kp$ matrix of 0s.

Consistent estimators of the cumulative dynamic-multiplier functions are given by

$$\bar{\mathbf{D}}_i = \sum_{j=0}^i \hat{\mathbf{D}}_j$$

Letting $\beta_x = \text{vec}[\mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_p \mathbf{B}_1 \mathbf{B}_2 \cdots \mathbf{B}_s \mathbf{B}_0]$ and letting $\Sigma_{\hat{\beta}_x}$ be the asymptotic variance-covariance estimator (VCE) of $\hat{\beta}_x$, Lütkepohl shows that an asymptotic VCE of $\hat{\mathbf{D}}_i$ is $\tilde{\mathbf{G}}_i \Sigma_{\hat{\beta}_x} \tilde{\mathbf{G}}_i'$ where

$$\tilde{\mathbf{G}}_i = \left[\sum_{j=0}^{i-1} \mathbf{B}_x' \tilde{\mathbf{A}}_x^{i-1-j} \otimes \mathbf{J}_x \tilde{\mathbf{A}}_x^j \mathbf{J}_x', \mathbf{I}_R \otimes \mathbf{J}_x \tilde{\mathbf{A}}_x^i \mathbf{J}_x \right]$$

Similarly, an asymptotic VCE of $\bar{\mathbf{D}}_i$ is $\left(\sum_{j=0}^i \tilde{\mathbf{G}}_j \right) \Sigma_{\hat{\beta}_x} \left(\sum_{j=0}^i \tilde{\mathbf{G}}_j' \right)$.

Forecast-error variance decomposition formulas for VARs

This section provides details of how `irf create` estimates the Cholesky FEVD, the structural FEVD, and their standard errors. Beginning with the Cholesky-based forecast-error decompositions, the fraction of the h -step-ahead forecast-error variance of variable j that is attributable to the Cholesky orthogonalized innovations in variable k can be estimated as

$$\hat{\omega}_{jk,h} = \frac{\sum_{i=0}^{h-1} (\mathbf{e}_j' \hat{\Theta}_i \mathbf{e}_k)^2}{\widehat{\text{MSE}}_j(h)}$$

where $\text{MSE}_j(h)$ is the j th diagonal element of

$$\sum_{i=0}^{h-1} \hat{\Phi}_i \hat{\Sigma} \hat{\Phi}_i'$$

(See Lütkepohl [2005, 109] for a discussion of this result.) $\hat{\omega}_{jk,h}$ and $\text{MSE}_j(h)$ are scalars. The square of the standard error of $\hat{\omega}_{jk,h}$ is

$$\mathbf{d}_{jk,h} \hat{\Sigma}_\alpha \mathbf{d}_{jk,h}' + \bar{\mathbf{d}}_{jk,h} \hat{\Sigma}_\sigma \bar{\mathbf{d}}_{jk,h}$$

where

$$\begin{aligned} \mathbf{d}_{jk,h} &= \frac{2}{\text{MSE}_j(h)^2} \sum_{i=0}^{h-1} \left\{ \text{MSE}_j(h) (\mathbf{e}_j' \hat{\Phi}_i \hat{\mathbf{P}}_c \mathbf{e}_k) (\mathbf{e}_k' \hat{\mathbf{P}}_c' \otimes \mathbf{e}_j') \mathbf{G}_i \right. \\ &\quad \left. - (\mathbf{e}_j' \hat{\Phi}_i \hat{\mathbf{P}}_c \mathbf{e}_k)^2 \sum_{m=0}^{h-1} (\mathbf{e}_j' \hat{\Phi}_m \hat{\Sigma} \otimes \mathbf{e}_j') \mathbf{G}_m \right\} \quad \mathbf{d}_{jk,h} \text{ is } 1 \times K^2 p \\ \bar{\mathbf{d}}_{jk,h} &= \sum_{i=0}^{h-1} \left\{ \text{MSE}_j(h) (\mathbf{e}_j' \hat{\Phi}_i \mathbf{P}_c \mathbf{e}_k) (\mathbf{e}_k' \otimes \mathbf{e}_j' \hat{\Phi}_i) \mathbf{H} \right. \\ &\quad \left. - (\mathbf{e}_j' \hat{\Phi}_i \hat{\mathbf{P}}_c \mathbf{e}_k)^2 \sum_{m=0}^{h-1} (\mathbf{e}_j' \hat{\Phi}_m \otimes \mathbf{e}_j' \hat{\Phi}_m) \mathbf{D}_K \right\} \frac{1}{\text{MSE}_j(h)^2} \quad \bar{\mathbf{d}}_{jk,h} \text{ is } 1 \times K \frac{(K+1)}{2} \\ \mathbf{G}_0 &= \mathbf{0} \quad \mathbf{G}_0 \text{ is } K^2 \times K^2 p \end{aligned}$$

and $\mathbf{D}_K$ is the $K^2 \times K\{(K+1)/2\}$ duplication matrix defined previously.

For the structural forecast-error decompositions, we follow [Amisano and Giannini \(1997, sec. 5.2\)](#). They define the matrix of structural forecast-error decompositions at horizon s , when a maximum of h periods are estimated, as

$$\begin{aligned}\widehat{\mathbf{W}}_s &= \widehat{\mathbf{F}}_s^{-1} \widehat{\mathbf{M}}_s \quad \text{for } s = 1, \dots, h+1 \\ \widehat{\mathbf{F}}_s &= \left(\sum_{i=0}^{s-1} \widehat{\boldsymbol{\Theta}}_i^{\text{sr}} \widehat{\boldsymbol{\Theta}}_i^{\text{sr}'} \right) \odot \mathbf{I}_K \\ \widehat{\mathbf{M}}_s &= \sum_{i=0}^{s-1} \widehat{\boldsymbol{\Theta}}_i^{\text{sr}} \odot \widehat{\boldsymbol{\Theta}}_i^{\text{sr}}\end{aligned}$$

where $\odot$ is the Hadamard, or element-by-element, product.

The $K^2 \times K^2$ asymptotic VCE of $\text{vec}(\widehat{\mathbf{W}}_s)$ is given by

$$\widetilde{\mathbf{Z}}_s \boldsymbol{\Sigma}(h) \widetilde{\mathbf{Z}}_s'$$

where $\widehat{\boldsymbol{\Sigma}}(h)$ is as derived previously, and

$$\begin{aligned}\widetilde{\mathbf{Z}}_s &= \left\{ \frac{\partial \text{vec}(\widehat{\mathbf{W}}_s)}{\partial \text{vec}(\widehat{\boldsymbol{\Theta}}_0^{\text{sr}})}, \frac{\partial \text{vec}(\widehat{\mathbf{W}}_s)}{\partial \text{vec}(\widehat{\boldsymbol{\Theta}}_1^{\text{sr}})}, \dots, \frac{\partial \text{vec}(\widehat{\mathbf{W}}_s)}{\partial \text{vec}(\widehat{\boldsymbol{\Theta}}_h^{\text{sr}})} \right\} \\ \frac{\partial \text{vec}(\widehat{\mathbf{W}}_s)}{\partial \text{vec}(\widehat{\boldsymbol{\Theta}}_j^{\text{sr}})} &= 2 \left\{ (\mathbf{I}_K \otimes \widehat{\mathbf{F}}_s^{-1}) \widetilde{\mathbf{D}}(\widehat{\boldsymbol{\Theta}}_j^{\text{sr}}) - (\widehat{\mathbf{W}}_s' \otimes \widehat{\mathbf{F}}_s^{-1}) \widetilde{\mathbf{D}}(\mathbf{I}_K) \mathbf{N}_K (\widehat{\boldsymbol{\Theta}}_j^{\text{sr}} \otimes \mathbf{I}_K) \right\}\end{aligned}$$

If $\mathbf{X}$ is an $n \times n$ matrix, then $\widetilde{\mathbf{D}}(\mathbf{X})$ is the $n^2 \times n^2$ matrix with $\text{vec}(\mathbf{X})$ on the diagonal and zeros in all the off-diagonal elements, and $\mathbf{N}_K$ is as defined previously.

Impulse–response function formulas for VECMs

We begin by providing the formulas for backing out the estimates of the $\mathbf{A}_i$ from the $\boldsymbol{\Gamma}_i$ estimated by `vec`. As discussed in [\[TS\] vec intro](#), the VAR in (1) can be rewritten as a VECM:

$$\Delta y_t = v + \boldsymbol{\Pi} y_{t-1} + \boldsymbol{\Gamma}_1 \Delta y_{t-1} + \boldsymbol{\Gamma}_{p-1} \Delta y_{p-2} + \epsilon_t$$

`vec` estimates $\boldsymbol{\Pi}$ and the $\boldsymbol{\Gamma}_i$. [Johansen \(1995, 25\)](#) notes that

$$\boldsymbol{\Pi} = \sum_{i=1}^p \mathbf{A}_i - \mathbf{I}_K \quad (9)$$

where $\mathbf{I}_K$ is the K -dimensional identity matrix, and

$$\boldsymbol{\Gamma}_i = - \sum_{j=i+1}^p \mathbf{A}_j \quad (10)$$

Defining

$$\mathbf{\Gamma} = \mathbf{I}_K - \sum_{i=1}^{p-1} \mathbf{\Gamma}_i$$

and using (9) and (10) allow us to solve for the $\mathbf{A}_i$ as

$$\mathbf{A}_1 = \mathbf{\Pi} + \mathbf{\Gamma}_1 + \mathbf{I}_K$$

$$\mathbf{A}_i = \mathbf{\Gamma}_i - \mathbf{\Gamma}_{i-1} \quad \text{for } i = \{2, \dots, p-1\}$$

and

$$\mathbf{A}_p = -\mathbf{\Gamma}_{p-1}$$

Using these formulas, we can back out estimates of $\mathbf{A}_i$ from the estimates of the $\mathbf{\Gamma}_i$ and $\mathbf{\Pi}$ produced by `vec`. Then we simply use the formulas for the IRFs and OIRFs presented in [Impulse-response function formulas for VARs](#).

The running sums of the IRFs and OIRFs over the steps within each impulse–response pair are the cumulative IRFs and OIRFs.

Algorithms for bootstrapping the VAR IRF and FEVD standard errors

`irf create` offers two bootstrap algorithms for estimating the standard errors of the various IRFs and FEVDs. Both `var` and `svar` contain estimators for the coefficients in a VAR that are conditional on the first p observations. The two bootstrap algorithms are also conditional on the first p observations.

Specifying the `bs` option calculates the standard errors by bootstrapping the residuals. For a bootstrap with R repetitions, this method uses the following algorithm:

1. Fit the model and save the estimated parameters.
2. Use the estimated coefficients to calculate the residuals.
3. Repeat steps 3a to 3c R times.
 - 3a. Draw a simple random sample of size T with replacement from the residuals. The random samples are drawn over the $K \times 1$ vectors of residuals. When the t th vector is drawn, all K residuals are selected. This preserves the contemporaneous correlations among the residuals.
 - 3b. Use the p initial observations, the sampled residuals, and the estimated coefficients to construct a new sample dataset.
 - 3c. Fit the model and calculate the different IRFs and FEVDs.
 - 3d. Save these estimates as observation r in the bootstrapped dataset.
4. For each IRF and FEVD, the estimated standard deviation from the R bootstrapped estimates is the estimated standard error of that impulse–response function or forecast-error variance decomposition.

Specifying the `bsp` option estimates the standard errors by a multivariate normal parametric bootstrap. The algorithm for the multivariate normal parametric bootstrap is identical to the one above, with the exception that 3a is replaced by 3a(bsp):

- 3a(bsp). Draw T pseudovariates from a multivariate normal distribution with covariance matrix $\hat{\mathbf{\Sigma}}$.

References

- Amisano, G., and C. Giannini. 1997. *Topics in Structural VAR Econometrics*. 2nd ed. Heidelberg: Springer.
- Christiano, L. J., M. Eichenbaum, and C. L. Evans. 1999. Monetary policy shocks: What have we learned and to what end? In *Handbook of Macroeconomics: Volume 1A*, ed. J. B. Taylor and M. Woodford. New York: Elsevier.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Johansen, S. 1995. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford: Oxford University Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Serfling, R. J. 1980. *Approximation Theorems of Mathematical Statistics*. New York: Wiley.
- Sims, C. A. 1980. Macroeconomics and reality. *Econometrica* 48: 1–48.
- Stock, J. H., and M. W. Watson. 2001. Vector autoregressions. *Journal of Economic Perspectives* 15: 101–115.

Also see

- [TS] **irf** — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
- [TS] **var intro** — Introduction to vector autoregressive models
- [TS] **vec intro** — Introduction to vector error-correction models

Syntax

```
irf ctable (spec1) [ (spec2) ... [ (specN) ] ] [ , options ]
```

where (spec_k) is

```
(irfname impulsevar responsevar stat [ , spec_options ])
```

irfname is the name of a set of IRF results in the active IRF file. *impulsevar* should be specified as an endogenous variable for all statistics except *dm* and *cdm*; for those, specify as an exogenous variable. *responsevar* is an endogenous variable name. *stat* is one or more statistics from the list below:

stat	description
irf	impulse–response function
oirf	orthogonalized impulse–response function
dm	dynamic-multiplier function
cirf	cumulative impulse–response function
coirf	cumulative orthogonalized impulse–response function
cdm	cumulative dynamic-multiplier function
fevd	Cholesky forecast-error variance decomposition
sirf	structural impulse–response function
sfevd	structural forecast-error variance decomposition

options	description
set(<i>filename</i>)	make <i>filename</i> active
noci	do not report confidence intervals
stderr	include standard errors for each statistic
individual	make an individual table for each combination
title("text")	use <i>text</i> as overall table title
step(#)	set common maximum step
level(#)	set confidence level; default is level(95)

spec_options	description
noci	do not report confidence intervals
stderr	include standard errors for each statistic
level(#)	set confidence level; default is level(95)
[†] ititle("text")	use <i>text</i> as individual subtitle for specific table

[†] ititle("text") is not shown in the dialog box.

spec_options may be specified within a table specification, globally, or both. When specified in a table specification, the *spec_options* affect only the specification in which they are used. When supplied globally, the *spec_options* affect all table specifications. When specified in both places, options for the table specification take precedence.

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Combined tables

Description

`irf ctable` makes a table or a combined table of IRF results. Each block within a pair of matching parentheses—each ($spec_k$)—specifies the information for a specific table. `irf ctable` combines these tables into one table, unless the `individual` option is specified, in which case separate tables for each block are created.

`irf ctable` operates on the active IRF file; see [TS] [irf set](#).

Options

`set(filename)` specifies the file to be made active; see [TS] [irf set](#). If `set()` is not specified, the active file is used.

`noci` suppresses reporting of the confidence intervals for each statistic. `noci` is assumed when the model was fit by `vec` because no confidence intervals were estimated.

`stderr` specifies that standard errors for each statistic also be included in the table.

`individual` places each block, or ($spec_k$), in its own table. By default, `irf ctable` combines all the blocks into one table.

`title("text")` specifies a title for the table or the set of tables.

`step(#)` specifies the maximum number of steps to use for all tables. By default, each table is constructed using all steps available.

`level(#)` specifies the default confidence level, as a percentage, for confidence intervals, when they are reported. The default is `level(95)` or as set by `set level`; see [U] [20.7 Specifying the width of confidence intervals](#).

The following option is available with `irf ctable` but is not shown in the dialog box:

`ititle("text")` specifies an individual subtitle for a specific table. `ititle()` may be specified only when the `individual` option is also specified.

Remarks

If you have not read [TS] [irf](#), please do so.

Also see [TS] [irf table](#) for a slightly easier to use, but less powerful, table command.

`irf ctable` creates a series of tables from IRF results. The information enclosed within each set of parentheses,

```
(irfname impulsevar responsevar stat [ , spec_options ])
```

forms a request for a specific table.

The first part—*irfname impulsevar responsevar*—identifies a set of IRF estimates or a set of variance decomposition estimates. The next part—*stat*—specifies which statistics are to be included in the table. The last part—*spec_options*—includes the `noci`, `level()`, and `stderr` options, and places (or suppresses) additional columns in the table.

Each specific table displays the requested statistics corresponding to the specified combination of *irfname*, *impulsevar*, and *responsevar* over the step horizon. By default, all the individual tables are combined into one table. Also by default, all the steps, or periods, available are included in the table. You can use the `step()` option to impose a common maximum for all tables.

➤ Example 1

In [example 1](#) of [\[TS\] irf table](#), we fit a model using `var` and we saved the IRFs for two different orderings. The commands we used were

```
. use http://www.stata-press.com/data/r11/lutkepohl2
. var dln_inv dln_inc dln_consump
. irf set results4
. irf create ordera, step(8)
. irf create orderb, order(dln_inc dln_inv dln_consump) step(8)
```

We then formed the desired table by typing

```
. irf table oirf fevd, impulse(dln_inc) response(dln_consump) noci std
> title("Ordera versus orderb")
```

Using `irf ctable`, we can form the equivalent table by typing

```
. irf ctable (ordera dln_inc dln_consump oirf fevd)
> (orderb dln_inc dln_consump oirf fevd),
> noci std title("Ordera versus orderb")

Ordera versus orderb
```

step	(1) oirf	(1) S.E.	(1) fevd	(1) S.E.
0	.005123	.000878	0	0
1	.001635	.000984	.288494	.077483
2	.002948	.000993	.294288	.073722
3	-.000221	.000662	.322454	.075562
4	.000811	.000586	.319227	.074063
5	.000462	.000333	.322579	.075019
6	.000044	.000275	.323552	.075371
7	.000151	.000162	.323383	.075314
8	.000091	.000114	.323499	.075386

step	(2) oirf	(2) S.E.	(2) fevd	(2) S.E.
0	.005461	.000925	0	0
1	.001578	.000988	.327807	.08159
2	.003307	.001042	.328795	.077519
3	-.00019	.000676	.370775	.080604
4	.000846	.000617	.366896	.079019
5	.000491	.000349	.370399	.079941
6	.000069	.000292	.371487	.080323
7	.000158	.000172	.371315	.080287
8	.000096	.000122	.371438	.080366

(1) *irfname* = *ordera*, *impulse* = *dln_inc*, and *response* = *dln_consump*
(2) *irfname* = *orderb*, *impulse* = *dln_inc*, and *response* = *dln_consump*

The output is displayed in one table. Because the table did not fit horizontally, it automatically wrapped. At the bottom of the table is a list of keys that appear at the top of each column. The

results in the table above indicate that the orthogonalized IRFs do not change by much. Because the estimated forecast-error variances do change, we might want to produce two tables that contain the estimated forecast-error variance decompositions and their 95% confidence intervals:

```
. irf ctable (ordera dln_inc dln_consump fevd)
>          (orderb dln_inc dln_consump fevd), individual
```

Table 1

step	(1) fevd	(1) Lower	(1) Upper
0	0	0	0
1	.288494	.13663	.440357
2	.294288	.149797	.43878
3	.322454	.174356	.470552
4	.319227	.174066	.464389
5	.322579	.175544	.469613
6	.323552	.175826	.471277
7	.323383	.17577	.470995
8	.323499	.175744	.471253

95% lower and upper bounds reported

(1) irfname = ordera, impulse = dln_inc, and response = dln_consump

Table 2

step	(2) fevd	(2) Lower	(2) Upper
0	0	0	0
1	.327807	.167893	.487721
2	.328795	.17686	.48073
3	.370775	.212794	.528757
4	.366896	.212022	.52177
5	.370399	.213718	.52708
6	.371487	.214058	.528917
7	.371315	.213956	.528674
8	.371438	.213923	.528953

95% lower and upper bounds reported

(2) irfname = orderb, impulse = dln_inc, and response = dln_consump

Because we specified the `individual` option, the output contains two tables, one for each specific table command. At the bottom of each table is a list of the keys used in that table and a note indicating the level of the confidence intervals that we requested. The results from table 1 and table 2 indicate that each estimated function is well within the confidence interval of the other, so we conclude that the functions are not significantly different.

4

(Continued on next page)

Saved results

`irf ctable` saves the following in `r()`:

Scalars

<code>r(ncols)</code>	number of columns in all tables
<code>r(k_ymax)</code>	number of distinct keys
<code>r(k)</code>	number of specific table commands

Macros

<code>r(key#)</code>	#th key
<code>r(tnotes)</code>	list of keys applied to each column

Methods and formulas

`irf ctable` is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Title

irf describe — Describe an IRF file

Syntax

```
irf describe [irf_resultslist] [, options]
```

<i>options</i>	description
<code>set(<i>filename</i>)</code>	make <i>filename</i> active
<code>using(<i>irf_filename</i>)</code>	describe <i>irf_filename</i> without making active
<code>detail</code>	show additional details of IRF results
<code>variables</code>	show underlying structure of the IRF dataset

Menu

Statistics > Multivariate time series > Manage IRF results and files > Describe IRF file

Description

`irf describe` describes the IRF results stored in an IRF file.

If `set()` or `using()` is not specified, the IRF results of the active IRF file are described.

Options

`set(filename)` specifies the IRF file to be described and set; see [\[TS\] irf set](#). If *filename* is specified without an extension, `.irf` is assumed.

`using(irf_filename)` specifies the IRF file to be described. The active IRF file, if any, remains unchanged. If *irf_filename* is specified without an extension, `.irf` is assumed.

`detail` specifies that `irf describe` display detailed information about each set of IRF results. `detail` is implied when *irf_resultslist* is specified.

`variables` is a programmer's option; additionally displays the output produced by the `describe` command.

Remarks

If you have not read [\[TS\] irf](#), please do so.

► Example 1

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
(output omitted)
```

We create three sets of IRF results:

```
. irf create order1, set(myirfs, replace)
(file myirfs.irf created)
(file myirfs.irf now active)
(file myirfs.irf updated)
. irf create order2, order(dln_inc dln_inv dln_consump)
(file myirfs.irf updated)
. irf create order3, order(dln_inc dln_consump dln_inv)
(file myirfs.irf updated)
. irf describe
Contains irf results from myirfs.irf (dated 4 Apr 2009 12:36)
```

irfname	model	endogenous variables and order (*)
order1	var	dln_inv dln_inc dln_consump
order2	var	dln_inc dln_inv dln_consump
order3	var	dln_inc dln_consump dln_inv

(*) order is relevant only when model is var

The output reveals the order in which we specified the variables.

```
. irf describe order1
irf results for order1
Estimation specification
  model:  var
  endog:  dln_inv dln_inc dln_consump
  sample: quarterly data from 1960q4 to 1978q4
  lags:   1 2
  constant: constant
  exog:   none
  exogvars: none
  exlags: none
  varcns: unconstrained

IRF specification
  step: 8
  order: dln_inv dln_inc dln_consump
  std error: asymptotic
  reps: none
```

Here we see a summary of the model we fit as well as the specification of the IRFs.



Saved results

`irf describe` saves the following in `r()`:

Scalars

<code>r(N)</code>	number of observations in the IRF file
<code>r(k)</code>	number of variables in the IRF file
<code>r(width)</code>	width of dataset in the IRF file
<code>r(N_max)</code>	maximum number of observations
<code>r(k_max)</code>	maximum number of variables
<code>r(widthmax)</code>	maximum width of the dataset
<code>r(changed)</code>	flag indicating that data have changed since last saved

Macros

<code>r(version)</code>	version of IRF results file
<code>r(irfnames)</code>	names of IRF results in the IRF file
<code>r(irfname_model)</code>	<code>var</code> , <code>sr var</code> , <code>lr var</code> , or <code>vec</code>
<code>r(irfname_order)</code>	Cholesky order assumed in IRF estimates
<code>r(irfname_exog)</code>	exogenous variables, and their lags, in VAR or underlying VAR
<code>r(irfname_exogvar)</code>	exogenous variables in VAR or underlying VAR
<code>r(irfname_constant)</code>	<code>constant</code> or <code>noconstant</code>
<code>r(irfname_lags)</code>	lags in model
<code>r(irfname_exlags)</code>	lags of exogenous variables in model
<code>r(irfname_tmin)</code>	minimum value of timevar in the estimation sample
<code>r(irfname_tmax)</code>	maximum value of timevar in the estimation sample
<code>r(irfname_timevar)</code>	name of <code>tsset</code> timevar
<code>r(irfname_tsfmt)</code>	format of timevar in the estimation sample
<code>r(irfname_varcns)</code>	<code>unconstrained</code> or colon-separated list of constraints placed on VAR coefficients
<code>r(irfname_svarcns)</code>	<code>"."</code> or colon-separated list of constraints placed on SVAR coefficients
<code>r(irfname_step)</code>	maximum step in IRF estimates
<code>r(irfname_stderror)</code>	<code>asymptotic</code> , <code>bs</code> , <code>bsp</code> , or <code>none</code> , depending on type of standard errors specified to <code>irf create</code>
<code>r(irfname_reps)</code>	<code>"."</code> or number of bootstrap replications performed
<code>r(irfname_version)</code>	version of IRF file that originally held <i>irfname</i> IRF results
<code>r(irfname_rank)</code>	<code>"."</code> or number of cointegrating equations
<code>r(irfname_trend)</code>	<code>"."</code> or <code>trend()</code> specified in <code>vec</code>
<code>r(irfname_veccns)</code>	<code>"."</code> or constraints placed on VECM parameters
<code>r(irfname_sind)</code>	<code>"."</code> or normalized seasonal indicators included in <code>vec</code>

Methods and formulas

`irf describe` is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Title

irf drop — Drop IRF results from the active IRF file

Syntax

```
irf drop irf_resultslist [ , set(filename) ]
```

Menu

Statistics > Multivariate time series > Manage IRF results and files > Drop IRF results

Description

irf drop removes IRF results from the active IRF file.

Option

set(filename) specifies the file to be made active; see [TS] irf set. If set() is not specified, the active file is used.

Remarks

If you have not read [TS] irf, please do so.

➤ Example 1

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
(output omitted)
```

We create three sets of IRF results:

```
. irf create order1, set(myirfs, replace)
(file myirfs.irf created)
(file myirfs.irf now active)
(file myirfs.irf updated)
. irf create order2, order(dln_inc dln_inv dln_consump)
(file myirfs.irf updated)
. irf create order3, order(dln_inc dln_consump dln_inv)
(file myirfs.irf updated)
. irf describe
Contains irf results from myirfs.irf (dated 4 Apr 2009 12:59)
```

irfname	model	endogenous variables and order (*)
order1	var	dln_inv dln_inc dln_consump
order2	var	dln_inc dln_inv dln_consump
order3	var	dln_inc dln_consump dln_inv

(*) order is relevant only when model is var

Now let's remove order1 and order2 from myirfs.irf.

```
. irf drop order1 order2
(order1 dropped)
(order2 dropped)
file myirfs.irf updated
```

```
. irf describe
```

Contains irf results from myirfs.irf (dated 4 Apr 2009 12:59)

irfname	model	endogenous variables and order (*)		
order3	var	dln_inc	dln_consump	dln_inv

(*) order is relevant only when model is var

order1 and order2 have been dropped.

◀

Methods and formulas

irf drop is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Syntax

```
irf graph stat [ , options ]
```

<i>stat</i>	description
<code>irf</code>	impulse–response function
<code>oirf</code>	orthogonalized impulse–response function
<code>dm</code>	dynamic-multiplier function
<code>cirf</code>	cumulative impulse–response function
<code>coirf</code>	cumulative orthogonalized impulse–response function
<code>cdm</code>	cumulative dynamic-multiplier function
<code>fevd</code>	Cholesky forecast-error variance decomposition
<code>sirf</code>	structural impulse–response function
<code>sfevd</code>	structural forecast-error variance decomposition

- Notes: 1. No statistic may appear more than once.
2. If confidence intervals are included (the default), only two statistics may be included.
3. If confidence intervals are suppressed (option `nocl`), up to four statistics may be included.

<i>options</i>	description
Main	
<code><u>set</u>(<i>filename</i>)</code>	make <i>filename</i> active
<code><u>irf</u>(<i>irfnames</i>)</code>	use <i>irfnames</i> IRF result sets
<code><u>impulse</u>(<i>impulsevar</i>)</code>	use <i>impulsevar</i> as impulse variables
<code><u>response</u>(<i>endogvars</i>)</code>	use endogenous variables as response variables
<code><u>nocl</u></code>	suppress confidence bands
<code><u>level</u>(#)</code>	set confidence level; default is <code>level(95)</code>
<code><u>lstep</u>(#)</code>	use # for first step
<code><u>ustep</u>(#)</code>	use # for maximum step

Advanced	
<code><u>individual</u></code>	graph each combination individually
<code><u>iname</u>(<i>namestub</i> [, <i>replace</i>])</code>	stub for naming the individual graphs
<code><u>isaving</u>(<i>filenamestub</i> [, <i>replace</i>])</code>	stub for saving the individual graphs to files

Plots	
<code><u>plot</u>#<i>opts</i>(<i>cline_options</i>)</code>	affect rendition of the line plotting the # <i>stat</i>

CI plots	
<code><u>ci</u>#<i>opts</i>(<i>area_options</i>)</code>	affect rendition of the confidence interval for the # <i>stat</i>

Y axis, X axis, Titles, Legend, Overall

twoway_options

any options other than `by()` documented in
[G] *twoway_options*

byopts(*by_option*)

how subgraphs are combined, labeled, etc.

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Graphs by impulse or response

Description

`irf graph` graphs impulse–response functions (IRFs), dynamic-multiplier functions, and forecast-error variance decompositions (FEVDs) over time.

Options

Main

`set(filename)` specifies the file to be made active; see [TS] `irf set`. If `set()` is not specified, the active file is used.

`irf(irfname)` specifies the IRF result sets to be used. If `irf()` is not specified, each of the results in the active IRF file is used. (Files often contain just one set of IRF results stored under one *irfname*; in that case, those results are used.)

`impulse(impulsevar)` and `response(endogvars)` specify the impulse and response variables. Usually one of each is specified, and one graph is drawn. If multiple variables are specified, a separate subgraph is drawn for each impulse–response combination. If `impulse()` and `response()` are not specified, subgraphs are drawn for all combination of impulse and response variables.

impulsevar should be specified as an endogenous variable for all statistics except `dm` or `cdm; for those, specify as an exogenous variable.`

`noci` suppresses graphing the confidence interval for each statistic. `noci` is assumed when the model was fit by `vec` because no confidence intervals were estimated.

`level(#)` specifies the default confidence level, as a percentage, for confidence intervals, when they are reported. The default is `level(95)` or as set by `set level`; see [U] 20.7 Specifying the width of confidence intervals. Also see [TS] `irf cgraph` for a graph command that allows the confidence level to vary over the graphs.

`lstep(#)` specifies the first step, or period, to be included in the graphs. `lstep(0)` is the default.

`ustep(#)`, $\# \geq 1$, specifies the maximum step, or period, to be included in the graphs.

Advanced

`individual` specifies that each graph be displayed individually. By default, `irf graph` combines the subgraphs into one image. When `individual` is specified, `byopts()` may not be specified, but the `isaving()` and `iname()` options may be specified.

`iname(namestub [, replace])` specifies that the *i*th individual graph be saved in memory under the name *namestubi*, which must be a valid Stata name of 24 characters or fewer. `iname()` may be specified only with the `individual` option.

`isaving(filenamestub [, replace])` specifies that the *i*th individual graph should be saved to disk in the current working directory under the name *filenamestubi.gph*. `isaving()` may be specified only when the `individual` option is also specified.

Plots

`plot1opts(cline_options)`, ..., `plot4opts(cline_options)` affect the rendition of the plotted statistics (the *stat*). `plot1opts()` affects the rendition of the first statistic; `plot2opts()`, the second; and so on. *cline_options* are as described in [G] [cline_options](#).

CI plots

`ci1opts(area_options)` and `ci2opts(area_options)` affect the rendition of the confidence intervals for the first (`ci1opts()`) and second (`ci2opts()`) statistics in *stat*.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)). Note that the `saving()` and `name()` options may not be combined with the `individual` option.

`byopts(by_option)` is as documented in [G] [by_option](#) and may not be specified when `individual` is specified. `byopts()` affects how the subgraphs are combined, labeled, etc.

Remarks

If you have not read [TS] [irf](#), please do so.

Also see [TS] [irf cgraph](#), which produces combined graphs; [TS] [irf ograph](#), which produces overlaid graphs; and [TS] [irf table](#), which displays results in tabular form.

`irf graph` produces one or more graphs and displays them arrayed into one image unless the `individual` option is specified, in which case the individual graphs are displayed separately. Each individual graph consists of all the specified *stat* and represents one impulse–response combination.

Because all the specified *stat* appear on the same graph, putting together statistics with very different scales is not recommended. For instance, sometimes `sirf` and `oirf` are on similar scales while `irf` is on a different scale. In such cases, combining `sirf` and `oirf` on the same graph looks fine, but combining either with `irf` produces an uninformative graph.

► Example 1

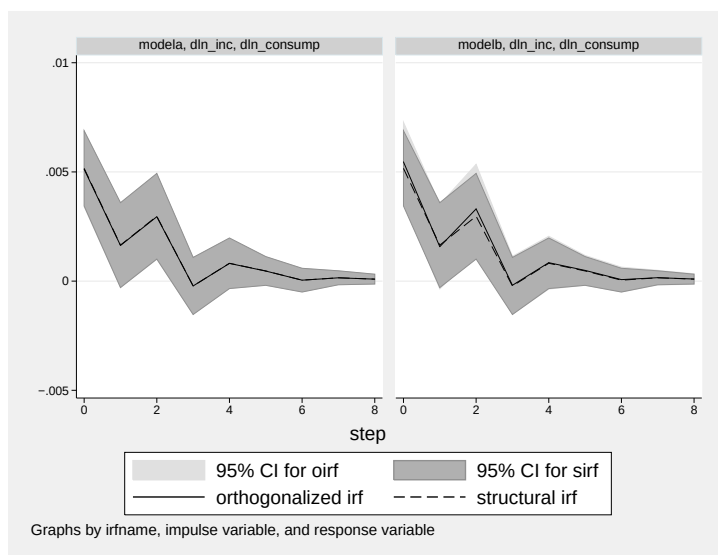
Suppose that we have results generated from two different SVAR models. We want to know whether the shapes of the structural IRFs and the structural FEVDs are similar in the two models. We are also interested in knowing whether the structural IRFs and the structural FEVDs differ significantly from their Cholesky counterparts.

Filling in the background, we have previously issued the commands

```
. use http://www.stata-press.com/data/r11/lutkepohl2
. mat a = (., 0, 0\0,.,0\.,.,.)
. mat b = I(3)
. svar dln_inv dln_inc dln_consump, aeq(a) beq(b)
. irf create modela, set(results3) step(8)
. svar dln_inc dln_inv dln_consump, aeq(a) beq(b)
. irf create modelb, step(8)
```

To see whether the shapes of the structural IRFs and the structural FEVDs are similar in the two models, we type

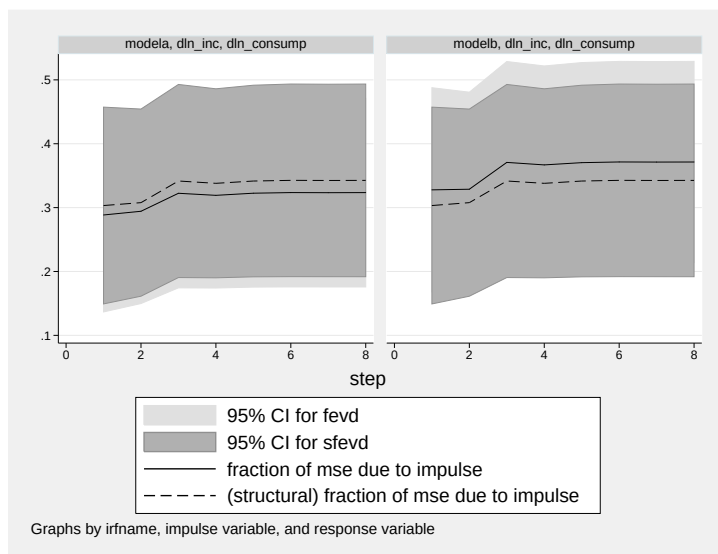
```
. irf graph oirf sirf, impulse(dln_inc) response(dln_consump)
```



The graph reveals that the `oirf` and the `sirf` estimates are essentially the same for both models and that the shapes of the functions are very similar for the two models.

To see whether the structural IRFs and the structural FEVDs differ significantly from their Cholesky counterparts, we type

```
. irf graph fevd sfevd, impulse(dln_inc) response(dln_consump) lstep(1)
> legend(cols(1))
```



This combined graph reveals that the shapes of these functions are also similar for the two models. However, the graph illuminates one minor difference between them: In `modela`, the estimated structural

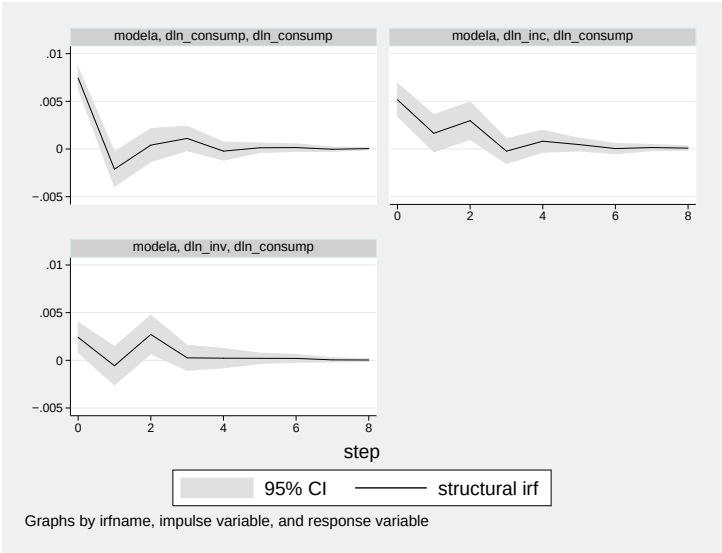
FEVD is slightly larger than the Cholesky-based estimates, whereas in `modelb` the Cholesky-based estimates are slightly larger than the structural estimates. For both models, however, the structural estimates are close to the center of the wide confidence intervals for the two estimates.



➤ Example 2

Let’s focus on the results from `modela`. Suppose that we were interested in examining how `dln_consump` responded to impulses in its own structural innovations, structural innovations to `dln_inc`, and structural innovations to `dln_inv`. We type

```
. irf graph sirf, irf(modela) response(dln_consump)
```



The upper-left graph shows the structural IRF of an innovation in `dln_consump` on `dln_consump`. It indicates that the identification restrictions used in `modela` imply that a positive shock to `dln_consump` causes an increase in `dln_consump`, followed by a decrease, followed by an increase, and so on, until the effect dies out after roughly 5 periods.

The upper-right graph shows the structural IRF of an innovation in `dln_inc` on `dln_consump`, indicating that a positive shock to `dln_inc` causes an increase in `dln_consump`, which dies out after 4 or 5 periods.



□ Technical note

[TS] **irf table** contains a [technical note](#) warning you to be careful in naming variables when you fit models. What is said there applies equally here.



Saved results

`irf graph` saves the following in `r()`:

Scalars

`r(k)` number of graphs

Macros

`r(stats)` *statlist*

`r(irfname)` *resultslst*

`r(impulse)` *impulselist*

`r(response)` *responselist*

`r(plot#)` contents of `plot#opts()`

`r(ci)` level applied to confidence
 intervals or `nocl`

`r(ciopts#)` contents of `ci#opts()`

`r(byopts)` contents of `byopts()`

`r(saving)` supplied `saving()` option

`r(name)` supplied `name()` option

`r(individual)` individual or blank

`r(isaving)` contents of `saving()`

`r(iname)` contents of `name()`

`r(subtitle#)` subtitle for individual graph #

Methods and formulas

`irf graph` is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Syntax

```
irf ograph (spec1) [ (spec2) ... [ (spec15) ] ] [ , options ]
```

where (*spec*_{*k*}) is

```
(irfname impulsevar responsevar stat [ , spec_options ])
```

irfname is the name of a set of IRF results in the active IRF file or “.”, which means the first named result in the active IRF file. *impulsevar* should be specified as an endogenous variable for all statistics except *dm* and *cdm*; for those, specify as an exogenous variable. *responsevar* is an endogenous variable name. *stat* is one or more statistics from the list below:

<i>stat</i>	description
<i>irf</i>	impulse–response function
<i>oirf</i>	orthogonalized impulse–response function
<i>dm</i>	dynamic-multiplier function
<i>cirf</i>	cumulative impulse–response function
<i>coirf</i>	cumulative orthogonalized impulse–response function
<i>cdm</i>	cumulative dynamic-multiplier function
<i>fevd</i>	Cholesky forecast-error variance decomposition
<i>sirf</i>	structural impulse–response function
<i>sfevd</i>	structural forecast-error variance decomposition

<i>options</i>	description

Plots	
<i>plot_options</i>	define the IRF plots
<i>set</i> (<i>filename</i>)	make <i>filename</i> active

Options	
<i>common_options</i>	level and steps

Y axis, X axis, Titles, Legend, Overall	
<i>twoway_options</i>	any options other than <i>by</i> () documented in [G] twoway_options

<i>plot_options</i>	description

Main	
<i>set</i> (<i>filename</i>)	make <i>filename</i> active
<i>irf</i> (<i>irfnames</i>)	use <i>irfnames</i> IRF result sets
<i>impulse</i> (<i>impulsevar</i>)	use <i>impulsevar</i> as impulse variables
<i>response</i> (<i>endogvars</i>)	use endogenous variables as response variables
<i>ci</i>	add confidence bands to the graph

<i>spec_options</i>	description
Options	
<i>common_options</i>	level and steps
Plot	
<i>cline_options</i>	affect rendition of the plotted lines
CI plot	
<i>ciopts(area_options)</i>	affect rendition of the confidence intervals

<i>common_options</i>	description
Options	
<i>level</i> (#)	set confidence level; default is <i>level</i> (95)
<i>lstep</i> (#)	use # for first step
<i>ustep</i> (#)	use # for maximum step

common_options may be specified within a plot specification, globally, or in both. When specified in a plot specification, the *common_options* affect only the specification in which they are used. When supplied globally, the *common_options* affect all plot specifications. When supplied in both places, options in the plot specification take precedence.

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Overlaid graph

Description

`irf ograph` displays plots of `irf` results on one graph (one pair of axes).
To become familiar with this command, type `db irf ograph`.

Options

Plots

plot_options defines the IRF plots and are found under the **Main**, **Plot**, and **CI plot** tabs.

`set(filename)` specifies the file to be made active; see [TS] [irf set](#). If `set()` is not specified, the active file is used.

Main

`set(filename)` specifies the file to be made active; see [TS] [irf set](#). If `set()` is not specified, the active file is used.

`irf(irfnames)` specifies the IRF result sets to be used. If `irf()` is not specified, each of the results in the active IRF file is used. (Files often contain just one set of IRF results stored under one *irfname*; in that case, those results are used.)

`impulse(varlist)` and `response(endogvars)` specify the impulse and response variables. Usually one of each is specified, and one graph is drawn. If multiple variables are specified, a separate subgraph is drawn for each impulse–response combination. If `impulse()` and `response()` are not specified, subgraphs are drawn for all combination of impulse and response variables.

`ci` adds confidence bands to the graph. The `noc` option may be used within a plot specification to suppress its confidence bands when the `ci` option is supplied globally.

Plot

cline_options affect the rendition of the plotted lines; see [G] [cline_options](#).

CI plot

`ciopts(area_options)` affects the rendition of the confidence bands for the plotted statistic; see [G] [area_options](#). `ciopts()` implies `ci`.

Options

`level(#)` specifies the confidence level, as a percentage, for confidence bands; see [U] [20.7 Specifying the width of confidence intervals](#).

`lstep(#)` specifies the first step, or period, to be included in the graph. `lstep(0)` is the default.

`ustep(#)`, $\# \geq 1$, specifies the maximum step, or period, to be included.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

Remarks

If you have not read [TS] [irf](#), please do so.

`irf ograph` overlays plots of IRFs and FEVDs on one graph.

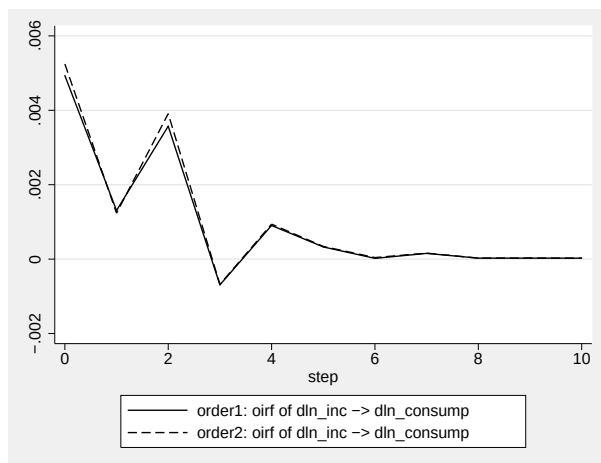
► Example 1

We have previously issued the commands

```
. use http://www.stata-press.com/data/r11/lutkepohl2
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
. irf create order1, step(10) set(myirf1, new)
. irf create order2, step(10) order(dln_inc dln_inv dln_consump)
```

We now wish to compare the `oirf` for impulse `dln_inc` and response `dln_consump` for two different Cholesky orderings:

```
. irf ograph (order1 dln_inc dln_consump oirf)
>           (order2 dln_inc dln_consump oirf)
```



◀

□ Technical note

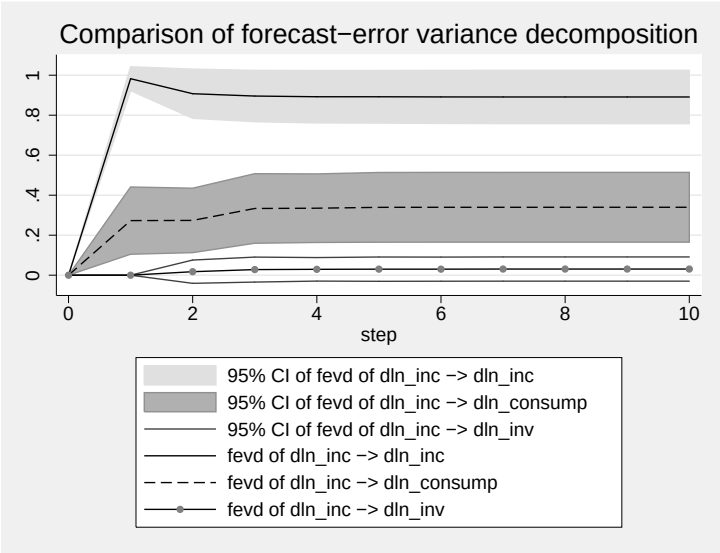
Graph options allow you to change the appearance of each plot. The following graph contains the plots of the FEVDs (FEVDs) for impulse `dln_inc` and each response using the results from the first collection of results in the active IRF file (using the “.” shortcut). In the second plot, we supply the `clpat(dash)` option (an abbreviation for `clpattern(dash)`) to give the line a dashed pattern. In the third plot, we supply the `m(o) clpat(dashdot) recast(connected)` options to get small circles connected by a line with a dash-dot pattern; the `cilines` option plots the confidence bands by using lines instead of areas. We use the `title()` option to add a descriptive title to the graph and supply the `ci` option globally to add confidence bands to all the plots.

(Continued on next page)

```

. irf ograph (. dln_inc dln_inc fevd)
>      (. dln_inc dln_consump fevd, clpat(dash))
>      (. dln_inc dln_inv fevd, cilines m(o) clpat(dashdot)
>      recast(connection))
>      , ci title("Comparison of forecast-error variance decomposition")

```



The `clpattern()` option is described in [G] [connect_options](#), `msymbol()` is described in [G] [marker_options](#), `title()` is described in [G] [title_options](#), and `recast()` is described in [G] [advanced_options](#).



Saved results

`irf ograph` saves the following in `r()`:

Scalars	
<code>r(plots)</code>	number of plot specifications
<code>r(ciplots)</code>	number of plotted confidence bands
Macros	
<code>r(irfname#)</code>	<i>irfname</i> from (<i>spec#</i>)
<code>r(impulse#)</code>	impulse from (<i>spec#</i>)
<code>r(response#)</code>	response from (<i>spec#</i>)
<code>r(stat#)</code>	statistics from (<i>spec#</i>)
<code>r(ci#)</code>	level from (<i>spec#</i>) or <i>noci</i>

Methods and formulas

`irf ograph` is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Title

irf rename — Rename an IRF result in an IRF file

Syntax

irf rename *oldname newname* [, set(*filename*)]

Menu

Statistics > Multivariate time series > Manage IRF results and files > Rename IRF results

Description

irf rename changes the name of a set of IRF results stored in the active IRF file.

Option

set(*filename*) specifies the file to be made active; see [TS] irf set. If set() is not specified, the active file is used.

Remarks

If you have not read [TS] irf, please do so.

➤ Example 1

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lags(1/2) dfk
(output omitted)
```

We create three sets of IRF results:

```
. irf create original, set(myirfs, replace)
(file myirfs.irf created)
(file myirfs.irf now active)
(file myirfs.irf updated)
. irf create order2, order(dln_inc dln_inv dln_consump)
(file myirfs.irf updated)
. irf create order3, order(dln_inc dln_consump dln_inv)
(file myirfs.irf updated)
. irf describe
Contains irf results from myirfs.irf (dated 4 Apr 2009 13:06)
```

irfname	model	endogenous variables and order (*)
original	var	dln_inv dln_inc dln_consump
order2	var	dln_inc dln_inv dln_consump
order3	var	dln_inc dln_consump dln_inv

(*) order is relevant only when model is var

Now let's rename IRF result `original` to `order1`.

```
. irf rename original order1
(81 real changes made)
original renamed to order1
. irf describe
Contains irf results from myirfs.irf (dated 4 Apr 2009 13:06)
```

irfname	model	endogenous variables and order (*)		
order1	var	dln_inv	dln_inc	dln_consump
order2	var	dln_inc	dln_inv	dln_consump
order3	var	dln_inc	dln_consump	dln_inv

(*) order is relevant only when model is var

`original` has been renamed to `order1`.

◀

Saved results

`irf rename` saves the following in `r()`:

```
Macros
      r(irfnames)  irfnames after rename
      r(oldnew)    oldname newname
```

Methods and formulas

`irf rename` is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Title

irf set — Set the active IRF file

Syntax

Report identity of active file

```
irf set
```

Set, and if necessary create, active file

```
irf set irf_filename
```

Create, and if necessary replace, active file

```
irf set irf_filename, replace
```

Clear any active IRF file

```
irf set, clear
```

Menu

Statistics > Multivariate time series > Manage IRF results and files > Set active IRF file

Description

In the first syntax, **irf set** reports the identity of the active file, if there is one. Also see [\[TS\] **irf describe**](#) for obtaining reports on the contents of an IRF file.

In the second syntax, **irf set irf_filename** specifies that the file be set as the active file and, if the file does not exist, that it be created as well.

In the third syntax, **irf set irf_filename, replace** specifies that even if file *irf_filename* exists, a new, empty file is to be created and set.

In the rarely used fourth syntax, **irf set, clear** specifies that, if any IRF file is set, it be unset and that there be no active IRF file.

IRF files are just files: they can be erased by **erase**, listed by **dir**, and copied by **copy**; see [\[D\] **erase**](#), [\[D\] **dir**](#), and [\[D\] **copy**](#).

If *irf_filename* is specified without an extension, **.irf** is assumed.

Options

replace specifies that if *irf_filename* already exists, the file is to be erased and a new, empty IRF file is to be created in its place. If it does not already exist, a new, empty file is created.

clear unsets the active IRF file.

Remarks

If you have not read [TS] [irf](#), please do so.

`irf set` reports the identity of the active IRF file:

```
. irf set
no irf file active
```

`irf set irf_filename` creates and sets an IRF file:

```
. irf set results1
(file results1.irf now active)
```

We specified the name `results1`, and `results1.irf` became the active file. The suffix `.irf` was added for us.

`irf set irf_filename` can also be used to create a new file:

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inc dln_consump, exog(1.dln_inv)
(output omitted)
. irf set results2
(file results2.irf created)
(file results2.irf now active)
. irf create order1
(file results2.irf updated)
```

Saved results

`irf set` saves the following in `r()`:

```
Macros
      r(Orville)  name of active IRF file, if there is an active IRF
```

Methods and formulas

`irf set` is implemented as an ado-file.

Also see

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

Syntax

```
irf table [stat] [, options]
```

<i>stat</i>	description
Main	
<code>irf</code>	impulse–response function
<code>oirf</code>	orthogonalized impulse–response function
<code>dm</code>	dynamic-multiplier function
<code>cirf</code>	cumulative impulse–response function
<code>coirf</code>	cumulative orthogonalized impulse–response function
<code>cdm</code>	cumulative dynamic-multiplier function
<code>fevd</code>	Cholesky forecast-error variance decomposition
<code>sirf</code>	structural impulse–response function
<code>sfevd</code>	structural forecast-error variance decomposition

If *stat* is not specified, all statistics are included, unless option `nostructural` is also specified, in which case `sirf` and `sfevd` are excluded. You may specify more than one *stat*.

<i>options</i>	description
Main	
<code>set(<i>filename</i>)</code>	make <i>filename</i> active
<code>irf(<i>irfnames</i>)</code>	use <i>irfnames</i> IRF result sets
<code>impulse(<i>impulsevar</i>)</code>	use <i>impulsevar</i> as impulse variables
<code>response(<i>endogvars</i>)</code>	use endogenous variables as response variables
<code>individual</code>	make an individual table for each result set
<code>title("text")</code>	use <i>text</i> for overall table title
Options	
<code>level(#)</code>	set confidence level; default is <code>level(95)</code>
<code>noci</code>	suppress confidence intervals
<code>stderror</code>	include standard errors in the tables
<code>nostructural</code>	suppress <code>sirf</code> and <code>sfevd</code> from the default list of statistics
<code>step(#)</code>	use common maximum step horizon <i>#</i> for all tables

Menu

Statistics > Multivariate time series > IRF and FEVD analysis > Tables by impulse or response

Description

`irf table` makes a table from the specified IRF results.

The rows of the tables are the time since impulse. Each column represents a combination of `impulse()` variable and `response()` variable for a *stat* from the `irf()` results.

Options

Main

`set(filename)` specifies the file to be made active; see [TS] [irf set](#). If `set()` is not specified, the active file is used.

All results are obtained from one IRF file. If you have results in different files that you want in one table, use `irf add` to copy results into one file; see [TS] [irf add](#).

`irf(irfnames)` specifies the IRF result sets to be used. If `irf()` is not specified, all the results in the active IRF file are used. (Files often contain just one set of IRF results, stored under one *irfname*; in that case, those results are used. When there are multiple IRF results, you may also wish to specify the individual option.)

`impulse(impulsevar)` specifies the impulse variables for which the statistics are to be reported. If `impulse()` is not specified, each model variable, in turn, is used. *impulsevar* should be specified as an endogenous variable for all statistics except `dm` or `cdm`; for those, specify as an exogenous variable.

`response(endogvars)` specifies the response variables for which the statistics are to be reported. If `response()` is not specified, each endogenous variable, in turn, is used.

`individual` specifies that each set of IRF results be placed in its own table, with its own title and footer. By default, `irf table` places all the IRF results in one table with one title and one footer. `individual` may not be combined with `title()`.

`title("text")` specifies a title for the overall table.

Options

`level(#)` specifies the default confidence level, as a percentage, for confidence intervals, when they are reported. The default is `level(95)` or as set by `set level`; see [U] [20.7 Specifying the width of confidence intervals](#).

`noci` suppresses reporting of the confidence intervals for each statistic. `noci` is assumed when the model was fit by `vec` because no confidence intervals were estimated.

`stderror` specifies that standard errors for each statistic also be included in the table.

`nostructural` specifies that *stat*, when not specified, exclude `sirf` and `sfevd`.

`step(#)` specifies the maximum step horizon for all tables. If `step()` is not specified, each table is constructed using all steps available.

Remarks

If you have not read [TS] [irf](#), please do so.

Also see [TS] [irf graph](#), which produces output in graphical form, and see [TS] [irf ctable](#), which also produces tabular output. `irf ctable` is more difficult to use but provides more control over how tables are formed.

➤ Example 1

We have fit a model with `var`, and we saved the IRFs from two different orderings. The commands we previously used were

```
. use http://www.stata-press.com/data/r11/lutkepohl2
. var dln_inv dln_inc dln_consump
. irf set results4
. irf create ordera, step(8)
. irf create orderb, order(dln_inc dln_inv dln_consump) step(8)
```

We now wish to compare the two orderings:

```
. irf table oirf fevd, impulse(dln_inc) response(dln_consump) noci std
> title("Ordera versus orderb")
```

Ordera versus orderb				
step	(1) oirf	(1) S.E.	(1) fevd	(1) S.E.
0	.005123	.000878	0	0
1	.001635	.000984	.288494	.077483
2	.002948	.000993	.294288	.073722
3	-.000221	.000662	.322454	.075562
4	.000811	.000586	.319227	.074063
5	.000462	.000333	.322579	.075019
6	.000044	.000275	.323552	.075371
7	.000151	.000162	.323383	.075314
8	.000091	.000114	.323499	.075386

step	(2) oirf	(2) S.E.	(2) fevd	(2) S.E.
0	.005461	.000925	0	0
1	.001578	.000988	.327807	.08159
2	.003307	.001042	.328795	.077519
3	-.00019	.000676	.370775	.080604
4	.000846	.000617	.366896	.079019
5	.000491	.000349	.370399	.079941
6	.000069	.000292	.371487	.080323
7	.000158	.000172	.371315	.080287
8	.000096	.000122	.371438	.080366

(1) irfname = ordera, impulse = dln_inc, and response = dln_consump
(2) irfname = orderb, impulse = dln_inc, and response = dln_consump

The output is displayed as a “single” table; because the table did not fit horizontally, it wrapped automatically. At the bottom of the table is a definition of the keys that appear at the top of each column. The results in the table above indicate that the orthogonalized IRFs do not change by much.



➤ Example 2

Because the estimated FEVDs do change significantly, we might want to produce two tables that contain the estimated FEVDs and their 95% confidence intervals:

```
. irf table fevd, impulse(dln_inc) response(dln_consump) individual
```

Results from ordera

step	(1) fevd	(1) Lower	(1) Upper
0	0	0	0
1	.288494	.13663	.440357
2	.294288	.149797	.43878
3	.322454	.174356	.470552
4	.319227	.174066	.464389
5	.322579	.175544	.469613
6	.323552	.175826	.471277
7	.323383	.17577	.470995
8	.323499	.175744	.471253

95% lower and upper bounds reported

(1) irfname = ordera, impulse = dln_inc, and response = dln_consump

Results from orderb

step	(1) fevd	(1) Lower	(1) Upper
0	0	0	0
1	.327807	.167893	.487721
2	.328795	.17686	.48073
3	.370775	.212794	.528757
4	.366896	.212022	.52177
5	.370399	.213718	.52708
6	.371487	.214058	.528917
7	.371315	.213956	.528674
8	.371438	.213923	.528953

95% lower and upper bounds reported

(1) irfname = orderb, impulse = dln_inc, and response = dln_consump

Because we specified the `individual` option, the output contains two tables, one for each set of IRF results. Examining the results in the tables indicates that each of the estimated functions is well within the confidence interval of the other, so we conclude that the functions are not significantly different.



□ Technical note

Be careful in how you name variables when you fit models. Say that you fit one model with `var` and used time-series operators to form one of the endogenous variables

```
. var d.ln_inv ...
```

and in another model, you created a new variable:

```
. gen dln_inv = d.ln_inv
. var dln_inv ...
```

Say that you saved IRF results from both (perhaps they differ in the number of lags). Now you wish to use `irf table` to compare them. You would not be able to specify `response(d.ln_inv)` or `response(dln_inv)` because neither variable is in both models. Similarly, you could not specify `impulse(d.ln_inv)` or `impulse(dln_inv)` for the same reason.

All is not lost; if `impulse()` is not specified, all endogenous variables are used, and similarly if `response()` is not specified, so you could obtain the result you desired by simply not specifying the options, but you will also obtain a lot more, besides. If you want to specify the `impulse()` or `response()` options, be sure to name variables consistently.

Also, you may forget how the endogenous variables were named. If so, `irf describe, detail` can provide the answer. In `irf describe`'s output, the endogenous variables are listed next to `endog`.



Saved results

If the `individual` option is not specified, `irf table` saves the following in `r()`:

Scalars	
<code>r(ncols)</code>	number of columns in table
<code>r(k_umax)</code>	number of distinct keys
<code>r(k)</code>	number of specific table commands
Macros	
<code>r(key#)</code>	#th key
<code>r(tnotes)</code>	list of keys applied to each column

If the `individual` option is specified, then for each *irfname*, `irf table` saves the following in `r()`:

Scalars	
<code>r(irfname_ncols)</code>	number of columns in table for <i>irfname</i>
<code>r(irfname_k_umax)</code>	number of distinct keys in table for <i>irfname</i>
<code>r(irfname_k)</code>	number of specific table commands used to create table for <i>irfname</i>
Macros	
<code>r(irfname_key#)</code>	#th key for <i>irfname</i> table
<code>r(irfname_tnotes)</code>	list of keys applied to each column in table for <i>irfname</i>

Methods and formulas

`irf table` is implemented as an ado-file.

Also see

- [TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs
- [TS] [var intro](#) — Introduction to vector autoregressive models
- [TS] [vec intro](#) — Introduction to vector error-correction models

Title

newey — Regression with Newey–West standard errors

Syntax

newey *depvar* [*indepvars*] [*if*] [*in*] [*weight*] , lag(#) [*options*]

<i>options</i>	description
Model	
*lag(#)	set maximum lag order of autocorrelation
noconstant	suppress constant term
Reporting	
level(#)	set confidence level; default is level(95)
display_options	control spacing and display of omitted variables and base and empty cells
†coeflegend	display coefficients' legend instead of coefficient table

*lag(#) is required.

†coeflegend does not appear in the dialog box.

You must `tsset` your data before using `newey`; see [TS] `tsset`.

indepvars may contain factor variables; see [U] 11.4.3 Factor variables.

depvar and *indepvars* may contain time-series operators; see [U] 11.4.4 Time-series varlists.

by, *rolling*, and *statsby* are allowed; see [U] 11.1.10 Prefix commands.

*aweight*s are allowed; see [U] 11.1.6 weight.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

Menu

Statistics > Time series > Regression with Newey-West std. errors

Description

`newey` produces Newey–West standard errors for coefficients estimated by OLS regression. The error structure is assumed to be heteroskedastic and possibly autocorrelated up to some lag.

Options

Model

`lag(#)` specifies the maximum lag to be considered in the autocorrelation structure. If you specify `lag(0)`, the output is the same as `regress, vce(robust)`. `lag()` is required.

`noconstant`; see [R] estimation options.

Reporting

`level(#)`; see [R] [estimation options](#).

`display_options`: `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`; see [R] [estimation options](#).

The following option is available with `newey` but is not shown in the dialog box:

`coeflegend`; see [R] [estimation options](#).

Remarks

The Huber/White/sandwich robust variance estimator (see [White \[1980\]](#)) produces consistent standard errors for OLS regression coefficient estimates in the presence of heteroskedasticity. The Newey–West ([1987](#)) variance estimator is an extension that produces consistent estimates when there is autocorrelation in addition to possible heteroskedasticity.

The Newey–West variance estimator handles autocorrelation up to and including a lag of m , where m is specified by stipulating the `lag()` option. Thus, it assumes that any autocorrelation at lags greater than m can be ignored.

If `lag(0)` is specified, the variance estimates produced by `newey` are simply the Huber/White/sandwich robust variances estimates calculated by `regress, vce(robust)`; see [R] [regress](#).

► Example 1

`newey, lag(0)` is equivalent to `regress, vce(robust)`:

```
. use http://www.stata-press.com/data/r11/auto
(1978 Automobile Data)

. regress price weight displ, vce(robust)
```

Linear regression

Number of obs = 74
F(2, 71) = 14.44
Prob > F = 0.0000
R-squared = 0.2909
Root MSE = 2518.4

price	Robust		t	P> t	[95% Conf. Interval]	
	Coef.	Std. Err.				
weight	1.823366	.7808755	2.34	0.022	.2663445	3.380387
displacement	2.087054	7.436967	0.28	0.780	-12.74184	16.91595
_cons	247.907	1129.602	0.22	0.827	-2004.455	2500.269

```
. generate t = _n
. tsset t
time variable: t, 1 to 74
delta: 1 unit
```



```
. newey price weight displ, lag(0)
```

```
Regression with Newey–West standard errors
maximum lag: 0
```

```
Number of obs =      74
F( 2,      71) =    14.44
Prob > F       =    0.0000
```

price	Newey–West		t	P> t	[95% Conf. Interval]	
	Coef.	Std. Err.				
weight	1.823366	.7808755	2.34	0.022	.2663445	3.380387
displacement	2.087054	7.436967	0.28	0.780	-12.74184	16.91595
_cons	247.907	1129.602	0.22	0.827	-2004.455	2500.269

Because `newey` requires the dataset to be `tsset`, we generated a dummy time variable `t`, which in this example played no role in the estimation.

◀

► Example 2

Say that we have time-series measurements on variables `usr` and `idle` and now wish to fit an OLS model but obtain Newey–West standard errors allowing for a lag of up to 3:

```
. use http://www.stata-press.com/data/r11/idle2, clear
```

```
. tsset time
```

```
    time variable:  time, 1 to 30
```

```
        delta: 1 unit
```

```
. newey usr idle, lag(3)
```

```
Regression with Newey–West standard errors
maximum lag: 3
```

```
Number of obs =      30
F( 1,      28) =    10.90
Prob > F       =    0.0026
```

usr	Newey–West		t	P> t	[95% Conf. Interval]	
	Coef.	Std. Err.				
idle	-.2281501	.0690927	-3.30	0.003	-.3696801	-.08662
_cons	23.13483	6.327031	3.66	0.001	10.17449	36.09516

◀

(Continued on next page)

Saved results

`newey` saves the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(df_m)</code>	model degrees of freedom
<code>e(df_r)</code>	residual degrees of freedom
<code>e(F)</code>	F statistic
<code>e(lag)</code>	maximum lag
<code>e(rank)</code>	rank of <code>e(V)</code>

Macros

<code>e(cmd)</code>	<code>newey</code>
<code>e(cmdline)</code>	command as typed
<code>e(depvar)</code>	name of dependent variable
<code>e(wtype)</code>	weight type
<code>e(wexp)</code>	weight expression
<code>e(title)</code>	title in estimation output
<code>e(vctype)</code>	title used to label Std. Err.
<code>e(properties)</code>	<code>b V</code>
<code>e(estat_cmd)</code>	program used to implement <code>estat</code>
<code>e(predict)</code>	program used to implement <code>predict</code>
<code>e(asbalanced)</code>	factor variables <code>fvset</code> as <code>asbalanced</code>
<code>e(asobserved)</code>	factor variables <code>fvset</code> as <code>asobserved</code>

Matrices

<code>e(b)</code>	coefficient vector
<code>e(V)</code>	variance–covariance matrix of the estimators

Functions

<code>e(sample)</code>	marks estimation sample

Methods and formulas

`newey` is implemented as an ado-file.

`newey` calculates the estimates

$$\hat{\beta}_{\text{OLS}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

$$\widehat{\text{Var}}(\hat{\beta}_{\text{OLS}}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\hat{\Omega}\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$$

That is, the coefficient estimates are simply those of OLS linear regression.

For `lag(0)` (no autocorrelation), the variance estimates are calculated using the White formulation:

$$\mathbf{X}'\hat{\Omega}\mathbf{X} = \mathbf{X}'\hat{\Omega}_0\mathbf{X} = \frac{n}{n-k} \sum_i \hat{e}_i^2 \mathbf{x}_i' \mathbf{x}_i$$

Here $\hat{e}_i = y_i - \mathbf{x}_i'\hat{\beta}_{\text{OLS}}$, where $\mathbf{x}_i$ is the i th row of the $\mathbf{X}$ matrix, n is the number of observations, and k is the number of predictors in the model, including the constant if there is one. The above formula is the same as that used by `regress`, `vce(robust)` with the regression-like formula (the default) for the multiplier q_c ; see [Methods and formulas](#) of [R] `regress`.

For $\text{lag}(m)$, $m > 0$, the variance estimates are calculated using the Newey–West (1987) formulation

$$\mathbf{X}'\hat{\Omega}\mathbf{X} = \mathbf{X}'\hat{\Omega}_0\mathbf{X} + \frac{n}{n-k} \sum_{l=1}^m \left(1 - \frac{l}{m+1}\right) \sum_{t=l+1}^n \hat{e}_t \hat{e}_{t-l} (\mathbf{x}'_t \mathbf{x}_{t-l} + \mathbf{x}'_{t-l} \mathbf{x}_t)$$

where $\mathbf{x}_t$ is the row of the $\mathbf{X}$ matrix observed at time t .

Whitney K. Newey (1954–) earned degrees in economics at Brigham Young University and MIT. After a period at Princeton, he returned to MIT as a professor in 1990. His interests in theoretical and applied econometrics include bootstrapping, nonparametric estimation of models, semiparametric models, and choosing the number of instrumental variables.

Kenneth D. West (1953–) earned a bachelor's degree in economics and mathematics at Wesleyan University and then a PhD in economics at MIT. After a period at Princeton, he joined the University of Wisconsin in 1988. His interests include empirical macroeconomics and time-series econometrics.

References

- Hardin, J. W. 1997. [sg72: Newey–West standard errors for probit, logit, and poisson models](#). *Stata Technical Bulletin* 39: 32–35. Reprinted in *Stata Technical Bulletin Reprints*, vol. 7, pp. 182–186. College Station, TX: Stata Press.
- Newey, W. K., and K. D. West. 1987. A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix. *Econometrica* 55: 703–708.
- White, H. 1980. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica* 48: 817–838.

Also see

- [\[TS\] newey postestimation](#) — Postestimation tools for newey
- [\[TS\] tsset](#) — Declare data to be time-series data
- [\[TS\] arima](#) — ARIMA, ARMAX, and other dynamic regression models
- [\[R\] regress](#) — Linear regression
- [\[U\] 20 Estimation and postestimation commands](#)

Description

The following postestimation commands are available for **newey**:

command	description
<code>estat</code>	VCE and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>linktest</code>	link test for model specification
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	point estimates, standard errors, testing, and inference for generalized predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

```
predict [type] newvar [if] [in] [ , statistic]
```

statistic	description
<code>xb</code>	linear prediction; the default
<code>stdp</code>	standard error of the linear prediction
<code>residuals</code>	residuals

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

- `xb`, the default, calculates the linear prediction.
- `stdp` calculates the standard error of the linear prediction.
- `residuals` calculates the residuals.

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Also see

[TS] [newey](#) — Regression with Newey–West standard errors

[U] [20 Estimation and postestimation commands](#)

Syntax

```
pergram varname [if] [in] [, options]
```

<i>options</i>	description
Main	
<u>generate</u> (<i>newvar</i>)	generate <i>newvar</i> to contain the raw periodogram values
Plot	
<i>cline_options</i>	affect rendition of the plotted points connected by lines
<i>marker_options</i>	change look of markers (color, size, etc.)
<i>marker_label_options</i>	add marker labels; change look or position
Add plots	
<u>addplot</u> (<i>plot</i>)	add other plots to the generated graph
Y axis, X axis, Titles, Legend, Overall	
<i>twoway_options</i>	any options other than <code>by()</code> documented in [G] twoway_options
[†] <u>nograph</u>	suppress the graph

[†] `nograph` is not shown in the dialog box.

You must `tsset` your data before using `pergram`; see [\[TS\] tsset](#). Also, the time series must be dense (nonmissing with no gaps in the time variable) in the specified sample.

varname may contain time-series operators; see [\[U\] 11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Graphs > Periodogram

Description

`pergram` plots the log-standardized periodogram for a dense time series.

Options

Main

`generate`(*newvar*) specifies a new variable to contain the raw periodogram values. The generated graph log-transforms and scales the values by the sample variance and then truncates them to the $[-6, 6]$ interval before graphing them.

Plot

cline_options affect the rendition of the plotted points connected by lines; see [G] [cline_options](#).

marker_options specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G] [marker_options](#).

marker_label_options specify if and how the markers are to be labeled; see [G] [marker_label_options](#).

Add plots

`addplot(plot)` adds specified plots to the generated graph; see [G] [addplot_option](#).

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

The following option is available with `pergram` but is not shown in the dialog box:

`nograph` prevents `pergram` from constructing a graph.

Remarks

A good discussion of the periodogram is provided in [Chatfield \(2004\)](#), [Hamilton \(1994\)](#), and [Newton \(1988\)](#). Chatfield is also a good introductory reference for time-series analysis. Another classic reference is [Box, Jenkins, and Reinsel \(1994\)](#). `pergram` produces a scatterplot in which the points of the scatterplot are connected. The points themselves represent the log-standardized periodogram, and the connections between points represent the (continuous) log-standardized sample spectral density.

In the following examples, we present the periodograms with an interpretation of the main features of the plots.

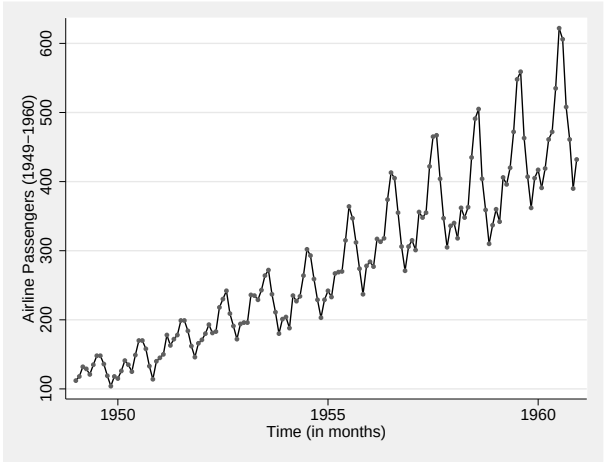
► Example 1

We have time-series data consisting of 144 observations on the monthly number of international airline passengers (in thousands) between 1949 and 1960 ([Box, Jenkins, and Reinsel 1994](#), Series G). We can graph the raw series and the log periodogram for these data by typing

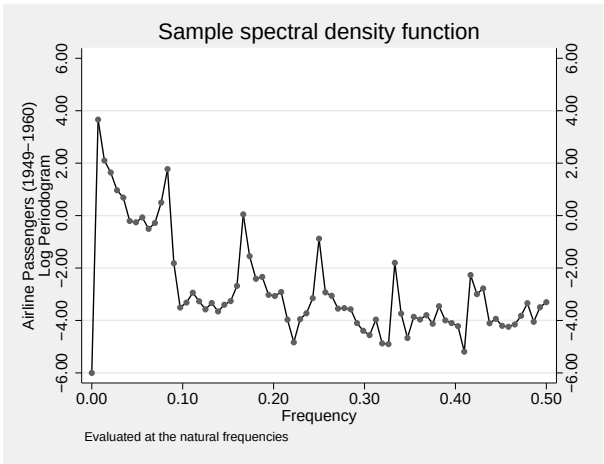
(Continued on next page)

```
. use http://www.stata-press.com/data/r11/air2
(TIMESLAB: Airline passengers)

. scatter air time, m(o) c(1)
```



```
. pergram air
```



The periodogram highlights the annual cycle together with the harmonics. Notice the peak at a frequency of about 0.08 cycles per month (cpm). The period is the reciprocal of frequency, and the reciprocal of 0.08 cpm is approximately 12 months per cycle. The similarity in shape of each group of 12 observations reveals the annual cycle. The magnitude of the cycle is increasing, resulting in the peaks in the periodogram at the harmonics of the principal annual cycle.

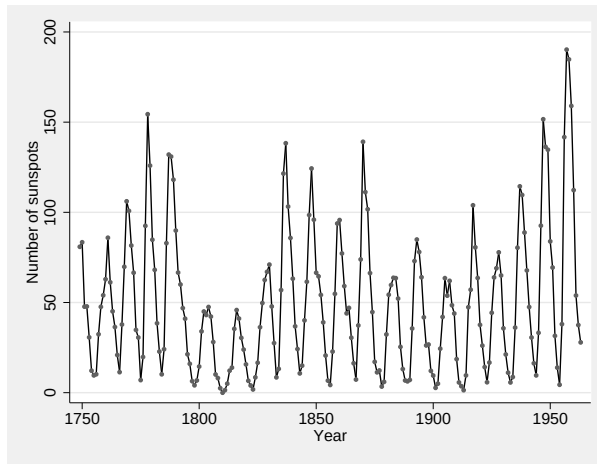


➤ Example 2

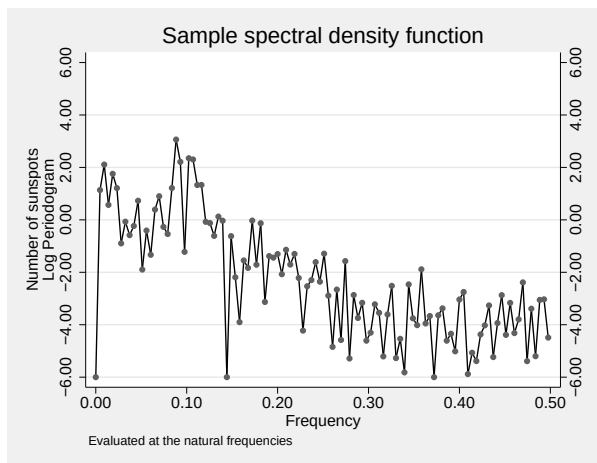
This example uses 215 observations on the annual number of sunspots from 1749 to 1963 (Box and Jenkins 1976, Series E). The graph of the raw series and the log periodogram for these data are given as




```
. use http://www.stata-press.com/data/r11/sunspot
(TIMESLAB: Wolfer sunspot data)
. scatter spot time, m(o) c(1)
```



```
. pergram spot
```



The periodogram peaks at a frequency of slightly less than 0.10 cycles per year, indicating a 10- to 12-year cycle in sunspot activity.

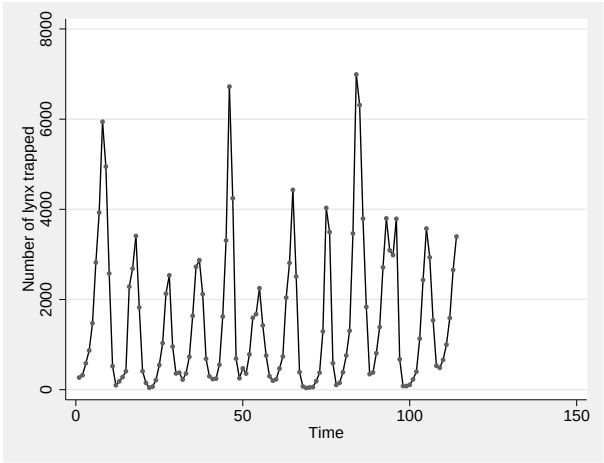


► Example 3

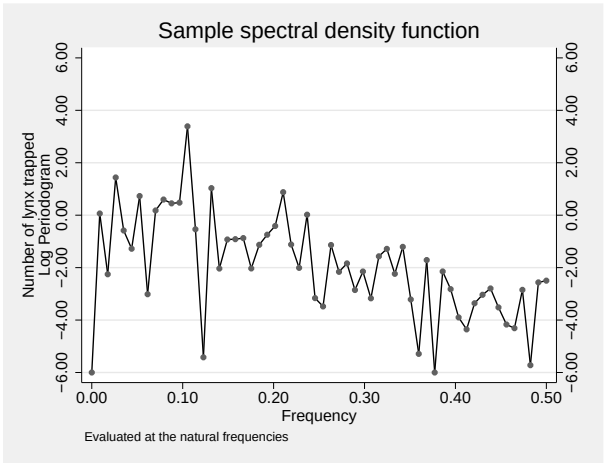
Here we examine the number of trapped Canadian lynx from 1821 through 1934 ([Newton 1988, 587](#)). The raw series and the log periodogram are given as

```
. use http://www.stata-press.com/data/r11/lynx2
(TIMESLAB: Canadian lynx)

. scatter lynx time, m(o) c(1)
```



```
. pergram lynx
```



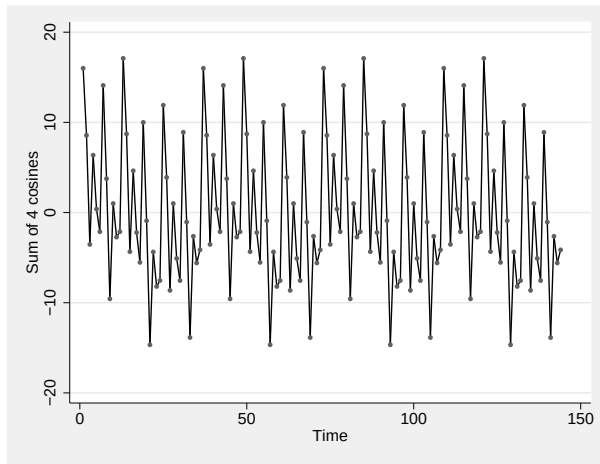
The periodogram indicates that there is a cycle with a duration of about 10 years for these data but that it is otherwise random.



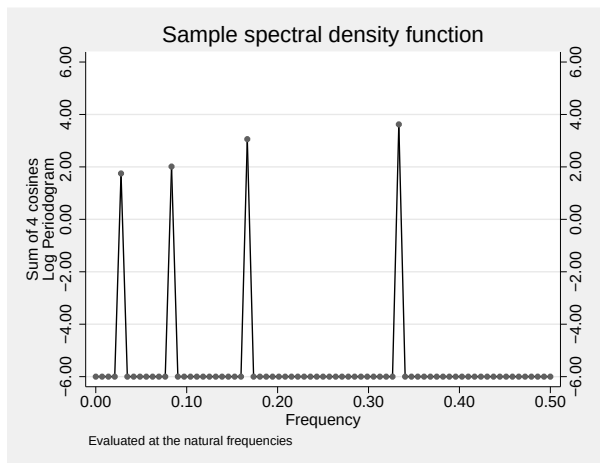
➤ Example 4

To more clearly highlight what the periodogram depicts, we present the result of analyzing a time series of the sum of four sinusoids (of different periods). The periodogram should be able to decompose the time series into four different sinusoids whose periods may be determined from the plot.

```
. use http://www.stata-press.com/data/r11/cos4
(TIMESLAB: Sum of 4 Cosines)
. scatter sumfc time, m(o) c(1)
```



```
. pergram sumfc, gen(ordinate)
```



The periodogram clearly shows the four contributions to the original time series. From the plot, we can see that the periods of the summands were 3, 6, 12, and 36, although you can confirm this by using

(Continued on next page)

```
. generate double omega = (_n-1)/144
. generate double period = 1/omega
(1 missing value generated)
. list period omega if ordinate> 1e-5 & omega <=.5
```

	period	omega
5.	36	.02777778
13.	12	.08333333
25.	6	.16666667
49.	3	.33333333

◀

Methods and formulas

pergram is implemented as an ado-file.

We use the notation of [Newton \(1988\)](#) in the following discussion.

A time series of interest is decomposed into a unique set of sinusoids of various frequencies and amplitudes.

A plot of the sinusoidal amplitudes (ordinates) versus the frequencies for the sinusoidal decomposition of a time series gives us the spectral density of the time series. If we calculate the sinusoidal amplitudes for a discrete set of “natural” frequencies $(1/n, 2/n, \dots, q/n)$, we obtain the periodogram.

Let $x(1), \dots, x(n)$ be a time series, and let $\omega_k = (k - 1)/n$ denote the natural frequencies for $k = 1, \dots, (n/2) + 1$. Define

$$C_k^2 = \frac{1}{n^2} \left| \sum_{t=1}^n x(t) e^{2\pi i(t-1)\omega_k} \right|^2$$

A plot of nC_k^2 versus ω_k is then called the periodogram.

The sample spectral density is defined for a continuous frequency ω as

$$\widehat{f}(\omega) = \begin{cases} \frac{1}{n} \left| \sum_{t=1}^n x(t) e^{2\pi i(t-1)\omega} \right|^2 & \text{if } \omega \in [0, .5] \\ \widehat{f}(1 - \omega) & \text{if } \omega \in [.5, 1] \end{cases}$$

The periodogram (and sample spectral density) is symmetric about $\omega = 0.5$. Further standardize the periodogram such that

$$\frac{1}{n} \sum_{k=2}^n \frac{nC_k^2}{\widehat{\sigma}^2} = 1$$

where $\widehat{\sigma}^2$ is the sample variance of the time series so that the average value of the ordinate is one.

Once the amplitudes are standardized, we may then take the natural log of the values and produce the log periodogram. In doing so, we truncate the graph at ± 6 . We drop the word “log” and simply refer to the “log periodogram” as the “periodogram” in text.

References

- Box, G. E. P., and G. M. Jenkins. 1976. *Time Series Analysis: Forecasting and Control*. Oakland, CA: Holden-Day.
- Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall.
- Chatfield, C. 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Newton, H. J. 1988. *TIMESLAB: A Time Series Analysis Laboratory*. Belmont, CA: Wadsworth.

Also see

- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [corrgram](#) — Tabulate and graph autocorrelations
- [TS] [cumsp](#) — Cumulative spectral distribution
- [TS] [wntestb](#) — Bartlett's periodogram-based test for white noise

Syntax

```
pperron varname [if] [in] [, options]
```

<i>options</i>	description
Main	
<code>noconstant</code>	suppress constant term
<code>trend</code>	include trend term in regression
<code>regress</code>	display regression table
<code>lags(#)</code>	use # Newey–West lags

You must `tsset` your data before using `pperron`; see [\[TS\] `tsset`](#).
varname may contain time-series operators; see [\[U\] 11.4.4 Time-series `varlists`](#).

Menu

Statistics > Time series > Tests > Phillips-Perron unit-root test

Description

`pperron` performs the Phillips–Perron (1988) test that a variable has a unit root. The null hypothesis is that the variable contains a unit root, and the alternative is that the variable was generated by a stationary process. `pperron` uses Newey–West (1987) standard errors to account for serial correlation, whereas the augmented Dickey–Fuller test implemented in `dfuller` uses additional lags of the first-differenced variable.

Options

Main

- `noconstant` suppresses the constant term (intercept) in the model.
- `trend` specifies that a trend term be included in the associated regression. This option may not be specified if `noconstant` is specified.
- `regress` specifies that the associated regression table appear in the output. By default, the regression table is not produced.
- `lags(#)` specifies the number of Newey–West lags to use in calculating the standard error. The default is to use $\text{int} \left\{ 4(T/100)^{2/9} \right\}$ lags.

Remarks

As noted in [TS] [dfuller](#), the Dickey–Fuller test involves fitting the regression model

$$y_t = \alpha + \rho y_{t-1} + \delta t + u_t \quad (1)$$

by ordinary least squares (OLS), but serial correlation will present a problem. To account for this, the augmented Dickey–Fuller test’s regression includes lags of the first differences of y_t .

The Phillips–Perron test involves fitting (1), and the results are used to calculate the test statistics. [Phillips and Perron \(1988\)](#) proposed two alternative statistics, which `pperron` presents. Phillips and Perron’s test statistics can be viewed as Dickey–Fuller statistics that have been made robust to serial correlation by using the Newey–West (1987) heteroskedasticity- and autocorrelation-consistent covariance matrix estimator.

[Hamilton \(1994, chap. 17\)](#) and [TS] [dfuller](#) discuss four different cases into which unit-root tests can be classified. The Phillips–Perron test applies to cases one, two, and four but not to case three. Cases one and two assume that the variable has a unit root without drift under the null hypothesis, the only difference being whether the constant term α is included in regression (1). Case four assumes that the variable has a random walk, with or without drift, under the null hypothesis. Case three, which assumes that the variable has a random walk with drift under the null hypothesis, is just a special case of case four, so the fact that the Phillips–Perron test does not apply is not restrictive. The table below summarizes the relevant cases:

Case	Process under null hypothesis	Regression restrictions	<code>dfuller</code> option
1	Random walk without drift	$\alpha = 0, \delta = 0$	<code>noconstant</code>
2	Random walk without drift	$\delta = 0$	(default)
4	Random walk with or without drift	(none)	<code>trend</code>

The critical values for the Phillips–Perron test are the same as those for the augmented Dickey–Fuller test. See [Hamilton \(1994, chap. 17\)](#) for more information.

► Example 1

Here we use the international airline passengers dataset ([Box, Jenkins, and Reinsel 1994, Series G](#)). This dataset has 144 observations on the monthly number of international airline passengers from 1949 through 1960. Because the data exhibit a clear upward trend over time, we will use the `trend` option.

(Continued on next page)

```
. use http://www.stata-press.com/data/r11/air2
(TIMESLAB: Airline passengers)
. pperron air, lags(4) trend regress
Phillips-Perron test for unit root
```

			Number of obs =	143
			Newey-West lags =	4
	Test Statistic	1% Critical Value	Interpolated Dickey-Fuller 5% Critical Value	10% Critical Value
Z(rho)	-46.405	-27.687	-20.872	-17.643
Z(t)	-5.049	-4.026	-3.444	-3.144

MacKinnon approximate p-value for Z(t) = 0.0002

air	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
air						
L1.	.7318116	.0578092	12.66	0.000	.6175196	.8461035
_trend	.7107559	.1670563	4.25	0.000	.3804767	1.041035
_cons	25.95168	7.325951	3.54	0.001	11.46788	40.43547

Just as in the [example](#) in [\[TS\] dfuller](#), we reject the null hypothesis of a unit root at all common significance levels. The interpolated critical values for Z_t differ slightly from those shown in the example in [\[TS\] dfuller](#) because the sample sizes are different: with the augmented Dickey–Fuller regression we lose observations because of the inclusion of lagged difference terms as regressors.

Saved results

pperron saves the following in `r()`:

Scalars					
<code>r(N)</code>	number of observations	<code>r(Zt)</code>	Phillips–Perron τ test statistic		
<code>r(lags)</code>	number of lagged differences used	<code>r(Zrho)</code>	Phillips–Perron ρ test statistic		
<code>r(pval)</code>	MacKinnon approximate p -value				
	(not included if <code>noconstant</code> specified)				

Methods and formulas

pperron is implemented as an ado-file.

In the OLS estimation of an AR(1) process with Gaussian errors,

$$y_i = \rho y_{i-1} + \epsilon_i$$

where ϵ_i are independently and identically distributed as $N(0, \sigma^2)$ and $y_0 = 0$, the OLS estimate (based on an n -observation time series) of the autocorrelation parameter ρ is given by

$$\hat{\rho}_n = \frac{\sum_{i=1}^n y_{i-1} y_i}{\sum_{i=1}^n y_i^2}$$

If $|\rho| < 1$, then $\sqrt{n}(\hat{\rho}_n - \rho) \rightarrow N(0, 1 - \rho^2)$. If this result were valid for when $\rho = 1$, then the resulting distribution would have a variance of zero. When $\rho = 1$, the OLS estimate $\hat{\rho}$ still converges to one, though we need to find a nondegenerate distribution so that we can test $H_0: \rho = 1$. See [Hamilton \(1994, chap. 17\)](#).

The Phillips–Perron test involves fitting the regression

$$y_i = \alpha + \rho y_{i-1} + \epsilon_i$$

where we may exclude the constant or include a trend term. There are two statistics, Z_ρ and Z_τ , calculated as

$$\begin{aligned} Z_\rho &= n(\hat{\rho}_n - 1) - \frac{1}{2} \frac{n^2 \hat{\sigma}^2}{s_n^2} (\hat{\lambda}_n^2 - \hat{\gamma}_{0,n}) \\ Z_\tau &= \sqrt{\frac{\hat{\gamma}_{0,n} \hat{\rho}_n - 1}{\hat{\lambda}_n^2}} \frac{\hat{\rho}_n - 1}{\hat{\sigma}} - \frac{1}{2} (\hat{\lambda}_n^2 - \hat{\gamma}_{0,n}) \frac{1}{\hat{\lambda}_n} \frac{n \hat{\sigma}}{s_n} \\ \hat{\gamma}_{j,n} &= \frac{1}{n} \sum_{i=j+1}^n \hat{u}_i \hat{u}_{i-j} \\ \hat{\lambda}_n^2 &= \hat{\gamma}_{0,n} + 2 \sum_{j=1}^q \left(1 - \frac{j}{q+1}\right) \hat{\gamma}_{j,n} \\ s_n^2 &= \frac{1}{n-k} \sum_{i=1}^n \hat{u}_i^2 \end{aligned}$$

where u_i is the OLS residual, k is the number of covariates in the regression, q is the number of Newey–West lags to use in calculating $\hat{\lambda}_n^2$, and $\hat{\sigma}$ is the OLS standard error of $\hat{\rho}$.

The critical values, which have the same distribution as the Dickey–Fuller statistic (see [Dickey and Fuller 1979](#)) included in the output, are linearly interpolated from the table of values that appear in [Fuller \(1996\)](#), and the MacKinnon approximate p -values use the regression surface published in [MacKinnon \(1994\)](#).

Peter Charles Bonest Phillips (1948–) was born in Weymouth, England, and earned degrees in economics at the University of Auckland–New Zealand, and the London School of Economics. After periods at the Universities of Essex and Birmingham, Phillips moved to Yale in 1979. He also holds appointments at the University of Auckland and the University of York. His main research interests are in econometric theory, financial econometrics, time-series and panel-data econometrics, and applied macroeconomics.

Pierre Perron (1959–) was born in Québec, Canada, and earned degrees at McGill, Queen’s, and Yale in economics. After posts at Princeton and the Université de Montréal, he joined Boston University in 1997. His research interests include time-series analysis, econometrics, and applied macroeconomics.

References

- Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control*. 3rd ed. Englewood Cliffs, NJ: Prentice–Hall.

- Dickey, D. A., and W. A. Fuller. 1979. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association* 74: 427–431.
- Fuller, W. A. 1996. *Introduction to Statistical Time Series*. 2nd ed. New York: Wiley.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- MacKinnon, J. G. 1994. Approximate asymptotic distribution functions for unit-root and cointegration tests. *Journal of Business and Economic Statistics* 12: 167–176.
- Newey, W. K., and K. D. West. 1987. A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix. *Econometrica* 55: 703–708.
- Phillips, P. C. B., and P. Perron. 1988. Testing for a unit root in time series regression. *Biometrika* 75: 335–346.

Also see

- [TS] **tsset** — Declare data to be time-series data
- [TS] **dfgls** — DF-GLS unit-root test
- [TS] **dfuller** — Augmented Dickey–Fuller unit-root test
- [XT] **xtunitroot** — Panel-data unit-root tests

Syntax

```
prais depvar [indepvars] [if] [in] [, options]
```

<i>options</i>	description
Model	
<code>rtype(<u>r</u>egress)</code>	base ρ on single-lag OLS of residuals; the default
<code>rtype(<u>r</u>eg)</code>	base ρ on single-lead OLS of residuals
<code>rtype(<u>t</u>scorr)</code>	base ρ on autocorrelation of residuals
<code>rtype(<u>d</u>w)</code>	base ρ on autocorrelation based on Durbin–Watson
<code>rtype(<u>t</u>heil)</code>	base ρ on adjusted autocorrelation
<code>rtype(<u>n</u>agar)</code>	base ρ on adjusted Durbin–Watson
<code>corc</code>	use Cochrane–Orcutt transformation
<code>ssesearch</code>	search for ρ that minimizes SSE
<code>twostep</code>	stop after the first iteration
<code><u>n</u>oconstant</code>	suppress constant term
<code><u>h</u>ascons</code>	has user-defined constant
<code><u>s</u>avespace</code>	conserve memory during estimation
SE/Robust	
<code>vce(<u>v</u>cetype)</code>	<i>vcetype</i> may be <code>ols</code> , <code>robust</code> , <code>cluster <i>clustvar</i></code> , <code>hc2</code> , or <code>hc3</code>
Reporting	
<code><u>l</u>evel(#)</code>	set confidence level; default is <code>level(95)</code>
<code><u>n</u>odw</code>	do not report the Durbin–Watson statistic
<code><i>display_options</i></code>	control spacing and display of omitted variables and base and empty cells
Optimization	
<code><i>optimize_options</i></code>	control the optimization process; seldom used
<code>[†]<u>c</u>oeflegend</code>	display coefficients’ legend instead of coefficient table

[†]`coeflegend` does not appear in the dialog box.
You must `tsset` your data before using `prais`; see [TS] `tsset`.
indepvars may contain factor variables; see [U] 11.4.3 Factor variables.
depvar and *indepvars* may contain time-series operators; see [U] 11.4.4 Time-series varlists.
`by`, `rolling`, and `statsby` are allowed; see [U] 11.1.10 Prefix commands.
See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

Menu

Statistics > Time series > Prais-Winsten regression

Description

`prais` uses the generalized least-squares method to estimate the parameters in a linear regression model in which the errors are serially correlated. Specifically, the errors are assumed to follow a first-order autoregressive process.

Options

Model

`rhotype(rhmethod)` selects a specific computation for the autocorrelation parameter ρ , where *rhmethod* can be

<code>regress</code>	$\rho_{\text{reg}} = \beta$ from the residual regression $\epsilon_t = \beta\epsilon_{t-1}$
<code>freg</code>	$\rho_{\text{freg}} = \beta$ from the residual regression $\epsilon_t = \beta\epsilon_{t+1}$
<code>tscorr</code>	$\rho_{\text{tscorr}} = \epsilon'\epsilon_{t-1}/\epsilon'\epsilon$, where ϵ is the vector of residuals
<code>dw</code>	$\rho_{\text{dw}} = 1 - \text{dw}/2$, where <i>dw</i> is the Durbin–Watson <i>d</i> statistic
<code>theil</code>	$\rho_{\text{theil}} = \rho_{\text{tscorr}}(N - k)/N$
<code>nagar</code>	$\rho_{\text{nagar}} = (\rho_{\text{dw}} * N^2 + k^2)/(N^2 - k^2)$

The `prais` estimator can use any consistent estimate of ρ to transform the equation, and each of these estimates meets that requirement. The default is `regress`, which produces the minimum sum-of-squares solution (`ssesearch` option) for the Cochrane–Orcutt transformation—none of these computations will produce the minimum sum-of-squares solution for the full Prais–Winsten transformation. See [Judge et al. \(1985\)](#) for a discussion of each estimate of ρ .

`corc` specifies that the Cochrane–Orcutt transformation be used to estimate the equation. With this option, the Prais–Winsten transformation of the first observation is not performed, and the first observation is dropped when estimating the transformed equation; see [Methods and formulas](#) below.

`ssesearch` specifies that a search be performed for the value of ρ that minimizes the sum-of-squared errors of the transformed equation (Cochrane–Orcutt or Prais–Winsten transformation). The search method is a combination of quadratic and modified bisection searches using golden sections.

`twostep` specifies that `prais` stop on the first iteration after the equation is transformed by ρ —the two-step efficient estimator. Although iterating these estimators to convergence is customary, they are efficient at each step.

`noconstant`; see [\[R\] estimation options](#).

`hascons` indicates that a user-defined constant, or a set of variables that in linear combination forms a constant, has been included in the regression. For some computational concerns, see the discussion in [\[R\] regress](#).

`savespace` specifies that `prais` attempt to save as much space as possible by retaining only those variables required for estimation. The original data are restored after estimation. This option is rarely used and should be used only if there is insufficient space to fit a model without the option.

SE/Robust

`vce(vcetype)` specifies the type of standard error reported, which includes types that are derived from asymptotic theory, that are robust to some kinds of misspecification, and that allow for intragroup correlation; see [\[R\] vce_option](#).

`vce(ols)`, the default, uses the standard variance estimator for ordinary least squares regression.

`prais` also allows the following:

`vce(hc2)` and `vce(hc3)` specify an alternative bias correction for the `vce(robust)` variance calculation; for more information, see [R] [regress](#). You may specify only one of `vce(hc2)`, `vce(hc3)`, or `vce(robust)`.

All estimates from `prais` are conditional on the estimated value of ρ . Robust variance estimates here are robust only to heteroskedasticity and are not generally robust to misspecification of the functional form or omitted variables. The estimation of the functional form is intertwined with the estimation of ρ , and all estimates are conditional on ρ . Thus estimates cannot be robust to misspecification of functional form. For these reasons, it is probably best to interpret `vce(robust)` in the spirit of White's (1980) original paper on estimation of heteroskedastic-consistent covariance matrices.

Reporting

`level(#)`; see [R] [estimation options](#).

`nodw` suppresses reporting of the Durbin–Watson statistic.

`display_options`: `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`; see [R] [estimation options](#).

Optimization

`optimize_options`: `iterate(#)`, `[no]log`, `tolerance(#)`. `iterate()` specifies the maximum number of iterations. `log/nolog` specifies whether to show the iteration log. `tolerance()` specifies the tolerance for the coefficient vector; `tolerance(1e-6)` is the default. These options are seldom used.

The following option is available with `prais` but is not shown in the dialog box:

`coeflegend`; see [R] [estimation options](#).

Remarks

`prais` fits a linear regression of *depvar* on *indepvars* that is corrected for first-order serially correlated residuals by using the Prais–Winsten (1954) transformed regression estimator, the Cochrane–Orcutt (1949) transformed regression estimator, or a version of the search method suggested by Hildreth and Lu (1960). Davidson and MacKinnon (1993) provide theoretical details on the three methods (see pages 333–335 for the latter two and pages 343–351 for Prais–Winsten).

The most common autocorrelated error process is the first-order autoregressive process. Under this assumption, the linear regression model can be written as

$$y_t = \mathbf{x}_t \boldsymbol{\beta} + u_t$$

where the errors satisfy

$$u_t = \rho u_{t-1} + e_t$$

and the e_t are independently and identically distributed as $N(0, \sigma^2)$. The covariance matrix $\boldsymbol{\Psi}$ of the error term e can then be written as

$$\boldsymbol{\Psi} = \frac{1}{1 - \rho^2} \begin{bmatrix} 1 & \rho & \rho^2 & \cdots & \rho^{T-1} \\ \rho & 1 & \rho & \cdots & \rho^{T-2} \\ \rho^2 & \rho & 1 & \cdots & \rho^{T-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^{T-1} & \rho^{T-2} & \rho^{T-3} & \cdots & 1 \end{bmatrix}$$

The Prais–Winsten estimator is a generalized least squares (GLS) estimator. The Prais–Winsten method (as described in [Judge et al. 1985](#)) is derived from the AR(1) model for the error term described above. Whereas the Cochrane–Orcutt method uses a lag definition and loses the first observation in the iterative method, the Prais–Winsten method preserves that first observation. In small samples, this can be a significant advantage.

□ Technical note

To fit a model with autocorrelated errors, you must specify your data as time series and have (or create) a variable denoting the time at which an observation was collected. The data for the regression should be equally spaced in time.

➤ Example 1

Say that we wish to fit a time-series model of `usr` on `idle` but are concerned that the residuals may be serially correlated. We will declare the variable `t` to represent time by typing

```
. use http://www.stata-press.com/data/r11/idle
. tsset t
      time variable:  t, 1 to 30
      delta: 1 unit
```

We can obtain Cochrane–Orcutt estimates by specifying the `corc` option:

```
. prais usr idle, corc
Iteration 0:  rho = 0.0000
Iteration 1:  rho = 0.3518
      (output omitted)
Iteration 13: rho = 0.5708
Cochrane-Orcutt AR(1) regression -- iterated estimates
```

Source	SS	df	MS	Number of obs =	29
Model	40.1309584	1	40.1309584	F(1, 27) =	6.49
Residual	166.898474	27	6.18142498	Prob > F =	0.0168
Total	207.029433	28	7.39390831	R-squared =	0.1938
				Adj R-squared =	0.1640
				Root MSE =	2.4862

usr	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
idle	-.1254511	.0492356	-2.55	0.017	-.2264742 -.024428
_cons	14.54641	4.272299	3.40	0.002	5.78038 23.31245
rho	.5707918				


```
Durbin-Watson statistic (original)    1.295766
Durbin-Watson statistic (transformed) 1.466222
```

The fitted model is

$$\text{usr}_t = -0.1255 \text{ idle}_t + 14.55 + u_t$$

and

$$u_t = 0.5708 u_{t-1} + e_t$$

We can also fit the model with the Prais–Winsten method,

```
. prais usr idle
Iteration 0: rho = 0.0000
Iteration 1: rho = 0.3518
(output omitted)
Iteration 14: rho = 0.5535
Prais-Winsten AR(1) regression -- iterated estimates
```

Source	SS	df	MS	Number of obs =	30
Model	43.0076941	1	43.0076941	F(1, 28) =	7.12
Residual	169.165739	28	6.04163354	Prob > F =	0.0125
				R-squared =	0.2027
				Adj R-squared =	0.1742
Total	212.173433	29	7.31632528	Root MSE =	2.458

usr	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
idle	-.1356522	.0472195	-2.87	0.008	-.2323769 -.0389275
_cons	15.20415	4.160391	3.65	0.001	6.681978 23.72633
rho	.5535476				

```
Durbin-Watson statistic (original) 1.295766
Durbin-Watson statistic (transformed) 1.476004
```

where the Prais–Winsten fitted model is

$$\text{usr}_t = -.1357 \text{idle}_t + 15.20 + u_t \quad \text{and} \quad u_t = .5535 u_{t-1} + e_t$$

As the results indicate, for these data there is little difference between the Cochrane–Orcutt and Prais–Winsten estimators, whereas the OLS estimate of the slope parameter is substantially different.

◀

► Example 2

We have data on quarterly sales, in millions of dollars, for 5 years, and we would like to use this information to model sales for company X. First, we fit a linear model by OLS and obtain the Durbin–Watson statistic by using `estat dwatson`; see [R] [regress postestimation time series](#).

```
. use http://www.stata-press.com/data/r11/qsales
. regress csales isales
```

Source	SS	df	MS	Number of obs =	20
Model	110.256901	1	110.256901	F(1, 18) =	14888.15
Residual	.133302302	18	.007405683	Prob > F =	0.0000
				R-squared =	0.9988
				Adj R-squared =	0.9987
Total	110.390204	19	5.81001072	Root MSE =	.08606

csales	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
isales	.1762828	.0014447	122.02	0.000	.1732475 .1793181
_cons	-1.454753	.2141461	-6.79	0.000	-1.904657 -1.004849

```
. estat dwatson
Durbin-Watson d-statistic( 2, 20) = .7347276
```

Because the Durbin–Watson statistic is far from 2 (the expected value under the null hypothesis of no serial correlation) and well below the 5% lower limit of 1.2, we conclude that the disturbances are serially correlated. (Upper and lower bounds for the d statistic can be found in most econometrics texts; e.g., [Harvey \[1990\]](#). The bounds have been derived for only a limited combination of regressors and observations.) To reinforce this conclusion, we use two other tests to test for serial correlation in the error distribution.

```
. estat bgodfrey, lags(1)
Breusch-Godfrey LM test for autocorrelation
```

lags(p)	chi2	df	Prob > chi2
1	7.998	1	0.0047

H0: no serial correlation

```
. estat durbinalt
Durbin's alternative test for autocorrelation
```

lags(p)	chi2	df	Prob > chi2
1	11.329	1	0.0008

H0: no serial correlation

`estat bgodfrey` reports the Breusch–Godfrey Lagrange-multiplier test statistic, and `estat durbinalt` reports the Durbin’s alternative test statistic. Both tests give a small p -value and thus reject the null hypothesis of no serial correlation. These two tests are asymptotically equivalent when testing for AR(1) process. See [\[R\] regress postestimation time series](#) if you are not familiar with these two tests.

We correct for autocorrelation with the `ssesearch` option of `prais` to search for the value of ρ that minimizes the sum-of-squared residuals of the Cochrane–Orcutt transformed equation. Normally, the default Prais–Winsten transformations is used with such a small dataset, but the less-efficient Cochrane–Orcutt transformation allows us to demonstrate an aspect of the estimator’s convergence.

```
. prais csales isales, corc ssesearch
Iteration 1: rho = 0.8944, criterion = -.07298558
Iteration 2: rho = 0.8944, criterion = -.07298558
(output omitted)
Iteration 15: rho = 0.9588, criterion = -.07167037
Cochrane-Orcutt AR(1) regression -- SSE search estimates
```

Source	SS	df	MS	Number of obs = 19		
Model	2.33199178	1	2.33199178	F(1, 17) = 553.14		
Residual	.071670369	17	.004215904	Prob > F = 0.0000		
				R-squared = 0.9702		
				Adj R-squared = 0.9684		
Total	2.40366215	18	.133536786	Root MSE = .06493		
csales	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
isales	.1605233	.0068253	23.52	0.000	.1461233	.1749234
_cons	1.738946	1.432674	1.21	0.241	-1.283732	4.761624
rho	.9588209					

```
Durbin-Watson statistic (original) 0.734728
Durbin-Watson statistic (transformed) 1.724419
```


We noted in [Options](#) that, with the default computation of ρ , the Cochrane–Orcutt method produces an estimate of ρ that minimizes the sum-of-squared residuals—the same criterion as the `ssesearch` option. Given that the two methods produce the same results, why would the search method ever be preferred? It turns out that the back-and-forth iterations used by Cochrane–Orcutt may have difficulty converging if the value of ρ is large. Using the same data, the Cochrane–Orcutt iterative procedure requires more than 350 iterations to converge, and a higher tolerance must be specified to prevent premature convergence:

```
. prais csales isales, corc tol(1e-9) iterate(500)
Iteration 0:  rho = 0.0000
Iteration 1:  rho = 0.6312
Iteration 2:  rho = 0.6866
(output omitted)
Iteration 377: rho = 0.9588
Iteration 378: rho = 0.9588
Iteration 379: rho = 0.9588
Cochrane-Orcutt AR(1) regression -- iterated estimates
```

Source	SS	df	MS	Number of obs =	19
Model	2.33199171	1	2.33199171	F(1, 17) =	553.14
Residual	.071670369	17	.004215904	Prob > F	= 0.0000
				R-squared	= 0.9702
				Adj R-squared	= 0.9684
Total	2.40366208	18	.133536782	Root MSE	= .06493

csales	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
isales	.1605233	.0068253	23.52	0.000	.1461233 .1749234
_cons	1.738946	1.432674	1.21	0.241	-1.283732 4.761625
rho	.9588209				

```
Durbin-Watson statistic (original)    0.734728
Durbin-Watson statistic (transformed) 1.724419
```

Once convergence is achieved, the two methods produce identical results.

◀

(Continued on next page)

Saved results

`prais` saves the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(N_gaps)</code>	number of gaps
<code>e(mss)</code>	model sum of squares
<code>e(df_m)</code>	model degrees of freedom
<code>e(rss)</code>	residual sum of squares
<code>e(df_r)</code>	residual degrees of freedom
<code>e(r2)</code>	R^2
<code>e(r2_a)</code>	adjusted R^2
<code>e(F)</code>	F statistic
<code>e(rmse)</code>	root mean squared error
<code>e(ll)</code>	log likelihood
<code>e(N_clust)</code>	number of clusters
<code>e(rho)</code>	autocorrelation parameter ρ
<code>e(dw)</code>	Durbin–Watson d statistic for transformed regression
<code>e(dw_0)</code>	Durbin–Watson d statistic of untransformed regression
<code>e(tol)</code>	target tolerance
<code>e(max_ic)</code>	maximum number of iterations
<code>e(ic)</code>	number of iterations

Macros

<code>e(cmd)</code>	<code>prais</code>
<code>e(cmdline)</code>	command as typed
<code>e(depvar)</code>	name of dependent variable
<code>e(title)</code>	title in estimation output
<code>e(clustvar)</code>	name of cluster variable
<code>e(cons)</code>	<code>noconstant</code> or not reported
<code>e(method)</code>	<code>twostep</code> , <code>iterated</code> , or <code>SSE search</code>
<code>e(tranmeth)</code>	<code>corc</code> or <code>prais</code>
<code>e(rhotype)</code>	method specified in <code>rhotype()</code> option
<code>e(vce)</code>	<i>vcetype</i> specified in <code>vce()</code>
<code>e(vcetype)</code>	title used to label Std. Err.
<code>e(properties)</code>	<code>b V</code>
<code>e(predict)</code>	program used to implement <code>predict</code>
<code>e(marginsok)</code>	predictions allowed by <code>margins</code>
<code>e(asbalanced)</code>	factor variables <code>fvset</code> as <code>asbalanced</code>
<code>e(asobserved)</code>	factor variables <code>fvset</code> as <code>asobserved</code>

Matrices

<code>e(b)</code>	coefficient vector
<code>e(V)</code>	variance–covariance matrix of the estimators
<code>e(V_modelbased)</code>	model-based variance

Functions

<code>e(sample)</code>	estimation sample

Methods and formulas

`prais` is implemented as an ado-file.

Consider the command ‘`prais y x z`’. The 0th iteration is obtained by estimating a , b , and c from the standard linear regression:

$$y_t = ax_t + bz_t + c + u_t$$

An estimate of the correlation in the residuals is then obtained. By default, `prais` uses the auxiliary regression:

$$u_t = \rho u_{t-1} + e_t$$

This can be changed to any computation noted in the `rhotype()` option.

Next we apply a Cochrane–Orcutt transformation (1) for observations $t = 2, \dots, n$

$$y_t - \rho y_{t-1} = a(x_t - \rho x_{t-1}) + b(z_t - \rho z_{t-1}) + c(1 - \rho) + v_t \quad (1)$$

and the transformation (1′) for $t = 1$

$$\sqrt{1 - \rho^2} y_1 = a(\sqrt{1 - \rho^2} x_1) + b(\sqrt{1 - \rho^2} z_1) + c\sqrt{1 - \rho^2} + \sqrt{1 - \rho^2} v_1 \quad (1')$$

Thus the differences between the Cochrane–Orcutt and the Prais–Winsten methods are that the latter uses (1′) in addition to (1), whereas the former uses only (1), necessarily decreasing the sample size by one.

Equations (1) and (1′) are used to transform the data and obtain new estimates of a , b , and c .

When the `twostep` option is specified, the estimation process stops at this point and reports these estimates. Under the default behavior of iterating to convergence, this process is repeated until the change in the estimate of ρ is within a specified tolerance.

The new estimates are used to produce fitted values

$$\hat{y}_t = \hat{a}x_t + \hat{b}z_t + \hat{c}$$

and then ρ is reestimated using, by default, the regression defined by

$$y_t - \hat{y}_t = \rho(y_{t-1} - \hat{y}_{t-1}) + u_t \quad (2)$$

We then reestimate (1) by using the new estimate of ρ and continue to iterate between (1) and (2) until the estimate of ρ converges.

Convergence is declared after `iterate()` iterations or when the absolute difference in the estimated correlation between two iterations is less than `tol()`; see [R] [maximize](#). Sargan (1964) has shown that this process will always converge.

Under the `ssesearch` option, a combined quadratic and bisection search using golden sections searches for the value of ρ that minimizes the sum-of-squared residuals from the transformed equation. The transformation may be either the Cochrane–Orcutt (1 only) or the Prais–Winsten (1 and 1′).

All reported statistics are based on the ρ -transformed variables, and ρ is assumed to be estimated without error. See [Judge et al. \(1985\)](#) for details.

The Durbin–Watson d statistic reported by `prais` and `estat dwatson` is

$$d = \frac{\sum_{j=1}^{n-1} (u_{j+1} - u_j)^2}{\sum_{j=1}^n u_j^2}$$

where u_j represents the residual of the j th observation.

This command supports the Huber/White/sandwich estimator of the variance and its clustered version using `vce(robust)` and `vce(cluster clustvar)`, respectively. See [P] [_robust](#), in particular, in [Introduction](#) and [Methods and formulas](#).

All estimates from `prais` are conditional on the estimated value of ρ . Robust variance estimates here are robust only to heteroskedasticity and are not generally robust to misspecification of the functional form or omitted variables. The estimation of the functional form is intertwined with the estimation of ρ , and all estimates are conditional on ρ . Thus estimates cannot be robust to misspecification of functional form. For these reasons, it is probably best to interpret `vce(robust)` in the spirit of White's original paper on estimation of heteroskedastic-consistent covariance matrices.

Acknowledgment

We thank Richard Dickens of the Centre for Economic Performance at the London School of Economics and Political Science for testing and assistance with an early version of this command.

Sigbert Jon Prais (1928–) was born in Frankfurt and moved to Britain in 1934 as a refugee. After earning degrees at the universities of Birmingham and Cambridge and serving in various posts in research and industry, he settled at the National Institute of Economic and Social Research. Prais's interests extend widely across economics, including studies of the influence of education on economic progress.

Christopher Blake Winsten (1923–2005) was born in Welwyn Garden City, England; the son of the writer Stephen Winsten and the painter and sculptress Clare Blake. He was educated at the University of Cambridge and worked with the Cowles Commission at the University of Chicago and at the universities of Oxford, London (Imperial College) and Essex, making many contributions to economics and statistics, including the Prais–Winsten transformation and joint authorship of a celebrated monograph on transportation economics.

Donald Cochrane (1917–1983) was an Australian economist and econometrician. He was born in Melbourne and earned degrees at Melbourne and Cambridge. After wartime service in the Royal Australian Air Force, he held chairs at Melbourne and Monash, being active also in work for various international organizations and national committees.

Guy Henderson Orcutt (1917–) was born in Michigan and earned degrees in physics and economics at the University of Michigan. He worked at Harvard, the University of Wisconsin, and Yale. He has contributed to econometrics and economics in several fields, most distinctively in developing microanalytical models of economic behavior.

References

- Chatterjee, S., and A. S. Hadi. 2006. *Regression Analysis by Example*. 4th ed. New York: Wiley.
- Cochrane, D., and G. H. Orcutt. 1949. Application of least squares regression to relationships containing auto-correlated error terms. *Journal of the American Statistical Association* 44: 32–61.
- Davidson, R., and J. G. MacKinnon. 1993. *Estimation and Inference in Econometrics*. New York: Oxford University Press.
- Durbin, J., and G. S. Watson. 1950. Testing for serial correlation in least squares regression. I. *Biometrika* 37: 409–428.
- . 1951. Testing for serial correlation in least squares regression. II. *Biometrika* 38: 159–177.
- Hardin, J. W. 1995. [stsl0: Prais–Winsten regression](#). *Stata Technical Bulletin* 25: 26–29. Reprinted in *Stata Technical Bulletin Reprints*, vol. 5, pp. 234–237. College Station, TX: Stata Press.
- Harvey, A. C. 1990. *The Econometric Analysis of Time Series*. 2nd ed. Cambridge, MA: MIT Press.
- Hildreth, C., and J. Y. Lu. 1960. Demand relations with autocorrelated disturbances. Reprinted in *Agricultural Experiment Station Technical Bulletin*, No. 276. East Lansing, MI: Michigan State University Press.
- Judge, G. G., W. E. Griffiths, R. C. Hill, H. Lütkepohl, and T.-C. Lee. 1985. *The Theory and Practice of Econometrics*. 2nd ed. New York: Wiley.
- King, M. L., and D. E. A. Giles, ed. 1987. *Specification Analysis in the Linear Model: Essays in Honor of Donald Cochrane*. London: Routledge & Kegan Paul.
- Kmenta, J. 1997. *Elements of Econometrics*. 2nd ed. Ann Arbor: University of Michigan Press.
- Prais, S. J., and C. B. Winsten. 1954. Trend estimators and serial correlation. Working paper 383, Cowles Commission. <http://cowles.econ.yale.edu/P/ccdp/st/s-0383.pdf>.
- Sargan, J. D. 1964. Wages and prices in the United Kingdom: A study in econometric methodology. In *Econometric Analysis for National Economic Planning*, ed. P. E. Hart, G. Mills, and J. K. Whitaker, 25–64. London: Butterworths.
- Theil, H. 1971. *Principles of Econometrics*. New York: Wiley.
- White, H. 1980. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica* 48: 817–838.
- Wooldridge, J. M. 2009. *Introductory Econometrics: A Modern Approach*. 4th ed. Cincinnati, OH: South-Western.
- Zellner, A. 1990. Guy H. Orcutt: Contributions to economic statistics. *Journal of Economic Behavior and Organization* 14: 43–51.

Also see

- [TS] [prais postestimation](#) — Postestimation tools for prais
- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [arima](#) — ARIMA, ARMAX, and other dynamic regression models
- [R] [regress](#) — Linear regression
- [R] [regress postestimation time series](#) — Postestimation tools for regress with time series
- [U] [20 Estimation and postestimation commands](#)

Description

The following postestimation commands are available for **prais**:

command	description
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>linktest</code>	link test for model specification
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	point estimates, standard errors, testing, and inference for generalized predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

```
predict [type] newvar [if] [in] [, statistic]
```

statistic	description
<code>xb</code>	linear prediction; the default
<code>stdp</code>	standard error of the linear prediction
<code>residuals</code>	residuals

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

`xb`, the default, calculates the fitted values—the prediction of $\mathbf{x}_j\mathbf{b}$ for the specified equation. This is the linear predictor from the fitted regression model; it does not apply the estimate of ρ to prior residuals.

`stdp` calculates the standard error of the prediction for the specified equation, i.e., the standard error of the predicted expected value or mean for the observation's covariate pattern. The standard error of the prediction is also referred to as the standard error of the fitted value.

As computed for `prais`, this is strictly the standard error from the variance in the estimates of the parameters of the linear model and assumes that ρ is estimated without error.

`residuals` calculates the residuals from the linear prediction.

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Also see

[TS] [prais](#) — Prais–Winsten and Cochrane–Orcutt regression

[U] [20 Estimation and postestimation commands](#)

Syntax

rolling [*exp_list*] [*if*] [*in*] [, *options*] : *command*

<i>options</i>	<i>description</i>
Main	
* <u>w</u> indow(<i>#</i>)	number of consecutive data points in each sample
<u>r</u> ecursive	use recursive samples
<u>r</u> recursive	use reverse recursive samples
Options	
<u>c</u> lear	replace data in memory with results
<u>s</u> aving(<i>filename</i> , ...)	save results to <i>filename</i> ; save statistics in double precision; save results to <i>filename</i> every # replications
<u>s</u> tepsize(<i>#</i>)	number of periods to advance window
<u>s</u> tart(<i>time_constant</i>)	period at which rolling is to start
<u>e</u> nd(<i>time_constant</i>)	period at which rolling is to end
<u>k</u> keep(<i>varname</i> [, start])	save <i>varname</i> along with results; optionally, use value at left edge of window
Reporting	
<u>n</u> odots	suppress the replication dots
<u>n</u> oisily	display any output from <i>command</i>
<u>t</u> race	trace <i>command</i> 's execution
Advanced	
<u>r</u> eject(<i>exp</i>)	identify invalid results

* `window(#)` is required.

You must `tsset` your data before using `rolling`; see [TS] [tsset](#).

`aweights` are allowed in *command* if *command* accepts `aweights`; see [U] [11.1.6 weight](#).

<i>exp_list</i> contains	(<i>name</i> : <i>elist</i>) <i>elist</i> <i>eexp</i>
<i>elist</i> contains	<i>newvar</i> = (<i>exp</i>) (<i>exp</i>)
<i>eexp</i> is	<i>specname</i> [<i>eqno</i>] <i>specname</i>
<i>specname</i> is	<code>_b</code> <code>_b[]</code> <code>_se</code> <code>_se[]</code>

eqno is **##**
 name

exp is a standard Stata expression; see [U] 13 Functions and expressions.

Distinguish between [], which are to be typed, and [], which indicate optional arguments.

Menu

Statistics > Time series > Rolling-window and recursive estimation

Description

rolling is a moving sampler that collects statistics from *command* after executing *command* on subsets of the data in memory. Typing

```
. rolling exp_list, window(50) clear: command
```

executes *command* on sample windows of span 50. That is, **rolling** will first execute *command* by using periods 1–50 of the dataset, and then using periods 2–51, 3–52, and so on. **rolling** can also perform recursive and reverse recursive analyses, in which the starting or ending period is held fixed and the window size grows.

command defines the statistical command to be executed. Most Stata commands and user-written programs can be used with **rolling**, as long as they follow standard Stata syntax and allow the if qualifier; see [U] 11 Language syntax. The by prefix cannot be part of *command*.

exp_list specifies the statistics to be collected from the execution of *command*. If no expressions are given, *exp_list* assumes a default of `_b` if *command* stores results in `e()` and of all the scalars if *command* stores results in `r()` and not in `e()`. Otherwise, not specifying an expression in *exp_list* is an error.

Options

Main

window(#) defines the window size used each time *command* is executed. The window size refers to calendar periods, not the number of observations. If there are missing data (for example, because of weekends), the actual number of observations used by *command* may be less than **window(#)**. **window(#)** is required.

recursive specifies that a recursive analysis be done. The starting period is held fixed, the ending period advances, and the window size grows.

rrecursive specifies that a reverse recursive analysis be done. Here the ending period is held fixed, the starting period advances, and the window size shrinks.

Options

clear specifies that Stata replace the data in memory with the collected statistics even though the current data in memory have not been saved to disk.

saving(filename[, suboptions]) creates a Stata data file (.dta file) consisting of, for each statistic in *exp_list*, a variable containing the window replicates.

`double` specifies that the results for each replication be stored as `doubles`, meaning 8-byte reals. By default, they are stored as `floats`, meaning 4-byte reals.

`every(#)` specifies that results be written to disk every `#`th replication. `every()` should be specified in conjunction only with `saving()` when *command* takes a long time for each replication. This will allow recovery of partial results should your computer crash. See [P] [postfile](#).

`stepsize(#)` specifies the number of periods the window is to be advanced each time *command* is executed.

`start(time_constant)` specifies the date on which `rolling` is to start. `start()` may be specified as an integer or as a date literal.

`end(time_constant)` specifies the date on which `rolling` is to end. `end()` may be specified as an integer or as a date literal.

`keep(varname[, start])` specifies a variable to be posted along with the results. The value posted is the value that corresponds to the right edge of the window. Specifying the `start()` option requests that the value corresponding to the left edge of the window be posted instead. This option is often used to record calendar dates.

Reporting

`nodots` suppresses display of the replication dot for each window on which *command* is executed. By default, one dot character is printed for each window. A red 'x' is printed if *command* returns with an error or if any value in *exp_list* is missing.

`noisily` causes the output of *command* to be displayed for each window on which *command* is executed. This option implies the `nodots` option.

`trace` causes a trace of the execution of *command* to be displayed. This option implies the `noisily` and `nodots` options.

Advanced

`reject(exp)` identifies an expression that indicates when results should be rejected. When *exp* is true, the saved statistics are set to missing values.

Remarks

`rolling` executes a command on each of a series of windows of observations and stores the results. `rolling` can perform what are commonly called rolling regressions, recursive regressions, and reverse recursive regressions. However, `rolling` is not limited to just linear regression analysis: any command that saves results in `e()` or `r()` can be used with `rolling`.

Suppose that you have data collected at 100 consecutive points in time, numbered 1–100, and you wish to perform a rolling regression with a window size of 20 periods. Typing

```
. rolling _b, window(20) clear: regress depvar indepvar
```

causes Stata to `regress depvar` on *indepvar* using periods 1–20, store the regression coefficients (`_b`), run the regression using periods 2–21, and so on, finishing with a regression using periods 81–100 (the last 20 periods).

The `stepsize()` option specifies how far ahead the window is moved each time. For example, if you specify `step(2)`, then *command* is executed on periods 1–20, and then 3–22, 5–24, etc. By default, `rolling` replaces the dataset in memory with the computed statistics unless the `saving()` option is specified, in which case the computed statistics are stored in the filename specified. If the

dataset in memory has been changed since it was last saved and you do not specify `saving()`, you must use `clear`.

`rolling` can also perform recursive and reverse recursive analyses. In a recursive analysis, the starting date is held fixed, and the window size grows as the ending date is advanced. In a reverse recursive analysis, the ending date is held fixed, and the window size shrinks as the starting date is advanced.

► Example 1

We have data on the daily returns to IBM stock (`ibm`), the S&P 500 (`spx`), and short-term interest rates (`irx`), and we want to create a series containing the beta of IBM by using the previous 200 trading days at each date. We will also record the standard errors, so that we can obtain 95% confidence intervals for the betas. See, for example, [Stock and Watson \(2007, 122\)](#) for more information on estimating betas. We type

```
. use http://www.stata-press.com/data/r11/ibm
(Source: Yahoo! Finance)
. tsset t
      time variable:  t, 1 to 494
              delta:  1 unit
. generate ibmadj = ibm - irx
(1 missing value generated)
. generate spxadj = spx - irx
(1 missing value generated)
. rolling _b _se, window(200) saving(betas, replace) keep(date): regress ibmadj
> spxadj
(running regress on estimation sample)
(note: file betas.dta not found)
Rolling replications (295)
-----|----- 1 -----|----- 2 -----|----- 3 -----|----- 4 -----|----- 5
.....
..... 50
..... 100
..... 150
..... 200
..... 250
.....
file betas.dta saved
```

Our dataset has both a time variable `t` that runs consecutively and a date variable `date` that measures the calendar date and therefore has gaps at weekends and holidays. Had we used the `date` variable as our time variable, `rolling` would have used windows consisting of 200 calendar days instead of 200 trading days, and each window would not have exactly 200 observations. We used the `keep(date)` option so that we could refer to the `date` variable when working with the results dataset.

We can list a portion of the dataset created by `rolling` to see what it contains:

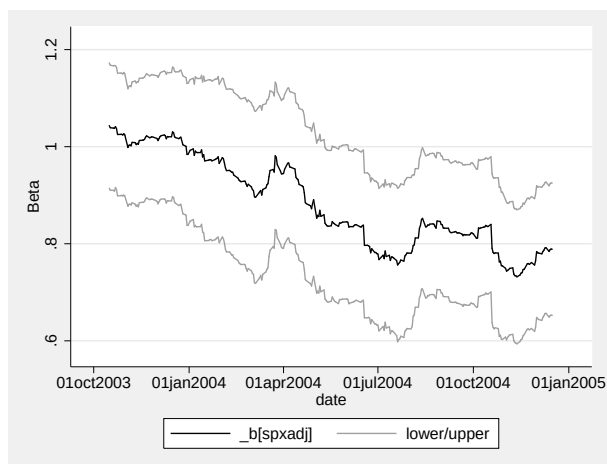
```
. use betas, clear
(rolling: regress)
. sort date
. list in 1/3, abbrev(10)
```

	start	end	date	_b_spxadj	_b_cons	_se_spxadj	_se_cons
1.	1	200	16oct2003	1.043422	-.0181504	.0658531	.0748295
2.	2	201	17oct2003	1.039024	-.0126876	.0656893	.074609
3.	3	202	20oct2003	1.038371	-.0235616	.0654591	.0743851

The variables `start` and `end` indicate the first and last observations used each time that `rolling` called `regress`, and the `date` variable contains the calendar date corresponding the period represented by `end`. The remaining variables are the estimated coefficients and standard errors from the regression. In our example, `_b_spxadj` contains the estimated betas, and `_b_cons` contains the estimated alphas. The variables `_se_spxadj` and `_se_cons` have the corresponding standard errors.

Finally, we compute the confidence intervals for the betas and examine how they have changed over time:

```
. generate lower = _b_spxadj - 1.96*_se_spxadj
. generate upper = _b_spxadj + 1.96*_se_spxadj
. twoway (line _b_spxadj date) (rline lower upper date) if date>=td(1oct2003),
> ytitle("Beta")
```



As 2004 progressed, IBM's stock returns were less influenced by returns in the broader market. Beginning in June of 2004, IBM's beta became significantly different from unity at the 95% confidence level, as indicated by the fact that the confidence interval does not contain one from then onward.



In addition to rolling-window analyses, `rolling` can also perform recursive ones. Suppose again that you have data collected at 100 consecutive points in time, and now you type

```
. rolling _b, window(20) recursive clear: regress depvar indepvar
```

Stata will first `regress depvar` on `indepvar` by using observations 1–20, store the coefficients, run the regression using observations 1–21, observations 1–22, and so on, finishing with a regression using all 100 observations. Unlike a rolling regression, in which case the number of observations is held constant and the starting and ending points are shifted, a recursive regression holds the starting point fixed and increases the number of observations. Recursive analyses are often used in forecasting situations. As time goes by, more information becomes available that can be used in making forecasts. See [Kmenta \(1997, 423–424\)](#).

► Example 2

Using the same dataset, we type

```
. use http://www.stata-press.com/data/r11/ibm, clear
(Source: Yahoo! Finance)
. tsset t
      time variable:  t, 1 to 494
              delta:  1 unit
. generate ibmadj = ibm - irx
(1 missing value generated)
. generate spxadj = spx - irx
(1 missing value generated)
. rolling _b _se, recursive window(200) clear: regress ibmadj spxadj
(output omitted)
. list in 1/3, abbrev(10)
```

	start	end	_b_spxadj	_b_cons	_se_spxadj	_se_cons
1.	1	200	1.043422	-.0181504	.0658531	.0748295
2.	1	201	1.039024	-.0126876	.0656893	.074609
3.	1	202	1.037687	-.016475	.0655896	.0743481

Here the starting period remains fixed and the window grows larger.

◀

In a reverse recursive analysis, the ending date is held fixed, and the window size becomes smaller as the starting date is advanced. For example, with a dataset that has observations numbered 1–100, typing

```
. rolling _b, window(20) reverse recursive clear: regress depvar indepvar
```

creates a dataset in which the first observation has the results based on periods 1–100, the second observation has the results based on 2–100, the third having 3–100, and so on, up to the last observation having results based on periods 81–100 (the last 20 observations).

► Example 3

Using the data on stock returns, we want to build a model in which we predict today's IBM stock return on the basis of yesterday's returns on IBM and the S&P 500. That is, letting i_t and s_t denote the returns to IBM and the S&P 500 on date t , we want to fit the regression model

$$i_t = \beta_0 + \beta_1 i_{t-1} + \beta_2 s_{t-1} + \epsilon_t$$

where ϵ_t is a regression error term, and then compute

$$\widehat{i_{t+1}} = \widehat{\beta_0} + \widehat{\beta_1} i_t + \widehat{\beta_2} s_t$$

We will use recursive regression because we suspect that the more data we have to fit the regression model, the better the model will predict returns. We will use at least 20 periods in fitting the regression.

```
. use http://www.stata-press.com/data/r11/ibm, clear
(Source: Yahoo! Finance)
. tsset t
      time variable:  t, 1 to 494
              delta:  1 unit
```

One alternative would be to use `rolling` with the `recursive` option to fit the regressions, collect the coefficients, and then compute the predicted values afterward. However, we will instead write a short program that computes the forecasts automatically and then use `rolling`, `recursive` on that program. The program must accept an `if` expression so that `rolling` can indicate to the program which observations are to be used. Our program is

```
program myforecast, rclass
    syntax [if]
    regress ibm L.ibm L.spx `if'
    // Find last time period of estimation sample and
    // make forecast for period just after that
    summ t if e(sample)
    local last = r(max)
    local fcast = _b[_cons] + _b[L.ibm]*ibm['last'] + ///
                 _b[L.spx]*spx['last']
    return scalar forecast = 'fcast'
    // Next period's actual return
    // Will return missing value for final period
    return scalar actual = ibm['last'+1]
end
```

Now we call `rolling`:

```
. rolling actual=r(actual) forecast=r(forecast), recursive window(20): myforecast
(output omitted)
. corr actual forecast
(obs=474)
```

	actual forecast	
actual	1.0000	
forecast	-0.0957	1.0000

Our model does not work too well—the correlation between actual returns and our forecasts is negative.



Saved results

`rolling` sets no `r-` or `e-`class macros. The results from the command used with `rolling`, depending on the last window of data used, are available after `rolling` has finished.

Methods and formulas

`rolling` is implemented as an ado-file.

Acknowledgment

We thank Christopher F. Baum for an earlier `rolling` regression command.

References

Kmenta, J. 1997. *Elements of Econometrics*. 2nd ed. Ann Arbor: University of Michigan Press.

Stock, J. H., and M. W. Watson. 2007. *Introduction to Econometrics*. 2nd ed. Boston: Addison–Wesley.

Also see

[D] [statsby](#) — Collect statistics for a command across a by list

[R] [saved results](#) — Saved results

Syntax

Covariance-form syntax

```
sspace state_ceq [state_ceq ... state_ceq] obs_ceq [obs_ceq ... obs_ceq]
[if] [in] [, options]
```

where each *state_ceq* is of the form

```
(statevar [lagged_statevars] [indepvars], state [noerror noconstant])
```

and each *obs_ceq* is of the form

```
(depvar [statevars] [indepvars] [, noerror noconstant])
```

Error-form syntax

```
sspace state_efeq [state_efeq ... state_efeq] obs_efeq [obs_efeq ... obs_efeq]
[if] [in] [, options]
```

where each *state_efeq* is of the form

```
(statevar [lagged_statevars] [indepvars] [state_errors], state [noconstant])
```

and each *obs_efeq* is of the form

```
(depvar [statevars] [indepvars] [obs_errors] [, noconstant])
```

statevar is the name of an unobserved state, not a variable. If there happens to be a variable of the same name, the variable is ignored and plays no role in the estimation.

lagged_statevars is a list of lagged *statevars*. Only first lags are allowed.

state_errors is a list of state-equation errors that enter a state equation. Each state error has the form *e.statevar*, where *statevar* is the name of a state in the model.

obs_errors is a list of observation-equation errors that enter an equation for an observed variable. Each error has the form *e.depvar*, where *depvar* is an observed dependent variable in the model.

equation-level options	description
Model	
state	specifies that the equation is a state equation
noerror	specifies that there is no error term in the equation
noconstant	suppresses the constant term from the equation

<i>options</i>	description
Model	
<u>covstate</u> (<i>covform</i>)	specifies the covariance structure for the errors in the state variables
<u>covobserved</u> (<i>covform</i>)	specifies the covariance structure for the errors in the observed dependent variables
<u>constraints</u> (<i>constraints</i>)	apply linear constraints
SE/Robust	
<u>vce</u> (<i>vcetype</i>)	<i>vcetype</i> may be <code>oim</code> or <code>robust</code>
Reporting	
<u>level</u> (#)	set confidence level; default is <code>level(95)</code>
<u>nocnsreport</u>	do not display constraints
<u>display_options</u>	control spacing and display of omitted variables and base and empty cells
Maximization	
<u>maximize_options</u>	control the maximization process; seldom used
<u>from</u> (<i>matname</i>)	specify initial values for the maximization process; seldom used
Advanced	
<u>method</u> (<i>method</i>)	specify the method for calculating the log likelihood; seldom used
[†] <u>coeflegend</u>	display coefficients' legend instead of coefficient table
<i>covform</i>	description
<u>identity</u>	identity matrix; the default for error-form syntax
<u>dscalar</u>	diagonal scalar matrix
<u>diagonal</u>	diagonal matrix; the default for covariance-form syntax
<u>unstructured</u>	symmetric, positive-definite matrix; not allowed with error-form syntax
<i>method</i>	description
<u>hybrid</u>	use the stationary Kalman filter and the De Jong diffuse Kalman filter; the default
<u>dejong</u>	use the stationary De Jong Kalman filter and the De Jong diffuse Kalman filter
<u>kdiffuse</u>	use the stationary Kalman filter and the nonstationary large- κ diffuse Kalman filter; seldom used

You must `tsset` your data before using `sspace`; see [TS] [tsset](#).

[†]`coeflegend` does not appear in the dialog box.

indepvars may contain factor variables; see [U] [11.4.3 Factor variables](#).

indepvars and *depvar* may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

`by`, `statsby`, and `rolling` are allowed; see [U] [11.1.10 Prefix commands](#).

See [U] [20 Estimation and postestimation commands](#) for more capabilities of estimation commands.

Description

sspace estimates the parameters of linear state-space models by maximum likelihood. Linear state-space models are very flexible and many linear time-series models can be written as linear state-space models.

sspace uses two forms of the Kalman filter to recursively obtain conditional means and variances of both the unobserved states and the measured dependent variables that are used to compute the likelihood.

The covariance-form syntax and the error-form syntax of **sspace** reflect the two different forms in which researchers specify state-space models. Choose the syntax that is easier for you; the two forms are isomorphic.

Options

Equation-level options

Model

state specifies that the equation is a state equation.

noerror specifies that there is no error term in the equation. **noerror** may not be specified in the error-form syntax.

noconstant suppresses the constant term from the equation.

Options

Model

covstate(covform) specifies the covariance structure for the state errors.

covstate(identity) specifies a covariance matrix equal to an identity matrix, and it is the default for the error-form syntax.

covstate(dscalar) specifies a covariance matrix equal to σ_{state}^2 times an identity matrix.

covstate(diagonal) specifies a diagonal covariance matrix, and it is the default for the covariance-form syntax.

covstate(unstructured) specifies a symmetric, positive-definite covariance matrix with parameters for all variances and covariances. **covstate(unstructured)** may not be specified with the error-form syntax.

covobserved(covform) specifies the covariance structure for the observation errors.

covobserved(identity) specifies a covariance matrix equal to an identity matrix, and it is the default for the error-form syntax.

covobserved(dscalar) specifies a covariance matrix equal to $\sigma_{\text{observed}}^2$ times an identity matrix.

covobserved(diagonal) specifies a diagonal covariance matrix, and it is the default for the covariance-form syntax.

covobserved(unstructured) specifies a symmetric, positive-definite covariance matrix with parameters for all variances and covariances. **covobserved(unstructured)** may not be specified with the error-form syntax.

constraints(constraints) specifies linear constraints to apply to the parameter estimates.

SE/Robust

`vce(vctype)` specifies the estimator for the variance–covariance matrix of the estimator.

`vce(oim)`, the default, causes `sspace` to use the observed information matrix estimator.

`vce(robust)` causes `sspace` to use the Huber/White/sandwich estimator.

Reporting

`level(#)`; see [R] [estimation options](#).

`nocnsreport`; see [R] [estimation options](#).

`display_options`: `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`; see [R] [estimation options](#).

Maximization

`maximize_options`: `difficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, `from(matname)`; see [R] [maximize](#) for all options except `from()`, and see below for information on `from()`. These options are seldom used.

`from(matname)` specifies initial values for the maximization process. `from(b0)` causes `sspace` to begin the maximization algorithm with the values in `b0`. `b0` must be a row vector; the number of columns must equal the number of parameters in the model; and the values in `b0` must be in the same order as the parameters in `e(b)`. This option is seldom used.

Advanced

`method(method)` specifies how to compute the log likelihood. This option is seldom used.

`method(hybrid)`, the default, uses the Kalman filter with model-based initial values for the states when the model is stationary and uses the [De Jong \(1988, 1991\)](#) diffuse Kalman filter when the model is nonstationary.

`method(dejong)` uses the Kalman filter with the [De Jong \(1988\)](#) method for estimating the initial values for the states when the model is stationary and uses the [De Jong \(1988, 1991\)](#) diffuse Kalman filter when the model is nonstationary.

`method(kdiffuse)` is a seldom used method that uses the Kalman filter with model-based initial values for the states when the model is stationary and uses the large- κ diffuse Kalman filter when the model is nonstationary.

The following option is available with `sspace` but is not shown in the dialog box:

`coeflegend`; see [R] [estimation options](#).

Remarks

Remarks are presented under the following headings:

An introduction to state-space models
Some stationary state-space models
Some nonstationary state-space models

An introduction to state-space models

Many linear time-series models can be written as linear state-space models, including vector autoregressive moving-average (VARMA) models, dynamic-factor (DF) models, and structural time-series (STS) models. The solutions to some stochastic dynamic-programming problems can also be written in the form of linear state-space models. We can estimate the parameters of a linear state-space model by maximum likelihood (ML). The Kalman filter or a diffuse Kalman filter is used to write the likelihood function in prediction-error form, assuming normally distributed errors. The quasi-maximum likelihood (QML) estimator, which drops the normality assumption, is consistent and asymptotically normal when the model is stationary. [Chang, Miller, and Park \(2009\)](#) establish consistency and asymptotic normality of the QML estimator for a class of nonstationary state-space models. The QML estimator differs from the ML estimator only in the VCE; specify the `vce(robust)` option to obtain the QML estimator.

[Hamilton \(1994a, 1994b\)](#), [Harvey \(1989\)](#), and [Brockwell and Davis \(1991\)](#) provide good introductions to state-space models. [Anderson and Moore's \(1979\)](#) text is a classic reference; they produced many results used subsequently. [Caines \(1988\)](#) and [Hannan and Deistler \(1988\)](#) provide excellent, more advanced, treatments.

`sspace` estimates linear state-space models with time-invariant coefficient matrices, which cover the models listed above and many others. `sspace` can estimate parameters from state-space models of the form

$$\begin{aligned}\mathbf{z}_t &= \mathbf{A}\mathbf{z}_{t-1} + \mathbf{B}\mathbf{x}_t + \mathbf{C}\boldsymbol{\epsilon}_t \\ \mathbf{y}_t &= \mathbf{D}\mathbf{z}_t + \mathbf{F}\mathbf{w}_t + \mathbf{G}\boldsymbol{\nu}_t\end{aligned}$$

where

$\mathbf{z}_t$ is an $m \times 1$ vector of unobserved state variables;

$\mathbf{x}_t$ is a $k_x \times 1$ vector of exogenous variables;

$\boldsymbol{\epsilon}_t$ is a $q \times 1$ vector of state-error terms, ($q \leq m$);

$\mathbf{y}_t$ is an $n \times 1$ vector of observed endogenous variables;

$\mathbf{w}_t$ is a $k_w \times 1$ vector of exogenous variables;

$\boldsymbol{\nu}_t$ is an $r \times 1$ vector of observation-error terms, ($r \leq n$); and

$\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{F}$, and $\mathbf{G}$ are parameter matrices.

The equations for $\mathbf{z}_t$ are known as the state equations, and the equations for $\mathbf{y}_t$ are known as the observation equations.

The error terms are assumed to be zero mean, normally distributed, serially uncorrelated, and uncorrelated with each other;

$$\begin{aligned}\boldsymbol{\epsilon}_t &\sim N(0, \mathbf{Q}) \\ \boldsymbol{\nu}_t &\sim N(0, \mathbf{R}) \\ E[\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_s'] &= \mathbf{0} \text{ for all } s \neq t \\ E[\boldsymbol{\epsilon}_t \boldsymbol{\nu}_s'] &= \mathbf{0} \text{ for all } s \text{ and } t\end{aligned}$$

The state-space form is used to derive the log likelihood of the observed endogenous variables conditional on their own past and any exogenous variables. When the model is stationary, a method for recursively predicting the current values of the states and the endogenous variables, known as the Kalman filter, is used to obtain the prediction error form of the log-likelihood function. When the model is nonstationary, a diffuse Kalman filter is used. How the Kalman filter and the diffuse

Kalman filter initialize their recursive computations depends on the `method()` option; see [Methods and formulas](#).

The linear state-space models with time-invariant coefficient matrices defined above can be specified in the covariance-form syntax and the error-form syntax. The covariance-form syntax requires that **C** and **G** be selection matrices, but places no restrictions on **Q** or **R**. In contrast, the error-form syntax places no restrictions **C** or **G**, but requires that **Q** and **R** be either diagonal, diagonal-scalar, or identity matrices. Some models are more easily specified in the covariance-form syntax, while others are more easily specified in the error-form syntax. Choose the syntax that is easiest for your application.

Some stationary state-space models

► Example 1

Following [Hamilton \(1994b, 373–374\)](#), we can write the first-order autoregressive (AR(1)) model

$$y_t - \mu = \alpha(y_{t-1} - \mu) + \epsilon_t$$

as a state-space model with the observation equation

$$y_t = \mu + u_t$$

and the state equation

$$u_t = \alpha u_{t-1} + \epsilon_t$$

where the unobserved state is $u_t = y_t - \mu$.

Here we fit this model to data on the capacity utilization rate. The variable `lncaputil` contains data on the natural log of the capacity utilization rate for the manufacturing sector of the U.S. economy. We treat the series as first-difference stationary and fit its first-difference to an AR(1) process. Here we estimate the parameters of the above state-space form of the AR(1) model:

(Continued on next page)

```
. use http://www.stata-press.com/data/r11/manufac
(St. Louis Fed (FRED) manufacturing data)

. constraint 1 [D.lncaputil]u = 1

. sspace (u L.u, state noconstant) (D.lncaputil u, noerror), constraints(1)
searching for initial values .....
(setting technique to bhhh)
Iteration 0:   log likelihood =   1483.3603
Iteration 1:   log likelihood =   1502.053
(output omitted)
Refining estimates:
Iteration 0:   log likelihood =    1516.44
Iteration 1:   log likelihood =    1516.44

State-space model
Sample: 1972m2 - 2008m12
                                     Number of obs   =      443
                                     Wald chi2(1)      =      61.73
Log likelihood =    1516.44          Prob > chi2     =      0.0000
( 1)  [D.lncaputil]u = 1
```

lncaputil	OIM		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
u						
u						
L1.	.3523983	.0448539	7.86	0.000	.2644862	.4403104
D.lncaputil						
u						
_cons	1 -.0003558	.0005781	-0.62	0.538	-.001489	.0007773
var(u)	.0000622	4.18e-06	14.88	0.000	.000054	.0000704

Note: Tests of variances against zero are conservative and are provided only for reference.

The iteration log has three parts: the dots from the search for initial values, the log from finding the maximum, and the log from a refining step. Here is a description of the logic behind each part:

1. The quality of the initial values affect the speed and robustness of the optimization algorithm. `sspace` takes a few iterations in a nonlinear least-squares (NLS) algorithm to find good initial values and reports a dot for each (NLS) iteration.
2. This iteration log is the standard method by which Stata reports the search for the maximum likelihood estimates of the parameters in a nonlinear model.
3. Some of the parameters are transformed in the maximization process that `sspace` reports in part 2. After a maximum candidate is found in part 2, `sspace` looks for a maximum in the unconstrained space, checks that the Hessian of the log-likelihood function is of full rank, and reports these iterations as the refining step.

The header in the output describes the estimation sample, reports the log-likelihood function at the maximum, and gives the results of a Wald test against the null hypothesis that the coefficients on all the independent variables, state variables, and lagged state variables are zero. In this example, the null hypothesis that the coefficient on `L1.u` is zero is rejected at all conventional levels.

The estimation table reports results for the state equations, the observation equations, and the variance–covariance parameters. The estimated autoregressive coefficient of 0.3524 indicates that there is persistence in the first-differences of the log of the manufacturing rate. The estimated mean of the differenced series is -0.0004 , which is smaller in magnitude than its standard error, indicating that there is no deterministic linear trend in the series.

The footer informs us that the reported tests of the variance parameters against zero are conservative. These tests are conservative because the variance parameters must be greater than or equal to zero.

Typing

```
. arima D.lncaputil, ar(1) technique(nr)
(output omitted)
```

produces nearly identical parameter estimates and standard errors for the mean and the autoregressive parameter. Because `sspace` estimates the variance of the state error while `arima` estimates the standard deviation, calculations are required to obtain the same results. The different parameterization of the variance parameter can cause small numerical differences.

◀

□ Technical note

In some situations, the second part of the iteration log terminates but the refining step never converges. Only when the refining step converges does the maximization algorithm find interpretable estimates. If the refining step iterates without convergence, the parameters of the specified model are not identified by the data. (See [Rothenberg \[1971\]](#), [Drukker and Wiggins \[2004\]](#), and [Davidson and MacKinnon \[1993, sec. 5.2\]](#) for discussions of identification.)

□

▷ Example 2

Following [Harvey \(1993, 95–96\)](#), we can write a zero-mean, first-order, autoregressive moving-average (ARMA(1,1)) model

$$y_t = \alpha y_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \quad (1)$$

as a state-space model with state equations

$$\begin{pmatrix} y_t \\ \theta \epsilon_t \end{pmatrix} = \begin{pmatrix} \alpha & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y_{t-1} \\ \theta \epsilon_{t-1} \end{pmatrix} + \begin{pmatrix} 1 \\ \theta \end{pmatrix} \epsilon_t \quad (2)$$

and observation equation

$$y_t = (1 \quad 0) \begin{pmatrix} y_t \\ \theta \epsilon_t \end{pmatrix} \quad (3)$$

The unobserved states in this model are $u_{1t} = y_t$ and $u_{2t} = \theta \epsilon_t$. We set the process mean to zero because economic theory and the previous example suggest that we should do so. Below we estimate the parameters in the state-space model by using the error-form syntax:

```
. constraint 2 [u1]L.u2 = 1
. constraint 3 [u1]e.u1 = 1
. constraint 4 [D.lncaputil]u1 = 1
. sspace (u1 L.u1 L.u2 e.u1, state noconstant) (u2 e.u1, state noconstant)
> (D.lncaputil u1, noconstant), constraints(2/4) covstate(diagonal)
searching for initial values .....
(setting technique to bhhh)
Iteration 0:   log likelihood =   1499.0918
Iteration 1:   log likelihood =   1528.6725
(output omitted)
Refining estimates:
Iteration 0:   log likelihood =   1531.255
Iteration 1:   log likelihood =   1531.255

State-space model
Sample: 1972m2 - 2008m12                                Number of obs   =          443
                                                           Wald chi2(2)    =          333.84
Log likelihood =   1531.255                                Prob > chi2     =          0.0000

( 1)  [u1]L.u2 = 1
( 2)  [u1]e.u1 = 1
( 3)  [D.lncaputil]u1 = 1
```

lncaputil		OIM		z	P> z	[95% Conf. Interval]	
		Coef.	Std. Err.				
u1	u1						
	L1.	.8056815	.0522661	15.41	0.000	.7032418	.9081212
	u2						
	L1.	1	.	.	.	.	.
	e.u1	1	.	.	.	.	.
u2	e.u1	-.5188453	.0701985	-7.39	0.000	-.6564317	-.3812588
D.lncaputil	u1	1	.	.	.	.	.
var(u1)		.0000582	3.91e-06	14.88	0.000	.0000505	.0000659

Note: Tests of variances against zero are conservative and are provided only for reference.

The command in the above output specifies two state equations, one observation equation, and two options. The first state equation defines u_{1t} and the second defines u_{2t} according to (2) above. The observation equation defines the process for D.lncaputil according to the one specified in (3) above. Several coefficients in (2) and (3) are set to 1, and constraints 2–4 place these restrictions on the model.

The estimated coefficient on L.u1 in equation u1, 0.806, is the estimate of α in (2), which is the autoregressive coefficient in the ARMA model in (1). The estimated coefficient on e.u1 in equation u2, -0.519 , is the estimate of θ , which is the moving-average term in the ARMA model in (1).

This example highlights a difference between the error-form syntax and the covariance-form syntax. The error-form syntax used in this example includes only explicitly included errors. In contrast, the covariance-form syntax includes an error term in each equation, unless the noerror option is specified.

The default for covstate() also differs between the error-form syntax and the covariance-form syntax. Because the coefficients on the errors in the error-form syntax are frequently used to

estimate the standard deviation of the errors, `covstate(identity)` is the default for the error-form syntax. In contrast, unit variances are less common in the covariance-form syntax, for which `covstate(diagonal)` is the default. In this example, we specified `covstate(diagonal)` to estimate a nonunitary variance for the state.

Typing

```
. arima D.lncaputil, noconstant ar(1) ma(1) technique(nr)
(output omitted)
```

produces nearly identical results. As in the AR(1) example above, `arima` estimates the standard deviation of the error term, while `sspace` estimates the variance. Although they are theoretically equivalent, the different parameterizations give rise to small numerical differences in the other parameters. ◀

► Example 3

The variable `lnhours` contains data on the log of manufacturing hours, which we treat as first-difference stationary. We have a theory in which the process driving the changes in the log utilization rate affects the changes in the log of hours, but changes in the log hours do not affect changes in the log utilization rate. In line with this theory, we estimate the parameters of a lower triangular, first-order vector autoregressive (VAR(1)) process

$$\begin{pmatrix} \Delta \text{lncaputil}_t \\ \Delta \text{lnhours}_t \end{pmatrix} = \begin{pmatrix} \alpha_1 & 0 \\ \alpha_2 & \alpha_3 \end{pmatrix} \begin{pmatrix} \Delta \text{lncaputil}_{t-1} \\ \Delta \text{lnhours}_{t-1} \end{pmatrix} + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \end{pmatrix} \quad (4)$$

where $\Delta y_t = y_t - y_{t-1}$, $\epsilon_t = (\epsilon_{1t}, \epsilon_{2t})'$ and $\text{Var}(\epsilon) = \Sigma$. We can write this VAR(1) process as a state-space model with state equations

$$\begin{pmatrix} u_{1t} \\ u_{2t} \end{pmatrix} = \begin{pmatrix} \alpha_1 & 0 \\ \alpha_2 & \alpha_3 \end{pmatrix} \begin{pmatrix} u_{1(t-1)} \\ u_{2(t-1)} \end{pmatrix} + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \end{pmatrix} \quad (5)$$

with $\text{Var}(\epsilon) = \Sigma$ and observation equations

$$\begin{pmatrix} \Delta \text{lncaputil} \\ \Delta \text{lnhours} \end{pmatrix} = \begin{pmatrix} u_{1t} \\ u_{2t} \end{pmatrix}$$

Below we estimate the parameters of the state-space model:

(Continued on next page)

```
. constraint 5 [D.lncaputil]u1 = 1
. constraint 6 [D.lnhours]u2 = 1
. sspace (u1 L.u1, state noconstant)
> (u2 L.u1 L.u2, state noconstant)
> (D.lncaputil u1, noconstant noerror)
> (D.lnhours u2, noconstant noerror),
> constraints(5/6) covstate(unstructured)
searching for initial values .....
(setting technique to bhhh)
Iteration 0: log likelihood = 3051.5168
Iteration 1: log likelihood = 3099.5168
(output omitted)
Refining estimates:
Iteration 0: log likelihood = 3211.7532
Iteration 1: log likelihood = 3211.7532
State-space model
Sample: 1972m2 - 2008m12
Log likelihood = 3211.7532
( 1) [D.lncaputil]u1 = 1
( 2) [D.lnhours]u2 = 1
```

Number of obs = 443
Wald chi2(3) = 166.87
Prob > chi2 = 0.0000

		OIM		z	P> z	[95% Conf. Interval]	
		Coef.	Std. Err.				
u1	u1						
	L1.	.353257	.0448456	7.88	0.000	.2653612	.4411528
u2	u1						
	L1.	.1286218	.0394742	3.26	0.001	.0512537	.2059899
	u2						
	L1.	-.3707083	.0434255	-8.54	0.000	-.4558208	-.2855959
D.lncaputil							
	u1	1	.	.	.	.	.
D.lnhours							
	u2	1	.	.	.	.	.
var(u1)		.0000623	4.19e-06	14.88	0.000	.0000541	.0000705
cov(u2,u1)		.000026	2.67e-06	9.75	0.000	.0000208	.0000312
var(u2)		.0000386	2.61e-06	14.76	0.000	.0000335	.0000437

Note: Tests of variances against zero are conservative and are provided only for reference.

Specifying `covstate(unstructured)` caused `sspace` to estimate the off-diagonal element of Σ . The output indicates that this parameter, `cov(u2,u1):_cons`, is small but statistically significant.

The estimated coefficient on `L.u1` in equation `u1`, 0.353, is the estimate of α_1 in (5). The estimated coefficient on `L.u1` in equation `u2`, 0.129, is the estimate of α_2 in (5). The estimated coefficient on `L.u1` in equation `u2`, -0.371 , is the estimate of α_3 in (5).

For the VAR(1) model in (4), the estimated autoregressive coefficient for `D.lncaputil` is similar to the corresponding estimate in the univariate results in [example 1](#). The estimated effect of `LD.lncaputil` on `D.lnhours` is 0.129, the estimated autoregressive coefficient of `D.lnhours` is -0.371 , and both are statistically significant.

These estimates can be compared with those produced by typing

```
. constraint 101 [D_lncaputil]LD.lnhours = 0
. var D.lncaputil D.lnhours, lags(1) noconstant constraints(101)
(output omitted)
. matrix list e(Sigma)
(output omitted)
```

The `var` estimates are not the same as the `sspace` estimates because the generalized least-squares estimator implemented in `var` is only asymptotically equivalent to the ML estimator implemented in `sspace`, but the point estimates are similar. The comparison is useful for pedagogical purposes because the `var` estimator is relatively simple.

Some problems require constraining a covariance term to zero. If we wanted to constrain `cov(u2,u1):_cons` to zero, we could type

```
. constraint 7 [cov(u2,u1)]_cons = 0
. sspace (u1 L.u1, state noconstant)
> (u2 L.u1 L.u2, state noconstant)
> (D.lncaputil u1, noconstant noerror)
> (D.lnhours u2, noconstant noerror),
> constraints(5/7) covstate(unstructured)
(output omitted)
```

◀

► Example 4

We now extend the previous example by modeling `D.lncaputil` and `D.lnhours` as a first-order vector autoregressive moving-average (VARMA(1,1)) process. Building on the previous examples, we allow the lag of `D.lncaputil` to affect `D.lnhours` but we do not allow the lag of `D.lnhours` to affect the lag of `D.lncaputil`. Previous univariate analysis revealed that `D.lnhours` is better modeled as an autoregressive process than as an ARMA(1,1) process. As a result, we estimate the parameters of

$$\begin{pmatrix} \Delta \text{lncaputil}_t \\ \Delta \text{lnhours}_t \end{pmatrix} = \begin{pmatrix} \alpha_1 & 0 \\ \alpha_2 & \alpha_3 \end{pmatrix} \begin{pmatrix} \Delta \text{lncaputil}_{t-1} \\ \Delta \text{lnhours}_{t-1} \end{pmatrix} + \begin{pmatrix} \theta_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \epsilon_{1(t-1)} \\ \epsilon_{2(t-1)} \end{pmatrix} + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \end{pmatrix}$$

We can write this VARMA(1,1) process as a state-space model with state equations

$$\begin{pmatrix} s_{1t} \\ s_{2t} \\ s_{3t} \end{pmatrix} = \begin{pmatrix} \alpha_1 & 1 & 0 \\ 0 & 0 & 0 \\ \alpha_2 & 0 & \alpha_3 \end{pmatrix} \begin{pmatrix} s_{1(t-1)} \\ s_{2(t-1)} \\ s_{3(t-1)} \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ \theta_1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \end{pmatrix}$$

where the states are

$$\begin{pmatrix} s_{1t} \\ s_{2t} \\ s_{3t} \end{pmatrix} = \begin{pmatrix} \Delta \text{lncaputil}_t \\ \theta_1 \epsilon_{1t} \\ \Delta \text{lnhours}_t \end{pmatrix}$$

and we simplify the problem by assuming that

$$\text{Var} \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix}$$

Below we estimate the parameters of this model by using `sspace`:

```
. constraint 7 [u1]L.u2      = 1
. constraint 8 [u1]e.u1      = 1
. constraint 9 [u3]e.u3      = 1
. constraint 10 [D.lncaputil]u1 = 1
. constraint 11 [D.lnhours]u3 = 1
. sspace (u1 L.u1 L.u2 e.u1, state noconstant)
>      (u2 e.u1, state noconstant)
>      (u3 L.u1 L.u3 e.u3, state noconstant)
>      (D.lncaputil u1, noconstant)
>      (D.lnhours u3, noconstant),
>      constraints(7/11) technique(nr) covstate(diagonal)
searching for initial values .....
(output omitted)
Refining estimates:
Iteration 0:  log likelihood = 3156.0564
Iteration 1:  log likelihood = 3156.0564

State-space model
Sample: 1972m2 - 2008m12
Log likelihood = 3156.0564
Number of obs   =      443
Wald chi2(4)    =     427.55
Prob > chi2     =      0.0000

( 1) [u1]L.u2 = 1
( 2) [u1]e.u1 = 1
( 3) [u3]e.u3 = 1
( 4) [D.lncaputil]u1 = 1
( 5) [D.lnhours]u3 = 1
```

		OIM		z	P> z	[95% Conf. Interval]	
		Coef.	Std. Err.				
u1	u1						
	L1.	.8058031	.0522493	15.42	0.000	.7033964	.9082098
	u2						
	L1.	1	.	.	.	.	.
	e.u1	1	.	.	.	.	.
u2							
	e.u1	-.518907	.0701848	-7.39	0.000	-.6564667	-.3813474
u3	u1						
	L1.	.1734868	.0405156	4.28	0.000	.0940776	.252896
	u3						
	L1.	-.4809376	.0498574	-9.65	0.000	-.5786563	-.3832188
	e.u3	1	.	.	.	.	.
D.lncaputil							
	u1	1	.	.	.	.	.
D.lnhours							
	u3	1	.	.	.	.	.
var(u1)		.0000582	3.91e-06	14.88	0.000	.0000505	.0000659
var(u3)		.0000382	2.56e-06	14.88	0.000	.0000331	.0000432

Note: Tests of variances against zero are conservative and are provided only for reference.

The estimates of the parameters in the model for `D.lncaputil` are similar to those in the univariate model fit in [example 2](#). The estimates of the parameters in the model for `D.lnhours` indicate that the lag of `D.lncaputil` has a positive effect on `D.lnhours`.

◀

□ Technical note

The `technique(nr)` option facilitates convergence in [example 4](#). Fitting state-space models is notoriously difficult. Convergence problems are common. Four methods for overcoming convergence problems are 1) selecting an alternate optimization algorithm by using the `technique()` option, 2) using alternative starting values by specifying the `from()` option, 3) using starting values obtained by estimating the parameters of a restricted version of the model of interest, or 4) putting the variables on the same scale.

□

▷ Example 5

[Stock and Watson \(1989, 1991\)](#) wrote a simple macroeconomic model as a dynamic-factor model, estimated the parameters by ML, and extracted an economic indicator. In this example, we estimate the parameters of a dynamic-factor model. In [\[TS\] sspace postestimation](#), we extend this example and extract an economic indicator for the differenced series.

We have data on an industrial-production index, `ipman`; an aggregate weekly hours index, `hours`; and aggregate unemployment, `unemp`. `income` is real disposable income divided by 100. We rescaled real disposable income to avoid convergence problems.

We postulate a latent factor that follows an AR(2) process. Each measured variable is then related to the current value of that latent variable by a parameter. The state-space form of our model is

$$\begin{pmatrix} f_t \\ f_{t-1} \end{pmatrix} = \begin{pmatrix} \theta_1 & \theta_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{t-1} \\ f_{t-2} \end{pmatrix} + \begin{pmatrix} \nu_t \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} \Delta ipman_t \\ \Delta income_t \\ \Delta hours_t \\ \Delta unemp_t \end{pmatrix} = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{pmatrix} f_t + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \\ \epsilon_{3t} \\ \epsilon_{4t} \end{pmatrix}$$

where

$$\text{Var} \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \\ \epsilon_{3t} \\ \epsilon_{4t} \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ 0 & 0 & \sigma_3^2 & 0 \\ 0 & 0 & 0 & \sigma_4^2 \end{pmatrix}$$

The parameter estimates are

(Continued on next page)

```
. use http://www.stata-press.com/data/r11/dfex
(St. Louis Fed (FRED) macro data)

. constraint 12 [lf]L.f = 1

. sspace (f L.f L.lf, state noconstant)
>      (lf L.f, state noconstant noerror)
>      (D.ipman f, noconstant)
>      (D.income f, noconstant)
>      (D.hours f, noconstant)
>      (D.unemp f, noconstant),
>      covstate(identity) constraints(12)
searching for initial values .....
(setting technique to bhhh)
Iteration 0:  log likelihood = -721.37028
Iteration 1:  log likelihood = -676.3876
(output omitted)
Refining estimates:
Iteration 0:  log likelihood = -662.09507
Iteration 1:  log likelihood = -662.09507

State-space model
Sample: 1972m2 - 2008m11                                Number of obs   =      442
                                                           Wald chi2(6)    =      751.95
Log likelihood = -662.09507                               Prob > chi2     =      0.0000
( 1)  [lf]L.f = 1
```

		OIM		z	P> z	[95% Conf. Interval]	
		Coef.	Std. Err.				
f	f						
	L1.	.2651932	.0568663	4.66	0.000	.1537372	.3766491
lf	lf						
	L1.	.4820398	.0624635	7.72	0.000	.3596136	.604466
lf	f						
	L1.	1	.	.	.	.	.
D.ipman	f	.3502249	.0287389	12.19	0.000	.2938976	.4065522
D.income	f	.0746338	.0217319	3.43	0.001	.0320401	.1172276
D.hours	f	.2177469	.0186769	11.66	0.000	.1811407	.254353
D.unemp	f	-.0676016	.0071022	-9.52	0.000	-.0815217	-.0536816
var(D.ipman)		.1383158	.0167086	8.28	0.000	.1055675	.1710641
var(D.income)		.2773808	.0188302	14.73	0.000	.2404743	.3142873
var(D.hours)		.0911446	.0080847	11.27	0.000	.0752988	.1069903
var(D.unemp)		.0237232	.0017932	13.23	0.000	.0202086	.0272378

Note: Tests of variances against zero are conservative and are provided only for reference.

The output indicates that the unobserved factor is quite persistent and that it is a significant predictor for each of the observed variables.

These models are frequently used to forecast the dependent variables and to estimate the unobserved factors. We present some illustrative examples in [TS] [sspace postestimation](#). The `dfactor` command estimates the parameters of dynamic-factor models; see [TS] [dfactor](#).



Some nonstationary state-space models

► Example 6

[Harvey \(1989\)](#) advocates the use of STS models. These models parameterize the trends and seasonal components of a set of time series. The simplest STS model is the local-level model, which is given by

$$y_t = \mu_t + \epsilon_t$$

where

$$\mu_t = \mu_{t-1} + \nu_t$$

The model is called a local-level model because the level of the series is modeled as a random walk plus an idiosyncratic noise term. (The model is also known as the random-walk-plus-noise model.) The local-level model is nonstationary because of the random-walk component. When the variance of the idiosyncratic-disturbance ϵ_t is zero and the variance of the level-disturbance ν_t is not zero, the local-level model reduces to a random walk. When the variance of the level-disturbance ν_t is zero and the variance of the idiosyncratic-disturbance ϵ_t is not zero,

$$\mu_t = \mu_{t-1} = \mu$$

and the local-level model reduces to

$$y_t = \mu + \epsilon_t$$

which is a simple regression with a time-invariant mean. The parameter μ is not estimated in the state-space formulation below.

In this example, we fit weekly levels of the Standard and Poor's 500 Index to a local-level model. Because this model is already in state-space form, we fit `close` by typing

(Continued on next page)

```
. use http://www.stata-press.com/data/r11/sp500w, clear
. constraint 13 [z]L.z = 1
. constraint 14 [close]z = 1
. sspace (z L.z, state noconstant) (close z, noconstant), constraints(13 14)
searching for initial values ...
(setting technique to bhhh)
Iteration 0:   log likelihood = -12651.724
Iteration 1:   log likelihood = -12581.815
(output omitted)
Refining estimates:
Iteration 0:   log likelihood = -12576.99
Iteration 1:   log likelihood = -12576.99

State-space model
Sample: 1 - 3093
Log likelihood = -12576.99
Number of obs   =      3093

( 1)  [z]L.z = 1
( 2)  [close]z = 1
```

close	OIM		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
z						
z						
L1.	1	.	.	.	.	.
close						
close						
z	1	.	.	.	.	.
var(z)	170.3456	7.584909	22.46	0.000	155.4794	185.2117
var(close)	15.24858	3.392457	4.49	0.000	8.599486	21.89767

Note: Model is not stationary.
Note: Tests of variances against zero are conservative and are provided only for reference.

The results indicate that both components have nonzero variances. The output footer informs us that the model is nonstationary at the estimated parameter values.

Technical note

In the previous example, we estimated the parameters of a nonstationary state-space model. The model is nonstationary because one of the eigenvalues of the **A** matrix has unit modulus. That all the coefficients in the **A** matrix are fixed is also important. See [Lütkepohl \(2005, 636–637\)](#) for why the ML estimator for the parameters of a nonstationary state-model that is nonstationary because of eigenvalues with unit moduli from a fixed **A** matrix is still consistent and asymptotically normal.

Example 7

In another basic STS model, known as the local linear-trend model, both the level and the slope of a linear time trend are random walks. Here are the state equations and the observation equation for a local linear-trend model for the level of industrial production contained in variable `ipman`:

$$\begin{pmatrix} \mu_t \\ \beta_t \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mu_{t-1} \\ \beta_{t-1} \end{pmatrix} + \begin{pmatrix} \nu_{1t} \\ \nu_{2t} \end{pmatrix}$$

$$\text{ipman}_t = \mu_t + \epsilon_t$$

The estimated parameters are

```
. use http://www.stata-press.com/data/r11/dfex, clear
(St. Louis Fed (FRED) macro data)

. constraint 15 [f1]L.f1 = 1

. constraint 16 [f1]L.f2 = 1

. constraint 17 [f2]L.f2 = 1

. constraint 18 [ipman]f1 = 1

.
. sspace (f1 L.f1 L.f2, state noconstant)
>       (f2 L.f2, state noconstant)
>       (ipman f1, noconstant), constraints(15/18)
searching for initial values ...
(setting technique to bhhh)
Iteration 0:   log likelihood = -423.64739
Iteration 1:   log likelihood = -402.80659
(output omitted)
Refining estimates:
Iteration 0:   log likelihood = -359.1266
Iteration 1:   log likelihood = -359.1266 (backed up)

State-space model

Sample: 1972m1 - 2008m11                      Number of obs   =          443
Log likelihood = -359.1266

( 1)  [f1]L.f1 = 1
( 2)  [f1]L.f2 = 1
( 3)  [f2]L.f2 = 1
( 4)  [ipman]f1 = 1
```

ipman	OIM		z	P> z	[95% Conf. Interval]	
	Coef.	Std. Err.				
f1	f1					
	L1.	1	.	.	.	.
f2	f2					
	L1.	1	.	.	.	.
ipman	f1					
	L1.	1	.	.	.	.
var(f1)		.1473071	.0407156	3.62	0.000	.067506 .2271082
var(f2)		.0178752	.0065743	2.72	0.007	.0049898 .0307606
var(ipman)		.0354429	.0148186	2.39	0.017	.0063989 .0644868

Note: Model is not stationary.

Note: Tests of variances against zero are conservative and are provided only for reference.

There is little evidence that either of the variance parameters are zero. The fit obtained indicates that we could now proceed with specification testing and checks to see how well this model forecasts these data.



Saved results

`sspace` saves the following in `e()`:

Scalars	
<code>e(N)</code>	number of observations
<code>e(k)</code>	number of parameters
<code>e(k_aux)</code>	number of auxiliary parameters
<code>e(k_eq)</code>	number of equations in <code>e(b)</code>
<code>e(k_autoCns)</code>	number of base, empty, and omitted constraints
<code>e(k_dv)</code>	number of dependent variables
<code>e(k_obser)</code>	number of observation equations
<code>e(k_state)</code>	number of state equations
<code>e(k_obser_err)</code>	number of observation-error terms
<code>e(k_state_err)</code>	number of state-error terms
<code>e(df_m)</code>	model degrees of freedom
<code>e(ll)</code>	log likelihood
<code>e(chi2)</code>	χ^2 statistic
<code>e(p)</code>	significance
<code>e(tmin)</code>	minimum time in sample
<code>e(tmax)</code>	maximum time in sample
<code>e(stationary)</code>	1 if the estimated parameters indicate a stationary model, 0 otherwise
<code>e(rank)</code>	rank of VCE
<code>e(ic)</code>	number of iterations
<code>e(rc)</code>	return code
<code>e(converged)</code>	1 if converged, 0 otherwise

Macros

<code>e(cmd)</code>	<code>sspace</code>
<code>e(cmdline)</code>	command as typed
<code>e(depvar)</code>	unoperated names of dependent variables in observation equations
<code>e(observ_deps)</code>	names of dependent variables in observation equations
<code>e(state_deps)</code>	names of dependent variables in state equations
<code>e(covariates)</code>	list of covariates
<code>e(indeps)</code>	independent variables
<code>e(tvar)</code>	time variable
<code>e(eqnames)</code>	names of equations
<code>e(title)</code>	title in estimation output
<code>e(tmins)</code>	formatted minimum time
<code>e(tmaxs)</code>	formatted maximum time
<code>e(R_structure)</code>	structure of observed-variable-error covariance matrix
<code>e(Q_structure)</code>	structure of state-error covariance matrix
<code>e(chi2type)</code>	Wald; type of model χ^2 test
<code>e(vce)</code>	<i>vcetype</i> specified in <code>vce()</code>
<code>e(vcetype)</code>	title used to label Std. Err.
<code>e(opt)</code>	type of optimization
<code>e(method)</code>	likelihood method
<code>e(initial_values)</code>	type of initial values
<code>e(technique)</code>	maximization technique(s)
<code>e(tech_steps)</code>	iterations taken in maximization technique(s)
<code>e(crittype)</code>	optimization criterion
<code>e(datasignature)</code>	the checksum
<code>e(datasignaturevars)</code>	variables used in calculation of checksum
<code>e(properties)</code>	<code>b</code> <code>V</code>
<code>e(estat_cmd)</code>	program used to implement <code>estat</code>
<code>e(predict)</code>	program used to implement <code>predict</code>
<code>e(marginsok)</code>	predictions allowed by <code>margins</code>
<code>e(marginsnotok)</code>	predictions disallowed by <code>margins</code>

(Continued on next page)

Matrices

<code>e(b)</code>	parameter vector
<code>e(Cns)</code>	constraints matrix
<code>e(ilog)</code>	iteration log (up to 20 iterations)
<code>e(gradient)</code>	gradient vector
<code>e(gamma)</code>	mapping from parameter vector to state-space matrices
<code>e(A)</code>	estimated A matrix
<code>e(B)</code>	estimated B matrix
<code>e(C)</code>	estimated C matrix
<code>e(D)</code>	estimated D matrix
<code>e(F)</code>	estimated F matrix
<code>e(G)</code>	estimated G matrix
<code>e(chol_R)</code>	Cholesky factor of estimated R matrix
<code>e(chol_Q)</code>	Cholesky factor of estimated Q matrix
<code>e(chol_Q1)</code>	Cholesky factor of estimated Q1 matrix
<code>e(chol_Q2)</code>	Cholesky factor of estimated Q2 matrix
<code>e(chol_Sz0)</code>	Cholesky factor of initial state covariance matrix
<code>e(z0)</code>	initial state vector
<code>e(d)</code>	addition term in diffuse initial state vector, if nonstationary model
<code>e(T)</code>	inner part of quadratic form for initial state covariance in a partially nonstationary model
<code>e(M)</code>	outer part of quadratic form for initial state covariance in a partially nonstationary model
<code>e(Z)</code>	select nonstationary components in a partially nonstationary model
<code>e(V)</code>	variance–covariance matrix of the estimators
<code>e(V_modelbased)</code>	model-based variance

Functions

<code>e(sample)</code>	marks estimation sample

Methods and formulas

`sspace` is implemented as an ado-file.

Recall that our notation for linear state-space models with time-invariant coefficient matrices is

$$\begin{aligned}\mathbf{z}_t &= \mathbf{A}\mathbf{z}_{t-1} + \mathbf{B}\mathbf{x}_t + \mathbf{C}\epsilon_t \\ \mathbf{y}_t &= \mathbf{D}\mathbf{z}_t + \mathbf{F}\mathbf{w}_t + \mathbf{G}\nu_t\end{aligned}$$

where

$\mathbf{z}_t$ is an $m \times 1$ vector of unobserved state variables;

$\mathbf{x}_t$ is a $k_x \times 1$ vector of exogenous variables;

ϵ_t is a $q \times 1$ vector of state-error terms, ($q \leq m$);

$\mathbf{y}_t$ is an $n \times 1$ vector of observed endogenous variables;

$\mathbf{w}_t$ is a $k_w \times 1$ vector of exogenous variables;

ν_t is an $r \times 1$ vector of observation-error terms, ($r \leq n$); and

$\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{F}$, and $\mathbf{G}$ are parameter matrices.

The equations for $\mathbf{z}_t$ are known as the state equations, and the equations for $\mathbf{y}_t$ are known as the observation equations.

The error terms are assumed to be zero mean, normally distributed, serially uncorrelated, and uncorrelated with each other;

$$\begin{aligned}\boldsymbol{\epsilon}_t &\sim N(0, \mathbf{Q}) \\ \boldsymbol{\nu}_t &\sim N(0, \mathbf{R}) \\ E[\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_s'] &= \mathbf{0} \text{ for all } s \neq t \\ E[\boldsymbol{\epsilon}_t \boldsymbol{\nu}_s'] &= \mathbf{0} \text{ for all } s \text{ and } t\end{aligned}$$

`sspace` estimates the parameters of linear state-space models by maximum likelihood. The Kalman filter is a method for recursively obtaining linear, least-squares forecasts of $\mathbf{y}_t$ conditional on past information. These forecasts are used to construct the log likelihood, assuming normality and stationarity. When the model is nonstationary, a diffuse Kalman filter is used.

Hamilton (1994a; 1994b, 389) shows that the QML estimator, obtained when the normality assumption is dropped, is consistent and asymptotically normal, although the variance–covariance matrix of the estimator (VCE) must be estimated by the Huber/White/sandwich estimator. Hamilton’s discussion applies to stationary models, and specifying `vce(robust)` produces a consistent estimator of the VCE when the errors are not normal.

Methods for computing the log likelihood differ in how they calculate initial values for the Kalman filter when the model is stationary, how they compute a diffuse Kalman filter when the model is nonstationary, and whether terms for initial states are included. `sspace` offers the `method(hybrid)`, `method(dejong)`, and `method(kdiffuse)` options for computing the log likelihood. All three methods handle both stationary and nonstationary models.

`method(hybrid)`, the default, uses the initial values for the states implied by stationarity to initialize the Kalman filter when the model is stationary. Hamilton (1994b, 378) discusses this method of computing initial values for the states and derives a log-likelihood function that does not include terms for the initial states. When the model is nonstationary, `method(hybrid)` uses the De Jong (1988, 1991) diffuse Kalman filter and log-likelihood function, which includes terms for the initial states.

`method(dejong)` uses the stationary De Jong (1988) method when the model is stationary and the De Jong (1988, 1991) diffuse Kalman filter when the model is nonstationary. The stationary De Jong (1988) method estimates initial values for the Kalman filter as part of the log-likelihood computation, as in De Jong (1988).

`method(kdiffuse)` implements the seldom-used large- κ diffuse approximation to the diffuse Kalman filter when the model is nonstationary and uses initial values for the states implied by stationarity when the model is stationary. The log likelihood does not include terms for the initial states in either case. We recommend that you do not use `method(kdiffuse)` except to replicate older results computed using this method.

De Jong (1988, 1991) and De Jong and Chu-Chun-Lin (1994) derive the log likelihood and a diffuse Kalman filter for handling nonstationary data. De Jong (1988) replaces the stationarity assumption with a time-immemorial assumption, which he uses to derive the log-likelihood function, an initial state vector, and a covariance of the initial state vector when the model is nonstationary. By default, and when `method(hybrid)` or `method(dejong)` is specified, `sspace` uses the diffuse Kalman filter given in definition 5 of De Jong and Chu-Chun-Lin (1994). This method uses theorem 3 of De Jong and Chu-Chun-Lin (1994) to compute the covariance of the initial states. When using this method,

`sspace` saves the matrices from their theorem 3 in `e()`, although the names are changed. `e(Z)` is their $\mathbf{U}_1$, `e(T)` is their $\mathbf{U}_2$, `e(A)` is their $\mathbf{T}$, and `e(M)` is their $\mathbf{M}$.

See [De Jong \(1988, 1991\)](#) and [De Jong and Chu-Chun-Lin \(1994\)](#) for the details of the De Jong diffuse Kalman filter.

Practical estimation and inference require that the maximum likelihood estimator be consistent and normally distributed in large samples. These statistical properties of the maximum likelihood estimator are well established when the model is stationary; see [Caines \(1988, chap. 5 and 7\)](#), [Hamilton \(1994b, 388–389\)](#), and [Hannan and Deistler \(1988, chap. 4\)](#). When the model is nonstationary, additional assumptions must hold for the maximum likelihood estimator to be consistent and asymptotically normal; see [Harvey \(1989, sec. 3.4\)](#), [Lütkepohl \(2005, 636–637\)](#), and [Schneider \(1988\)](#). [Chang, Miller, and Park \(2009\)](#) show that the ML and the QML estimators are consistent and asymptotically normal for a class of nonstationary state-space models.

We now give an intuitive version of the Kalman filter. `sspace` uses theoretically equivalent, but numerically more stable, methods. For each time t , the Kalman filter produces the conditional expected state vector $\mathbf{z}_{t|t}$ and the conditional covariance matrix $\mathbf{\Omega}_{t|t}$; both are conditional on information up to and including time t . Using the model and previous period results, for each t we begin with

$$\begin{aligned}\mathbf{z}_{t|t-1} &= \mathbf{A}\mathbf{z}_{t-1|t-1} + \mathbf{B}\mathbf{x}_t \\ \mathbf{\Omega}_{t|t-1} &= \mathbf{A}\mathbf{\Omega}_{t-1|t-1}\mathbf{A}' + \mathbf{CQC}' \\ \mathbf{y}_{t|t-1} &= \mathbf{D}\mathbf{z}_{t|t-1} + \mathbf{F}\mathbf{w}_t\end{aligned}\tag{6}$$

The residuals and the mean squared error (MSE) matrix of the forecast error are

$$\begin{aligned}\tilde{\mathbf{v}}_{t|t} &= \mathbf{y}_t - \mathbf{y}_{t|t-1} \\ \mathbf{\Sigma}_{t|t} &= \mathbf{D}\mathbf{\Omega}_{t|t-1}\mathbf{D}' + \mathbf{GRG}'\end{aligned}\tag{7}$$

In the last steps, we update the conditional expected state vector and the conditional covariance with the time t information:

$$\begin{aligned}\mathbf{z}_{t|t} &= \mathbf{z}_{t|t-1} + \mathbf{\Omega}_{t|t-1}\mathbf{D}\mathbf{\Sigma}_{t|t}^{-1}\tilde{\mathbf{v}}_{t|t} \\ \mathbf{\Omega}_{t|t} &= \mathbf{\Omega}_{t|t-1} - \mathbf{\Omega}_{t|t-1}\mathbf{D}\mathbf{\Sigma}_{t|t}^{-1}\mathbf{D}'\mathbf{\Omega}_{t|t-1}\end{aligned}\tag{8}$$

Equations (6)–(8) are the Kalman filter. The equations denoted by (6) are the one-step predictions. The one-step predictions do not use contemporaneous values of $\mathbf{y}_t$; only past values of $\mathbf{y}_t$, past values of the exogenous $\mathbf{x}_t$, and contemporaneous values of $\mathbf{x}_t$ are used. Equations (7) and (8) form the update step of the Kalman filter; they incorporate the contemporaneous dependent variable information into the predicted states.

The Kalman filter requires initial values for the states and a covariance matrix for the initial states to start off the recursive process. [Hamilton \(1994b\)](#) discusses how to compute initial values for the Kalman filter assuming stationarity. This method is used by default when the model is stationary. [De Jong \(1988\)](#) discusses how to estimate initial values by maximum likelihood; this method is used when `method(dejong)` is specified.

Letting δ be the vector of parameters in the model, [Lütkepohl \(2005\)](#) and [Harvey \(1989\)](#) show that the log-likelihood function for the parameters of a stationary model is given by

$$\ln L(\delta) = -0.5 \left\{ nT \ln(2\pi) + \sum_{t=1}^T \ln(|\Sigma_{t|t-1}|) + \sum_{t=1}^T \mathbf{e}_t' \Sigma_{t|t-1}^{-1} \mathbf{e}_t \right\}$$

where $\mathbf{e}_t = (\mathbf{y}_t - \mathbf{y}_{t|t-1})$ depends on δ and Σ also depends on δ .

The variance–covariance matrix of the estimator (VCE) is estimated by the observed information matrix (OIM) estimator by default. Specifying `vce(robust)` causes `sspace` to use the Huber/White/sandwich estimator. Both estimators of the VCE are standard and documented in [Hamilton \(1994b\)](#).

[Hamilton \(1994b\)](#), [Hannan and Deistler \(1988\)](#), and [Caines \(1988\)](#) show that the ML estimator is consistent and asymptotically normal when the model is stationary. [Schneider \(1988\)](#) establishes consistency and asymptotic normality when the model is nonstationary because $\mathbf{A}$ has some eigenvalues with modulus 1 and there are no unknown parameters in $\mathbf{A}$.

Not all state-space models are identified, as discussed in [Hamilton \(1994b\)](#) and [Lütkepohl \(2005\)](#). `sspace` checks for local identification at the optimum. `sspace` will not declare convergence unless the Hessian is full rank. This check for local identifiability is due to [Rothenberg \(1971\)](#).

Specifying `method(dejong)` causes `sspace` to maximize the log-likelihood function given in section 2 (vii) of [De Jong \(1988\)](#). This log-likelihood function includes the initial states as parameters to be estimated. We use some of the methods in [Casals, Sotoca, and Jerez \(1999\)](#) for computing the [De Jong \(1988\)](#) log-likelihood function.

References

- Anderson, B. D. O., and J. B. Moore. 1979. *Optimal Filtering*. Englewood Cliffs, NJ: Prentice–Hall.
- Brockwell, P. J., and R. A. Davis. 1991. *Time Series: Theory and Methods*. 2nd ed. New York: Springer.
- Caines, P. E. 1988. *Linear Stochastic Systems*. New York: Wiley.
- Casals, J., S. Sotoca, and M. Jerez. 1999. A fast and stable method to compute the likelihood of time invariant state-space models. *Economics Letters* 65: 329–337.
- Chang, Y., J. I. Miller, and J. Y. Park. 2009. Extracting a common stochastic trend: Theory with some applications. *Journal of Econometrics* 150: 231–247.
- Durbin, J., and G. S. Maddala. 1993. *Estimation and Inference in Econometrics*. New York: Oxford University Press.
- De Jong, P. 1988. The likelihood for a state space model. *Biometrika* 75: 165–169.
- . 1991. The diffuse Kalman filter. *Annals of Statistics* 19: 1073–1083.
- De Jong, P., and S. Chu-Chun-Lin. 1994. Stationary and non-stationary state space models. *Journal of Time Series Analysis* 15: 151–166.
- Drukker, D. M., and V. L. Wiggins. 2004. Verifying the solution from a nonlinear solver: A case study: Comment. *American Economic Review* 94: 397–399.
- Hamilton, J. D. 1994a. State-space models. In Vol. 4 of *Handbook of Econometrics*, ed. R. F. Engle and D. L. McFadden, 3039–3080. New York: Elsevier.
- . 1994b. *Time Series Analysis*. Princeton: Princeton University Press.
- Hannan, E. J., and M. Deistler. 1988. *The Statistical Theory of Linear Systems*. New York: Wiley.
- Harvey, A. C. 1989. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge: Cambridge University Press.
- . 1993. *Time Series Models*. 2nd ed. Cambridge, MA: MIT Press.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Rothenberg, T. J. 1971. Identification in parametric models. *Econometrica* 39: 577–591.

Schneider, W. 1988. Analytical uses of Kalman filtering in econometrics: A survey. *Statistical Papers* 29: 3–33.

Stock, J. H., and M. W. Watson. 1989. New indexes of coincident and leading economic indicators. In *NBER Macroeconomics Annual 1989*, ed. O. J. Blanchard and S. Fischer, vol. 4, 351–394. Cambridge, MA: MIT Press.

———. 1991. A probability model of the coincident economic indicators. In *Leading Economic Indicators: New Approaches and Forecasting Records*, ed. K. Lahiri and G. H. Moore, 63–89. Cambridge: Cambridge University Press.

Also see

[TS] **sspace postestimation** — Postestimation tools for `sspace`

[TS] **dfactor** — Dynamic-factor models

[TS] **dfactor postestimation** — Postestimation tools for `dfactor`

[TS] **var** — Vector autoregressive models

[TS] **arima** — ARIMA, ARMAX, and other dynamic regression models

[U] **20 Estimation and postestimation commands**

Description

The following standard postestimation commands are available for `sspace`:

<i>command</i>	<i>description</i>
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	nonlinear predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

```
predict [type] {stub*|newvarlist} [if] [in] [, statistic options]
```

<i>statistic</i>	<i>description</i>
Main	
<code>xb</code>	observable variables
<code>states</code>	latent state variables
<code>residuals</code>	residuals
<code>rstandard</code>	standardized residuals

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

options	description
Options	
<code>equation(<i>eqnames</i>)</code>	name(s) of equation(s) for which predictions are to be made
<code>rmse(<i>stub*</i> <i>newvarlist</i>)</code>	put estimated root mean squared errors of predicted statistics in new variables
<code>dynamic(<i>time_constant</i>)</code>	begin dynamic forecast at specified time
Advanced	
<code>smethod(<i>method</i>)</code>	method for predicting unobserved states
method	description
<code>onestep</code>	predict using past information
<code>smooth</code>	predict using all sample information
<code>filter</code>	predict using past and contemporaneous information

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

Main

`xb`, `states`, `residuals`, and `rstandard` specify the statistic to be predicted.

`xb`, the default, calculates the linear predictions of the observed variables.

`states` calculates the linear predictions of the latent state variables.

`residuals` calculates the residuals in the equations for observable variables. `residuals` may not be specified with `dynamic()`.

`rstandard` calculates the standardized residuals, which are the residuals normalized to be uncorrelated and to have unit variances. `rstandard` may not be specified with `smethod(filter)`, `smethod(smooth)`, or `dynamic()`.

Options

`equation(eqnames)` specifies the equation(s) for which the predictions are to be calculated. If you do not specify `equation()` or `stub*`, the results are the same as if you had specified the name of the first equation for the predicted statistic.

You specify a list of equation names, such as `equation(income consumption)` or `equation(factor1 factor2)`, to identify the equations. Specify names of state equations when predicting `states` and names of observable equations in all other cases.

`equation()` may not be specified with `stub*`.

`rmse(stub* | newvarlist)` puts the root mean squared errors of the predicted statistics into the specified new variables. The root mean squared errors measure the variances due to the disturbances but do not account for estimation error.

`dynamic(time_constant)` specifies when `predict` starts producing dynamic forecasts. The specified *time_constant* must be in the scale of the time variable specified in `tsset`, and the *time_constant* must be inside a sample for which observations on the dependent variables are available. For example, `dynamic(tq(2008q4))` causes dynamic predictions to begin in the fourth quarter of 2008, assuming that your time variable is quarterly; see [D] [dates and times](#). If the model contains exogenous variables, they must be present for the whole predicted sample. `dynamic()` may not be specified with `rstandard`, `residuals`, or `smethod(smooth)`.

Advanced

`smethod(method)` specifies the method for predicting the unobserved states; `smethod(onestep)`, `smethod(filter)`, and `smethod(smooth)` cause different amounts of information on the dependent variables to be used in predicting the states at each time period.

`smethod(onestep)`, the default, causes `predict` to estimate the states at each time period using previous information on the dependent variables. The full Kalman filter is performed on previous periods, but only the one-step predictions are made for the current period.

`smethod(smooth)` causes `predict` to estimate the states at each time period using all the sample data by the Kalman smoother. `smethod(smooth)` may not be specified with `rstandard`.

`smethod(filter)` causes `predict` to estimate the states at each time period using previous and contemporaneous data by the Kalman filter. The full Kalman filter is performed on previous periods and the current period. `smethod(filter)` may be specified only with `states`.

Remarks

We assume that you have already read [TS] [sspace](#). In this entry, we illustrate some of the features of `predict` after using `sspace` to estimate the parameters of a state-space model.

All the predictions after `sspace` depend on the unobserved states, which are estimated recursively. Changing the sample can alter the state estimates, which can change all other predictions.

► Example 1

In [example 5](#) of [TS] [sspace](#), we estimated the parameters of the dynamic-factor model

$$\begin{pmatrix} f_t \\ f_{t-1} \end{pmatrix} = \begin{pmatrix} \theta_1 & \theta_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{t-1} \\ f_{t-2} \end{pmatrix} + \begin{pmatrix} \nu_t \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} \Delta \text{ipman}_t \\ \Delta \text{income}_t \\ \Delta \text{hours}_t \\ \Delta \text{unemp}_t \end{pmatrix} = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{pmatrix} f_t + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \\ \epsilon_{3t} \\ \epsilon_{4t} \end{pmatrix}$$

where

$$\text{Var} \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \\ \epsilon_{3t} \\ \epsilon_{4t} \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ 0 & 0 & \sigma_3^2 & 0 \\ 0 & 0 & 0 & \sigma_4^2 \end{pmatrix}$$

by typing

```
. use http://www.stata-press.com/data/r11/dfex
(St. Louis Fed (FRED) macro data)

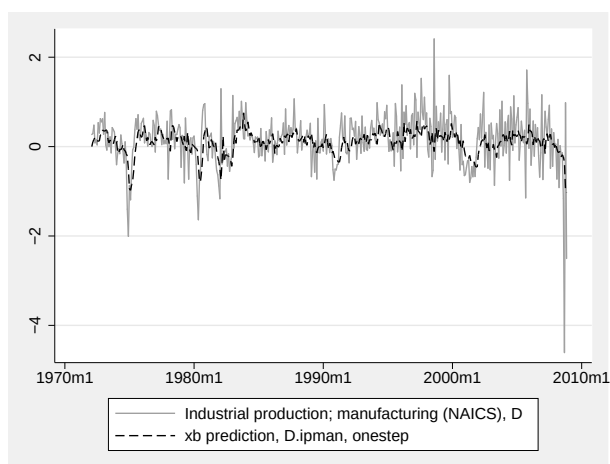
. constraint 1 [lf]L.f = 1

. sspace (f L.f L.lf, state noconstant)
>      (lf L.f, state noconstant noerror)
>      (D.ipman f, noconstant)
>      (D.income f, noconstant)
>      (D.hours f, noconstant)
>      (D.unemp f, noconstant),
>      covstate(identity) constraints(1)
(output omitted)
```

Below we obtain the one-step predictions for each of the four dependent variables in the model, and then we graph the actual and predicted ipman:

```
. predict dep*
(option xb assumed; fitted values)

. tsline D.ipman dep1, lcolor(gs10) xtitle("") legend(rows(2))
```



The graph shows that the one-step predictions account for only a small part of the swings in the realized ipman.

◀

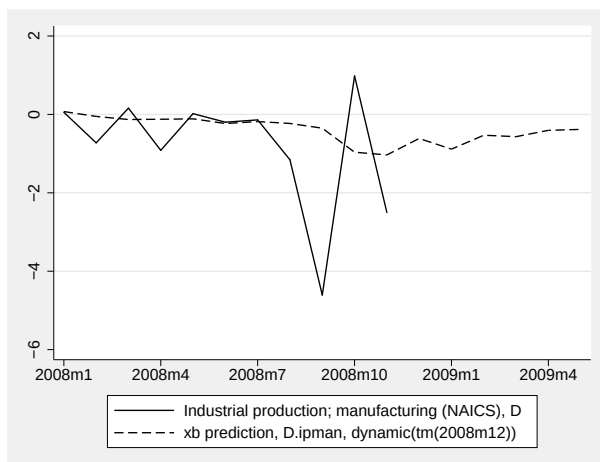
► Example 2

We use the estimates from the previous example to make out-of-sample predictions. After using `tsappend` to extend the dataset by six periods, we use `predict` with the `dynamic()` option and graph the result.

```
. tsappend, add(6)

. predict Dipman_f, dynamic(tm(2008m12)) equation(D.ipman)

. tsline D.ipman Dipman_f if month>=tm(2008m1), xtitle("") legend(rows(2))
```



The model predicts that the changes in industrial production will remain negative for the forecast horizon, although they increase toward zero.

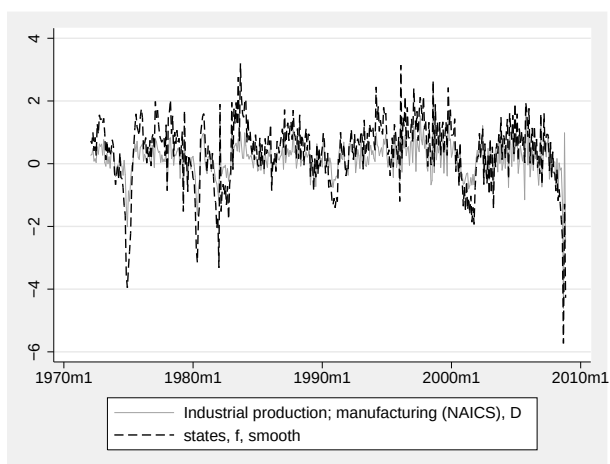


► Example 3

In this example, we want to estimate the unobserved factor instead of predicting a dependent variable. Specifying `smethod(smooth)` causes `predict` to use all sample information in estimating the states by the Kalman smoother.

Below we estimate the unobserved factor by using the estimation sample, and we graph `ipman` and the estimated factor:

```
. predict fac if e(sample), states smethod(smooth) equation(f)
. tsline D.ipman fac, xtitle("") legend(rows(2))
```

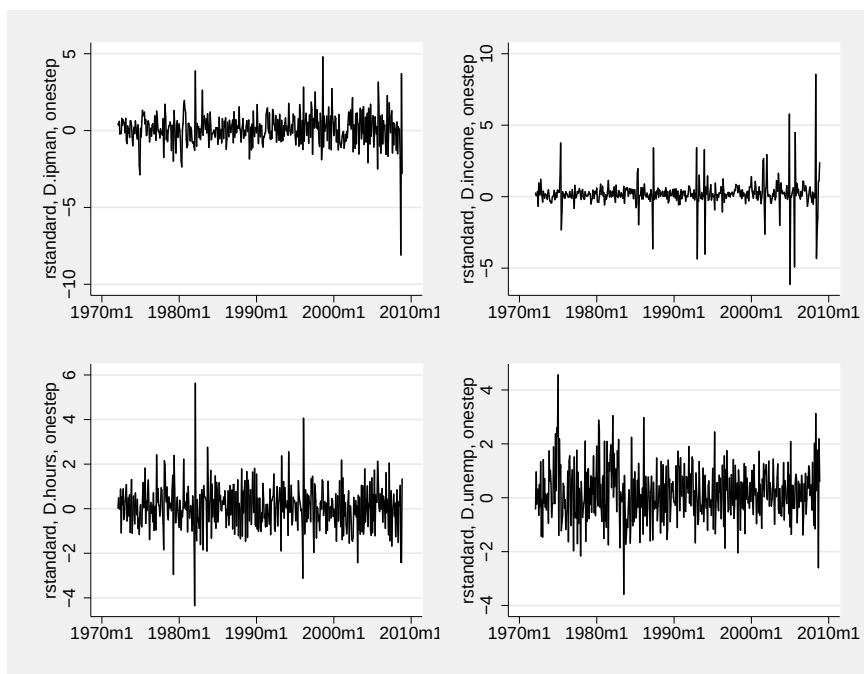


► Example 4

The residuals and the standardized residuals are frequently used to review the specification of the model.

Below we calculate the standardized residuals for each of the series and display them in a combined graph:

```
. predict sres1-sres4 if e(sample), rstandard
. tsline sres1, xtitle("") name(sres1)
. tsline sres2, xtitle("") name(sres2)
. tsline sres3, xtitle("") name(sres3)
. tsline sres4, xtitle("") name(sres4)
. graph combine sres1 sres2 sres3 sres4, name(combined)
```



◀

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Estimating the unobserved states is key to predicting the dependent variables.

By default and with the `smethod(onestep)` option, `predict` estimates the states in each period by applying the full Kalman filter to all previous periods and only making the one-step predictions to the current period. (See [Methods and formulas](#) of [\[TS\] sspace](#) for the Kalman filter equations.)

With the `smethod(filter)` option, `predict` estimates the states in each period by applying the full Kalman filter on all previous periods and the current period. The computational difference between `smethod(onestep)` and `smethod(filter)` is that `smethod(filter)` performs the update

step on the current period while `smethod(onestep)` does not. The statistical difference between `smethod(onestep)` and `smethod(filter)` is that `smethod(filter)` uses contemporaneous information on the dependent variables while `smethod(onestep)` does not.

As noted in [TS] [sspace](#), `sspace` has both a stationary and a diffuse Kalman filter. `predict` uses the same Kalman filter used for estimation.

With the `smethod(smooth)` option, `predict` estimates the states in each period using all the sample information by applying the Kalman smoother. `predict` uses the [Harvey \(1989, sec. 3.6.2\)](#) fixed-interval smoother with model-based initial values to estimate the states when the estimated parameters imply a stationary model. [De Jong \(1989\)](#) provides a computationally efficient method. [Hamilton \(1994\)](#) discusses the model-based initial values for stationary state-space models. When the model is nonstationary, the [De Jong \(1989\)](#) diffuse Kalman smoother is used to predict the states. The smoothed estimates of the states are subsequently used to predict the dependent variables.

The dependent variables are predicted by plugging in the estimated states. The residuals are calculated as the differences between the predicted and the realized dependent variables. The root mean squared errors are the square roots of the diagonal elements of the mean squared error matrices that are computed by the Kalman filter. The standardized residuals are the residuals normalized by the Cholesky factor of their mean squared error produced by the Kalman filter.

`predict` uses the [Harvey \(1989, sec. 3.5\)](#) methods to compute the dynamic forecasts and the root mean squared errors. Let τ be the period at which the dynamic forecasts begin; τ must either be in the specified sample or be in the period immediately following the specified sample.

The dynamic forecasts depend on the predicted states in the period $\tau - 1$, which `predict` obtains by running the Kalman filter or the diffuse Kalman filter on the previous sample observations. The states in the periods prior to starting the dynamic predictions may be estimated using `smethod(onestep)` or `smethod(smooth)`.

Using an `if` or `in` qualifier to alter the prediction sample can change the estimate of the unobserved states in the period prior to beginning the dynamic predictions and hence alter the dynamic predictions. The initial states are estimated using `e(b)` and the prediction sample.

References

- De Jong, P. 1988. The likelihood for a state space model. *Biometrika* 75: 165–169.
- . 1989. Smoothing and interpolation with the state-space model. *Journal of the American Statistical Association* 84: 1085–1088.
- . 1991. The diffuse Kalman filter. *Annals of Statistics* 19: 1073–1083.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Harvey, A. C. 1989. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge: Cambridge University Press.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Also see

- [TS] [sspace](#) — State-space models
- [TS] [dfactor](#) — Dynamic-factor models
- [TS] [dfactor postestimation](#) — Postestimation tools for `dfactor`
- [U] [20 Estimation and postestimation commands](#)

Syntax

tsappend , { add(#) | last(date | clock) tsfmt(string) } [options]

options	description
*add(#)	add # observations
*last(date clock)	add observations at date or clock
*tsfmt(string)	use time-series function string with last(date clock)
panel(panel_id)	add observations to panel panel_id

* Either add(#) is required, or last(date | clock) and tsfmt(string) are required.
You must tsset your data before using tsappend; see [\[TS\] tsset](#).

Menu

Statistics > Time series > Setup and utilities > Add observations to time-series dataset

Description

tsappend appends observations to a time-series dataset or to a panel dataset. tsappend uses and updates the information set by tsset.

Options

add(#) specifies the number of observations to add.

last(date | clock) and tsfmt(string) must be specified together and are an alternative to add().

last(date | clock) specifies the date or the date and time of the last observation to add.

tsfmt(string) specifies the name of the Stata time-series function to use in converting the date specified in last() to an integer. The function names are tc (clock), tC (Clock), td (daily), tw (weekly), tm (monthly), tq (quarterly), th (half-yearly), and ty (yearly).

For clock times, the last time added (if any) will be earlier than the time requested in last(date | clock) if last() is not a multiple of delta units from the last time in the data.

For instance, you might specify last(17may2007) tsfmt(td), last(2001m1) tsfmt(tm), or last(17may2007 15:30:00) tsfmt(tc).

panel(panel_id) specifies that observations be added only to panels with the ID specified in panel().

Remarks

- Remarks are presented under the following headings:
- Introduction
 - Using tsappend with time-series data
 - Using tsappend with panel data

Introduction

`tsappend` adds observations to a time-series dataset or to a panel dataset. You must `tsset` your data before using `tsappend`. `tsappend` simultaneously removes any gaps from the dataset.

There are two ways to use `tsappend`: you can specify the `add(#)` option to request that # observations be added, or you can specify the `last(date|clock)` option to request that observations be appended until the date specified is reached. If you specify `last()`, you must also specify `tsfmt()`. `tsfmt()` specifies the Stata time-series date function that converts the date held in `last()` to an integer.

`tsappend` works with time series of panel data. With panel data, `tsappend` adds the requested observations to all the panels, unless the `panel()` option is also specified.

Using tsappend with time-series data

`tsappend` can be useful for appending observations when dynamically predicting a time series. Consider an example in which `tsappend` adds the extra observations before dynamically predicting from an AR(1) regression:

```
. use http://www.stata-press.com/data/r11/tsappend1
```

```
. regress y l.y
```

Source	SS	df	MS			
Model	115.349555	1	115.349555	Number of obs =	479	
Residual	461.241577	477	.966963473	F(1, 477) =	119.29	
				Prob > F =	0.0000	
				R-squared =	0.2001	
				Adj R-squared =	0.1984	
Total	576.591132	478	1.2062576	Root MSE =	.98334	

y	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
y						
L1.	.4493507	.0411417	10.92	0.000	.3685093	.5301921
_cons	11.11877	.8314581	13.37	0.000	9.484993	12.75254

```
. mat b = e(b)
```

```
. mat colnames b = L.xb one
```

```
. tsset
```

```
time variable: t2, 1960m2 to 2000m1
delta: 1 month
```

```
. tsappend, add(12)
```

```
. tsset
```

```
time variable: t2, 1960m2 to 2001m1
delta: 1 month
```

```
. predict xb if t2<=tm(2000m2)
```

```
(option xb assumed; fitted values)
```

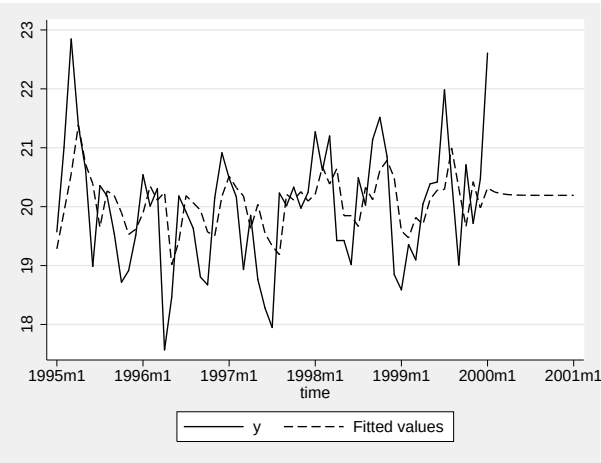
```
(12 missing values generated)
```

```
. gen one=1
```

```
. mat score xb=b if t2>=tm(2000m2), replace
```

The calls to `tsset` before and after `tsappend` were unnecessary. Their output reveals that `tsappend` added another year of observations. We then used `predict` and `matrix score` to obtain the dynamic predictions, which allows us to produce the following graph:

```
. line y xb t2 if t2>=tm(1995m1), ytitle("") xtitle("time")
```



In the call to `tsappend`, instead of saying that we wanted to add 12 observations, we could have specified that we wanted to fill in observations through the first month of 2001:

```
. use http://www.stata-press.com/data/r11/tsappend1, clear
. tsset
    time variable:  t2, 1960m2 to 2000m1
      delta: 1 month
. tsappend, last(2001m1) tsfmt(tm)
. tsset
    time variable:  t2, 1960m2 to 2001m1
      delta: 1 month
```

We specified the `tm()` function in the `tsfmt()` option. [\[D\] functions](#) contains a list of time-series functions for translating date literals to integers. Because we have monthly data, and since [\[D\] functions](#) tells us that we want to use the `tm()` function, we specified the `tsfmt(tm)` option. The following table shows the most common types of time-series data, their formats, the appropriate translation functions, and the corresponding options for `tsappend`:

Description	Format	Function	Option
time	%tc	tc()	tsfmt(tc)
time	%tC	tC()	tsfmt(tC)
daily	%td	td()	tsfmt(td)
weekly	%tw	tw()	tsfmt(tw)
monthly	%tm	tm()	tsfmt(tm)
quarterly	%tq	tq()	tsfmt(tq)
half-yearly	%th	th()	tsfmt(th)
yearly	%ty	ty()	tsfmt(ty)

Using tsappend with panel data

tsappend's actions on panel data are similar to its action on time-series data, except that tsappend performs those actions on each time series within the panels.

If the end dates vary over panels, last() and add() will produce different results. add(#) always adds # observations to each panel. If the data end at different periods before tsappend, add() is used, the data will still end at different periods after tsappend, add(). In contrast, tsappend, last() tsfmt() will cause all the panels to end on the specified last date. If the beginning dates differ across panels, using tsappend, last() tsfmt() to provide a uniform ending date will not create balanced panels because the number of observations per panel will still differ.

Consider the panel data summarized in the output below:

```
. use http://www.stata-press.com/data/r11/tsappend3, clear
```

```
. xtdescribe
```

```
id: 1, 2, ..., 3                      n = 3
t2: 1998m1, 1998m2, ..., 2000m1       T = 25
Delta(t2) = 1 month
Span(t2) = 25 periods
(id*t2 uniquely identifies each observation)
```

Distribution of T_i:	min	5%	25%	50%	75%	95%	max
	13	13	13	20	24	24	24

Freq.	Percent	Cum.	Pattern
1	33.33	33.33	111111111111
1	33.33	66.67	1111.1111111111111111
1	33.33	100.00	111111111111111111....
3	100.00		XXXXXXXXXXXXXXXXXXXX

```
. by id: sum t2
```

```
-> id = 1
```

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	13	474	3.89444	468	480

```
-> id = 2
```

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	20	465.5	5.91608	456	475

```
-> id = 3
```

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	24	468.3333	7.322786	456	480

The output from xtdescribe and summarize on these data tells us that one panel starts later than the other, that another panel ends before the other two, and that the remaining panel has a gap in the time variable but otherwise spans the entire time frame.

(Continued on next page)

Now consider the data after a call to `tsappend, add(6)`:

```
. tsappend, add(6)
. xtdescribe
      id:  1, 2, ..., 3                                n =           3
      t2: 1998m1, 1998m2, ..., 2000m7                  T =          31
           Delta(t2) = 1 month
           Span(t2)  = 31 periods
           (id*t2 uniquely identifies each observation)
```

Distribution of T_i:

	min	5%	25%	50%	75%	95%	max
	19	19	19	26	31	31	31
Freq.	Percent	Cum.	Pattern				
1	33.33	33.33	11111111111111111111				
1	33.33	66.67	1111111111111111111111111111.....				
1	33.33	100.00	1111111111111111111111111111111111				
3	100.00		XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX				

```
. by id: sum t2
```

-> id = 1

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	19	477	5.627314	468	486

-> id = 2

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	26	468.5	7.648529	456	481

-> id = 3

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	31	471	9.092121	456	486

This output from `xtdescribe` and `summarize` after the call to `tsappend` shows that the call to `tsappend, add(6)` added 6 observations to each panel and filled in the gap in the time variable in the second panel. `tsappend, add()` did not cause a uniform end date over the panels.

The following output illustrates the contrast between `tsappend, add()` and `tsappend, last()` `tsfmt()` with panel data that end at different dates. The output from `xtdescribe` and `summarize` shows that the call to `tsappend, last()` `tsfmt()` filled in the gap in `t2` and caused all the panels to end at the specified end date. The output also shows that the panels remain unbalanced because one panel has a later entry date than the other two.

```
. use http://www.stata-press.com/data/r11/tsappend2, clear
. tsappend, last(2000m7) tsfmt(tm)
```

```
. xtdescribe
      id:  1, 2, ..., 3                      n =          3
      t2: 1998m1, 1998m2, ..., 2000m7       T =         31
      Delta(t2) = 1 month
      Span(t2)  = 31 periods
      (id*t2 uniquely identifies each observation)

Distribution of T_i:  min      5%    25%    50%    75%    95%    max
                   19      19      19      31      31      31      31

      Freq.  Percent   Cum. | Pattern
-----|-----
      2      66.67   66.67 | 11111111111111111111111111111111
      1      33.33  100.00 | .....11111111111111111111111111111111
-----|-----
      3     100.00             | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

. by id: sum t2
```

-> id = 1

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	19	477	5.627314	468	486

-> id = 2

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	31	471	9.092121	456	486

-> id = 3

Variable	Obs	Mean	Std. Dev.	Min	Max
t2	31	471	9.092121	456	486

Saved results

tsappend saves the following in `r()`:

```
Scalars
      r(add)  number of observations added
```

Methods and formulas

tsappend is implemented as an ado-file.

Also see

- [TS] **tsset** — Declare data to be time-series data
- [TS] **tsfill** — Fill in gaps in time variable

Title

tsfill — Fill in gaps in time variable

Syntax

```
tsfill [ , full ]
```

You must `tsset` your data before using `tsfill`; see [\[TS\] tsset](#).

Menu

Statistics > Time series > Setup and utilities > Fill in gaps in time variable

Description

`tsfill` is used after `tsset` to fill in gaps in time-series data and gaps in panel data with new observations, which contain missing values. For instance, perhaps observations for *timevar* = 1, 3, 5, 6, ..., 22 exist. `tsfill` would create observations for *timevar* = 2 and *timevar* = 4 containing all missing values. There is seldom reason to do this because Stata's time-series operators consider *timevar* not the observation number. Referring to `L.gnp` to obtain lagged `gnp` values would correctly produce a missing value for *timevar* = 3, even if the data were not filled in. Referring to `L2.gnp` would correctly return the value of `gnp` in the first observation for *timevar* = 3, even if the data were not filled in.

Option

`full` is for use with panel data only. With panel data, `tsfill` by default fills in observations for each panel according to the minimum and maximum values of *timevar* for the panel. Thus if the first panel spanned the times 5–20 and the second panel the times 1–15, after `tsfill` they would still span the same periods; observations would be created to fill in any missing times from 5–20 in the first panel and from 1–15 in the second.

If `full` is specified, observations are created so that both panels span the time 1–20, the overall minimum and maximum of *timevar* across panels.

Remarks

Remarks are presented under the following headings:

Using tsfill with time-series data

Using tsfill with panel data

Using tsfill with time-series data

You have monthly data, with gaps:

```
. use http://www.stata-press.com/data/r11/tsfillxmpl
. tsset
      time variable:  mdate, 1995m7 to 1996m3, but with gaps
                delta:  1 month
. list mdate income
```

	mdate	income
1.	1995m7	1153
2.	1995m8	1181
3.	1995m11	1236
4.	1995m12	1297
5.	1996m1	1265
6.	1996m3	1282

You can fill in the gaps by interpolation easily with `tsfill` and `ipolate`. `tsfill` creates the missing observations:

```
. tsfill
. list mdate income
```

	mdate	income	
1.	1995m7	1153	
2.	1995m8	1181	
3.	1995m9	.	← new
4.	1995m10	.	← new
5.	1995m11	1236	
6.	1995m12	1297	
7.	1996m1	1265	
8.	1996m2	.	← new
9.	1996m3	1282	

We can now use `ipolate` (see [\[D\] ipolate](#)) to fill them in:

```
. ipolate income mdate, gen(ipinc)
. list mdate income ipinc
```

	mdate	income	ipinc
1.	1995m7	1153	1153
2.	1995m8	1181	1181
3.	1995m9	.	1199.3333
4.	1995m10	.	1217.6667
5.	1995m11	1236	1236
6.	1995m12	1297	1297
7.	1996m1	1265	1265
8.	1996m2	.	1273.5
9.	1996m3	1282	1282

Using tsfill with panel data

You have the following panel dataset:

```
. use http://www.stata-press.com/data/r11/tsfillxmpl2, clear
. tsset
    panel variable:  edlevel (unbalanced)
    time variable:  year, 1988 to 1992, but with a gap
                  delta: 1 unit
. list edlevel year income
```

	edlevel	year	income
1.	1	1988	14500
2.	1	1989	14750
3.	1	1990	14950
4.	1	1991	15100
5.	2	1989	22100
6.	2	1990	22200
7.	2	1992	22800

Just as with nonpanel time-series datasets, you can use `tsfill` to fill in the gaps:

```
. tsfill
. list edlevel year income
```

	edlevel	year	income	
1.	1	1988	14500	
2.	1	1989	14750	
3.	1	1990	14950	
4.	1	1991	15100	
5.	2	1989	22100	
6.	2	1990	22200	
7.	2	1991	.	← new
8.	2	1992	22800	

You could instead use `tsfill` to produce fully balanced panels with the `full` option:

```
. tsfill, full
. list edlevel year income, sep(0)
```

	edlevel	year	income	
1.	1	1988	14500	
2.	1	1989	14750	
3.	1	1990	14950	
4.	1	1991	15100	
5.	1	1992	.	← new
6.	2	1988	.	← new
7.	2	1989	22100	
8.	2	1990	22200	
9.	2	1991	.	← new
10.	2	1992	22800	

Methods and formulas

`tsfill` is implemented as an ado-file.

Also see

[TS] [tsset](#) — Declare data to be time-series data

[TS] [tsappend](#) — Add observations to a time-series dataset

Syntax

Time-series line plot

```
[twoway] tsline varlist [if] [in] [, tsline_options]
```

Time-series range plot with lines

```
[twoway] tsrline y1 y2 [if] [in] [, tsrline_options]
```

where the time variable is assumed set by `tsset`, *varlist* has the interpretation $y_1[y_2 \dots y_k]$.

tsline_options	description
Plots	
scatter_options	any of the options allowed by [G] graph twoway scatter with the exception of <i>marker_options</i> , <i>marker_placement_options</i> , and <i>marker_label_options</i> , which will be ignored if specified
Y axis, Time axis, Titles, Legend, Overall, By	
twoway_options	any options documented in [G] <i>twoway_options</i>

tsrline_options	description
Plots	
rline_options	any of the options allowed by [G] graph twoway rline
Y axis, Time axis, Titles, Legend, Overall, By	
twoway_options	any options documented in [G] <i>twoway_options</i>

Menu

Statistics > Time series > Graphs > Line plots

Description

`tsline` draws line plots for time-series data.

`tsrline` draws a range plot with lines for time-series data.

`tsline` and `tsrline` are both commands and *plottypes* as defined in [G] **graph twoway**. Thus the syntax for `tsline` is

```
. graph twoway tsline ...  
. twoway tsline ...
```

```
. tsline ...
```

and similarly for `tsrline`. Being plot types, these commands may be combined with other plot types in the `twoway` family, as in,

```
. twoway (tsrline ...) (tsline ...) (lfit ...) ...
```

which can equivalently be written

```
. tsrline ... || tsline ... || lfit ... || ...
```

Options

Plots

scatter_options are any of the options allowed by the `graph twoway scatter` command except that *marker_options*, *marker_placement_option*, and *marker_label_options* will be ignored if specified; see [G] [graph twoway scatter](#).

rline_options are any of the options allowed by the `graph twoway rline` command; see [G] [graph twoway rline](#).

Y axis, Time axis, Titles, Legend, Overall, By

twoway_options are any of the options documented in [G] [twoway_options](#). These include options for titling the graph (see [G] [title_options](#)), for saving the graph to disk (see [G] [saving_option](#)), and the `by()` option, which will allow you to simultaneously plot different subsets of the data (see [G] [by_option](#)).

Also see the `recast()` option discussed in [G] [advanced_options](#) for information on how to plot spikes, bars, etc., instead of lines.

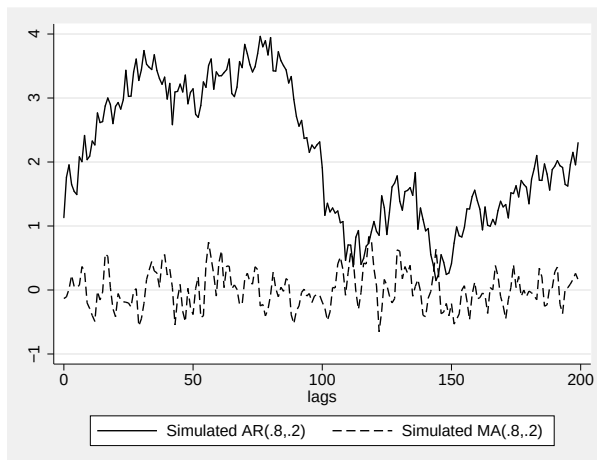
Remarks

► Example 1

We simulated two separate time series (each of 200 observations) and placed them in a Stata dataset, `tsline1.dta`. The first series simulates an AR(2) process with $\phi_1 = 0.8$ and $\phi_2 = 0.2$; the second series simulates an MA(2) process with $\theta_1 = 0.8$ and $\theta_2 = 0.2$. We use `tsline` to graph these two series.

```
. use http://www.stata-press.com/data/r11/tsline1
. tsset lags
      time variable:  lags, 0 to 199
                delta:  1 unit
. tsline ar ma
```

(Continued on next page)

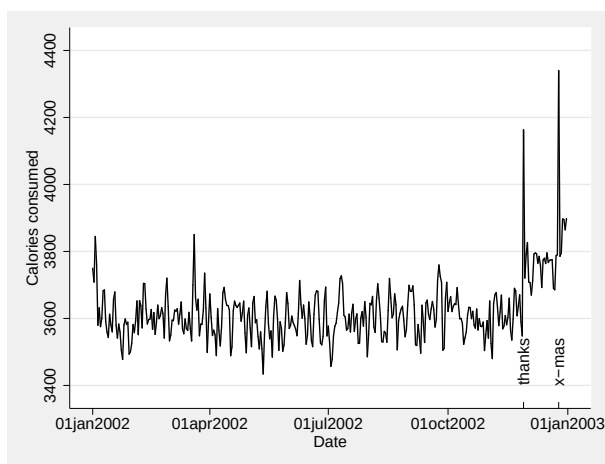


◀

► Example 2

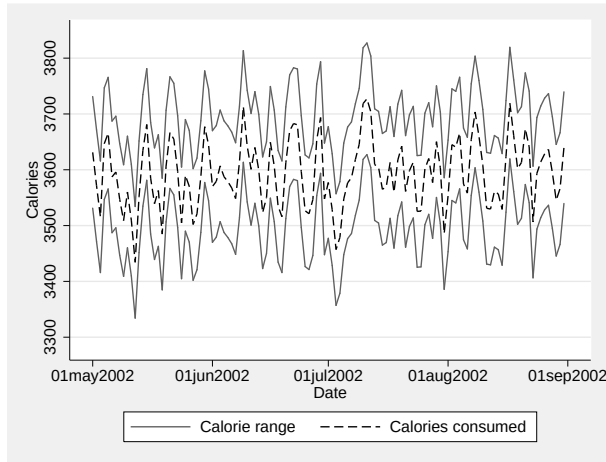
Suppose that we kept a calorie log for an entire calendar year. At the end of the year, we would have a dataset (e.g., `tsline2.dta`) that contains the number of calories consumed for 365 days. We could then use `tsset` to identify the date variable and `tsline` to plot calories versus time. Knowing that we tend to eat a little more food on Thanksgiving and Christmas day, we use the `ttick()` and `tttext()` options to point these days out on the time axis.

```
. use http://www.stata-press.com/data/r11/tsline2
. tsset day
    time variable:  day, 01jan2002 to 31dec2002
        delta: 1 day
. tsline calories, ttick(28nov2002 25dec2002, tpos(in))
> tttext(3470 28nov2002 "thanks" 3470 25dec2002 "x-mas", orient(vert))
```



We were uncertain of the exact values we logged, so we also gave a range for each day. Here is a plot of the summer months.

```
. tsline lcalories ucalories if tin(1may2002,31aug2002) || tsline cal ||
> if tin(1may2002,31aug2002), ytitle(Calories)
```



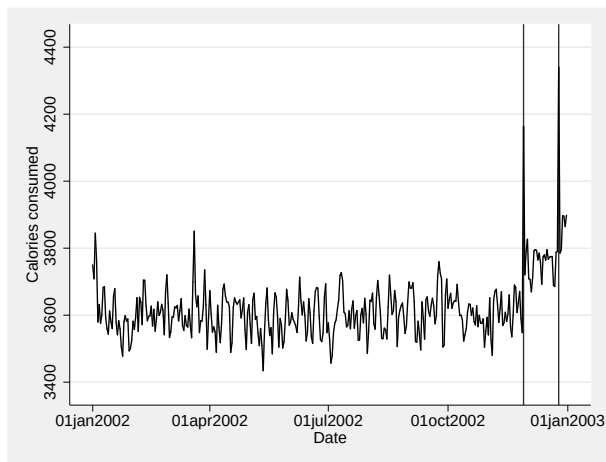
Options associated with the time axis allow dates (and times) to be specified in place of numeric date (and time) values. For instance, we used

```
ttick(28nov2002 25dec2002, tpos(in))
```

to place tick marks at the specified dates. This works similarly for `tlabel`, `tmlabel`, and `tmtick`.

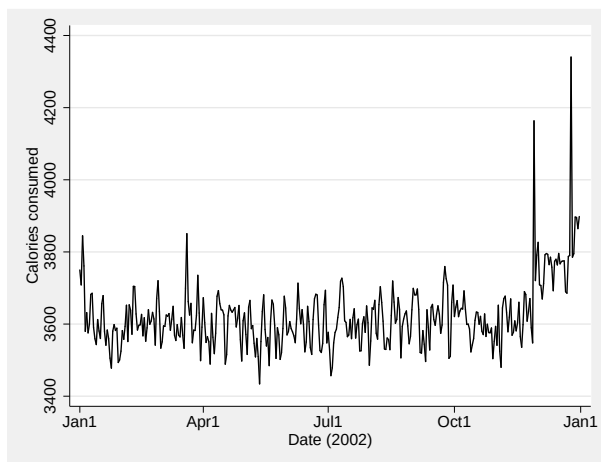
Suppose that we wanted to place vertical lines for the previously mentioned holidays. We could specify the dates in the `tline()` option as follows:

```
. tsline calories, tline(28nov2002 25dec2002)
```



We could also modify the format of the time axis so that only the day in the year is displayed in the labeled ticks:

```
. tsline calories, tlabel(, format(%tdmd)) tttitle("Date (2002)")
```



◀

Methods and formulas

`tsline` and `tsrline` are implemented as ado-files.

Also see

[TS] [tsset](#) — Declare data to be time-series data

[G] [graph twoway](#) — Twoway graphs

[XT] [xtline](#) — Panel-data line plots

Syntax

```
tsreport [if] [in] [, options]
```

options	description
Main	
<u>r</u> eport	display number of gaps in the time series
<u>r</u> epor <u>t</u> 0	display count of gaps even if no gaps
<u>l</u> ist	display gaps data in tabular list
<u>p</u> anel	ignore panel changes

tsreport typed without options produces no output but provides its standard saved results.

Menu

Statistics > Time series > Setup and utilities > Report time-series aspects of dataset

Description

tsreport reports time gaps in a sample of observations. The report option displays a one-line statement showing the count of the gaps, and list displays a list of records that follow gaps. A return value r(N_gaps) indicates the number of gaps in the sample.

Options

Main

- report specifies that the number of gaps in the time series be reported, if any gaps exist.
- report0 specifies that the count of gaps be reported, even if there are no gaps.
- list specifies that a tabular list of gaps be displayed.
- panel specifies that panel changes not be counted as gaps. Whether panel changes are counted as gaps usually depends on how the calling command handles panels.

Remarks

Time-series commands sometimes require that observations be on a fixed time interval with no gaps, or the command may not function properly. tsreport provides a tool for reporting the gaps in a sample.

➤ Example 1

The following monthly panel data have two panels and a missing month (March) in the second panel:

```
. use http://www.stata-press.com/data/r11/tsrptxmpl
. list edlevel month income in 1/6, sep(0)
```

	edlevel	month	income
1.	1	1998m1	687
2.	1	1998m2	783
3.	1	1998m3	790
4.	2	1998m1	1435
5.	2	1998m2	1522
6.	2	1998m4	1532

Invoking `tsreport` without the `panel` option gives us the following report:

```
. tsreport, report
Number of gaps in sample: 2 (gap count includes panel changes)
```

We could get a list of gaps and better see what has been counted as a gap by using the `list` option:

```
. tsreport, report list
Number of gaps in sample: 2 (gap count includes panel changes)
Observations with preceding time gaps
(gaps include panel changes)
```

Record	edlevel	month
4	2	1998m1
6	2	1998m4

We now see why `tsreport` is reporting two gaps. It is counting the known gap in March of the second panel and counting the change from the first to the second panel. (If we are programmers writing a procedure that does not account for panels, a change from one panel to the next represents a break in the time series just as a gap in the data does.)

We may prefer that the changes in panels not be counted as gaps. We can obtain a count without the panel change by using the `panel` option:

```
. tsreport, report panel
Number of gaps in sample: 1
```

To obtain a fuller report, we type

```
. tsreport, report list panel
Number of gaps in sample: 1
Observations with preceding time gaps
```

Record	edlevel	month
6	2	1998m4

Saved results

`tsreport` saves the following in `r()`:

Scalars

`r(N_gaps)` number of gaps in sample

Methods and formulas

`tsreport` is implemented as an ado-file.

Also see

[TS] [tsset](#) — Declare data to be time-series data

Title

tsrevar — Time-series operator programming command

Syntax

```
tsrevar [varlist] [if] [in] [ , substitute list ]
```

You must `tsset` your data before using `tsrevar`; see [TS] `tsset`.

Description

`tsrevar`, `substitute` takes a *varlist* that might contain *op.varname* combinations and substitutes equivalent temporary variables for the combinations.

`tsrevar`, `list` creates no new variables. It returns in `r(varlist)` the list of base variables corresponding to *varlist*.

Options

`substitute` specifies that `tsrevar` resolve *op.varname* combinations by creating temporary variables as described above. `substitute` is the default action taken by `tsrevar`; you do not need to specify the option.

`list` specifies that `tsrevar` return a list of base variable names.

Remarks

`tsrevar` substitutes temporary variables for any *op.varname* combinations in a variable list. For instance, the original *varlist* might be “`gnp L.gnp r`”, and `tsrevar`, `substitute` would create `newvar = L.gnp` and create the equivalent varlist “`gnp newvar r`”. This new varlist could then be used with commands that do not otherwise support time-series operators, or it could be used in a program to make execution faster at the expense of using more memory.

`tsrevar`, `substitute` might create no new variables, one new variable, or many new variables, depending on the number of *op.varname* combinations appearing in *varlist*. Any new variables created are temporary. The new, equivalent varlist is returned in `r(varlist)`. The new varlist corresponds one to one with the original *varlist*.

`tsrevar`, `list` returns in `r(varlist)` the list of base variable names of *varlist* with the time-series operators removed. `tsrevar`, `list` creates no new variables. For instance, if the original *varlist* were “`gnp l.gnp l2.gnp r l.cd`”, then `r(varlist)` would contain “`gnp r cd`”. This is useful for programmers who might want to create programs to keep only the variables corresponding to *varlist*.

► Example 1

```
. use http://www.stata-press.com/data/r11/tsrevarex
. tsrevar l.gnp d.gnp r
```

creates two temporary variables containing the values for `l.gnp` and `d.gnp`. The variable `r` appears in the new variable list but does not require a temporary variable.

The resulting variable list is

```
. display "r(varlist)"
__00014P __00014Q r
```

(Your temporary variable names may be different, but that is of no consequence.)

We can see the results by listing the new variables alongside the original value of `gnp`.

```
. list gnp 'r(varlist)' in 1/5
```

	gnp	__00014P	__00014Q	r
1.	128	.	.	3.2
2.	135	128	7	3.8
3.	132	135	-3	2.6
4.	138	132	6	3.9
5.	145	138	7	4.2

Temporary variables automatically vanish when the program concludes.

If we had needed only the base variable names, we could have specified

```
. tsrevar l.gnp d.gnp r, list
. display "r(varlist)"
gnp r
```

The order of the list will probably differ from that of the original list; base variables are listed only once and are listed in the order that they appear in the dataset.

◀

□ Technical note

`tsrevar`, `substitute` avoids creating duplicate variables. Consider

```
. tsrevar gnp l.gnp r cd l.cd l.gnp
```

`l.gnp` appears twice in the varlist. `tsrevar` will create only one new variable for `l.gnp` and use that new variable twice in the resulting `r(varlist)`. Moreover, `tsrevar` will even do this across multiple calls:

```
. tsrevar gnp l.gnp cd l.cd
. tsrevar cpi l.gnp
```

`l.gnp` appears in two separate calls. At the first call, `tsrevar` creates a temporary variable corresponding to `l.gnp`. At the second call, `tsrevar` remembers what it has done and uses that same temporary variable for `l.gnp` again.

□

Saved results

`tsrevar` saves the following in `r()`:

Macros

`r(varlist)` the modified variable list or list of base variable names

Also see

[P] [syntax](#) — Parse Stata syntax

[P] [unab](#) — Unabbreviate variable list

[U] [11 Language syntax](#)

[U] [11.4.4 Time-series varlists](#)

[U] [18 Programming Stata](#)

tsset — Declare data to be time-series data

Syntax

Declare data to be time series

```
tsset timevar [ , options ]
tsset panelvar timevar [ , options ]
```

Display how data are currently tsset

```
tsset
```

Clear time-series settings

```
tsset, clear
```

In the declare syntax, *panelvar* identifies the panels and *timevar* identifies the times.

options	description

Main

<i>unitoptions</i>	specify units of <i>timevar</i>

Delta

<i>deltaoption</i>	specify period of <i>timevar</i>

<i>unitoptions</i>	description

<i>(default)</i>	<i>timevar</i> 's units to be obtained from <i>timevar</i> 's display format
<u>c</u> locktime	<i>timevar</i> is %tc: 0 = 1jan1960 00:00:00.000, 1 = 1jan1960 00:00:00.001, ...
<u>d</u> aily	<i>timevar</i> is %td: 0 = 1jan1960, 1 = 2jan1960, ...
<u>w</u> eekly	<i>timevar</i> is %tw: 0 = 1960w1, 1 = 1960w2, ...
<u>m</u> onthly	<i>timevar</i> is %tm: 0 = 1960m1, 1 = 1960m2, ...
<u>q</u> uarterly	<i>timevar</i> is %tq: 0 = 1960q1, 1 = 1960q2, ...
<u>h</u> alfyearly	<i>timevar</i> is %th: 0 = 1960h1, 1 = 1960h2, ...
<u>y</u> early	<i>timevar</i> is %ty: 1960 = 1960, 1961 = 1961, ...
<u>g</u> eneric	<i>timevar</i> is %tg: 0 = ?, 1 = ?, ...
<u>f</u> ormat(<i>%fmt</i>)	specify <i>timevar</i> 's format and then apply default rule

In all cases, negative *timevar* values are allowed.

deltaoption specifies the period between observations in *timevar* units and may be specified as

<i>deltaoption</i>	example
<code>delta(#)</code>	<code>delta(1)</code> or <code>delta(2)</code>
<code>delta((exp))</code>	<code>delta((7*24))</code>
<code>delta(# units)</code>	<code>delta(7 days)</code> or <code>delta(15 minutes)</code> or <code>delta(7 days 15 minutes)</code>
<code>delta((exp) units)</code>	<code>delta((2+3) weeks)</code>

Allowed units for %tc and %tC *timevars* are

seconds	second	secs	sec
minutes	minute	mins	min
hours	hour		
days	day		
weeks	week		

and for all other %t *timevars*, units specified must match the frequency of the data; e.g., for %ty, units must be year or years.

Menu

Statistics > Time series > Setup and utilities > Declare dataset to be time-series data

Description

`tsset` declares the data in memory to be a time series. `tssetting` the data is what makes Stata’s time-series operators such as `L.` and `F.` (lag and lead) work; the operators are discussed under [Remarks](#) below. Also, before using the other `ts` commands, you must `tsset` the data first. If you `save` the data after `tsset`, the data will be remembered to be time series and you will not have to `tsset` again.

There are two syntaxes for setting the data:

```
tsset timevar
tsset panelvar timevar
```

In the first syntax—`tsset timevar`—the data are set to be a straight time series.

In the second syntax—`tsset panelvar timevar`—the data are set to be a collection of time series, one for each value of *panelvar*, also known as panel data, cross-sectional time-series data, and `xt` data. Such datasets can be analyzed by `xt` commands as well as `ts` commands. If you `tsset panelvar timevar`, you do not need to `xtset panelvar timevar` to use the `xt` commands.

`tsset` without arguments—`tsset`—displays how the data are currently `tsset` and sorts the data on *timevar* or *panelvar timevar* if they are sorted differently from that.

`tsset, clear` is a rarely used programmer’s command to declare that the data are no longer a time series.

Options

Main

unitoptions *clocktime*, *daily*, *weekly*, *monthly*, *quarterly*, *halfyearly*, *yearly*, *generic*, and *format(%fmt)* specify the units in which *timevar* is recorded.

timevar will usually be a %t variable; see [D] [dates and times](#). If *timevar* already has a %t display format assigned to it, you do not need to specify a *unitoption*; *tsset* will obtain the units from the format. If you have not yet bothered to assign the appropriate %t format, however, you can use the *unitoptions* to tell *tsset* the units. Then *tsset* will set *timevar*'s display format for you. Thus, the *unitoptions* are convenience options; they allow you to skip formatting the time variable. The following all have the same net result:

Alternative 1	Alternative 2	Alternative 3
<code>format t %td</code>	<i>(t not formatted)</i>	<i>(t not formatted)</i>
<code>tsset t</code>	<code>tsset t, daily</code>	<code>tsset t, format(%td)</code>

timevar is not required to be a %t variable; it can be any variable of your own concocting so long as it takes on only integer values. In such cases, it is called *generic* and considered to be %tg. Specifying the *unitoption* *generic* or attaching a special format to *timevar*, however, is not necessary because *tsset* will assume that the variable is *generic* if it has any numerical format other than a %t format (or if it has a %tg format).

clear—used in *tsset*, *clear*—makes Stata forget that the data ever were *tsset*. This is a rarely used programmer's option.

Delta

delta() specifies the period of *timevar* and is commonly used when *timevar* is %tc. *delta()* is only sometimes used with the other %t formats or with *generic* time variables.

If *delta()* is not specified, *delta(1)* is assumed. This means that at *timevar* = 5, the previous time is *timevar* = 5 − 1 = 4 and the next time would be *timevar* = 5 + 1 = 6. Lag and lead operators, for instance, would work this way. This would be assumed regardless of the units of *timevar*.

If you specified *delta(2)*, then at *timevar* = 5, the previous time would be *timevar* = 5 − 2 = 3 and the next time would be *timevar* = 5 + 2 = 7. Lag and lead operators would work this way. In the observation with *timevar* = 5, *L.price* would be the value of *price* in the observation for which *timevar* = 3 and *F.price* would be the value of *price* in the observation for which *timevar* = 7. If you then add an observation with *timevar* = 4, the operators will still work appropriately; i.e., at *timevar* = 5, *L.price* will still have the value of *price* at *timevar* = 3.

There are two aspects of *timevar*: its units and its periodicity. The *unitoptions* set the units. *delta()* sets the periodicity.

We mentioned that *delta()* is commonly used with %tc *timevars* because Stata's %tc variables have units of milliseconds. If *delta()* is not specified and in some model you refer to *L.price*, you will be referring to the value of *price* 1 ms ago. Few people have data with periodicity of a millisecond. Perhaps your data are hourly. You could specify *delta(3600000)*. Or you could specify *delta((60*60*1000))*, because *delta()* will allow expressions if you include an extra pair of parentheses. Or you could specify *delta(1 hour)*. They all mean the same thing: *timevar* has periodicity of 3,600,000 ms. In an observation for which *timevar* = 1,489,572,000,000 (corresponding to 15mar2007 10:00:00), *L.price* would be the observation for which *timevar* = 1,489,572,000,000 − 3,600,000 = 1,489,568,400,000 (corresponding to 15mar2007 9:00:00).

When you `tsset` the data and specify `delta()`, `tsset` verifies that all the observations follow the specified periodicity. For instance, if you specified `delta(2)`, then *timevar* could contain any subset of $\{\dots, -4, -2, 0, 2, 4, \dots\}$ or it could contain any subset of $\{\dots, -3, -1, 1, 3, \dots\}$. If *timevar* contained a mix of values, `tsset` would issue an error message. If you also specify a *panelvar*—you type `tsset panelvar timevar, delta(2)`—the check is made on each panel independently. One panel might contain *timevar* values from one set and the next, another, and that would be fine.

Remarks

`tsset` sets *timevar* so that Stata’s time-series operators are understood in varlists and expressions. The time-series operators are

Operator	Meaning
L.	lag x_{t-1}
L2.	2-period lag x_{t-2}
...	
F.	lead x_{t+1}
F2.	2-period lead x_{t+2}
...	
D.	difference $x_t - x_{t-1}$
D2.	difference of difference $x_t - x_{t-1} - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2}$
...	
S.	“seasonal” difference $x_t - x_{t-1}$
S2.	lag-2 (seasonal) difference $x_t - x_{t-2}$
...	

Time-series operators may be repeated and combined. `L3.gnp` refers to the third lag of variable `gnp`, as do `LLL.gnp`, `LL2.gnp`, and `L2L.gnp`. `LF.gnp` is the same as `gnp`. `DS12.gnp` refers to the one-period difference of the 12-period difference. `LDS12.gnp` refers to the same concept, lagged once.

`D1.` = `S1.`, but `D2.` $\neq$ `S2.`, `D3.` $\neq$ `S3.`, and so on. `D2.` refers to the difference of the difference. `S2.` refers to the two-period difference. If you wanted the difference of the difference of the 12-period difference of `gnp`, you would write `D2S12.gnp`.

Operators may be typed in uppercase or lowercase. Most users would type `d2s12.gnp` instead of `D2S12.gnp`.

You may type operators however you wish; Stata internally converts operators to their canonical form. If you typed `ld2ls12d.gnp`, Stata would present the operated variable as `L2D3S12.gnp`.

Stata also understands *operator(numlist)* . to mean a set of operated variables. For instance, typing `L(1/3).gnp` in a varlist is the same as typing ‘`L.gnp L2.gnp L3.gnp`’. The operators can also be applied to a list of variables by enclosing the variables in parentheses; e.g.,


```
. list year L(1/3).(gnp cpi)
```

	year	L.gnp	L2.gnp	L3.gnp	L.cpi	L2.cpi	L3.cpi
1.	1989	.	.	.	.	.	.
2.	1990	5452.8	.	.	100	.	.
3.	1991	5764.9	5452.8	.	105	100	.
4.	1992	5932.4	5764.9	5452.8	108	105	100
			(output omitted)				
8.	1996	7330.1	6892.2	6519.1	122	119	112

In *operator#.*, making # zero returns the variable itself. *L0.gnp* is *gnp*. Thus, you can type `list year l(0/3).gnp` to mean `list year gnp L.gnp L2.gnp L3.gnp`.

The parenthetical notation may be used with any operator. Typing `D(1/3).gnp` would return the first through third differences.

The parenthetical notation may be used in operator lists with multiple operators, such as `L(0/3)D2S12.gnp`.

Operator lists may include up to one set of parentheses, and the parentheses may enclose a *numlist*; see [U] 11.1.8 *numlist*.

Before you can use these time-series operators, however, the dataset must satisfy two requirements:

1. the dataset must be *tsset* and
2. the dataset must be sorted by *timevar* or, if it is a cross-sectional time-series dataset, by *panelvar timevar*.

tsset handles both requirements. As you use Stata, however, you may later use a command that re-sorts that data, and if you do, the time-series operators will not work:

```
. tsset time
(output omitted)
. regress y x l.x
(output omitted)
. (you continue to use Stata and, sometime later:)
. regress y x l.x
not sorted
r(5);
```

Then typing *tsset* without arguments will reestablish the sort order:

```
. tsset
(output omitted)
. regress y x l.x
(output omitted)
```

Here typing *tsset* is the same as typing `sort time`. Had we previously `tsset country time`, however, typing *tsset* would be the same as typing `sort country time`. You can type the `sort` command or type *tsset* without arguments; it makes no difference.

There are two syntaxes for setting your data:

```
tsset timevar
tsset panelvar timevar
```

In both, *timevar* must contain integer values. If *panelvar* is specified, it too must contain integer values, and the dataset is declared to be a cross-section of time series, such as a collection of time series for different countries.

► Example 1: Numeric time variable

You have monthly data on personal income. Variable `t` records the time of an observation, but there is nothing special about the name of the variable. There is nothing special about the values of the variable, either. `t` is not required to be `%tm` variable—perhaps you do not even know what that means. `t` is just a numeric variable containing integer values that represent the month, and we will imagine that `t` takes on the values 1, 2, ..., 9, although it could just as well be -3, -2 ..., 5, or 1,023, 1,024, ..., 1,031. What is important is that the values are dense: adjacent months have a time value that differs by 1.

```
. use http://www.stata-press.com/data/r11/tssetxmpl
. list t income
```

	t	income
1.	1	1153
2.	2	1181
	(output omitted)	
9.	9	1282

```
. tsset t
      time variable:  t, 1 to 9
              delta:  1 unit

. regress income l.income
(output omitted)
```

◀

► Example 2: Adjusting the starting date

In the example above, that `t` started at 1 was not important. As we said, the `t` variable could just as well be recorded -3, -2 ..., 5, or 1,023, 1,024, ..., 1,031. What is important is that the difference in `t` between observations be `delta()` when there are no gaps.

Although how time is measured makes no difference, Stata has formats to display time nicely if it is recorded in certain ways; you can learn about the formats by seeing [D] [dates and times](#). Stata likes time variables in which 1jan1960 is recorded as 0. In our previous example, if `t = 1` corresponds to July 1995, then we could make a variable that fits Stata's preference by typing

```
. generate newt = tm(1995m7) + t - 1
```

`tm()` is the function that returns a month equivalent; `tm(1995m7)` evaluates to the constant 426, meaning 426 months after January 1960. We now have variable `newt` containing

```
. list t newt income
```

	t	newt	income
1.	1	426	1153
2.	2	427	1181
3.	3	428	1208
	(output omitted)		
9.	9	434	1282

If we put a `%tm` format on `newt`, it will display more cleanly:

```
. format newt %tm
. list t newt income
```

	t	newt	income
1.	1	1995m7	1153
2.	2	1995m8	1181
3.	3	1995m9	1208
		<i>(output omitted)</i>	
9.	9	1996m3	1282

We could now `tsset newt` rather than `t`:

```
. tsset newt
      time variable:  newt, 1995m7 to 1996m3
              delta:  1 month
```



□ Technical note

In addition to monthly, Stata understands clock times (to the millisecond level) as well as daily, weekly, quarterly, half-yearly, and yearly data. See [\[D\] dates and times](#) for a description of these capabilities.

Let's reconsider the previous example, but rather than monthly, let's assume the data are daily, weekly, etc. The only thing to know is that, corresponding to function `tm()`, there are functions `td()`, `tw()`, `tq()`, `th()`, and `ty()` and that, corresponding to format `%tm`, there are formats `%td`, `%tw`, `%tq`, `%th`, and `%ty`. Here is what we would have typed had our data been on a different time scale:

```
Daily:      if your t variable had t=1 corresponding to 15mar1993
             . gen newt = td(15mar1993) + t - 1
             . tsset newt, daily

Weekly:     if your t variable had t=1 corresponding to 1994w1:
             . gen newt = tw(1994w1) + t - 1
             . tsset newt, weekly

Monthly:    if your t variable had t=1 corresponding to 2004m7:
             . gen newt = tm(2004m7) + t - 1
             . tsset newt, monthly

Quarterly:  if your t variable had t=1 corresponding to 1994q1:
             . gen newt = tq(1994q1) + t - 1
             . tsset newt, quarterly

Half-yearly: if your t variable had t=1 corresponding to 1921h2:
             . gen newt = th(1921h2) + t - 1
             . tsset newt, halfyearly

Yearly:     if your t variable had t=1 corresponding to 1842:
             . gen newt = 1842 + t - 1
             . tsset newt, yearly
```

In each example above, we subtracted one from our time variable in constructing the new time variable `newt` because we assumed that our starting time value was 1. For the quarterly example, if our starting time value were 5 and that corresponded to 1994q1, we would type

```
. generate newt = tq(1994q1) + t - 5
```

Had our initial time value been `t = 742` and that corresponded to 1994q1, we would have typed

```
. generate newt = tq(1994q1) + t - 742
```



➤ Example 3: Time-series data but no time variable

Perhaps we have the same time-series data but no time variable:

```
. use http://www.stata-press.com/data/r11/tssetxmpl2, clear
. list income
```

	income
1.	1153
2.	1181
3.	1208
4.	1272
5.	1236
6.	1297
7.	1265
8.	1230
9.	1282

Say that we know that the first observation corresponds to July 1995 and continues without gaps. We can create a monthly time variable and format it by typing

```
. generate t = tm(1995m7) + _n - 1
. format t %tm
```

We can now `tsset` our dataset and `list` it:

```
. tsset t
      time variable:  t, 1995m7 to 1996m3
              delta:  1 month
. list t income
```

	t	income
1.	1995m7	1153
2.	1995m8	1181
3.	1995m9	1208
	<i>(output omitted)</i>	
9.	1996m3	1282



➤ Example 4: Time variable as a string

Your data might include a time variable that is encoded into a string. In the example below each monthly observation is identified by string variable `yrmo` containing the month and year of the observation, sometimes with punctuation between:

```
. use http://www.stata-press.com/data/r11/tssetxmpl, clear
. list yrmo income
```

	yrmo	income
1.	7/1995	1153
2.	8/1995	1181
3.	9-1995	1208
4.	10,1995	1272
5.	11 1995	1236
6.	12 1995	1297
7.	1/1996	1265
8.	2.1996	1230
9.	3- 1996	1282

The first step is to convert the string to a numeric representation. Doing so is easy using the `monthly()` function; see [\[D\] dates and times](#).

```
. gen mdate = monthly(yrmo, "MY")
. list yrmo mdate income
```

	yrmo	mdate	income
1.	7/1995	426	1153
2.	8/1995	427	1181
3.	9-1995	428	1208
<i>(output omitted)</i>			
9.	3- 1996	434	1282

Our new variable, `mdate`, contains the number of months from January 1960. Now that we have numeric variable `mdate`, we can `tsset` the data:

```
. format mdate %tm
. tsset mdate
      time variable:  mdate, 1995m7 to 1996m3
              delta:  1 month
```

In fact, we can combine the two and type

```
. tsset mdate, format(%tm)
      time variable:  mdate, 1995m7 to 1996m3
              delta:  1 month
```

or type

```
. tsset mdate, monthly
      time variable:  mdate, 1995m7 to 1996m3
              delta:  1 month
```

(Continued on next page)

In all cases, we obtain

```
. list yrmo mdate income
```

	yrmo	mdate	income
1.	7/1995	1995m7	1153
2.	8/1995	1995m8	1181
3.	9-1995	1995m9	1208
4.	10,1995	1995m10	1272
5.	11 1995	1995m11	1236
6.	12 1995	1995m12	1297
7.	1/1996	1996m1	1265
8.	2.1996	1996m2	1230
9.	3- 1996	1996m3	1282

Stata can translate many different date formats, including strings like 12jan2009; January 12, 2009; 12-01-2009; 01/12/2009; 01/12/09; 12jan2009 8:14; 12-01-2009 13:12; 01/12/09 1:12 pm; Wed Jan 31 13:03:25 CST 2009; 1998q1; and more. See [\[D\] dates and times](#).



➤ Example 5: Time-series data with gaps

Gaps in the time series cause no difficulties:

```
. use http://www.stata-press.com/data/r11/tssetxmpl3, clear
. list yrmo income
```

	yrmo	income
1.	7/1995	1153
2.	8/1995	1181
3.	11 1995	1236
4.	12 1995	1297
5.	1/1996	1265
6.	3- 1996	1282

```
. gen mdate = monthly(yrmo, "MY")
. tsset mdate, monthly
    time variable:  mdate, 1995m7 to 1996m3, but with gaps
                delta:  1 month
```

Once the dataset has been `tsset`, we can use the time-series operators. The `D` operator specifies first differences:

```
. list mdate income d.income
```

	mdate	income	D.income
1.	1995m7	1153	.
2.	1995m8	1181	28
3.	1995m11	1236	.
4.	1995m12	1297	61
5.	1996m1	1265	-32
6.	1996m3	1282	.

We can use the operators in an expression or varlist context; we do not have to create a new variable to hold `D.income`. We can use `D.income` with the `list` command, with `regress` or any other Stata command that allows time-series varlists.



► Example 6: Clock times

We have data from a large hotel in Las Vegas that changes the reservation prices for its rooms hourly. A piece of the data looks like

```
. use http://www.stata-press.com/data/r11/tssetxmpl14, clear
. list in 1/5
```

	time	price
1.	02.13.2007 08:00	140
2.	02.13.2007 09:00	155
3.	02.13.2007 10:00	160
4.	02.13.2007 11:00	155
5.	02.13.2007 12:00	160

Variable `time` is a string variable. The first step in making this dataset a time-series dataset is to translate the string to a numeric variable:

```
. generate double t = clock(time, "MDY hm")
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	1.487e+12
2.	02.13.2007 09:00	155	1.487e+12
3.	02.13.2007 10:00	160	1.487e+12
4.	02.13.2007 11:00	155	1.487e+12
5.	02.13.2007 12:00	160	1.487e+12

See [D] [dates and times](#) for an explanation of what is going on here. `clock()` is the function that converts strings to date–time (`%tc`) values. We typed `clock(time, "MDY hm")` to convert string variable `time`, and we told `clock()` that the values in `time` were in the order month, day, year, hour, and minute. We stored new variable `t` as a `double` because time values are large, and doing so is required to prevent rounding. Even so, the resulting values `1.487e+12` look rounded, but that is only because of the default display format for new variables. We can see the values better if we change the format:

```
. format t %20.0gc
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	1,486,972,800,000
2.	02.13.2007 09:00	155	1,486,976,400,000
3.	02.13.2007 10:00	160	1,486,980,000,000
4.	02.13.2007 11:00	155	1,486,983,600,000
5.	02.13.2007 12:00	160	1,486,987,200,000

Even better would be to change the format to %tc—Stata’s clock-time format:

```
. format t %tc
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	13feb2007 08:00:00
2.	02.13.2007 09:00	155	13feb2007 09:00:00
3.	02.13.2007 10:00	160	13feb2007 10:00:00
4.	02.13.2007 11:00	155	13feb2007 11:00:00
5.	02.13.2007 12:00	160	13feb2007 12:00:00

We could drop variable `time`. New variable `t` contains the same information as `time` and `t` is better because it is a Stata time variable, the most important property of which being that it is numeric rather than string. We can `tsset` it. Here, however, we also need to specify the period with `tsset’s` `delta()` option. Stata’s time variables are numeric, but they record milliseconds since 01jan1960 00:00:00. By default, `tsset` uses `delta(1)`, and that means the time-series operators would not work as we want them to work. For instance, `L.price` would look back only 1 ms (and find nothing). We want `L.price` to look back 1 hour (3,600,000 ms):

```
. tsset t, delta(1 hour)
      time variable:  t,
                    13feb2007 08:00:00.000 to 13feb2007 14:00:00.000
      delta: 1 hour
. list t price l.price in 1/5
```

	t	price	L.price
1.	13feb2007 08:00:00	140	.
2.	13feb2007 09:00:00	155	140
3.	13feb2007 10:00:00	160	155
4.	13feb2007 11:00:00	155	160
5.	13feb2007 12:00:00	160	155

4

➤ Example 7: Clock times must be double

In the previous example, it was of vital importance that when we generated the %tc variable `t`,

```
. generate double t = clock(time, "MDY hm")
```

we generated it as a double. Let’s see what would have happened had we forgotten and just typed `generate t = clock(time, "MDY hm")`. Let’s go back and start with the same original data:

```
. use http://www.stata-press.com/data/r11/tssetxmpl4, clear
. list in 1/5
```

	time	price
1.	02.13.2007 08:00	140
2.	02.13.2007 09:00	155
3.	02.13.2007 10:00	160
4.	02.13.2007 11:00	155
5.	02.13.2007 12:00	160

Remember, variable `time` is a string variable, and we need to translate it to numeric. So we translate, but this time we forget to make the new variable a double:

```
. generate t = clock(time, "MDY hm")
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	1.49e+12
2.	02.13.2007 09:00	155	1.49e+12
3.	02.13.2007 10:00	160	1.49e+12
4.	02.13.2007 11:00	155	1.49e+12
5.	02.13.2007 12:00	160	1.49e+12

We see the first difference—`t` now lists as 1.49e+12 rather than 1.487e+12 as it did previously—but this is nothing that would catch our attention. We would not even know that the value is different. Let's continue.

We next put a `%20.0gc` format on `t` to better see the numerical values. In fact, that is not something we would usually do in an analysis. We did that in the example to emphasize to you that the `t` values were really big numbers. We will repeat the exercise just to be complete, but in real analysis, we would not bother.

```
. format t %20.0gc
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	1,486,972,780,544
2.	02.13.2007 09:00	155	1,486,976,450,560
3.	02.13.2007 10:00	160	1,486,979,989,504
4.	02.13.2007 11:00	155	1,486,983,659,520
5.	02.13.2007 12:00	160	1,486,987,198,464

Okay, we see big numbers in `t`. Let's continue.

Next we put a `%tc` format on `t`, and that is something we would usually do, and you should always do. You should also list a bit of the data, as we did:

```
. format t %tc
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	13feb2007 07:59:40
2.	02.13.2007 09:00	155	13feb2007 09:00:50
3.	02.13.2007 10:00	160	13feb2007 09:59:49
4.	02.13.2007 11:00	155	13feb2007 11:00:59
5.	02.13.2007 12:00	160	13feb2007 11:59:58

By now, you should see a problem: the translated date–time values are off by a second or two. That was caused by rounding. Dates and times should be the same, not approximately the same, and when you see a difference like this, you should say to yourself, “The translation is off a little. Why is that?” and then you should think, “Of course, rounding. I bet that I did not create `t` as a double.”

Let us assume, however, that you do not do this. You instead plow ahead:

```
. tsset t, delta(1 hour)
time values with period less than delta() found
r(451);
```

And that is what will happen when you forget to create `t` as a double. The rounding will cause uneven period, and `tsset` will complain.

By the way, it is only important that clock times (`%tc` and `%tC` variables) be stored as doubles. The other date values `%td`, `%tw`, `%tm`, `%tq`, `%th`, and `%ty` are small enough that they can safely be stored as floats, although forgetting and storing them as doubles does no harm.

◀

□ Technical note

Stata provides two clock-time formats, `%tc` and `%tC`. `%tC` provides a clock with leap seconds. Leap seconds are occasionally inserted to account for randomness of the earth's rotation, which gradually slows. Unlike the extra day inserted in leap years, the timing of when leap seconds will be inserted cannot be foretold. The authorities in charge of such matters announce a leap second approximately 6 months before insertion. Leap seconds are inserted at the end of the day, and the leap second is called 23:59:60 (i.e., 11:59:60 pm), which is then followed by the usual 00:00:00 (12:00:00 am). Most nonastronomers find these leap seconds vexing. The added seconds cause problems because of their lack of predictability—knowing how many seconds there will be between 01jan2012 and 01jan2013 is not possible—and because there are not necessarily 24 hours in a day. If you use a leap second adjusted-clock, most days have 24 hours, but a few have 24 hours and 1 second. You must look at a table to find out.

From a time-series analysis point of view, the nonconstant day causes the most problems. Let's say that you have data on blood pressure, taken hourly at 1:00, 2:00, . . . , and that you have `tsset` your data with `delta(1 hour)`. On most days, `L24.bp` would be blood pressure at the same time yesterday. If the previous day had a leap second, however, and your data were recorded using a leap second adjusted-clock, there would be no observation `L24.bp` because 86,400 seconds before the current reading does not correspond to an on-the-hour time; 86,401 seconds before the current reading corresponds to yesterday's time. Thus, whenever possible, using Stata's `%tc` encoding rather than `%tC` is better.

When times are recorded by computers using leap second-adjusted clocks, however, avoiding `%tC` is not possible. For performing most time-series analysis, the recommended procedure is to map the `%tC` values to `%tc` and then `tsset` those. You must ask yourself whether the process you are studying is based on the clock—the nurse does something at 2 o'clock every day—or the true passage of time—the emitter spits out an electron every 86,400,000 ms.

When dealing with computer-recorded times, first find out whether the computer (and its time-recording software) use a leap second-adjusted clock. If it does, translate that to a `%tC` value. Then use function `cofC()` to convert to a `%tc` value and `tsset` that. If variable `T` contains the `%tC` value,

```
. gen double t = cofC(T)
. format t %tc
. tsset t, delta(...)
```

Function `cofC()` moves leap seconds forward: 23:59:60 becomes 00:00:00 of the next day.

□

Panel data

► Example 8: Time-series data for multiple groups

Assume that we have a time series on average annual income and that we have the series for two groups: individuals who have not completed high school (`edlevel = 1`) and individuals who have (`edlevel = 2`).

```
. use http://www.stata-press.com/data/r11/tssetxmpl5, clear
. list edlevel year income, sep(0)
```

	edlevel	year	income
1.	1	1988	14500
2.	1	1989	14750
3.	1	1990	14950
4.	1	1991	15100
5.	2	1989	22100
6.	2	1990	22200
7.	2	1992	22800

We declare the data to be a panel by typing

```
. tsset edlevel year, yearly
      panel variable:  edlevel, (unbalanced)
      time variable:   year, 1988 to 1992, but with a gap
                      delta: 1 year
```

Having `tsset` the data, we can now use time-series operators. The difference operator, for example, can be used to list annual changes in income:

```
. list edlevel year income d.income, sep(0)
```

	edlevel	year	income	D.income
1.	1	1988	14500	.
2.	1	1989	14750	250
3.	1	1990	14950	200
4.	1	1991	15100	150
5.	2	1989	22100	.
6.	2	1990	22200	100
7.	2	1992	22800	.

We see that in addition to producing missing values due to missing times, the difference operator correctly produced a missing value at the start of each panel. Once we have `tsset` our panel data, we can use time-series operators and be assured that they will handle missing time periods and panel changes correctly.

Saved results

`tsset` saves the following in `r()`:

Scalars

<code>r(imin)</code>	minimum panel ID
<code>r(imax)</code>	maximum panel ID
<code>r(tmin)</code>	minimum time
<code>r(tmax)</code>	maximum time
<code>r(tdelta)</code>	delta

Macros

<code>r(panelvar)</code>	name of panel variable
<code>r(timevar)</code>	name of time variable
<code>r(tdeltas)</code>	formatted delta
<code>r(tmins)</code>	formatted minimum time
<code>r(tmaxs)</code>	formatted maximum time
<code>r(tsfmt)</code>	%fmt of time variable
<code>r(unit)</code>	units of time variable: Clock, clock, daily, weekly, monthly, quarterly, halfyearly, yearly, or generic
<code>r(unit1)</code>	units of time variable: C, c, d, w, m, q, h, y, or ""
<code>r(balanced)</code>	unbalanced, weakly balanced, or strongly balanced; a set of panels are strongly balanced if they all have the same time values, otherwise balanced if same number of time values, otherwise unbalanced

Methods and formulas

`tsset` is implemented as an ado-file.

Reference

Baum, C. F. 2000. [sts17: Compacting time series data](#). *Stata Technical Bulletin* 57: 44–45. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 369–370. College Station, TX: Stata Press.

Also see

[TS] [tsfill](#) — Fill in gaps in time variable

Syntax

```
tssmooth smoother [type] newvar = exp [if] [in] [, ...]
```

Smoother category	<i>smoother</i>
Moving average	
with uniform weights	ma
with specified weights	ma
Recursive	
exponential	exponential
double exponential	dexponential
nonseasonal Holt–Winters	hwinters
seasonal Holt–Winters	shwinters
Nonlinear filter	nl

See [\[TS\] tssmooth ma](#), [\[TS\] tssmooth exponential](#), [\[TS\] tssmooth dexponential](#), [\[TS\] tssmooth hwinters](#), [\[TS\] tssmooth shwinters](#), and [\[TS\] tssmooth nl](#).

Description

tssmooth creates new variable *newvar* and fills it in by passing the specified expression (usually a variable name) through the requested smoother.

Remarks

The recursive smoothers may also be used for forecasting univariate time series; indeed, the Holt–Winters methods are used almost exclusively for this. All can perform dynamic out-of-sample forecasts, and the smoothing parameters may be chosen to minimize the in-sample sum-of-squared prediction errors.

The moving-average and nonlinear smoothers are generally used to extract the trend—or signal—from a time series while omitting the high-frequency or noise components.

All smoothers work both with time-series data and panel data. When used with panel data, the calculation is performed separately within panel.

Several texts provide good introductions to the methods available in tssmooth. [Chatfield \(2004\)](#) discusses how these methods fit into time-series analysis in general. [Abraham and Ledolter \(1983\)](#); [Montgomery, Johnson, and Gardiner \(1990\)](#); [Bowerman and O’Connell \(1993\)](#); and [Chatfield \(2001\)](#) discuss using these methods for modern time-series forecasting.

References

- Abraham, B., and J. Ledolter. 1983. *Statistical Methods for Forecasting*. New York: Wiley.
- Bowerman, B. L., and R. T. O'Connell. 1993. *Forecasting and Time Series: An Applied Approach*. 3rd ed. Pacific Grove, CA: Duxbury.
- Chatfield, C. 2001. *Time-Series Forecasting*. London: Chapman & Hall/CRC.
- . 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- Chatfield, C., and M. Yar. 1988. Holt-Winters forecasting: Some practical issues. *Statistician* 37: 129–140.
- Holt, C. C. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20: 5–10.
- Montgomery, D. C., L. A. Johnson, and J. S. Gardiner. 1990. *Forecasting and Time Series Analysis*. 2nd ed. New York: McGraw-Hill.
- Winters, P. R. 1960. Forecasting sales by exponentially weighted moving averages. *Management Science* 6: 324–342.

Also see

- [TS] **tsset** — Declare data to be time-series data
- [TS] **arima** — ARIMA, ARMAX, and other dynamic regression models
- [TS] **sspace** — State-space models
- [R] **smooth** — Robust nonlinear smoother

Syntax

```
tssmooth dexponential [type] newvar = exp [if] [in] [, options]
```

options	description
Main	
replace	replace <i>newvar</i> if it already exists
parms(α)	use α as smoothing parameter
samp0(<i>#</i>)	use <i>#</i> observations to obtain initial values for recursions
s0(<i>#</i> ₁ <i>#</i> ₂)	use <i>#</i> ₁ and <i>#</i> ₂ as initial values for recursions
forecast(<i>#</i>)	use <i>#</i> periods for the out-of-sample forecast
You must <code>tsset</code> your data before using <code>tssmooth</code> ; see [TS] tsset . <i>exp</i> may contain time-series operators; see [U] 11.4.4 Time-series varlists .	

Menu

Statistics > Time series > Smoothers/univariate forecasters > Double-exponential smoothing

Description

`tssmooth dexponential` models the trend of a variable whose difference between changes from the previous values is serially correlated. More precisely, it models a variable whose second difference follows a low-order, moving-average process.

Options

Main
<code>replace</code> replaces <i>newvar</i> if it already exists.
<code>parms(α)</code> specifies the parameter α for the double-exponential smoothers; $0 < \alpha < 1$. If <code>parms(α)</code> is not specified, the smoothing parameter is chosen to minimize the in-sample sum-of-squared forecast errors.
<code>samp0(<i>#</i>)</code> and <code>s0(<i>#</i>₁ <i>#</i>₂)</code> are mutually exclusive ways of specifying the initial values for the recursion. By default, initial values are obtained by fitting a linear regression with a time trend, using the first half of the observations in the dataset; see <i>Remarks</i> .
<code>samp0(<i>#</i>)</code> specifies that the first <i>#</i> be used in that regression.
<code>s0(<i>#</i>₁ <i>#</i>₂)</code> specifies that <i>#</i> ₁ <i>#</i> ₂ be used as initial values.
<code>forecast(<i>#</i>)</code> specifies the number of periods for the out-of-sample prediction; $0 \leq \# \leq 500$. The default is <code>forecast(0)</code> , which is equivalent to not performing an out-of-sample forecast.

Remarks

The double-exponential smoothing procedure is designed for series that can be locally approximated as

$$\hat{x}_t = m_t + b_t t$$

where $\hat{x}_t$ is the smoothed or predicted value of the series x , and the terms m_t and b_t change over time. [Abraham and Ledolter \(1983\)](#), [Bowerman and O’Connell \(1993\)](#), and [Montgomery, Johnson, and Gardiner \(1990\)](#) all provide good introductions to double-exponential smoothing. [Chatfield \(2001, 2004\)](#) provides helpful discussions of how double-exponential smoothing relates to modern time-series methods.

The double-exponential method has been used both as a smoother and as a prediction method. Recall from [\[TS\] tssmooth exponential](#) that the single-exponential smoothed series is given by

$$S_t = \alpha x_t + (1 - \alpha) S_{t-1}$$

where α is the smoothing constant and x_t is the original series. The double-exponential smoother is obtained by smoothing the smoothed series,

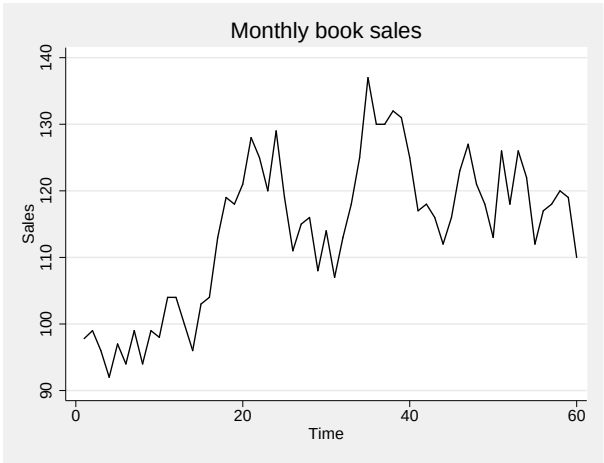
$$S_t^{[2]} = \alpha S_t + (1 - \alpha) S_{t-1}^{[2]}$$

Values of S_0 and $S_0^{[2]}$ are necessary to begin the process. Per [Montgomery, Johnson, and Gardiner \(1990\)](#), the default method is to obtain S_0 and $S_0^{[2]}$ from a regression of the first N_{pre} values of x_t on $\tilde{t} = (1, \dots, N_{\text{pre}} - t_0)'$. By default, N_{pre} is equal to one-half the number of observations in the sample. N_{pre} can be specified using the `samp0()` option.

The values of S_0 and $S_0^{[2]}$ can also be specified using the option `s0()`.

► Example 1

Suppose that we had some data on the monthly sales of a book and that we wanted to smooth this series. The graph below illustrates that this series is locally trending over time, so we would not want to use single-exponential smoothing.



The following example illustrates that double-exponential smoothing is simply smoothing the smoothed series. Because the starting values are treated as time-zero values, we actually lose 2 observations when smoothing the smoothed series.

```
. use http://www.stata-press.com/data/r11/sales2
. tssmooth exponential double sm1=sales, p(.7) s0(1031)
exponential coefficient =      0.7000
sum-of-squared residuals =      13923
root mean squared error =      13.192
. tssmooth exponential double sm2=sm1, p(.7) s0(1031)
exponential coefficient =      0.7000
sum-of-squared residuals =      7698.6
root mean squared error =      9.8098
. tssmooth dexpontential double sm2b=sales, p(.7) s0(1031 1031)
double-exponential coefficient =      0.7000
sum-of-squared residuals =      3724.4
root mean squared error =      6.8231
. generate double sm2c = f2.sm2
(2 missing values generated)
. list sm2b sm2c in 1/10
```

	sm2b	sm2c
1.	1031	1031
2.	1028.3834	1028.3834
3.	1030.6306	1030.6306
4.	1017.8182	1017.8182
5.	1022.938	1022.938
6.	1026.0752	1026.0752
7.	1041.8587	1041.8587
8.	1042.8341	1042.8341
9.	1035.9571	1035.9571
10.	1030.6651	1030.6651

◀

The double-exponential method can also be viewed as a forecasting mechanism. The exponential forecast method is a constrained version of the Holt–Winters method implemented in [TS] **tssmooth hwinters** (as discussed by [Gardner Jr. \[1985\]](#) and [Chatfield \[2001\]](#)). [Chatfield \(2001\)](#) also notes that the double-exponential method arises when the underlying model is an ARIMA(0,2,2) with equal roots.

This method produces predictions $\hat{x}_t$ for $t = t_1, \dots, T + \text{forecast}()$. These predictions are obtained as a function of the smoothed series and the smoothed-smoothed series. For $t \in [t_0, T]$,

$$\hat{x}_t = \left(2 + \frac{\alpha}{1 - \alpha}\right) S_t - \left(1 + \frac{\alpha}{1 - \alpha}\right) S_t^{[2]}$$

where S_t and $S_t^{[2]}$ are as given above.

The out-of-sample predictions are obtained as a function of the constant term, the linear term of the smoothed series at the last observation in the sample, and time. The constant term is $a_T = 2S_T - S_T^{[2]}$, and the linear term is $b_T = \frac{\alpha}{1 - \alpha}(S_T - S_T^{[2]})$. The τ th-step-ahead out-of-sample prediction is given by

$$\hat{x}_t = a_t + \tau b_T$$

➤ Example 2

Specifying the forecast option puts the double-exponential forecast into the new variable instead of the double-exponential smoothed series. The code given below uses the smoothed series `sm1` and `sm2` that were generated above to illustrate how the double-exponential forecasts are computed.

```
. tssmooth dexponential double f1=sales, p(.7) s0(1031 1031) forecast(4)
double-exponential coefficient =      0.7000
sum-of-squared residuals      =      20737
root mean squared error       =      16.1
. generate double xhat = (2 + .7/.3) * sm1 - (1 + .7/.3) * f.sm2
(5 missing values generated)
. list xhat f1 in 1/10
```

	xhat	f1
1.	1031	1031
2.	1031	1031
3.	1023.524	1023.524
4.	1034.8039	1034.8039
5.	994.0237	994.0237
6.	1032.4463	1032.4463
7.	1031.9015	1031.9015
8.	1071.1709	1071.1709
9.	1044.6454	1044.6454
10.	1023.1855	1023.1855



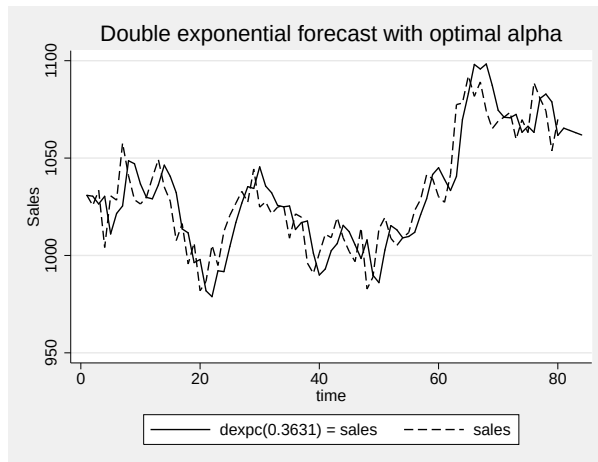
➤ Example 3

Generally, when you are forecasting, you do not know the smoothing parameter. `tssmooth dexponential` computes the double-exponential forecasts of a series and obtains the optimal smoothing parameter by finding the smoothing parameter that minimizes the in-sample sum-of-squared forecast errors.

```
. tssmooth dexponential f2=sales, forecast(4)
computing optimal double-exponential coefficient (0,1)
optimal double-exponential coefficient =      0.3631
sum-of-squared residuals              =     16075.805
root mean squared error                =     14.175598
```

The following graph describes the fit that we obtained by applying the double-exponential forecast method to our sales data. The out-of-sample dynamic predictions are not constant, as in the single-exponential case.

```
. line f2 sales t, title("Double exponential forecast with optimal alpha")
> ytitle(Sales) xtitle(time)
```



◀

`tssmooth dexponential` automatically detects panel data from the information provided when the dataset was `tsset`. The starting values are chosen separately for each series. If the smoothing parameter is chosen to minimize the sum-of-squared prediction errors, the optimization is performed separately on each panel. The saved results contain the results from the last panel. Missing values at the beginning of the sample are excluded from the sample. After at least one value has been found, missing values are filled in using the one-step-ahead predictions from the previous period.

Saved results

`tssmooth` saves the following in `r()`:

Scalars

<code>r(N)</code>	number of observations
<code>r(alpha)</code>	α smoothing parameter
<code>r(rss)</code>	sum-of-squared errors
<code>r(rmse)</code>	root mean squared error
<code>r(N_pre)</code>	number of observations used in calculating starting values, if starting values calculated
<code>r(s2_0)</code>	initial value for linear term, i.e., $S_0^{[2]}$
<code>r(s1_0)</code>	initial value for constant term, i.e., S_0
<code>r(linear)</code>	final value of linear term
<code>r(constant)</code>	final value of constant term
<code>r(period)</code>	period, if filter is seasonal

Macros

<code>r(method)</code>	smoothing method
<code>r(exp)</code>	expression specified
<code>r(timevar)</code>	time variable specified in <code>tsset</code>
<code>r(panelvar)</code>	panel variable specified in <code>tsset</code>

Methods and formulas

`tssmooth dexponential` is implemented as an ado-file.

A truncated description of the specified double-exponential filter is used to label the new variable. See [D] [label](#) for more information on labels.

An untruncated description of the specified double-exponential filter is saved in the characteristic `tssmooth` for the new variable. See [P] [char](#) for more information on characteristics.

The updating equations for the smoothing and forecasting versions are as given previously.

The starting values for both the smoothing and forecasting versions of double-exponential are obtained using the same method, which begins with the model

$$x_t = \beta_0 + \beta_1 t$$

where x_t is the series to be smoothed and t is a time variable that has been normalized to equal 1 in the first period included in the sample. The regression coefficient estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are obtained via OLS. The sample is determined by the option `samp0()`. By default, `samp0()` includes the first half of the observations. Given the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, the starting values are

$$S_0 = \hat{\beta}_0 - \{(1 - \alpha)/\alpha\}\hat{\beta}_1$$

$$S_0^{[2]} = \hat{\beta}_0 - 2\{(1 - \alpha)/\alpha\}\hat{\beta}_1$$

References

- Abraham, B., and J. Ledolter. 1983. *Statistical Methods for Forecasting*. New York: Wiley.
- Bowerman, B. L., and R. T. O'Connell. 1993. *Forecasting and Time Series: An Applied Approach*. 3rd ed. Pacific Grove, CA: Duxbury.
- Chatfield, C. 2001. *Time-Series Forecasting*. London: Chapman & Hall/CRC.
- . 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- Chatfield, C., and M. Yar. 1988. Holt-Winters forecasting: Some practical issues. *Statistician* 37: 129–140.
- Gardner Jr., E. S. 1985. Exponential smoothing: The state of the art. *Journal of Forecasting* 4: 1–28.
- Holt, C. C. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20: 5–10.
- Montgomery, D. C., L. A. Johnson, and J. S. Gardiner. 1990. *Forecasting and Time Series Analysis*. 2nd ed. New York: McGraw-Hill.
- Winters, P. R. 1960. Forecasting sales by exponentially weighted moving averages. *Management Science* 6: 324–342.

Also see

[TS] [tsset](#) — Declare data to be time-series data

[TS] [tssmooth](#) — Smooth and forecast univariate time-series data

Syntax

tssmooth exponential [*type*] *newvar* = *exp* [*if*] [*in*] [, *options*]

<i>options</i>	description
Main	
<code>replace</code>	replace <i>newvar</i> if it already exists
<code>parms($\#_{\alpha}$)</code>	use $\#_{\alpha}$ as smoothing parameter
<code>samp0($\#$)</code>	use $\#$ observations to obtain initial value for recursion
<code>s0($\#$)</code>	use $\#$ as initial value for recursion
<code>forecast($\#$)</code>	use $\#$ periods for the out-of-sample forecast

You must `tsset` your data before using `tssmooth`; see [TS] [tsset](#).
exp may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Smoothers/univariate forecasters > Single-exponential smoothing

Description

`tssmooth exponential` models the trend of a variable whose change from the previous value is serially correlated. More precisely, it models a variable whose first difference follows a low-order, moving-average process.

Options

Main

- `replace` replaces *newvar* if it already exists.
- `parms( $\#_{\alpha}$ )` specifies the parameter α for the exponential smoother; $0 < \#_{\alpha} < 1$. If `parms( $\#_{\alpha}$ )` is not specified, the smoothing parameter is chosen to minimize the in-sample sum-of-squared forecast errors.
- `samp0( $\#$ )` and `s0( $\#$ )` are mutually exclusive ways of specifying the initial value for the recursion.
- `samp0( $\#$ )` specifies that the initial value be obtained by calculating the mean over the first $\#$ observations of the sample.
- `s0( $\#$ )` specifies the initial value to be used.
- If neither option is specified, the default is to use the mean calculated over the first half of the sample.
- `forecast( $\#$ )` gives the number of observations for the out-of-sample prediction; $0 \leq \# \leq 500$. The default value is `forecast(0)` and is equivalent to not forecasting out of sample.

Remarks

[Introduction](#)

[Examples](#)

[Treatment of missing values](#)

Introduction

Exponential smoothing can be viewed either as an adaptive-forecasting algorithm or, equivalently, as a geometrically weighted moving-average filter. Exponential smoothing is most appropriate when used with time-series data that exhibit no linear or higher-order trends but that do exhibit low-velocity, aperiodic variation in the mean. [Abraham and Ledolter \(1983\)](#), [Bowerman and O’Connell \(1993\)](#), and [Montgomery, Johnson, and Gardiner \(1990\)](#) all provide good introductions to single-exponential smoothing. [Chatfield \(2001, 2004\)](#) discusses how single-exponential smoothing relates to modern time-series methods. For example, simple exponential smoothing produces optimal forecasts for several underlying models, including ARIMA(0,1,1) and the random-walk-plus-noise state-space model. (See [Chatfield \[2001, sec. 4.3.1\]](#).)

The exponential filter with smoothing parameter α creates the series S_t , where

$$S_t = \alpha X_t + (1 - \alpha)S_{t-1} \quad \text{for } t = 1, \dots, T$$

and S_0 is the initial value. This is the adaptive forecast-updating form of the exponential smoother. This implies that

$$S_t = \alpha \sum_{k=0}^{T-1} (1 - \alpha)^k X_{T-k} + (1 - \alpha)^T S_0$$

which is the weighted moving-average representation, with geometrically declining weights. The choice of the smoothing constant α determines how quickly the smoothed series or forecast will adjust to changes in the mean of the unfiltered series. For small values of α , the response will be slow because more weight is placed on the previous estimate of the mean of the unfiltered series, whereas larger values of α will put more emphasis on the most recently observed value of the unfiltered series.

Examples

► Example 1

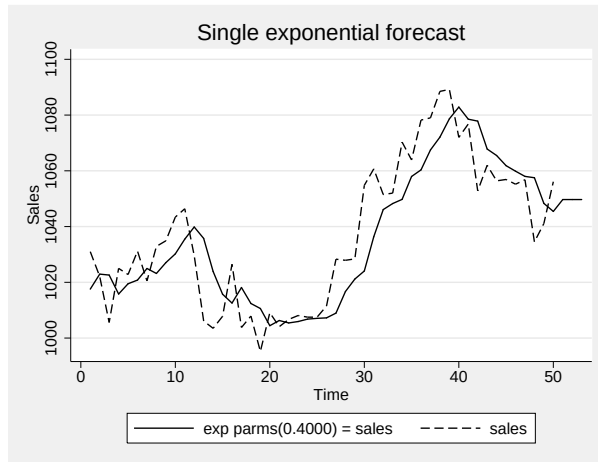
Let’s consider some examples using sales data. Here we forecast sales for three periods with a smoothing parameter of 0.4:

```
. use http://www.stata-press.com/data/r11/sales1
. tssmooth exponential sm1=sales, parms(.4) forecast(3)

exponential coefficient =      0.4000
sum-of-squared residuals =      8345
root mean squared error =     12.919
```

To compare our forecast with the actual data, we graph the series and the forecasted series over time.

```
. line sm1 sales t, title("Single exponential forecast") ytitle(Sales) xtitle(Time)
```

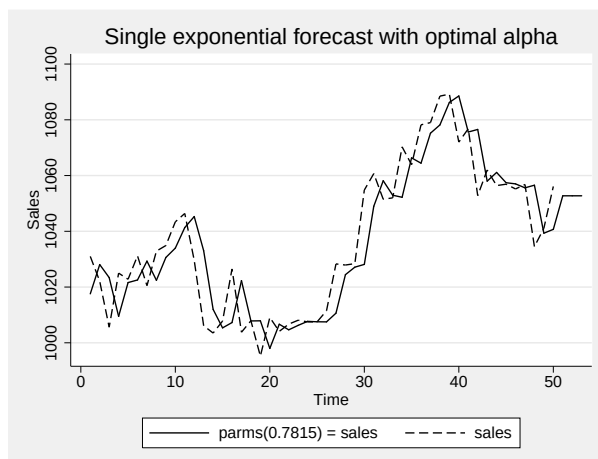


The graph indicates that our forecasted series may not be adjusting rapidly enough to the changes in the actual series. The smoothing parameter α controls the rate at which the forecast adjusts. Smaller values of α adjust the forecasts more slowly. Thus we suspect that our chosen value of 0.4 is too small. One way to investigate this suspicion is to ask `tssmooth exponential` to choose the smoothing parameter that minimizes the sum-of-squared forecast errors.

```
. tssmooth exponential sm2=sales, forecast(3)
computing optimal exponential coefficient (0,1)
optimal exponential coefficient =      0.7815
sum-of-squared residuals      =    6727.7056
root mean squared error      =    11.599746
```

The output suggests that the value of $\alpha = 0.4$ is too small. The graph below indicates that the new forecast tracks the series much more closely than the previous forecast.

```
. line sm2 sales t, title ("Single exponential forecast with optimal alpha")
> ytitle(sales) xtitle(Time)
```



We noted above that simple exponential forecasts are optimal for an ARIMA (0,1,1) model. (See [TS] [arima](#) for fitting ARIMA models in Stata.) [Chatfield \(2001, 90\)](#) gives the following useful derivation that relates the MA coefficient in an ARIMA (0,1,1) model to the smoothing parameter in single-exponential smoothing. An ARIMA (0,1,1) is given by

$$x_t - x_{t-1} = \epsilon_t + \theta \epsilon_{t-1}$$

where ϵ_t is an identically and independently distributed white-noise error term. Thus given $\hat{\theta}$, an estimate of θ , an optimal one-step prediction of $\hat{x}_{t+1}$ is $\hat{x}_{t+1} = x_t + \hat{\theta}\epsilon_t$. Because ϵ_t is not observable, it can be replaced by

$$\hat{\epsilon}_t = x_t - \hat{x}_{t-1}$$

yielding

$$\hat{x}_{t+1} = x_t + \hat{\theta}(x_t - \hat{x}_{t-1})$$

Letting $\hat{\alpha} = 1 + \hat{\theta}$ and doing more rearranging implies that

$$\begin{aligned}\hat{x}_{t+1} &= (1 + \hat{\theta})x_t - \hat{\theta}\hat{x}_{t-1} \\ \hat{x}_{t+1} &= \hat{\alpha}x_t - (1 - \hat{\alpha})\hat{x}_{t-1}\end{aligned}$$

► Example 2

Let’s compare the estimate of the optimal smoothing parameter of 0.7815 with the one we could obtain using [TS] [arima](#). Below we fit an ARIMA(0,1,1) to the sales data and then remove the estimate of α . The two estimates of α are quite close, given the large estimated standard error of $\hat{\theta}$.

```
. arima sales, arima(0,1,1)
(setting optimization to BHHH)
Iteration 0:  log likelihood = -189.91037
Iteration 1:  log likelihood = -189.62405
Iteration 2:  log likelihood = -189.60468
Iteration 3:  log likelihood = -189.60352
Iteration 4:  log likelihood = -189.60343
(switching optimization to BFGS)
Iteration 5:  log likelihood = -189.60342
ARIMA regression
Sample: 2 - 50                                Number of obs   =          49
                                                Wald chi2(1)    =           1.41
Log likelihood = -189.6034                     Prob > chi2     =          0.2347
```

D.sales	OPG					[95% Conf. Interval]
	Coef.	Std. Err.	z	P> z		
sales						
_cons	.5025469	1.382727	0.36	0.716	-2.207548	3.212641
ARMA						
ma						
L1.	-.1986561	.1671699	-1.19	0.235	-.5263031	.1289908
/sigma	11.58992	1.240607	9.34	0.000	9.158378	14.02147

```
. di 1 + _b[ARMA:L.ma]
.80134387
```


► Example 3

`tssmooth exponential` automatically detects panel data. Suppose that we had sales figures for five companies in long form. Running `tssmooth exponential` on the variable that contains all five series puts the smoothed series and the predictions in one variable in long form. When the smoothing parameter is chosen to minimize the squared prediction error, an optimal value for the smoothing parameter is chosen separately for each panel.

```
. use http://www.stata-press.com/data/r11/sales_cert, clear
. tsset
    panel variable:  id (strongly balanced)
    time variable:  t, 1 to 100
                delta:  1 unit
. tssmooth exponential sm5=sales, forecast(3)
```

```
-> id = 1
computing optimal exponential coefficient (0,1)
optimal exponential coefficient =      0.8702
sum-of-squared residuals      =    16070.567
root mean squared error      =    12.676974
```

```
-> id = 2
computing optimal exponential coefficient (0,1)
optimal exponential coefficient =      0.7003
sum-of-squared residuals      =    20792.393
root mean squared error      =    14.419568
```

```
-> id = 3
computing optimal exponential coefficient (0,1)
optimal exponential coefficient =      0.6927
sum-of-squared residuals      =      21629
root mean squared error      =    14.706801
```

```
-> id = 4
computing optimal exponential coefficient (0,1)
optimal exponential coefficient =      0.3866
sum-of-squared residuals      =    22321.334
root mean squared error      =    14.940326
```

```
-> id = 5
computing optimal exponential coefficient (0,1)
optimal exponential coefficient =      0.4540
sum-of-squared residuals      =    20714.095
root mean squared error      =    14.392392
```

`tssmooth exponential` computed starting values and chose an optimal α for each panel individually.

Treatment of missing values

Missing values in the middle of the data are filled in with the one-step-ahead prediction using the previous values. Missing values at the beginning or end of the data are treated as if the observations were not there.

`tssmooth exponential` treats observations excluded from the sample by `if` and `in` just as if they were missing.

➤ Example 4

Here the 28th observation is missing. The prediction for the 29th observation is repeated in the new series.

```
. use http://www.stata-press.com/data/r11/sales1, clear
. tssmooth exponential sm1=sales, parms(.7) forecast(3)
  (output omitted)
. generate sales2=sales if t!=28
(4 missing values generated)
. tssmooth exponential sm3=sales2, parms(.7) forecast(3)
exponential coefficient =      0.7000
sum-of-squared residuals =    6842.4
root mean squared error =     11.817
. list t sales2 sm3 if t>25 & t<31
```

	t	sales2	sm3
26.	26	1011.5	1007.5
27.	27	1028.3	1010.3
28.	28	.	1022.9
29.	29	1028.4	1022.9
30.	30	1054.8	1026.75

Because the data for $t = 28$ are missing, the prediction for period 28 has been used in its place. This implies that the updating equation for period 29 is

$$S_{29} = \alpha S_{28} + (1 - \alpha)S_{28} = S_{28}$$

which explains why the prediction for $t = 28$ is repeated.

Because this is a single-exponential procedure, the loss of that one observation will not be noticed several periods later.

```
. generate diff = sm3-sm1 if t>28
(28 missing values generated)
. list t diff if t>28 & t<39
```

	t	diff
29.	29	-3.5
30.	30	-1.050049
31.	31	-.3150635
32.	32	-.0946045
33.	33	-.0283203
34.	34	-.0085449
35.	35	-.0025635
36.	36	-.0008545
37.	37	-.0003662
38.	38	-.0001221

◀

► Example 5

Now consider an example in which there are data missing at the beginning and end of the sample.

```
. generate sales3=sales if t>2 & t<49
(7 missing values generated)
. tssmooth exponential sm4=sales3, parms(.7) forecast(3)
exponential coefficient =      0.7000
sum-of-squared residuals =    6215.3
root mean squared error =    11.624
. list t sales sales3 sm4 if t<5 | t>45
```

	t	sales	sales3	sm4
1.	1	1031	.	.
2.	2	1022.1	.	.
3.	3	1005.6	1005.6	1016.787
4.	4	1025	1025	1008.956
46.	46	1055.2	1055.2	1057.2
47.	47	1056.8	1056.8	1055.8
48.	48	1034.5	1034.5	1056.5
49.	49	1041.1	.	1041.1
50.	50	1056.1	.	1041.1
51.	51	.	.	1041.1
52.	52	.	.	1041.1
53.	53	.	.	1041.1

The output above illustrates that missing values at the beginning or end of the sample cause the sample to be truncated. The new series begins with nonmissing data and begins predicting immediately after it stops.

One period after the actual data concludes, the exponential forecast becomes a constant. After the actual end of the data, the forecast at period t is substituted for the missing data. This also illustrates why the forecasted series is a constant.

◀

Saved results

`tssmooth` saves the following in `r()`:

Scalars

<code>r(N)</code>	number of observations
<code>r(alpha)</code>	α smoothing parameter
<code>r(rss)</code>	sum-of-squared prediction errors
<code>r(rmse)</code>	root mean squared error
<code>r(N_pre)</code>	number of observations used in calculating starting values
<code>r(s1_0)</code>	initial value for S_t

Macros

<code>r(method)</code>	smoothing method
<code>r(exp)</code>	expression specified
<code>r(timevar)</code>	time variable specified in <code>tsset</code>
<code>r(panelvar)</code>	panel variable specified in <code>tsset</code>

Methods and formulas

`tssmooth exponential` is implemented as an ado-file.

The formulas for deriving smoothed series are as given in the text. When the value of α is not specified, an optimal value is found that minimizes the mean squared forecast error. A method of bisection is used to find the solution to this optimization problem.

A truncated description of the specified exponential filter is used to label the new variable. See [\[D\] label](#) for more information about labels.

An untruncated description of the specified exponential filter is saved in the characteristic `tssmooth` for the new variable. See [\[P\] char](#) for more information about characteristics.

References

- Abraham, B., and J. Ledolter. 1983. *Statistical Methods for Forecasting*. New York: Wiley.
- Bowerman, B. L., and R. T. O'Connell. 1993. *Forecasting and Time Series: An Applied Approach*. 3rd ed. Pacific Grove, CA: Duxbury.
- Chatfield, C. 2001. *Time-Series Forecasting*. London: Chapman & Hall/CRC.
- . 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- Chatfield, C., and M. Yar. 1988. Holt-Winters forecasting: Some practical issues. *Statistician* 37: 129–140.
- Holt, C. C. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20: 5–10.
- Montgomery, D. C., L. A. Johnson, and J. S. Gardiner. 1990. *Forecasting and Time Series Analysis*. 2nd ed. New York: McGraw-Hill.
- Winters, P. R. 1960. Forecasting sales by exponentially weighted moving averages. *Management Science* 6: 324–342.

Also see

[\[TS\] tsset](#) — Declare data to be time-series data

[\[TS\] tssmooth](#) — Smooth and forecast univariate time-series data

Syntax

```
tssmooth hwinters [type] newvar = exp [if] [in] [, options]
```

options	description
Main	
replace	replace <i>newvar</i> if it already exists
parms($\#_{\alpha}$ $\#_{\beta}$)	use $\#_{\alpha}$ and $\#_{\beta}$ as smoothing parameters
samp0($\#$)	use $\#$ observations to obtain initial values for recursion
s0($\#_{\text{cons}}$ $\#_{\text{lt}}$)	use $\#_{\text{cons}}$ and $\#_{\text{lt}}$ as initial values for recursion
forecast($\#$)	use $\#$ periods for the out-of-sample forecast
Options	
diff	alternative initial-value specification; see <i>Options</i>
Maximization	
maximize_options	control the maximization process; seldom used
from($\#_{\alpha}$ $\#_{\beta}$)	use $\#_{\alpha}$ and $\#_{\beta}$ as starting values for the parameters

You must `tsset` your data before using `tssmooth hwinters`; see [TS] [tsset](#).
exp may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Smoothers/univariate forecasters > Holt-Winters nonseasonal smoothing

Description

`tssmooth hwinters` is used in smoothing or forecasting a series that can be modeled as a linear trend in which the intercept and the coefficient on time vary over time.

Options

Main

`replace` replaces *newvar* if it already exists.

`parms( $\#_{\alpha}$   $\#_{\beta}$ )`, $0 \leq \#_{\alpha} \leq 1$ and $0 \leq \#_{\beta} \leq 1$, specifies the parameters. If `parms()` is not specified, the values are chosen by an iterative process to minimize the in-sample sum-of-squared prediction errors.

If you experience difficulty converging (many iterations and “not concave” messages), try using `from()` to provide better starting values.

`samp0( $\#$ )` and `s0( $\#_{\text{cons}}$   $\#_{\text{lt}}$ )` specify how the initial values $\#_{\text{cons}}$ and $\#_{\text{lt}}$ for the recursion are obtained.

By default, initial values are obtained by fitting a linear regression with a time trend using the first half of the observations in the dataset.

`samp0(#)` specifies that the first `#` observations be used in that regression.

`s0(#cons #lt)` specifies that `#cons` and `#lt` be used as initial values.

`forecast(#)` specifies the number of periods for the out-of-sample prediction; $0 \leq \# \leq 500$. The default is `forecast(0)`, which is equivalent to not performing an out-of-sample forecast.

Options

`diff` specifies that the linear term is obtained by averaging the first difference of exp_t and the intercept is obtained as the difference of exp in the first observation and the mean of $D.exp_t$.

If the `diff` option is not specified, a linear regression of exp_t on a constant and t is fit.

Maximization

`maximize_options` controls the process for solving for the optimal α and β when `parms()` is not specified.

`maximize_options`: `nodifficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, `nonrtolerance`, see [R] [maximize](#). These options are seldom used.

`from(# $\alpha$  # $\beta$ )`, $0 < \#_{\alpha} < 1$ and $0 < \#_{\beta} < 1$, specifies starting values from which the optimal values of α and β will be obtained. If `from()` is not specified, `from(.5 .5)` is used.

Remarks

The Holt–Winters method forecasts series of the form

$$\hat{x}_{t+1} = a_t + b_t t$$

where $\hat{x}_t$ is the forecast of the original series x_t , a_t is a mean that drifts over time, and b_t is a coefficient on time that also drifts. In fact, as [Gardner Jr. \(1985\)](#) has noted, the Holt–Winters method produces optimal forecasts for an ARIMA(0,2,2) model and some local linear models. See [\[TS\] arima](#) and the references in that entry for ARIMA models, and see [Harvey \(1989\)](#) for a discussion of the local linear model and its relationship to the Holt–Winters method. [Abraham and Ledolter \(1983\)](#), [Bowerman and O’Connell \(1993\)](#), and [Montgomery, Johnson, and Gardiner \(1990\)](#) all provide good introductions to the Holt–Winters method. [Chatfield \(2001, 2004\)](#) provides helpful discussions of how this method relates to modern time-series analysis.

The Holt–Winters method can be viewed as an extension of double-exponential smoothing with two parameters, which may be explicitly set or chosen to minimize the in-sample sum-of-squared forecast errors. In the latter case, as discussed in [Methods and formulas](#), the smoothing parameters are chosen to minimize the in-sample sum-of-squared forecast errors plus a penalty term that helps to achieve convergence when one of the parameters is too close to the boundary.

Given the series x_t , the smoothing parameters α and β , and the starting values a_0 and b_0 , the updating equations are

$$a_t = \alpha x_t + (1 - \alpha)(a_{t-1} + b_{t-1})$$

$$b_t = \beta(a_t - a_{t-1}) + (1 - \beta)b_{t-1}$$

After computing the series of constant and linear terms, a_t and b_t , respectively, the τ -step-ahead prediction of x_t is given by

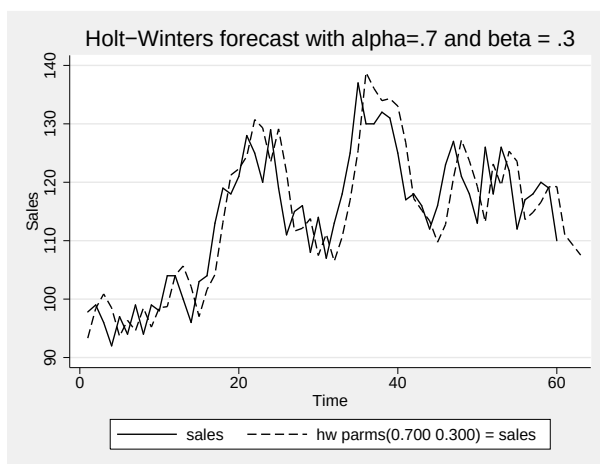
$$\hat{x}_{t+\tau} = a_t + b_t\tau$$

► Example 1

Below we show how to use `tssmooth hwinters` with specified smoothing parameters. This example also shows that the Holt–Winters method can closely follow a series in which both the mean and the time coefficient drift over time.

Suppose that we have data on the monthly sales of a book and that we want to forecast this series with the Holt–Winters method.

```
. use http://www.stata-press.com/data/r11/bsales
. tssmooth hwinters hw1=sales, parms(.7 .3) forecast(3)
Specified weights:
      alpha = 0.7000
      beta  = 0.3000
sum-of-squared residuals = 2301.046
root mean squared error = 6.192799
. line sales hw1 t, title("Holt-Winters Forecast with alpha=.7 and beta=.3")
> ytitle(Sales) xtitle(Time)
```



The graph indicates that the forecasts are for linearly decreasing sales. Given a_T and b_T , the out-of-sample predictions are linear functions of time. In this example, the slope appears to be too steep, probably because our choice of α and β .

◀

► Example 2

The graph in the previous example illustrates that the starting values for the linear and constant series can affect the in-sample fit of the predicted series for the first few observations. The previous example used the default method for obtaining the initial values for the recursion. The output below illustrates that, for some problems, the differenced-based initial values provide a better in-sample fit

for the first few observations. However, the differenced-based initial values do not always outperform the regression-based initial values. Furthermore, as shown in the output below, for series of reasonable length, the predictions produced are nearly identical.

```
. tssmooth hwinters hw2=sales, parms(.7 .3) forecast(3) diff
Specified weights:
              alpha = 0.7000
              beta  = 0.3000
sum-of-squared residuals = 2261.173
  root mean squared error = 6.13891
. list hw1 hw2 if _n<6 | _n>57
```

	hw1	hw2
1.	93.31973	97.80807
2.	98.40002	98.11447
3.	100.8845	99.2267
4.	98.50404	96.78276
5.	93.62408	92.2452
58.	116.5771	116.5771
59.	119.2146	119.2146
60.	119.2608	119.2608
61.	111.0299	111.0299
62.	109.2815	109.2815
63.	107.5331	107.5331

When the smoothing parameters are chosen to minimize the in-sample sum-of-squared forecast errors, changing the initial values can affect the choice of the optimal α and β . When changing the initial values results in different optimal values for α and β , the predictions will also differ.



When the Holt–Winters model fits the data well, finding the optimal smoothing parameters generally proceeds well. When the model fits poorly, finding the α and β that minimize the in-sample sum-of-squared forecast errors can be difficult.

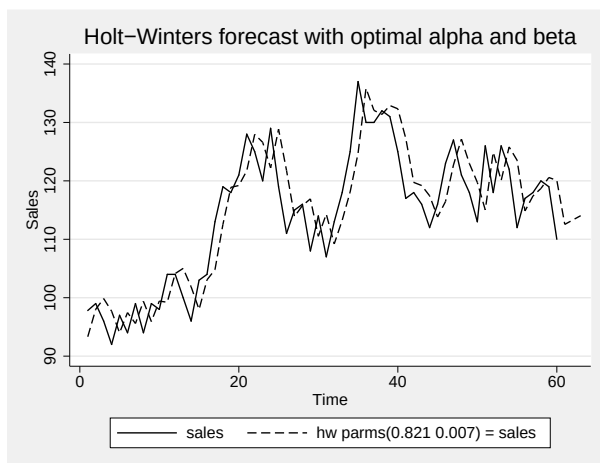
➤ Example 3

In this example, we forecast the book sales data using the α and β that minimize the in-sample squared forecast errors.

```
. tssmooth hwinters hw3=sales, forecast(3)
computing optimal weights
Iteration 0:   penalized RSS = -2632.2073   (not concave)
Iteration 1:   penalized RSS = -1982.8431
Iteration 2:   penalized RSS = -1976.4236
Iteration 3:   penalized RSS = -1975.9175
Iteration 4:   penalized RSS = -1975.9036
Iteration 5:   penalized RSS = -1975.9036
Optimal weights:
              alpha = 0.8209
              beta  = 0.0067
penalized sum of squared residuals = 1975.904
      sum of squared residuals = 1975.904
      root mean-squared error = 5.738617
```


The following graph contains the data and the forecast using the optimal α and β . Comparing this graph with the one above illustrates how different choices of α and β can lead to very different forecasts. Instead of linearly decreasing sales, the new forecast is for linearly increasing sales.

```
. line sales hw3 t, title("Holt-Winters Forecast with optimal alpha and beta")
> ytitle(Sales) xtitle(Time)
```



◀

Saved results

tssmooth saves the following in `r()`:

Scalars

<code>r(N)</code>	number of observations	<code>r(N_pre)</code>	number of observations used in calculating starting values
<code>r(alpha)</code>	α smoothing parameter	<code>r(s2_0)</code>	initial value for linear term
<code>r(beta)</code>	β smoothing parameter	<code>r(s1_0)</code>	initial value for constant term
<code>r(rss)</code>	sum-of-squared errors	<code>r(linear)</code>	final value of linear term
<code>r(prss)</code>	penalized sum-of-squared errors, if <code>parms()</code> not specified	<code>r(constant)</code>	final value of constant term
<code>r(rmse)</code>	root mean squared error		

Macros

<code>r(method)</code>	smoothing method	<code>r(timevar)</code>	time variables specified in <code>tsset</code>
<code>r(exp)</code>	expression specified	<code>r(panelvar)</code>	panel variables specified in <code>tsset</code>

Methods and formulas

tssmooth hwinters is implemented as an ado-file.

A truncated description of the specified Holt–Winters filter is used to label the new variable. See [\[D\] label](#) for more information on labels.

An untruncated description of the specified Holt–Winters filter is saved in the characteristic named tssmooth for the new variable. See [\[P\] char](#) for more information on characteristics.

Given the series, x_t ; the smoothing parameters, α and β ; and the starting values, a_0 and b_0 , the updating equations are

$$a_t = \alpha x_t + (1 - \alpha)(a_{t-1} + b_{t-1})$$

$$b_t = \beta(a_t - a_{t-1}) + (1 - \beta)b_{t-1}$$

By default, the initial values are found by fitting a linear regression with a time trend. The time variable in this regression is normalized to equal one in the first period included in the sample. By default, one-half of the data is used in this regression, but this sample can be changed using `samp0()`. a_0 is then set to the estimate of the constant, and b_0 is set to the estimate of the coefficient on the time trend. Specifying the `diff` option sets b_0 to the mean of $D.x$ and a_0 to $x_1 - b_0$. `s0()` can also be used to specify the initial values directly.

Sometimes, one or both of the optimal parameters may lie on the boundary of $[0, 1]$. To keep the estimates inside $[0, 1]$, `tssmooth hwinters` parameterizes the objective function in terms of their inverse logits, i.e., in terms of $\exp(\alpha)/\{1 + \exp(\alpha)\}$ and $\exp(\beta)/\{1 + \exp(\beta)\}$. When one of these parameters is actually on the boundary, this can complicate the optimization. For this reason, `tssmooth hwinters` optimizes a penalized sum-of-squared forecast errors. Let $\hat{x}_t(\tilde{\alpha}, \tilde{\beta})$ be the forecast for the series x_t , given the choices of $\tilde{\alpha}$ and $\tilde{\beta}$. Then the in-sample penalized sum-of-squared prediction errors is

$$P = \sum_{t=1}^T \left[\{x_t - \hat{x}_t(\tilde{\alpha}, \tilde{\beta})\}^2 + I_{|f(\tilde{\alpha})| > 12} (|f(\tilde{\alpha})| - 12)^2 + I_{|f(\tilde{\beta})| > 12} (|f(\tilde{\beta})| - 12)^2 \right]$$

where $f(x) = \ln\{x(1-x)\}$. The penalty term is zero unless one of the parameters is close to the boundary. When one of the parameters is close to the boundary, the penalty term will help to obtain convergence.

Acknowledgment

We thank Nicholas J. Cox of Durham University for his helpful comments.

References

- Abraham, B., and J. Ledolter. 1983. *Statistical Methods for Forecasting*. New York: Wiley.
- Bowerman, B. L., and R. T. O'Connell. 1993. *Forecasting and Time Series: An Applied Approach*. 3rd ed. Pacific Grove, CA: Duxbury.
- Chatfield, C. 2001. *Time-Series Forecasting*. London: Chapman & Hall/CRC.
- . 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- Chatfield, C., and M. Yar. 1988. Holt-Winters forecasting: Some practical issues. *Statistician* 37: 129–140.
- Gardner Jr., E. S. 1985. Exponential smoothing: The state of the art. *Journal of Forecasting* 4: 1–28.
- Holt, C. C. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20: 5–10.
- Montgomery, D. C., L. A. Johnson, and J. S. Gardiner. 1990. *Forecasting and Time Series Analysis*. 2nd ed. New York: McGraw–Hill.
- Winters, P. R. 1960. Forecasting sales by exponentially weighted moving averages. *Management Science* 6: 324–342.

Also see

[TS] [tsset](#) — Declare data to be time-series data

[TS] [tssmooth](#) — Smooth and forecast univariate time-series data

Syntax

Moving average with uniform weights

```
tssmooth ma [type] newvar = exp [if] [in] , window(#l [#c [#f]]) [replace]
```

Moving average with specified weights

```
tssmooth ma [type] newvar = exp [if] [in] , weights([numlistl] <#c> [numlistf])  
[replace]
```

You must `tsset` your data before using `tssmooth`; see [TS] [tsset](#).

exp may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Smoothers/univariate forecasters > Moving-average filter

Description

`tssmooth ma` creates a new series in which each observation is an average of nearby observations in the original series.

In the first syntax, `window()` is required and specifies the span of the filter. `tssmooth ma` constructs a uniformly weighted moving average of the expression.

In the second syntax, `weights()` is required and specifies the weights to be used. `tssmooth ma` then applies the specified weights to construct a weighted moving average of the expression.

Options

`window(#l [#c [#f]])` describes the span of the uniformly weighted moving average.

*#*_{*l*} specifies the number of lagged terms to be included, $0 \leq \#_l \leq$ one-half the number of observations in the sample.

*#*_{*c*} is optional and specifies whether to include the current observation in the filter. A 0 indicates exclusion and 1, inclusion. The current observation is excluded by default.

*#*_{*f*} is optional and specifies the number of forward terms to be included, $0 \leq \#_f \leq$ one-half the number of observations in the sample.

`weights([numlistl] <#c> [numlistf])` is required for the weighted moving average and describes the span of the moving average, as well as the weights to be applied to each term in the average. The middle term literally is surrounded by < and >, so you might type `weights(1/2 <3> 2/1)`.

$numlist_l$ is optional and specifies the weights to be applied to the lagged terms when computing the moving average.

$\#_c$ is required and specifies the weight to be applied to the current term.

$numlist_f$ is optional and specifies the weights to be applied to the forward terms when computing the moving average.

The number of elements in each $numlist$ is limited to one-half the number of observations in the sample.

`replace` replaces `newvar` if it already exists.

Remarks

Moving averages are simple linear filters of the form

$$\hat{x}_t = \frac{\sum_{i=-l}^f w_i x_{t+i}}{\sum_{i=-l}^f w_i}$$

where

$\hat{x}_t$ is the moving average

x_t is the variable or expression to be smoothed

w_i are the weights being applied to the terms in the filter

l is the longest lag in the span of the filter

f is the longest lead in the span of the filter

Moving averages are used primarily to reduce noise in time-series data. Using moving averages to isolate signals is problematic, however, because the moving averages themselves are serially correlated, even when the underlying data series is not. Still, [Chatfield \(2004\)](#) discusses moving-average filters and provides several specific moving-average filters for extracting certain trends.

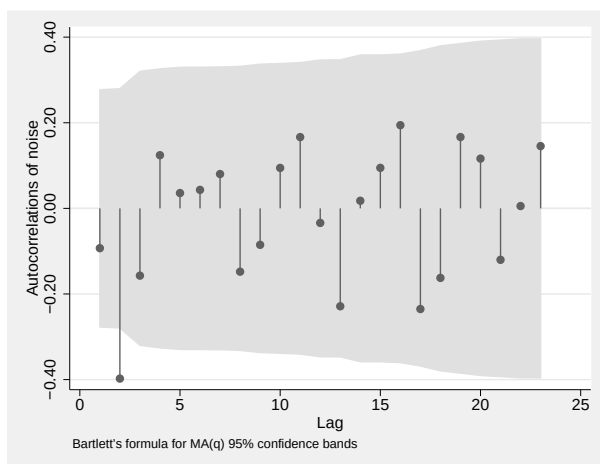
► Example 1

Suppose that we have a time series of sales data, and we want to separate the data into two components: signal and noise. To eliminate the noise, we apply a moving-average filter. In this example, we use a symmetric moving average with a span of 5. This means that we will average the first two lagged values, the current value, and the first two forward terms of the series, with each term in the average receiving a weight of 1.

```
. use http://www.stata-press.com/data/r11/sales1
. tsset
      time variable:  t, 1 to 50
      delta: 1 unit
. tssmooth ma sm1 = sales, window(2 1 2)
The smoother applied was
      (1/5)*[x(t-2) + x(t-1) + 1*x(t) + x(t+1) + x(t+2)]; x(t)= sales
```

We would like to smooth our series so that there is no autocorrelation in the noise. Below we compute the noise as the difference between the smoothed series and the series itself. Then we use `ac` (see [\[TS\] corrgram](#)) to check for autocorrelation in the noise.

```
. generate noise = sales-sm1
. ac noise
```



◀

► Example 2

In the previous [example](#), there is some evidence of negative second-order autocorrelation, possibly due to the uniform weighting or the length of the filter. We are going to specify a shorter filter in which the weights decline as the observations get farther away from the current observation.

The weighted moving-average filter requires that we supply the weights to apply to each element with the `weights()` option. In specifying the weights, we implicitly specify the span of the filter.

Below we use the filter

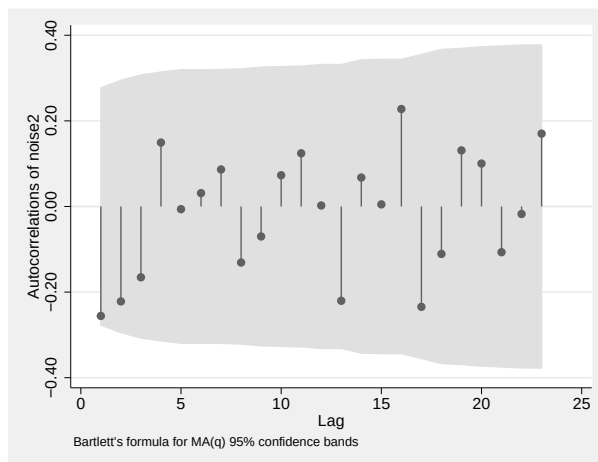
$$\hat{x}_t = (1/9)(1x_{t-2} + 2x_{t-1} + 3x_t + 2x_{t+1} + 1x_{t+2})$$

In what follows, 1/2 does not mean one-half, it means the numlist 1 2:

```
. tssmooth ma sm2 = sales, weights( 1/2 <3> 2/1)
The smoother applied was
      (1/9)*[1*x(t-2) + 2*x(t-1) + 3*x(t) + 2*x(t+1) + 1*x(t+2)]; x(t)= sales
. generate noise2 = sales-sm2
```

We compute the noise and use `ac` to check for autocorrelation.

```
. ac noise2
```



The graph shows no significant evidence of autocorrelation in the noise from the second filter.



□ Technical note

`tssmooth ma` gives any missing observations a coefficient of zero in both the uniformly weighted and weighted moving-average filters. This simply means that missing values or missing periods are excluded from the moving average.

Sample restrictions, via `if` and `in`, cause the expression smoothed by `tssmooth ma` to be missing for the excluded observations. Thus sample restrictions have the same effect as missing values in a variable that is filtered in the expression. Also, gaps in the data that are longer than the span of the filter will generate missing values in the filtered series.

Because the first l observations and the last f observations will be outside the span of the filter, those observations will be set to missing in the moving-average series.



Saved results

`tssmooth` saves the following in `r()`:

Scalars

<code>r(N)</code>	number of observations
<code>r(w0)</code>	weight on the current observation
<code>r(wlead#)</code>	weight on lead #, if leads are specified
<code>r(wlag#)</code>	weight on lag #, if lags are specified

Macros

<code>r(method)</code>	smoothing method
<code>r(exp)</code>	expression specified
<code>r(timevar)</code>	time variable specified in <code>tsset</code>
<code>r(panelvar)</code>	panel variable specified in <code>tsset</code>

Methods and formulas

`tssmooth ma` is implemented as an ado-file. The formula for moving averages is the same as previously given.

A truncated description of the specified moving-average filter labels the new variable. See [D] [label](#) for more information on labels.

An untruncated description of the specified moving-average filter is saved in the characteristic `tssmooth` for the new variable. See [P] [char](#) for more information on characteristics.

Reference

Chatfield, C. 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.

Also see

[TS] [tsset](#) — Declare data to be time-series data

[TS] [tssmooth](#) — Smooth and forecast univariate time-series data

Syntax

```
tssmooth nl [type] newvar = exp [if] [in] , smoother(smoother[, twice])  
[replace]
```

where *smoother* is specified as *Sm*[*Sm*[...]] and *Sm* is one of

```
{ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 } [R]  
3 [R] S [S | R] [S | R] ...  
E  
H
```

The numbers specified in *smoother* represent the span of a running median smoother. For example, a number 3 specifies that each value be replaced by the median of the point and the two adjacent data values. The letter H indicates that a Hanning linear smoother, which is a span-3 smoother with binomial weights, be applied.

The letters E, S, and R are three refinements that can be combined with the running median and Hanning smoothers. First, the end points of a smooth can be given special treatment. This is specified by the E operator. Second, smoothing by 3, the span-3 running median, tends to produce flat-topped hills and valleys. The splitting operator, S, “splits” these repeated values, applies the end-point operator to them, and then “rejoins” the series. Third, it is sometimes useful to repeat an odd-span median smoother or the splitting operator until the smooth no longer changes. Following a digit or an S with an R specifies this type of repetition.

Finally, the *twice* operator specifies that after smoothing, the smoother be reapplied to the resulting rough, and any recovered signal be added back to the original smooth.

Letters may be specified in lowercase, if preferred. Examples of *smoother*[, *twice*] include

3RSSH	3RSSH,twice	4253H	4253H,twice	43RSR2H,twice
3rssh	3rssh,twice	4253h	4253h,twice	43rsr2h,twice

You must *tsset* your data before using *tssmooth nl*; see [TS] *tsset*.
exp may contain time-series operators; see [U] 11.4.4 Time-series *varlists*.

Menu

Statistics > Time series > Smoothers/univariate forecasters > Nonlinear filter

Description

`tssmooth nl` uses nonlinear smoothers to identify the underlying trend in a series.

Options

Main

`smoother(smoother [, twice])` specifies the nonlinear smoother to be used.
`replace` replaces *newvar* if it already exists.

Remarks

`tssmooth nl` works as a front end to `smooth`. See [\[R\] smooth](#) for details.

Saved results

`tssmooth` saves the following in `r()`:

Scalars	
<code>r(N)</code>	number of observations
Macros	
<code>r(method)</code>	<code>nl</code>
<code>r(smoother)</code>	specified smoother
<code>r(timevar)</code>	time variable specified in <code>tsset</code>
<code>r(panelvar)</code>	panel variable specified in <code>tsset</code>

Methods and formulas

`tssmooth nl` is implemented as an ado-file. The methods are documented in [\[R\] smooth](#).

A truncated description of the specified nonlinear filter labels the new variable. See [\[D\] label](#) for more information on labels.

An untruncated description of the specified nonlinear filter is saved in the characteristic `tssmooth` for the new variable. See [\[P\] char](#) for more information on characteristics.

Also see

- [\[TS\] tsset](#) — Declare data to be time-series data
- [\[TS\] tssmooth](#) — Smooth and forecast univariate time-series data

Syntax

```
tssmooth shwinters [type] newvar = exp [if] [in] [, options]
```

options	description
Main	
replace	replace <i>newvar</i> if it already exists
parms($\#_{\alpha}$ $\#_{\beta}$ $\#_{\gamma}$)	use $\#_{\alpha}$, $\#_{\beta}$, and $\#_{\gamma}$ as smoothing parameters
samp0($\#$)	use $\#$ observations to obtain initial values for recursion
s0($\#_{\text{cons}}$ $\#_{\text{lt}}$)	use $\#_{\text{cons}}$ and $\#_{\text{lt}}$ as initial values for recursion
forecast($\#$)	use $\#$ periods for the out-of-sample forecast
period($\#$)	use $\#$ for period of the seasonality
additive	use additive seasonal Holt–Winters method
Options	
sn0_0(<i>varname</i>)	use initial seasonal values in <i>varname</i>
sn0_v(<i>newvar</i>)	store estimated initial values for seasonal terms in <i>newvar</i>
snt_v(<i>newvar</i>)	store final year’s estimated seasonal terms in <i>newvar</i>
normalize	normalize seasonal values
altstarts	use alternative method for computing the starting values
Maximization	
maximize_options	control the maximization process; seldom used
from($\#_{\alpha}$ $\#_{\beta}$ $\#_{\gamma}$)	use $\#_{\alpha}$, $\#_{\beta}$, and $\#_{\gamma}$ as starting values for the parameters

You must `tsset` your data before using `tssmooth shwinters`; see [TS] `tsset`.
exp may contain time-series operators; see [U] 11.4.4 Time-series varlists.

Menu

Statistics > Time series > Smoothers/univariate forecasters > Holt-Winters seasonal smoothing

Description

`tssmooth shwinters` performs the seasonal Holt–Winters method on a user-specified expression, which is usually just a variable name, and generates a new variable containing the forecasted series.

Options

Main
replace replaces <i>newvar</i> if it already exists.

`parms(# $\alpha$  # $\beta$  # $\gamma$ )`, $0 \leq \#_{\alpha} \leq 1$, $0 \leq \#_{\beta} \leq 1$, and $0 \leq \#_{\gamma} \leq 1$, specifies the parameters. If `parms()` is not specified, the values are chosen by an iterative process to minimize the in-sample sum-of-squared prediction errors.

If you experience difficulty converging (many iterations and “not concave” messages), try using `from()` to provide better starting values.

`samp0(#)` and `s0(# $_{\text{cons}}$  # $_{\text{lt}}$ )` have to do with how the initial values $\#_{\text{cons}}$ and $\#_{\text{lt}}$ for the recursion are obtained.

`s0(# $_{\text{cons}}$  # $_{\text{lt}}$ )` specifies the initial values to be used.

`samp0(#)` specifies that the initial values be obtained using the first $\#$ observations of the sample. This calculation is described under [Methods and formulas](#) and depends on whether the `altstart` and `additive` options are also specified.

If neither option is specified, the first half of the sample is used to obtain initial values.

`forecast(#)` specifies the number of periods for the out-of-sample prediction; $0 \leq \# \leq 500$. The default is `forecast(0)`, which is equivalent to not performing an out-of-sample forecast.

`period(#)` specifies the period of the seasonality. If `period()` is not specified, the seasonality is obtained from the `tsset` options `daily`, `weekly`, ..., `yearly`. If you did not specify one of those options when you `tsset` the data, you must specify the `period()` option. For instance, if your data are quarterly and you did not specify `tsset`’s `quarterly` option, you must now specify `period(4)`.

By default, seasonal values are calculated, but you may specify the initial seasonal values to be used via the `sn0_0(varname)` option. The first `period()` observations of *varname* are to contain the initial seasonal values.

`additive` uses the additive seasonal Holt–Winters method instead of the default multiplicative seasonal Holt–Winters method.

Options

`sn0_0(varname)` specifies the initial seasonal values to use. *varname* must contain a complete year’s worth of seasonal values, beginning with the first observation in the estimation sample. For example, if you have monthly data, the first 12 observations of *varname* must contain nonmissing data. `sn0_0()` cannot be used with `sn0_v()`.

`sn0_v(newvar)` stores in *newvar* the initial seasonal values after they have been estimated. `sn0_v()` cannot be used with `sn0_0()`.

`snt_v(newvar)` stores in *newvar* the seasonal values for the final year’s worth of data.

`normalize` specifies that the seasonal values be normalized. In the multiplicative model, they are normalized to sum to one. In the additive model, the seasonal values are normalized to sum to zero.

`altstarts` uses an alternative method to compute the starting values for the constant, the linear, and the seasonal terms. The default and the alternative methods are described in [Methods and formulas](#). `altstarts` may not be specified with `s0()`.

Maximization

`maximize_options` controls the process for solving for the optimal α , β , and γ when the `parms()` option is not specified.

maximize_options: `nodifficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, `nonrntolerance`, see [R] [maximize](#). These options are seldom used.

`from(# $\alpha$  # $\beta$  # $\gamma$ )`, $0 < \#_\alpha < 1$, $0 < \#_\beta < 1$, and $0 < \#_\gamma < 1$, specifies starting values from which the optimal values of α , β , and γ will be obtained. If `from()` is not specified, `from(.5 .5 .5)` is used.

Remarks

Remarks are presented under the following headings:

Introduction

Holt–Winters seasonal multiplicative method

Holt–Winters seasonal additive method

Introduction

The seasonal Holt–Winters methods forecast univariate series that have a seasonal component. If the amplitude of the seasonal component grows with the series, the Holt–Winters multiplicative method should be used. If the amplitude of the seasonal component is not growing with the series, the Holt–Winters additive method should be used. [Abraham and Ledolter \(1983\)](#), [Bowerman and O’Connell \(1993\)](#), and [Montgomery, Johnson, and Gardiner \(1990\)](#) provide good introductions to the Holt–Winters methods in recursive univariate forecasting methods. [Chatfield \(2001, 2004\)](#) provides introductions in the broader context of modern time-series analysis.

Like the other recursive methods in `tssmooth`, `tssmooth shwinters` uses the information stored by `tsset` to detect panel data. When applied to panel data, each series is smoothed separately, and the starting values are computed separately for each panel. If the smoothing parameters are chosen to minimize the in-sample sum-of-squared forecast errors, the optimization is performed separately on each panel.

When there are missing values at the beginning of the series, the sample begins with the first nonmissing observation. Missing values after the first nonmissing observation are filled in with forecasted values.

Holt–Winters seasonal multiplicative method

This method forecasts seasonal time series in which the amplitude of the seasonal component grows with the series. [Chatfield \(2001\)](#) notes that there are some nonlinear state-space models whose optimal prediction equations correspond to the multiplicative Holt–Winters method. This procedure is best applied to data that could be described by

$$x_{t+j} = (\mu_t + \beta j)S_{t+j} + \epsilon_{t+j}$$

where x_t is the series, μ_t is the time-varying mean at time t , β is a parameter, S_t is the seasonal component at time t , and ϵ_t is an idiosyncratic error. See [Methods and formulas](#) for the updating equations.

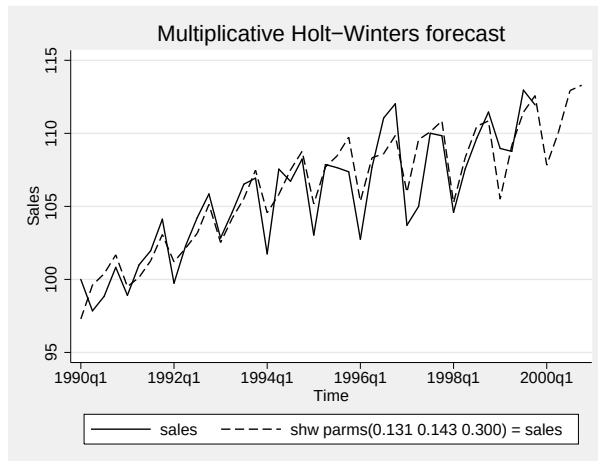
► Example 1

We have quarterly data on turkey sales by a new producer in the 1990s. The data have a strong seasonal component and an upward trend. We use the multiplicative Holt–Winters method to forecast sales for the year 2000. Because we have already `tsset` our data to the quarterly format, we do not need to specify the `period()` option.

```
. use http://www.stata-press.com/data/r11/turksales
. tssmooth shwinters shw1 = sales, forecast(4)
computing optimal weights
Iteration 0:   penalized RSS = -189.34609   (not concave)
Iteration 1:   penalized RSS = -108.68038   (not concave)
Iteration 2:   penalized RSS = -106.24548
Iteration 3:   penalized RSS =  -106.141
Iteration 4:   penalized RSS = -106.14093
Iteration 5:   penalized RSS = -106.14093
Optimal weights:
                    alpha = 0.1310
                    beta  = 0.1428
                    gamma = 0.2999
penalized sum-of-squared residuals = 106.1409
sum-of-squared residuals = 106.1409
root mean squared error  = 1.628964
```

The graph below describes the fit and the forecast that was obtained.

```
. line sales shw1 t, title("Multiplicative Holt–Winters forecast")
> xtitle(Time) ytitle(Sales)
```



◀

Holt–Winters seasonal additive method

This method is similar to the previous one, but the seasonal effect is assumed to be additive rather than multiplicative. This method forecasts series that can be described by the equation

$$x_{t+j} = (\mu_t + \beta j) + S_{t+j} + \epsilon_{t+j}$$

See [Methods and formulas](#) for the updating equations.

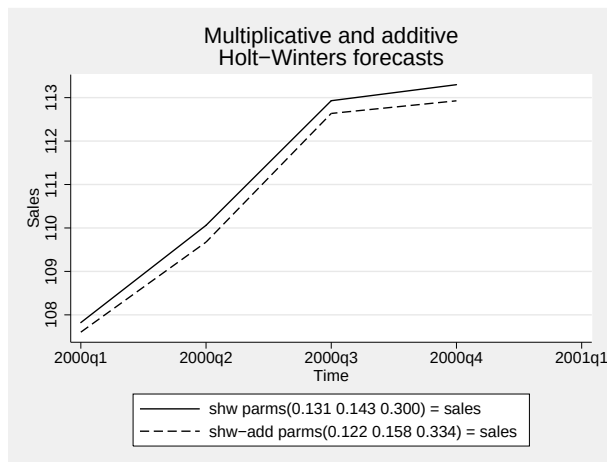
► Example 2

In this example, we fit the data from the previous example to the additive model to forecast sales in the coming year. We use the `snt_v()` option to save the last year's seasonal terms in the new variable `seas`.

```
. tssmooth shwinters shwa = sales, forecast(4) snt_v(seas) normalize additive
computing optimal weights
Iteration 0:   penalized RSS = -190.90242   (not concave)
Iteration 1:   penalized RSS = -108.8357
Iteration 2:   penalized RSS = -108.25359
Iteration 3:   penalized RSS = -107.68187
Iteration 4:   penalized RSS = -107.66444
Iteration 5:   penalized RSS = -107.66442
Iteration 6:   penalized RSS = -107.66442
Optimal weights:
                alpha = 0.1219
                beta  = 0.1580
                gamma = 0.3340
penalized sum-of-squared residuals = 107.6644
sum-of-squared residuals = 107.6644
root mean squared error = 1.640613
```

The output reveals that the multiplicative model has a better in-sample fit, and the graph below shows that the forecast from the multiplicative model is higher than that of the additive model.

```
. line shw1 shwa t if t>=tq(2000q1), title("Multiplicative and additive"
> "Holt-Winters forecasts") xtitle("Time") ytitle("Sales") legend(cols(1))
```



To check whether the estimated seasonal components are intuitively sound, we list the last year's seasonal components.

(Continued on next page)

```
. list t seas if seas < .
```

	t	seas
37.	1999q1	-2.7533393
38.	1999q2	-.91752567
39.	1999q3	1.8082417
40.	1999q4	1.8626233

The output indicates that the signs of the estimated seasonal components agree with our intuition.



Saved results

tssmooth saves the following in `r()`:

Scalars

<code>r(N)</code>	number of observations	<code>r(N_pre)</code>	number of seasons used
<code>r(alpha)</code>	α smoothing parameter		in calculating starting values
<code>r(beta)</code>	β smoothing parameter	<code>r(s2_0)</code>	initial value for linear term
<code>r(gamma)</code>	γ smoothing parameter	<code>r(s1_0)</code>	initial value for constant term
<code>r(prss)</code>	penalized sum-of-squared errors	<code>r(linear)</code>	final value of linear term
<code>r(rss)</code>	sum-of-squared errors	<code>r(constant)</code>	final value of constant term
<code>r(rmse)</code>	root mean squared error	<code>r(period)</code>	period, if filter is seasonal

Macros

<code>r(method)</code>	shwinters, additive or shwinters, multiplicative	<code>r(exp)</code>	expression specified
<code>r(normalize)</code>	normalize, if specified	<code>r(timevar)</code>	time variable specified in <code>tsset</code>
		<code>r(panelvar)</code>	panel variable specified in <code>tsset</code>

Methods and formulas

tssmooth shwinters is implemented as an ado-file.

A truncated description of the specified seasonal Holt–Winters filter labels the new variable. See [\[D\] label](#) for more information on labels.

An untruncated description of the specified seasonal Holt–Winters filter is saved in the characteristic named tssmooth for the new variable. See [\[P\] char](#) for more information on characteristics.

When the `parms()` option is not specified, the smoothing parameters are chosen to minimize the in-sample sum of penalized squared-forecast errors. Sometimes, one or more of the three optimal parameters lies on the boundary $[0, 1]$. To keep the estimates inside $[0, 1]$, tssmooth shwinters parameterizes the objective function in terms of their inverse logits, i.e., $\exp(\alpha)/\{1 + \exp(\alpha)\}$, $\exp(\beta)/\{1 + \exp(\beta)\}$, and $\exp(\gamma)/\{1 + \exp(\gamma)\}$. When one of these parameters is actually on the boundary, this can complicate the optimization. For this reason, tssmooth shwinters optimizes a penalized sum-of-squared forecast errors. Let $\hat{x}_t(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$ be the forecast for the series x_t given the choices of $\tilde{\alpha}$, $\tilde{\beta}$, and $\tilde{\gamma}$. Then the in-sample penalized sum-of-squared prediction errors is

$$P = \sum_{t=1}^T \left[\{x_t - \hat{x}_t(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})\}^2 + I_{|f(\tilde{\alpha})| > 12} (|f(\tilde{\alpha})| - 12)^2 + I_{|f(\tilde{\beta})| > 12} (|f(\tilde{\beta})| - 12)^2 + I_{|f(\tilde{\gamma})| > 12} (|f(\tilde{\gamma})| - 12)^2 \right]$$

where $f(x) = \ln\left(\frac{x}{1-x}\right)$. The penalty term is zero unless one of the parameters is close to the boundary. When one of the parameters is close to the boundary, the penalty term will help to obtain convergence.

Holt–Winters seasonal multiplicative procedure

As with the other recursive methods in `tssmooth`, there are three aspects to implementing the Holt–Winters seasonal multiplicative procedure: the forecasting equation, the initial values, and the updating equations. Unlike in the other methods, the data are now assumed to be seasonal with period L .

Given the estimates $a(t)$, $b(t)$, and $s(t + \tau - L)$, a τ step-ahead point forecast of x_t , denoted by $\hat{y}_{t+\tau}$, is

$$\hat{y}_{t+\tau} = \{a(t) + b(t)\tau\} s(t + \tau - L)$$

Given the smoothing parameters α , β , and γ , the updating equations are

$$a(t) = \alpha \frac{x_t}{s(t - L)} + (1 - \alpha) \{a(t - 1) + b(t - 1)\}$$

$$b(t) = \beta \{a(t) - a(t - 1)\} + (1 - \beta) b(t - 1)$$

and

$$s(t) = \gamma \left\{ \frac{x_t}{a(t)} \right\} + (1 - \gamma) s(t - L)$$

To restrict the seasonal terms to sum to 1 over each year, specify the `normalize` option.

The updating equations require the $L + 2$ initial values $a(0)$, $b(0)$, $s(1 - L)$, $s(2 - L)$, $\dots$, $s(0)$. Two methods calculate the initial values with the first m years, each of which contains L seasons. By default, m is set to the number of seasons in half the sample.

The initial value of the trend component, $b(0)$, can be estimated by

$$b(0) = \frac{\bar{x}_m - \bar{x}_1}{(m - 1)L}$$

where $\bar{x}_m$ is the average level of x_t in year m and $\bar{x}_1$ is the average level of x_t in the first year.

The initial value for the linear term, $a(0)$, is then calculated as

$$a(0) = \bar{x}_1 - \frac{L}{2} b(0)$$

To calculate the initial values for the seasons $1, 2, \dots, L$, we first calculate the deviation-adjusted values,

$$S(t) = \frac{x_t}{\bar{x}_i - \left\{ \frac{(L+1)}{2} - j \right\} b(0)}$$

where i is the year that corresponds to time t , j is the season that corresponds to time t , and $\bar{x}_i$ is the average level of x_t in year i .

Next, for each season $l = 1, 2, \dots, L$, we define $\bar{s}_l$ as the average S_t over the years. That is,

$$\bar{s}_l = \frac{1}{m} \sum_{k=0}^{m-1} S_{l+kL} \quad \text{for } l = 1, 2, \dots, L$$

Then the initial seasonal estimates are

$$s_{0l} = \bar{s}_l \left(\frac{L}{\sum_{l=1}^L \bar{s}_l} \right) \quad \text{for } l = 1, 2, \dots, L$$

and these values are used to fill in $s(1-L), \dots, s(0)$.

If the `altstarts` option is specified, the starting values are computed based on a regression with seasonal indicator variables. Specifically, the series x_t is regressed on a time variable normalized to equal one in the first period in the sample and on a constant. Then $b(0)$ is set to the estimated coefficient on the time variable, and $a(0)$ is set to the estimated constant term. To calculate the seasonal starting values, x_t is regressed on a set of L seasonal dummy variables. The l th seasonal starting value is set to $(\frac{1}{\mu})\hat{\beta}_l$, where μ is the mean of x_t and $\hat{\beta}_l$ is the estimated coefficient on the l th seasonal dummy variable. The sample used in both regressions and the mean computation is restricted to include the first `samp0()` years. By default, `samp0()` includes half the data.

□ Technical note

If there are missing values in the first few years, a small value of m can cause the starting value methods for seasonal term to fail. Here you should either specify a larger value of m by using `samp0()` or directly specify the seasonal starting values by using the `snt0_0()` option. □

Holt–Winters seasonal additive procedure

This procedure is similar to the previous one, except that the data are assumed to be described by

$$x_t = (\beta_0 + \beta_1 t) + s_t + \epsilon_t$$

As in the multiplicative case, there are three smoothing parameters, α , β , and γ , which can either be set or chosen to minimize the in-sample sum-of-squared forecast errors.

The updating equations are

$$a(t) = \alpha \{x_t - s(t-L)\} + (1-\alpha) \{a(t-1) + b(t-1)\}$$

$$b(t) = \beta \{a(t) - a(t-1)\} + (1-\beta)b(t-1)$$

and

$$s(t) = \gamma \{x_t - a(t)\} + (1-\gamma)s(t-L)$$

To restrict the seasonal terms to sum to 0 over each year, specify the `normalize` option.

A τ -step-ahead forecast, denoted by $\hat{y}_{t+\tau}$, is given by

$$\hat{x}_{t+\tau} = a(t) + b(t)\tau + s(t+\tau-L)$$

As in the multiplicative case, there are two methods for setting the initial values.

The default method is to obtain the initial values for $a(0), b(0), s(1-L), \dots, s(0)$ from the regression

$$x_t = a(0) + b(0)t + \beta_{s,1-L}D_1 + \beta_{s,2-L}D_2 + \dots + \beta_{s,0}D_L + e_t$$

where the $D_1, \dots, D_L$ are dummy variables with

$$D_i = \begin{cases} 1 & \text{if } t \text{ corresponds to season } i \\ 0 & \text{otherwise} \end{cases}$$

When `altstarts` is specified, an alternative method is used that regresses the x_t series on a time variable that has been normalized to equal one in the first period in the sample and on a constant term. $b(0)$ is set to the estimated coefficient on the time variable, and $a(0)$ is set to the estimated constant term. Then the demeaned series $\tilde{x}_t = x_t - \mu$ is created, where μ is the mean of the x_t . The $\tilde{x}_t$ are regressed on L seasonal dummy variables. The l th seasonal starting value is then set to β_l , where β_l is the estimated coefficient on the l th seasonal dummy variable. The sample in both the regression and the mean calculation is restricted to include the first `samp0` years, where, by default, `samp0()` includes half the data.

Acknowledgment

We thank Nicholas J. Cox of Durham University for his helpful comments.

References

- Abraham, B., and J. Ledolter. 1983. *Statistical Methods for Forecasting*. New York: Wiley.
- Bowerman, B. L., and R. T. O'Connell. 1993. *Forecasting and Time Series: An Applied Approach*. 3rd ed. Pacific Grove, CA: Duxbury.
- Chatfield, C. 2001. *Time-Series Forecasting*. London: Chapman & Hall/CRC.
- . 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- Chatfield, C., and M. Yar. 1988. Holt-Winters forecasting: Some practical issues. *Statistician* 37: 129–140.
- Holt, C. C. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20: 5–10.
- Montgomery, D. C., L. A. Johnson, and J. S. Gardiner. 1990. *Forecasting and Time Series Analysis*. 2nd ed. New York: McGraw–Hill.
- Winters, P. R. 1960. Forecasting sales by exponentially weighted moving averages. *Management Science* 6: 324–342.

Also see

- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [tssmooth](#) — Smooth and forecast univariate time-series data

Description

Stata has a suite of commands for fitting, forecasting, interpreting, and performing inference on vector autoregressive (VAR) models and structural vector autoregressive (SVAR) models. The suite includes several commands for estimating and interpreting impulse–response functions (IRFs), dynamic-multiplier functions, and forecast-error variance decompositions (FEVDs). The table below describes the available commands.

Fitting a VAR or SVAR

var	[TS] var	Fit vector autoregressive models
svar	[TS] var svar	Fit structural vector autoregressive models
varbasic	[TS] varbasic	Fit a simple VAR and graph IRFs or FEVDs

Model diagnostics and inference

varstable	[TS] varstable	Check the stability condition of VAR or SVAR estimates
varsoc	[TS] varsoc	Obtain lag-order selection statistics for VARs and VECMs
varwle	[TS] varwle	Obtain Wald lag-exclusion statistics after <code>var</code> or <code>svar</code>
vargranger	[TS] vargranger	Perform pairwise Granger causality tests after <code>var</code> or <code>svar</code>
varlmar	[TS] varlmar	Perform LM test for residual autocorrelation after <code>var</code> or <code>svar</code>
varnorm	[TS] varnorm	Test for normally distributed disturbances after <code>var</code> or <code>svar</code>

Forecasting after fitting a VAR or SVAR

fcast compute	[TS] fcast compute	Compute dynamic forecasts of dependent variables after <code>var</code> , <code>svar</code> , or <code>vec</code>
fcast graph	[TS] fcast graph	Graph forecasts of variables computed by <code>fcast compute</code>

Working with IRFs, dynamic-multiplier functions, and FEVDs

irf	[TS] irf	Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

This entry provides an overview of vector autoregressions and structural vector autoregressions. More rigorous treatments can be found in [Hamilton \(1994\)](#), [Lütkepohl \(2005\)](#), and [Amisano and Giannini \(1997\)](#). [Stock and Watson \(2001\)](#) provide an excellent nonmathematical treatment of vector autoregressions and their role in macroeconomics.

Remarks

Remarks are presented under the following headings:

Introduction to VARs
Introduction to SVARs
Short-run SVAR models
Long-run restrictions
IRFs and FEVDs

Introduction to VARs

A VAR is a model in which K variables are specified as linear functions of p of their own lags, p lags of the other $K - 1$ variables, and possibly additional exogenous variables. Algebraically, a p -order VAR model, written VAR(p), with exogenous variables $\mathbf{x}_t$ is given by

$$\mathbf{y}_t = \mathbf{v} + \mathbf{A}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{A}_p \mathbf{y}_{t-p} + \mathbf{B}_0 \mathbf{x}_t + \mathbf{B}_1 \mathbf{x}_{t-1} + \cdots + \mathbf{B}_s \mathbf{x}_{t-s} + \mathbf{u}_t \quad t \in \{-\infty, \infty\} \quad (1)$$

where

$\mathbf{y}_t = (y_{1t}, \dots, y_{Kt})'$ is a $K \times 1$ random vector,
 $\mathbf{A}_1$ through $\mathbf{A}_p$ are $K \times K$ matrices of parameters,
 $\mathbf{x}_t$ is an $M \times 1$ vector of exogenous variables,
 $\mathbf{B}_0$ through $\mathbf{B}_s$ are $K \times M$ matrices of coefficients,
 $\mathbf{v}$ is a $K \times 1$ vector of parameters, and
 $\mathbf{u}_t$ is assumed to be white noise; that is,
 $E(\mathbf{u}_t) = \mathbf{0}$,
 $E(\mathbf{u}_t \mathbf{u}_t') = \Sigma$, and
 $E(\mathbf{u}_t \mathbf{u}_s') = \mathbf{0}$ for $t \neq s$

There are $K^2 \times p + K \times (M(s+1) + 1)$ parameters in the equation for $\mathbf{y}_t$, and there are $\{K \times (K+1)\}/2$ parameters in the covariance matrix Σ . One way to reduce the number of parameters is to specify an incomplete VAR, in which some of the $\mathbf{A}$ or $\mathbf{B}$ matrices are set to zero. Another way is to specify linear constraints on some of the coefficients in the VAR.

A VAR can be viewed as the reduced form of a system of dynamic simultaneous equations. Consider the system

$$\mathbf{W}_0 \mathbf{y}_t = \mathbf{a} + \mathbf{W}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{W}_p \mathbf{y}_{t-p} + \widetilde{\mathbf{W}}_1 \mathbf{x}_t + \widetilde{\mathbf{W}}_2 \mathbf{x}_{t-2} + \cdots + \widetilde{\mathbf{W}}_s \mathbf{x}_{t-s} + \mathbf{e}_t \quad (2)$$

where $\mathbf{a}$ is a $K \times 1$ vector of parameters, each $\mathbf{W}_i$, $i = 0, \dots, p$, is a $K \times K$ matrix of parameters, and $\mathbf{e}_t$ is a $K \times 1$ disturbance vector. In the traditional dynamic simultaneous equations approach, sufficient restrictions are placed on the $\mathbf{W}_i$ to obtain identification. Assuming that $\mathbf{W}_0$ is nonsingular, (2) can be rewritten as

$$\begin{aligned} \mathbf{y}_t = & \mathbf{W}_0^{-1} \mathbf{a} + \mathbf{W}_0^{-1} \mathbf{W}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{W}_0^{-1} \mathbf{W}_p \mathbf{y}_{t-p} \\ & + \mathbf{W}_0^{-1} \widetilde{\mathbf{W}}_1 \mathbf{x}_t + \mathbf{W}_0^{-1} \widetilde{\mathbf{W}}_2 \mathbf{x}_{t-2} + \cdots + \mathbf{W}_0^{-1} \widetilde{\mathbf{W}}_s \mathbf{x}_{t-s} + \mathbf{W}_0^{-1} \mathbf{e}_t \end{aligned} \quad (3)$$

which is a VAR with

$$\begin{aligned} \mathbf{v} &= \mathbf{W}_0^{-1} \mathbf{a} \\ \mathbf{A}_i &= \mathbf{W}_0^{-1} \mathbf{W}_i \\ \mathbf{B}_i &= \mathbf{W}_0^{-1} \widetilde{\mathbf{W}}_i \\ \mathbf{u}_t &= \mathbf{W}_0^{-1} \mathbf{e}_t \end{aligned}$$

The cross-equation error variance–covariance matrix Σ contains all the information about contemporaneous correlations in a VAR and may be the VAR’s greatest strength and its greatest weakness. Because no questionable a priori assumptions are imposed, fitting a VAR allows the dataset to speak for itself. However, without imposing some restrictions on the structure of Σ , we cannot make a causal interpretation of the results.

If we make additional technical assumptions, we can derive another representation of the VAR in (1). If the VAR is stable (see [TS] [varstable](#)), we can rewrite $\mathbf{y}_t$ as

$$\mathbf{y}_t = \boldsymbol{\mu} + \sum_{i=0}^{\infty} \mathbf{D}_i \mathbf{x}_{t-i} + \sum_{i=0}^{\infty} \boldsymbol{\Phi}_i \mathbf{u}_{t-i} \quad (4)$$

where $\boldsymbol{\mu}$ is the $K \times 1$ time-invariant mean of the process and $\mathbf{D}_i$ and $\boldsymbol{\Phi}_i$ are $K \times R$ and $K \times K$ matrices of parameters, respectively. Equation (4) states that the process by which the variables in $\mathbf{y}_t$ fluctuate about their time-invariant means, $\boldsymbol{\mu}$, is completely determined by the parameters in $\mathbf{D}_i$ and $\boldsymbol{\Phi}_i$ and the (infinite) past history of the exogenous variables $\mathbf{x}_t$ and the independent and identically distributed (i.i.d.) shocks or innovations, $\mathbf{u}_{t-1}, \mathbf{u}_{t-2}, \dots$. Equation (4) is known as the vector moving-average representation of the VAR. The $\mathbf{D}_i$ are the dynamic-multiplier functions, or transfer functions. The moving-average coefficients $\boldsymbol{\Phi}_i$ are also known as the simple IRFs at horizon i . The precise relationships between the VAR parameters and the $\mathbf{D}_i$ and $\boldsymbol{\Phi}_i$ are derived in [Methods and formulas](#) of [TS] [irf create](#).

The joint distribution of $\mathbf{y}_t$ is determined by the distributions of $\mathbf{x}_t$ and $\mathbf{u}_t$ and the parameters $\mathbf{v}$, $\mathbf{B}_i$, and $\mathbf{A}_i$. Estimating the parameters in a VAR requires that the variables in $\mathbf{y}_t$ and $\mathbf{x}_t$ be covariance stationary, meaning that their first two moments exist and are time invariant. If the $\mathbf{y}_t$ are not covariance stationary, but their first differences are, a vector error-correction model (VECM) can be used. See [TS] [vec intro](#) and [TS] [vec](#) for more information about those models.

If the $\mathbf{u}_t$ form a zero mean, i.i.d. vector process, and $\mathbf{y}_t$ and $\mathbf{x}_t$ are covariance stationary and are not correlated with the $\mathbf{u}_t$, consistent and efficient estimates of the $\mathbf{B}_i$, the $\mathbf{A}_i$, and $\mathbf{v}$ are obtained via seemingly unrelated regression, yielding estimators that are asymptotically normally distributed. When the equations for the variables $\mathbf{y}_t$ have the same set of regressors, equation-by-equation OLS estimates are the conditional maximum likelihood estimates.

Much of the interest in VAR models is focused on the forecasts, IRFs, dynamic-multiplier functions, and the FEVDs, all of which are functions of the estimated parameters. Estimating these functions is straightforward, but their asymptotic standard errors are usually obtained by assuming that $\mathbf{u}_t$ forms a zero mean, i.i.d. Gaussian (normal) vector process. Also, some of the specification tests for VARs have been derived using the likelihood-ratio principle and the stronger Gaussian assumption.

In the absence of contemporaneous exogenous variables, the disturbance variance–covariance matrix contains all the information about contemporaneous correlations among the variables. VARs are sometimes classified into three types by how they account for this contemporaneous correlation. (See [Stock and Watson \[2001\]](#) for one derivation of this taxonomy.) A reduced-form VAR, aside from estimating the variance–covariance matrix of the disturbance, does not try to account for contemporaneous correlations. In a recursive VAR, the K variables are assumed to form a recursive dynamic structural equation model in which the first variable is a function of lagged variables, the second is a function of contemporaneous values of the first variable and lagged values, and so on. In a structural VAR, the theory you are working with places restrictions on the contemporaneous correlations that are not necessarily recursive.

Stata has two commands for fitting reduced-form VARs: `var` and `varbasic`. `var` allows for constraints to be imposed on the coefficients. `varbasic` allows you to fit a simple VAR quickly without constraints and graph the IRFs.

Because fitting a VAR of the correct order can be important, `varsoc` offers several methods for choosing the lag order p of the VAR to fit. After fitting a VAR, and before proceeding with inference, interpretation, or forecasting, checking that the VAR fits the data is important. `varlmar` can be used to check for autocorrelation in the disturbances. `varwle` performs Wald tests to determine whether certain lags can be excluded. `varnorm` tests the null hypothesis that the disturbances are normally distributed. `varstable` checks the eigenvalue condition for stability, which is needed to interpret the IRFs and IRFS.

Introduction to SVARs

As discussed in [TS] [irf create](#), a problem with VAR analysis is that, because Σ is not restricted to be a diagonal matrix, an increase in an innovation to one variable provides information about the innovations to other variables. This implies that no causal interpretation of the simple IRFs is possible: there is no way to determine whether the shock to the first variable caused the shock in the second variable or vice versa.

However, suppose that we had a matrix $\mathbf{P}$ such that $\Sigma = \mathbf{P}\mathbf{P}'$. We can then show that the variables in $\mathbf{P}^{-1}\mathbf{u}_t$ have zero mean and that $E\{\mathbf{P}^{-1}\mathbf{u}_t(\mathbf{P}^{-1}\mathbf{u}_t)'\} = \mathbf{I}_K$. We could rewrite (4) as

$$\begin{aligned} \mathbf{y}_t &= \boldsymbol{\mu} + \sum_{s=0}^{\infty} \boldsymbol{\Phi}_s \mathbf{P} \mathbf{P}^{-1} \mathbf{u}_{t-s} \\ &= \boldsymbol{\mu} + \sum_{s=0}^{\infty} \boldsymbol{\Theta}_s \mathbf{P}^{-1} \mathbf{u}_{t-s} \\ &= \boldsymbol{\mu} + \sum_{s=0}^{\infty} \boldsymbol{\Theta}_s \mathbf{w}_{t-s} \end{aligned} \tag{5}$$

where $\boldsymbol{\Theta}_s = \boldsymbol{\Phi}_s \mathbf{P}$ and $\mathbf{w}_t = \mathbf{P}^{-1} \mathbf{u}_t$. If we had such a $\mathbf{P}$, the $\mathbf{w}_k$ would be mutually orthogonal, and the $\boldsymbol{\Theta}_s$ would allow the causal interpretation that we seek.

SVAR models provide a framework for estimation of and inference about a broad class of $\mathbf{P}$ matrices. As described in [TS] [irf create](#), the estimated $\mathbf{P}$ matrices can then be used to estimate structural IRFs and structural FEVDs. There are two types of SVAR models. Short-run SVAR models identify a $\mathbf{P}$ matrix by placing restrictions on the contemporaneous correlations between the variables. Long-run SVAR models, on the other hand, do so by placing restrictions on the long-term accumulated effects of the innovations.

Short-run SVAR models

A short-run SVAR model without exogenous variables can be written as

$$\mathbf{A}(\mathbf{I}_K - \mathbf{A}_1 L - \mathbf{A}_2 L^2 - \cdots - \mathbf{A}_p L^p) \mathbf{y}_t = \mathbf{A} \boldsymbol{\epsilon}_t = \mathbf{B} \mathbf{e}_t \tag{6}$$

where L is the lag operator; $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{A}_1, \dots, \mathbf{A}_p$ are $K \times K$ matrices of parameters; $\boldsymbol{\epsilon}_t$ is a $K \times 1$ vector of innovations with $\boldsymbol{\epsilon}_t \sim N(\mathbf{0}, \Sigma)$ and $E[\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_s'] = \mathbf{0}_K$ for all $s \neq t$; and $\mathbf{e}_t$ is a $K \times 1$ vector of orthogonalized disturbances; i.e., $\mathbf{e}_t \sim N(\mathbf{0}, \mathbf{I}_K)$ and $E[\mathbf{e}_t \mathbf{e}_s'] = \mathbf{0}_K$ for all $s \neq t$. These transformations of the innovations allow us to analyze the dynamics of the system in terms of a change to an element of $\mathbf{e}_t$. In a short-run SVAR model, we obtain identification by placing restrictions on $\mathbf{A}$ and $\mathbf{B}$, which are assumed to be nonsingular.

Equation (6) implies that $\mathbf{P}_{\text{sr}} = \mathbf{A}^{-1}\mathbf{B}$, where $\mathbf{P}_{\text{sr}}$ is the $\mathbf{P}$ matrix identified by a particular short-run SVAR model. The latter equality in (6) implies that

$$\mathbf{A}\epsilon_t\epsilon_t'\mathbf{A}' = \mathbf{B}\mathbf{e}_t\mathbf{e}_t'\mathbf{B}'$$

Taking the expectation of both sides yields

$$\Sigma = \mathbf{P}_{\text{sr}}\mathbf{P}_{\text{sr}}'$$

Assuming that the underlying VAR is stable (see [TS] [varstable](#) for a discussion of stability), we can invert the autoregressive representation of the model in (6) to an infinite-order, moving-average representation of the form

$$\mathbf{y}_t = \boldsymbol{\mu} + \sum_{s=0}^{\infty} \boldsymbol{\Theta}_s^{\text{sr}} \mathbf{e}_{t-s} \quad (7)$$

whereby $\mathbf{y}_t$ is expressed in terms of the mutually orthogonal, unit-variance structural innovations $\mathbf{e}_t$. The $\boldsymbol{\Theta}_s^{\text{sr}}$ contain the structural IRFs at horizon s .

In a short-run SVAR model, the $\mathbf{A}$ and $\mathbf{B}$ matrices model all the information about contemporaneous correlations. The $\mathbf{B}$ matrix also scales the innovations $\mathbf{u}_t$ to have unit variance. This allows the structural IRFs constructed from (7) to be interpreted as the effect on variable i of a one-time unit increase in the structural innovation to variable j after s periods.

$\mathbf{P}_{\text{sr}}$ identifies the structural IRFs by defining a transformation of Σ , and $\mathbf{P}_{\text{sr}}$ is identified by the restrictions placed on the parameters in $\mathbf{A}$ and $\mathbf{B}$. Because there are only $K(K+1)/2$ free parameters in Σ , only $K(K+1)/2$ parameters may be estimated in an identified $\mathbf{P}_{\text{sr}}$. Because there are $2K^2$ total parameters in $\mathbf{A}$ and $\mathbf{B}$, the order condition for identification requires that at least $2K^2 - K(K+1)/2$ restrictions be placed on those parameters. Just as in the simultaneous-equations framework, this order condition is necessary but not sufficient. [Amisano and Giannini \(1997\)](#) derive a method to check that an SVAR model is locally identified near some specified values for $\mathbf{A}$ and $\mathbf{B}$.

Before moving on to models with long-run constraints, consider these limitations. We cannot place constraints on the elements of $\mathbf{A}$ in terms of the elements of $\mathbf{B}$, or vice versa. This limitation is imposed by the form of the check for identification derived by [Amisano and Giannini \(1997\)](#). As noted in [Methods and formulas](#) of [TS] [var svar](#), this test requires separate constraint matrices for the parameters in $\mathbf{A}$ and $\mathbf{B}$. Also, we cannot mix short-run and long-run constraints.

Long-run restrictions

A general short-run SVAR has the form

$$\mathbf{A}(\mathbf{I}_K - \mathbf{A}_1L - \mathbf{A}_2L^2 - \cdots - \mathbf{A}_pL^p)\mathbf{y}_t = \mathbf{B}\mathbf{e}_t$$

To simplify the notation, let $\bar{\mathbf{A}} = (\mathbf{I}_K - \mathbf{A}_1L - \mathbf{A}_2L^2 - \cdots - \mathbf{A}_pL^p)$. The model is assumed to be stable (see [TS] [varstable](#)), so $\bar{\mathbf{A}}^{-1}$, the matrix of estimated long-run effects of the reduced-form VAR shocks, is well defined. Constraining $\mathbf{A}$ to be an identity matrix allows us to rewrite this equation as

$$\mathbf{y}_t = \bar{\mathbf{A}}^{-1}\mathbf{B}\mathbf{e}_t$$

which implies that $\Sigma = \mathbf{B}\mathbf{B}'$. Thus $\mathbf{C} = \bar{\mathbf{A}}^{-1}\mathbf{B}$ is the matrix of long-run responses to the orthogonalized shocks, and

$$\mathbf{y}_t = \mathbf{C}\mathbf{e}_t$$

In long-run models, the constraints are placed on the elements of $\mathbf{C}$, and the free parameters are estimated. These constraints are often exclusion restrictions. For instance, constraining $\mathbf{C}[1, 2]$ to be zero can be interpreted as setting the long-run response of variable 1 to the structural shocks driving variable 2 to be zero.

Stata's `svr` command estimates the parameters of structural VARs. See [TS] [var svr](#) for more information and examples.

IRFs and FEVDs

IRFs describe how the K endogenous variables react over time to a one-time shock to one of the K disturbances. Because the disturbances may be contemporaneously correlated, these functions do not explain how variable i reacts to a one-time increase in the innovation to variable j after s periods, holding everything else constant. To explain this, we must start with orthogonalized innovations so that the assumption to hold everything else constant is reasonable. Recursive VARs use a Cholesky decomposition to orthogonalize the disturbances and thereby obtain structurally interpretable IRFs. Structural VARs use theory to impose sufficient restrictions, which need not be recursive, to decompose the contemporaneous correlations into orthogonal components.

FEVDs are another tool for interpreting how the orthogonalized innovations affect the K variables over time. The FEVD from j to i gives the fraction of the s -step forecast-error variance of variable i that can be attributed to the j th orthogonalized innovation.

Dynamic-multiplier functions describe how the endogenous variables react over time to a unit change in an exogenous variable. This is a different experiment from that in IRFs and FEVDs because dynamic-multiplier functions consider a change in an exogenous variable instead of a shock to an endogenous variable.

`irf create` estimates IRFs, Cholesky orthogonalized IRFs, dynamic-multiplier functions, and structural IRFs and their standard errors. It also estimates Cholesky and structural FEVDs. The `irf graph`, `irf cgraph`, `irf ograph`, `irf table`, and `irf ctable` commands graph and tabulate these estimates. Stata also has several other commands to manage IRF and FEVD results. See [TS] [irf](#) for a description of these commands.

`fcast compute` computes dynamic forecasts and their standard errors from VARs. `fcast graph` graphs the forecasts that are generated using `fcast compute`.

VARs allow researchers to investigate whether one variable is useful in predicting another variable. A variable x is said to Granger-cause a variable y if, given the past values of y , past values of x are useful for predicting y . The Stata command `vargranger` performs Wald tests to investigate Granger causality between the variables in a VAR.

References

- Amisano, G., and C. Giannini. 1997. *Topics in Structural VAR Econometrics*. 2nd ed. Heidelberg: Springer.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Stock, J. H., and M. W. Watson. 2001. Vector autoregressions. *Journal of Economic Perspectives* 15: 101–115.
- Watson, M. W. 1994. Vector autoregressions and cointegration. In Vol. IV of *Handbook of Econometrics*, ed. R. F. Engle and D. L. McFadden. Amsterdam: Elsevier.

Also see

[TS] [var](#) — Vector autoregressive models

[TS] [var svar](#) — Structural vector autoregressive models

[TS] [vec intro](#) — Introduction to vector error-correction models

[TS] [vec](#) — Vector error-correction models

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

Syntax

var *depvarlist* [*if*] [*in*] [, *options*]

<i>options</i>	description
Model	
<u>no</u> constant	suppress constant term
<u>l</u> ags(<i>numlist</i>)	use lags <i>numlist</i> in the VAR
<u>e</u> xog(<i>varlist</i>)	use exogenous variables <i>varlist</i>
Model 2	
<u>c</u> onstraints(<i>numlist</i>)	apply specified linear constraints
<u>n</u> olog	suppress SURE iteration log
<u>i</u> terate(#)	set maximum number of iterations for SURE; default is <code>iterate(1600)</code>
<u>t</u> olerance(#)	set convergence tolerance of SURE
<u>n</u> oiseure	use one-step SURE
<u>d</u> fk	make small-sample degrees-of-freedom adjustment
<u>s</u> mall	report small-sample <i>t</i> and <i>F</i> statistics
<u>n</u> obigf	do not compute parameter vector for coefficients implicitly set to zero
Reporting	
<u>l</u> evel(#)	set confidence level; default is <code>level(95)</code>
<u>l</u> utstats	report Lütkepohl lag-order selection statistics
<u>n</u> ocnsreport	do not display constraints
<u>d</u> isplay_options	control spacing
[†] <u>c</u> oefflegend	display coefficients' legend instead of coefficient table

[†]`coefflegend` does not appear in the dialog box.
You must `tsset` your data before using `var`; see [TS] `tsset`.
depvarlist and *varlist* may contain time-series operators; see [U] 11.4.4 Time-series varlists.
`by`, `rolling`, `statsby`, and `xi` are allowed; see [U] 11.1.10 Prefix commands.
See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

Menu

Statistics > Multivariate time series > Vector autoregression (VAR)

Description

`var` fits a multivariate time-series regression of each dependent variable on lags of itself and on lags of all the other dependent variables. `var` also fits a variant of vector autoregressive (VAR) models known as the VARX model, which also includes exogenous variables. See [TS] [var intro](#) for a list of commands that are used in conjunction with `var`.

Options

Model

`noconstant`; see [R] [estimation options](#).

`lags(numlist)` specifies the lags to be included in the model. The default is `lags(1 2)`. This option takes a *numlist* and not simply an integer for the maximum lag. For example, `lags(2)` would include only the second lag in the model, whereas `lags(1/2)` would include both the first and second lags in the model. See [U] [11.1.8 numlist](#) and [U] [11.4.4 Time-series varlists](#) for more discussion of numlists and lags.

`exog(varlist)` specifies a list of exogenous variables to be included in the VAR.

Model 2

`constraints(numlist)`; see [R] [estimation options](#).

`nolog` suppresses the log from the iterated seemingly unrelated regression algorithm. By default, the iteration log is displayed when the coefficients are estimated through iterated seemingly unrelated regression. When the `constraints()` option is not specified, the estimates are obtained via OLS, and `nolog` has no effect. For this reason, `nolog` can be specified only when `constraints()` is specified. Similarly, `nolog` cannot be combined with `noisure`.

`iterate(#)` specifies an integer that sets the maximum number of iterations when the estimates are obtained through iterated seemingly unrelated regression. By default, the limit is 1,600. When `constraints()` is not specified, the estimates are obtained using OLS, and `iterate()` has no effect. For this reason, `iterate()` can be specified only when `constraints()` is specified. Similarly, `iterate()` cannot be combined with `noisure`.

`tolerance(#)` specifies a number greater than zero and less than 1 for the convergence tolerance of the iterated seemingly unrelated regression algorithm. By default, the tolerance is `1e-6`. When the `constraints()` option is not specified, the estimates are obtained using OLS, and `tolerance()` has no effect. For this reason, `tolerance()` can be specified only when `constraints()` is specified. Similarly, `tolerance()` cannot be combined with `noisure`.

`noisure` specifies that the estimates in the presence of constraints be obtained through one-step seemingly unrelated regression. By default, `var` obtains estimates in the presence of constraints through iterated seemingly unrelated regression. When `constraints()` is not specified, the estimates are obtained using OLS, and `noisure` has no effect. For this reason, `noisure` can be specified only when `constraints()` is specified.

`dfk` specifies that a small-sample degrees-of-freedom adjustment be used when estimating Σ , the error variance–covariance matrix. Specifically, $1/(T - \overline{m})$ is used instead of the large-sample divisor $1/T$, where $\overline{m}$ is the average number of parameters in the functional form for $\mathbf{y}_t$ over the K equations.

`small` causes `var` to report small-sample t and F statistics instead of the large-sample normal and chi-squared statistics.

`nobigf` requests that `var` not save the estimated parameter vector that incorporates coefficients that have been implicitly constrained to be zero, such as when some lags have been omitted from a model. `e(bf)` is used for computing asymptotic standard errors in the postestimation commands `irf create` and `fcast compute`. Therefore, specifying `nobigf` implies that the asymptotic standard errors will not be available from `irf create` and `fcast compute`. See [Fitting models with some lags excluded](#).

Reporting

`level(#)`; see [\[R\] estimation options](#).

`lutstats` specifies that the Lütkepohl (2005) versions of the lag-order selection statistics be reported.

See [Methods and formulas](#) in [\[TS\] varsoc](#) for a discussion of these statistics.

`nocnsreport`; see [\[R\] estimation options](#).

`display_options`: `vsquish`; see [\[R\] estimation options](#).

The following option is available with `var` but is not shown in the dialog box:

`coeflegend`; see [\[R\] estimation options](#).

Remarks

Remarks are presented under the following headings:

[Introduction](#)

[Fitting models with some lags excluded](#)

[Fitting models with exogenous variables](#)

[Fitting models with constraints on the coefficients](#)

Introduction

A VAR is a model in which K variables are specified as linear functions of p of their own lags, p lags of the other $K - 1$ variables, and possibly exogenous variables. A VAR with p lags is usually denoted a VAR(p). For more information, see [\[TS\] var intro](#).

► Example 1: VAR model

To illustrate the basic usage of `var`, we replicate the example in Lütkepohl (2005, 77–78). The data consists of three variables: the first difference of the natural log of investment, `dln_inv`; the first difference of the natural log of income, `dln_inc`; and the first difference of the natural log of consumption, `dln_consump`. The dataset contains data through the fourth quarter of 1982, though Lütkepohl uses only the observations through the fourth quarter of 1978.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. tsset
      time variable:  qtr, 1960q1 to 1982q4
              delta:  1 quarter
```

(Continued on next page)

```
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lutstats dfk
Vector autoregression
Sample: 1960q4 - 1978q4                                No. of obs      =          73
Log likelihood =    606.307                        (lutstats) AIC      = -24.63163
FPE           =  2.18e-11                            HQIC           = -24.40656
Det(Sigma_ml) =  1.23e-11                            SBIC           = -24.06686
```

Equation	Parms	RMSE	R-sq	chi2	P>chi2
dln_inv	7	.046148	0.1286	9.736909	0.1362
dln_inc	7	.011719	0.1142	8.508289	0.2032
dln_consump	7	.009445	0.2513	22.15096	0.0011

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
dln_inv						
dln_inv						
L1.	-.3196318	.1254564	-2.55	0.011	-.5655218	-.0737419
L2.	-.1605508	.1249066	-1.29	0.199	-.4053633	.0842616
dln_inc						
L1.	.1459851	.5456664	0.27	0.789	-.9235013	1.215472
L2.	.1146009	.5345709	0.21	0.830	-.9331388	1.162341
dln_consump						
L1.	.9612288	.6643086	1.45	0.148	-.3407922	2.26325
L2.	.9344001	.6650949	1.40	0.160	-.369162	2.237962
_cons	-.0167221	.0172264	-0.97	0.332	-.0504852	.0170409
dln_inc						
dln_inv						
L1.	.0439309	.0318592	1.38	0.168	-.018512	.1063739
L2.	.0500302	.0317196	1.58	0.115	-.0121391	.1121995
dln_inc						
L1.	-.1527311	.1385702	-1.10	0.270	-.4243237	.1188615
L2.	.0191634	.1357525	0.14	0.888	-.2469067	.2852334
dln_consump						
L1.	.2884992	.168699	1.71	0.087	-.0421448	.6191431
L2.	-.0102	.1688987	-0.06	0.952	-.3412354	.3208353
_cons	.0157672	.0043746	3.60	0.000	.0071932	.0243412
dln_consump						
dln_inv						
L1.	-.002423	.0256763	-0.09	0.925	-.0527476	.0479016
L2.	.0338806	.0255638	1.33	0.185	-.0162235	.0839847
dln_inc						
L1.	.2248134	.1116778	2.01	0.044	.005929	.4436978
L2.	.3549135	.1094069	3.24	0.001	.1404798	.5693471
dln_consump						
L1.	-.2639695	.1359595	-1.94	0.052	-.5304451	.0025062
L2.	-.0222264	.1361204	-0.16	0.870	-.2890175	.2445646
_cons	.0129258	.0035256	3.67	0.000	.0060157	.0198358

The output has two parts: a header and the standard Stata output table for the coefficients, standard errors, and confidence intervals. The header contains summary statistics for each equation in the VAR and statistics used in selecting the lag order of the VAR. Although there are standard formulas for all the lag-order statistics, [Lütkepohl \(2005\)](#) gives different versions of the three information criteria that drop the constant term from the likelihood. To obtain the [Lütkepohl \(2005\)](#) versions, we specified the `lutstats` option. The formulas for the standard and Lütkepohl versions of these statistics are given in [Methods and formulas](#) of `[TS] varsoc`.

The `dfk` option specifies that the small-sample divisor $1/(T - \overline{m})$ be used in estimating Σ instead of the maximum likelihood (ML) divisor $1/T$, where $\overline{m}$ is the average number of parameters included in each of the K equations. All the lag-order statistics are computed using the ML estimator of Σ . Thus, specifying `dfk` will not change the computed lag-order statistics, but it will change the estimated variance–covariance matrix. Also, when `dfk` is specified, a `dfk`-adjusted log likelihood is computed and saved in `e(11_dfk)`.

◀

The `lag()` option takes a *numlist* of lags. To specify a model that includes the first and second lags, type

```
. var y1 y2 y3, lags(1/2)
```

not

```
. var y1 y2 y3, lags(2)
```

because the latter specification would fit a model that included only the second lag.

Fitting models with some lags excluded

To fit a model that has only a fourth lag, that is,

$$\mathbf{y}_t = \mathbf{v} + \mathbf{A}_4 \mathbf{y}_{t-4} + \mathbf{u}_t$$

you would specify the `lags(4)` option. Doing so is equivalent to fitting the more general model

$$\mathbf{y}_t = \mathbf{v} + \mathbf{A}_1 \mathbf{y}_{t-1} + \mathbf{A}_2 \mathbf{y}_{t-2} + \mathbf{A}_3 \mathbf{y}_{t-3} + \mathbf{A}_4 \mathbf{y}_{t-4} + \mathbf{u}_t$$

with $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ constrained to be $\mathbf{0}$. When you fit a model with some lags excluded, `var` estimates the coefficients included in the specification ($\mathbf{A}_4$ here) and saves these estimates in `e(b)`. To obtain the asymptotic standard errors for impulse–response functions and other postestimation statistics, Stata needs the complete set of parameter estimates, including those that are constrained to be zero; `var` stores them in `e(bf)`. Because you can specify models for which the full set of parameter estimates exceeds Stata’s limit on the size of matrices, the `nobigf` option specifies that `var` not compute and store `e(bf)`. This means that the asymptotic standard errors of the postestimation functions cannot be obtained, although bootstrap standard errors are still available. Building `e(bf)` can be time consuming, so if you do not need this full matrix, and speed is an issue, use `nobigf`.

Fitting models with exogenous variables

► Example 2: VAR model with exogenous variables

We use the `exog()` option to include exogenous variables in a VAR.

```
. var dln_inc dln_consump if qtr<=tq(1978q4), dfk exog(dln_inv)
Vector autoregression
Sample: 1960q4 - 1978q4                      No. of obs      =          73
Log likelihood = 478.5663                     AIC              = -12.78264
FPE           = 9.64e-09                      HQIC            = -12.63259
Det(Sigma_ml) = 6.93e-09                      SBIC           = -12.40612

Equation      Parms      RMSE      R-sq      chi2      P>chi2
-----
dln_inc       6          .011917  0.0702   5.059587  0.4087
dln_consump   6          .009197  0.2794  25.97262  0.0001
```

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
dln_inc						
dln_inc						
L1.	-.1343345	.1391074	-0.97	0.334	-.4069801	.1383111
L2.	.0120331	.1380346	0.09	0.931	-.2585097	.2825759
dln_consump						
L1.	.3235342	.1652769	1.96	0.050	-.0004027	.647471
L2.	.0754177	.1648624	0.46	0.647	-.2477066	.398542
dln_inv	.0151546	.0302319	0.50	0.616	-.0440987	.074408
_cons	.0145136	.0043815	3.31	0.001	.0059259	.0231012
dln_consump						
dln_inc						
L1.	.2425719	.1073561	2.26	0.024	.0321578	.452986
L2.	.3487949	.1065281	3.27	0.001	.1400036	.5575862
dln_consump						
L1.	-.3119629	.1275524	-2.45	0.014	-.5619611	-.0619648
L2.	-.0128502	.1272325	-0.10	0.920	-.2622213	.2365209
dln_inv	.0503616	.0233314	2.16	0.031	.0046329	.0960904
_cons	.0131013	.0033814	3.87	0.000	.0064738	.0197288

All the postestimation commands for analyzing VARs work when exogenous variables are included in a model, but the asymptotic standard errors for the h -step-ahead forecasts are not available.



Fitting models with constraints on the coefficients

`var` permits model specifications that include constraints on the coefficient, though `var` does not allow for constraints on Σ . See [TS] [var intro](#) and [TS] [var svar](#) for ways to constrain Σ .

► Example 3: VAR model with constraints

In the first example, we fit a full VAR(2) to a three-equation model. The coefficients in the equation for `dln_inv` were jointly insignificant, as were the coefficients in the equation for `dln_inc`; and many individual coefficients were not significantly different from zero. In this example, we constrain the coefficient on `L2.dln_inc` in the equation for `dln_inv` and the coefficient on `L2.dln_consump` in the equation for `dln_inc` to be zero.

```
. constraint 1 [dln_inv]L2.dln_inc = 0
. constraint 2 [dln_inc]L2.dln_consump = 0
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lutstats dfk
> constraints(1 2)
Estimating VAR coefficients
Iteration 1:    tolerance = .00737681
Iteration 2:    tolerance = 3.998e-06
Iteration 3:    tolerance = 2.730e-09

Vector autoregression
Sample: 1960q4 - 1978q4
Log likelihood = 606.2804          (lutstats) AIC          = -31.69254
FPE            = 1.77e-14          HQIC           = -31.46747
Det(Sigma_ml)  = 1.05e-14          SBIC          = -31.12777

Equation      Parms    RMSE    R-sq    chi2    P>chi2
-----
dln_inv       6       .043895  0.1280  9.842338  0.0798
dln_inc       6       .011143  0.1141  8.584446  0.1268
dln_consump   7       .008981  0.2512  22.86958  0.0008

( 1) [dln_inv]L2.dln_inc = 0
( 2) [dln_inc]L2.dln_consump = 0
```

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
dln_inv						
dln_inv						
L1.	-.320713	.1247512	-2.57	0.010	-.5652208	-.0762051
L2.	-.1607084	.124261	-1.29	0.196	-.4042555	.0828386
dln_inc						
L1.	.1195448	.5295669	0.23	0.821	-.9183873	1.157477
L2.	5.66e-19	9.33e-18	0.06	0.952	-1.77e-17	1.89e-17
dln_consump						
L1.	1.009281	.623501	1.62	0.106	-.2127586	2.231321
L2.	1.008079	.5713486	1.76	0.078	-.1117438	2.127902
_cons	-.0162102	.016893	-0.96	0.337	-.0493199	.0168995

(Continued on next page)

dln_inc							
dln_inv							
L1.	.0435712	.0309078	1.41	0.159	-.017007	.1041495	
L2.	.0496788	.0306455	1.62	0.105	-.0103852	.1097428	
dln_inc							
L1.	-.1555119	.1315854	-1.18	0.237	-.4134146	.1023908	
L2.	.0122353	.1165811	0.10	0.916	-.2162595	.2407301	
dln_consump							
L1.	.29286	.1568345	1.87	0.062	-.01453	.6002501	
L2.	-1.53e-18	1.89e-17	-0.08	0.935	-3.85e-17	3.55e-17	
_cons	.015689	.003819	4.11	0.000	.0082039	.0231741	
dln_consump							
dln_inv							
L1.	-.0026229	.0253538	-0.10	0.918	-.0523154	.0470696	
L2.	.0337245	.0252113	1.34	0.181	-.0156888	.0831378	
dln_inc							
L1.	.2224798	.1094349	2.03	0.042	.0079912	.4369683	
L2.	.3469758	.1006026	3.45	0.001	.1497984	.5441532	
dln_consump							
L1.	-.2600227	.1321622	-1.97	0.049	-.519056	-.0009895	
L2.	-.0146825	.1117618	-0.13	0.895	-.2337315	.2043666	
_cons	.0129149	.003376	3.83	0.000	.0062981	.0195317	

None of the free parameter estimates changed by much. Whereas the coefficients in the equation `dln_inv` are now significant at the 10% level, the coefficients in the equation for `dln_inc` remain jointly insignificant.

Saved results

`var` saves the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(N_gaps)</code>	number of gaps in sample
<code>e(k)</code>	number of parameters in all equations
<code>e(k_eq)</code>	number of equations in <code>e(b)</code>
<code>e(k_dv)</code>	number of dependent variables
<code>e(df_eq)</code>	average number of parameters in an equation
<code>e(df_m)</code>	model degrees of freedom
<code>e(df_r)</code>	residual degrees of freedom (small only)
<code>e(ll)</code>	log likelihood
<code>e(ll_dfk)</code>	dfk adjusted log likelihood (dfk only)
<code>e(obs_#)</code>	number of observations on equation #
<code>e(k_#)</code>	number of parameters in equation #
<code>e(df_m#)</code>	model degrees of freedom for equation #
<code>e(df_r#)</code>	residual degrees of freedom for equation # (small only)
<code>e(r2_#)</code>	R -squared for equation #
<code>e(ll_#)</code>	log likelihood for equation #
<code>e(chi2_#)</code>	χ^2 for equation #
<code>e(F_#)</code>	F statistic for equation # (small only)
<code>e(rmse_#)</code>	root mean squared error for equation #
<code>e(aic)</code>	Akaike information criterion
<code>e(hqic)</code>	Hannan–Quinn information criterion
<code>e(sbic)</code>	Schwarz–Bayesian information criterion
<code>e(fpe)</code>	final prediction error
<code>e(mlag)</code>	highest lag in VAR
<code>e(tmin)</code>	first time period in sample
<code>e(tmax)</code>	maximum time
<code>e(detsig)</code>	determinant of <code>e(Sigma)</code>
<code>e(detsig_ml)</code>	determinant of $\hat{\Sigma}_{ml}$
<code>e(rank)</code>	rank of <code>e(V)</code>

(Continued on next page)

Macros

<code>e(cmd)</code>	<code>var</code>
<code>e(cmdline)</code>	command as typed
<code>e(depvar)</code>	names of dependent variables
<code>e(endog)</code>	names of endogenous variables, if specified
<code>e(exog)</code>	names of exogenous variables, and their lags, if specified
<code>e(exogvars)</code>	names of exogenous variables, if specified
<code>e(eqnames)</code>	names of equations
<code>e(lags)</code>	lags in model
<code>e(exlags)</code>	lags of exogenous variables in model, if specified
<code>e(title)</code>	title in estimation output
<code>e(constraints)</code>	constraints, if specified
<code>e(cnslist_var)</code>	list of specified constraints
<code>e(small)</code>	small, if specified
<code>e(lutstats)</code>	lutstats, if specified
<code>e(timevar)</code>	time variable specified in <code>tsset</code>
<code>e(tsfmt)</code>	format for the current time variable
<code>e(dfk)</code>	dfk, if specified
<code>e(properties)</code>	b V
<code>e(predict)</code>	program used to implement <code>predict</code>
<code>e(marginsok)</code>	predictions allowed by <code>margins</code>
<code>e(marginsnotok)</code>	predictions disallowed by <code>margins</code>

Matrices

<code>e(b)</code>	coefficient vector
<code>e(Cns)</code>	constraints matrix
<code>e(Sigma)</code>	Σ matrix
<code>e(V)</code>	variance–covariance matrix of the estimators
<code>e(bf)</code>	constrained coefficient vector
<code>e(exlagsm)</code>	matrix mapping lags to exogenous variables
<code>e(G)</code>	Gamma matrix; see Methods and formulas

Functions

<code>e(sample)</code>	marks estimation sample

Methods and formulas

`var` is implemented as an ado-file.

When there are no constraints placed on the coefficients, the $\text{VAR}(p)$ is a seemingly unrelated regression model with the same explanatory variables in each equation. As discussed in [Lütkepohl \(2005\)](#) and [Greene \(2008\)](#), performing linear regression on each equation produces the maximum likelihood estimates of the coefficients. The estimated coefficients can then be used to calculate the residuals, which in turn are used to estimate the cross-equation error variance–covariance matrix Σ .

Per [Lütkepohl \(2005\)](#), we write the $\text{VAR}(p)$ with exogenous variables as

$$\mathbf{y}_t = \mathbf{A}\mathbf{Y}_{t-1} + \mathbf{B}_0\mathbf{x}_t + \mathbf{u}_t \tag{5}$$

where

$\mathbf{y}_t$ is the $K \times 1$ vector of endogenous variables,

$\mathbf{A}$ is a $K \times Kp$ matrix of coefficients,

$\mathbf{B}_0$ is a $K \times M$ matrix of coefficients,

$\mathbf{x}_t$ is the $M \times 1$ vector of exogenous variables,

$\mathbf{u}_t$ is the $K \times 1$ vector of white noise innovations, and

$\mathbf{Y}_t$ is the $Kp \times 1$ matrix given by $\mathbf{Y}_t = \begin{pmatrix} \mathbf{y}_t \\ \vdots \\ \mathbf{y}_{t-p+1} \end{pmatrix}$

Although (5) is easier to read, the formulas are much easier to manipulate if it is instead written as

$$\mathbf{Y} = \mathbf{B}\mathbf{Z} + \mathbf{U}$$

where

$$\begin{aligned} \mathbf{Y} &= (\mathbf{y}_1, \dots, \mathbf{y}_T) & \mathbf{Y} \text{ is } K \times T \\ \mathbf{B} &= (\mathbf{A}, \mathbf{B}_0) & \mathbf{B} \text{ is } K \times (Kp + M) \\ \mathbf{Z} &= \begin{pmatrix} \mathbf{Y}_0 \dots, \mathbf{Y}_{T-1} \\ \mathbf{x}_1 \dots, \mathbf{x}_T \end{pmatrix} & \mathbf{Z} \text{ is } (Kp + M) \times T \\ \mathbf{U} &= (\mathbf{u}_1, \dots, \mathbf{u}_T) & \mathbf{U} \text{ is } K \times T \end{aligned}$$

Intercept terms in the model are included in $\mathbf{x}_t$. If there are no exogenous variables and no intercept terms in the model, $\mathbf{x}_t$ is empty.

The coefficients are estimated by iterated seemingly unrelated regression. Because the estimation is actually performed by `reg3`, the methods are documented in [R] [reg3](#). See [P] [makecns](#) for more on estimation with constraints.

Let $\hat{\mathbf{U}}$ be the matrix of residuals that are obtained via $\mathbf{Y} - \hat{\mathbf{B}}\mathbf{Z}$, where $\hat{\mathbf{B}}$ is the matrix of estimated coefficients. Then the estimator of Σ is

$$\hat{\Sigma} = \frac{1}{\tilde{T}} \hat{\mathbf{U}}' \hat{\mathbf{U}}$$

By default, the maximum likelihood divisor of $\tilde{T} = T$ is used. When `dfk` is specified, a small-sample degrees-of-freedom adjustment is used; then, $\tilde{T} = T - \overline{m}$ where $\overline{m}$ is the average number of parameters per equation in the functional form for $\mathbf{y}_t$ over the K equations.

`small` specifies that Wald tests after `var` be assumed to have F or t distributions instead of chi-squared or standard normal distributions. The standard errors from each equation are computed using the degrees of freedom for the equation.

The “gamma” matrix saved in `e(g)` referred to in *Saved results* is the $(Kp + 1) \times (Kp + 1)$ matrix given by

$$\frac{1}{\tilde{T}} \sum_{t=1}^T (1, \mathbf{Y}_t') (1, \mathbf{Y}_t')$$

The formulas for the lag-order selection criteria and the log likelihood are discussed in [TS] [varsoc](#).

Acknowledgment

We thank Christopher F. Baum of Boston College for his helpful comments.

References

- Greene, W. H. 2008. *Econometric Analysis*. 6th ed. Upper Saddle River, NJ: Prentice–Hall.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Stock, J. H., and M. W. Watson. 2001. Vector autoregressions. *Journal of Economic Perspectives* 15: 101–115.
- Watson, M. W. 1994. Vector autoregressions and cointegration. In Vol. IV of *Handbook of Econometrics*, ed. R. F. Engle and D. L. McFadden. Amsterdam: Elsevier.

Also see

- [TS] **var postestimation** — Postestimation tools for var
- [TS] **tsset** — Declare data to be time-series data
- [TS] **dfactor** — Dynamic-factor models
- [TS] **dvech** — Diagonal vech multivariate GARCH models
- [TS] **sspace** — State-space models
- [TS] **var svar** — Structural vector autoregressive models
- [TS] **varbasic** — Fit a simple VAR and graph IRFS or FEVDs
- [TS] **vec** — Vector error-correction models
- [U] **20 Estimation and postestimation commands**
- [TS] **var intro** — Introduction to vector autoregressive models

Description

The following postestimation commands are of special interest after `var`:

command	description
<code>fcast compute</code>	obtain dynamic forecasts
<code>fcast graph</code>	graph dynamic forecasts obtained from <code>fcast compute</code>
<code>irf</code>	create and analyze IRFs and FEVDs
<code>vargranger</code>	Granger causality tests
<code>varlmar</code>	LM test for autocorrelation in residuals
<code>varnorm</code>	test for normally distributed residuals
<code>varsoc</code>	lag-order selection criteria
<code>varstable</code>	check stability condition of estimates
<code>varwle</code>	Wald lag-exclusion statistics

For information about these commands, see the corresponding entries in this manual.

The following standard postestimation commands are also available:

command	description
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	point estimates, standard errors, testing, and inference for generalized predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

`predict` [*type*] *newvar* [*if*] [*in*] [, e*quation*(*eqno*|*eqname*) *statistic*]

<i>statistic</i>	description
<code>xb</code>	linear prediction; the default
<code>stdp</code>	standard error of the linear prediction
<code>residuals</code>	residuals

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

`equation(eqno | eqname)` specifies the equation to which you are referring.

`equation()` is filled in with one *eqno* or *eqname* for options `xb`, `stdp`, and `residuals`. For example, `equation(#1)` would mean that the calculation is to be made for the first equation, `equation(#2)` would mean the second, and so on. You could also refer to the equation by its name; thus, `equation(income)` would refer to the equation named `income` and `equation(hours)`, to the equation named `hours`.

If you do not specify `equation()`, the results are the same as if you specified `equation(#1)`.

`xb`, the default, calculates the linear prediction for the specified equation.

`stdp` calculates the standard error of the linear prediction for the specified equation.

`residuals` calculates the residuals.

For more information on using `predict` after multiple-equation estimation commands, see [\[R\] predict](#).

Remarks

Remarks are presented under the following headings:

Model selection and inference
Forecasting

Model selection and inference

See the following sections for information on model selection and inference after `var`.

[\[TS\] irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[\[TS\] vargranger](#) — Perform pairwise Granger causality tests after `var` or `svar`

[\[TS\] varlmar](#) — Perform LM test for residual autocorrelation after `var` or `svar`

[\[TS\] varnorm](#) — Test for normally distributed disturbances after `var` or `svar`

[\[TS\] varsoc](#) — Obtain lag-order selection statistics for VARs and VECMs

[\[TS\] varstable](#) — Check the stability condition of VAR or SVAR estimates

[\[TS\] varwle](#) — Obtain Wald lag-exclusion statistics after `var` or `svar`

Forecasting

Two types of forecasts are available after you fit a $\text{VAR}(p)$: a one-step-ahead forecast and a dynamic h -step-ahead forecast.

The one-step-ahead forecast produces a prediction of the value of an endogenous variable in the current period by using the estimated coefficients, the past values of the endogenous variables, and any exogenous variables. If you include contemporaneous values of exogenous variables in your model, you must have observations on the exogenous variables that are contemporaneous with the period in which the prediction is being made to compute the prediction. In Stata terms, these one-step-ahead predictions are just the standard linear predictions available after any estimation command. Thus `predict, xb eq(eqno | eqname)` produces one-step-ahead forecasts for the specified equation. `predict, stdp eq(eqno | eqname)` produces the standard error of the linear prediction for the specified equation. The standard error of the forecast includes an estimate of the variability due to innovations, whereas the standard error of the linear prediction does not.

The dynamic h -step-ahead forecast begins by using the estimated coefficients, the lagged values of the endogenous variables, and any exogenous variables to predict one step ahead for each endogenous variable. Then the one-step-ahead forecast produces two-step-ahead forecasts for each endogenous variable. The process continues for h periods. Because each step uses the predictions of the previous steps, these forecasts are known as dynamic forecasts. See the following sections for information on obtaining forecasts after `svar`:

[TS] [fcast compute](#) — Compute dynamic forecasts of dependent variables after `var`, `svar`, or `vec`

[TS] [fcast graph](#) — Graph forecasts of variables computed by `fcast compute`

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Formulas for predict

`predict` with the `xb` option provides the one-step-ahead forecast. If exogenous variables are specified, the forecast is conditional on the exogenous $\mathbf{x}_t$ variables. Specifying the `residuals` option causes `predict` to calculate the errors of the one-step-ahead forecasts. Specifying the `stdp` option causes `predict` to calculate the standard errors of the one-step-ahead forecasts.

Also see

[TS] [var](#) — Vector autoregressive models

[U] [20 Estimation and postestimation commands](#)

Syntax

Short-run constraints

```
svar depvarlist [if] [in], { aconstraints(constraintsa) aeq(matrixaeq)  
      acns(matrixacns) bconstraints(constraintsb) beq(matrixbeq) bcns(matrixbcns) }  
      [short_run_options]
```

Long-run constraints

```
svar depvarlist [if] [in], { lrconstraints(constraintslr) lreq(matrixlreq)  
      lrcns(matrixlrcns) } [long_run_options]
```

<i>short_run_options</i>	description
Model	
<u>noconstant</u>	suppress constant term
* <u>aconstraints</u> (<i>constraints_a</i>)	apply previously defined <i>constraints_a</i> to A
* <u>aeq</u> (<i>matrix_{aeq}</i>)	define and apply to A equality constraint matrix <i>matrix_{aeq}</i>
* <u>acns</u> (<i>matrix_{acns}</i>)	define and apply to A cross-parameter constraint matrix <i>matrix_{acns}</i>
* <u>bconstraints</u> (<i>constraints_b</i>)	apply previously defined <i>constraints_b</i> to B
* <u>beq</u> (<i>matrix_{beq}</i>)	define and apply to B equality constraint matrix <i>matrix_{beq}</i>
* <u>bcns</u> (<i>matrix_{bcns}</i>)	define and apply to B cross-parameter constraint matrix <i>matrix_{bcns}</i>
<u>lags</u> (<i>numlist</i>)	use lags <i>numlist</i> in the underlying VAR
Model 2	
<u>exog</u> (<i>varlist_{exog}</i>)	use exogenous variables <i>varlist</i>
<u>varconstraints</u> (<i>constraints_v</i>)	apply <i>constraints_v</i> to underlying VAR
<u>noislog</u>	suppress SURE iteration log
<u>isiterate</u> (#)	set maximum number of iterations for SURE; default is <code>isiterate(1600)</code>
<u>istolerance</u> (#)	set convergence tolerance of SURE
<u>noisure</u>	use one-step SURE
<u>dfk</u>	make small-sample degrees-of-freedom adjustment
<u>small</u>	report small-sample <i>t</i> and <i>F</i> statistics
<u>noidencheck</u>	do not check for local identification
<u>nobigf</u>	do not compute parameter vector for coefficients implicitly set to zero
Reporting	
<u>level</u> (#)	set confidence level; default is <code>level(95)</code>
<u>full</u>	show constrained parameters in table
<u>var</u>	display underlying var output
<u>lutstats</u>	report Lütkepohl lag-order selection statistics
<u>nocnsreport</u>	do not display constraints
Maximization	
<i>maximize_options</i>	control the maximization process; seldom used
† <u>coeflegend</u>	display coefficients' legend instead of coefficient table

* aconstraints(*constraints_a*), aeq(*matrix_{aeq}*), acns(*matrix_{acns}*), bconstraints(*constraints_b*), beq(*matrix_{beq}*), bcns(*matrix_{bcns}*): at least one of these options must be specified.

† coeflegend does not appear in the dialog box.

<i>long_run_options</i>	description
Model	
<u>noconstant</u>	suppress constant term
* <u>lrconstraints</u> (<i>constraints</i> _{lr})	apply previously defined <i>constraints</i> _{lr} to C
* <u>lreq</u> (<i>matrix</i> _{lreq})	define and apply to C equality constraint matrix <i>matrix</i> _{lreq}
* <u>lrcns</u> (<i>matrix</i> _{lrcns})	define and apply to C cross-parameter constraint matrix <i>matrix</i> _{lrcns}
<u>lags</u> (<i>numlist</i>)	use lags <i>numlist</i> in the underlying VAR
Model 2	
<u>exog</u> (<i>varlist</i> _{exog})	use exogenous variables <i>varlist</i>
<u>varconstraints</u> (<i>constraints</i> _v)	apply <i>constraints</i> _v to underlying VAR
<u>noislog</u>	suppress SURE iteration log
<u>isiterate</u> (#)	set maximum number of iterations for SURE; default is <code>isiterate(1600)</code>
<u>istolerance</u> (#)	set convergence tolerance of SURE
<u>noisure</u>	use one-step SURE
<u>dfk</u>	make small-sample degrees-of-freedom adjustment
<u>small</u>	report small-sample <i>t</i> and <i>F</i> statistics
<u>noidencheck</u>	do not check for local identification
<u>nobigf</u>	do not compute parameter vector for coefficients implicitly set to zero
Reporting	
<u>level</u> (#)	set confidence level; default is <code>level(95)</code>
<u>full</u>	show constrained parameters in table
<u>var</u>	display underlying var output
<u>lutstats</u>	report Lütkepohl lag-order selection statistics
<u>nocnsreport</u>	do not display constraints
Maximization	
<i>maximize_options</i>	control the maximization process; seldom used
† <u>coeflegend</u>	display coefficients' legend instead of coefficient table
* <code>lrconstraints(<i>constraints</i>_{lr})</code> , <code>lreq(<i>matrix</i>_{lreq})</code> , <code>lrcns(<i>matrix</i>_{lrcns})</code> : at least one of these options must be specified.	
† <code>coeflegend</code> does not appear in the dialog box.	

You must `tsset` your data before using `svar`; see [TS] [tsset](#).
depvarlist and *varlist*_{exog} may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).
`by`, `rolling`, `statsby`, and `xi` are allowed; see [U] [11.1.10 Prefix commands](#).
 See [U] [20 Estimation and postestimation commands](#) for more capabilities of estimation commands.

Menu

Statistics > Multivariate time series > Structural vector autoregression (SVAR)

Description

`svar` fits a vector autoregressive model subject to short- or long-run constraints you place on the resulting impulse–response functions (IRFs). Economic theory typically motivates the constraints, allowing a causal interpretation of the IRFs to be made. See [TS] [var intro](#) for a list of commands that are used in conjunction with `svar`.

Options

Model

`noconstant`; see [R] [estimation options](#).

`aconstraints(constraintsa)`, `aeq(matrixaeq)`, `acns(matrixacns)`

`bconstraints(constraintsb)`, `beq(matrixbeq)`, `bcns(matrixbcns)`

These options specify the short-run constraints in an SVAR. To specify a short-run SVAR model, you must specify at least one of these options. The first list of options specifies constraints on the parameters of the **A** matrix; the second list specifies constraints on the parameters of the **B** matrix (see [Short-run SVAR models](#)). If at least one option is selected from the first list and none are selected from the second list, `svar` sets **B** to the identity matrix. Similarly, if at least one option is selected from the second list and none are selected from the first list, `svar` sets **A** to the identity matrix.

None of these options may be specified with any of the options that define long-run constraints.

`aconstraints(constraintsa)` specifies a *numlist* of previously defined Stata constraints that are to be applied to **A** during estimation.

`aeq(matrixaeq)` specifies a matrix that defines a set of equality constraints. This matrix must be square with dimension equal to the number of equations in the underlying VAR. The elements of this matrix must be *missing* or real numbers. A missing value in the (i, j) element of this matrix specifies that the (i, j) element of **A** is a free parameter. A real number in the (i, j) element of this matrix constrains the (i, j) element of **A** to this real number. For example,

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ . & 1.5 \end{bmatrix}$$

specifies that $\mathbf{A}[1, 1] = 1$, $\mathbf{A}[1, 2] = 0$, $\mathbf{A}[2, 2] = 1.5$, and $\mathbf{A}[2, 1]$ is a free parameter.

(Continued on next page)

`acns(matrixacns)` specifies a matrix that defines a set of exclusion or cross-parameter equality constraints on **A**. This matrix must be square with dimension equal to the number of equations in the underlying VAR. Each element of this matrix must be *missing*, 0, or a positive integer. A missing value in the (i, j) element of this matrix specifies that no constraint be placed on this element of **A**. A zero in the (i, j) element of this matrix constrains the (i, j) element of **A** to be zero. Any strictly positive integers must be in two or more elements of this matrix. A strictly positive integer in the (i, j) element of this matrix constrains the (i, j) element of **A** to be equal to all the other elements of **A** that correspond to elements in this matrix that contain the same integer. For example, consider the matrix

$$\mathbf{A} = \begin{bmatrix} \cdot & 1 \\ 1 & 0 \end{bmatrix}$$

Specifying `acns(A)` in a two-equation SVAR constrains $\mathbf{A}[2, 1] = \mathbf{A}[1, 2]$ and $\mathbf{A}[2, 2] = 0$ while leaving $\mathbf{A}[1, 1]$ free.

`bconstraints(constraintsb)` specifies a *numlist* of previously defined Stata constraints to be applied to **B** during estimation.

`beq(matrixbeq)` specifies a matrix that defines a set of equality constraints. This matrix must be square with dimension equal to the number of equations in the underlying VAR. The elements of this matrix must be either *missing* or real numbers. The syntax of implied constraints is analogous to the one described in `aeq()`, except that it applies to **B** rather than to **A**.

`bcns(matrixbcns)` specifies a matrix that defines a set of exclusion or cross-parameter equality constraints on **B**. This matrix must be square with dimension equal to the number of equations in the underlying VAR. Each element of this matrix must be *missing*, 0, or a positive integer. The format of the implied constraints is the same as the one described in the `acns()` option above.

`lrconstraints(constraintslr)`, `lreq(matrixlreq)`, `lrcns(matrixlrcns)`

These options specify the long-run constraints in an SVAR. To specify a long-run SVAR model, you must specify at least one of these options. The list of options specifies constraints on the parameters of the long-run **C** matrix (see [Long-run SVAR models](#) for the definition of **C**). None of these options may be specified with any of the options that define short-run constraints.

`lrconstraints(constraintslr)` specifies a *numlist* of previously defined Stata constraints to be applied to **C** during estimation.

`lreq(matrixlreq)` specifies a matrix that defines a set of equality constraints on the elements of **C**. This matrix must be square with dimension equal to the number of equations in the underlying VAR. The elements of this matrix must be either *missing* or real numbers. The syntax of implied constraints is analogous to the one described in option `aeq()`, except that it applies to **C**.

`lrcns(matrixlrcns)` specifies a matrix that defines a set of exclusion or cross-parameter equality constraints on **C**. This matrix must be square with dimension equal to the number of equations in the underlying VAR. Each element of this matrix must be *missing*, 0, or a positive integer. The syntax of the implied constraints is the same as the one described for the `acns()` option above.

`lags(numlist)` specifies the lags to be included in the underlying VAR model. The default is `lags(1 2)`. This option takes a *numlist* and not simply an integer for the maximum lag. For instance, `lags(2)` would include only the second lag in the model, whereas `lags(1/2)` would include both the first and second lags in the model. See [\[U\] 11.1.8 numlist](#) and [\[U\] 11.4.4 Time-series varlists](#) for further discussion of *numlists* and lags.

Model 2

- `exog(varlistexog)` specifies a list of exogenous variables to be included in the underlying VAR.
- `varconstraints(constraintsv)` specifies a list of constraints to be applied to coefficients in the underlying VAR. Because `svr` estimates multiple equations, the constraints must specify the equation name for all but the first equation.
- `noislog` prevents `svr` from displaying the iteration log from the iterated seemingly unrelated regression algorithm. When the `varconstraints()` option is not specified, the VAR coefficients are estimated via OLS, a noniterative procedure. As a result, `noislog` may be specified only with `varconstraints()`. Similarly, `noislog` may not be combined with `noisure`.
- `isiterate(#)` sets the maximum number of iterations for the iterated, seemingly unrelated regression algorithm. The default limit is 1,600. When the `varconstraints()` option is not specified, the VAR coefficients are estimated via OLS, a noniterative procedure. As a result, `isiterate()` may be specified only with `varconstraints()`. Similarly, `isiterate()` may not be combined with `noisure`.
- `istolerance(#)` specifies the convergence tolerance of the iterated seemingly unrelated regression algorithm. The default tolerance is $1\text{e-}6$. When the `varconstraints()` option is not specified, the VAR coefficients are estimated via OLS, a noniterative procedure. As a result, `istolerance()` may be specified only with `varconstraints()`. Similarly, `istolerance()` may not be combined with `noisure`.
- `noisure` specifies that the VAR coefficients be estimated via one-step seemingly unrelated regression when `varconstraints()` is specified. By default, `svr` estimates the coefficients in the VAR via iterated, seemingly unrelated regression when `varconstraints()` is specified. When the `varconstraints()` option is not specified, the VAR coefficient estimates are obtained via OLS, a noniterative procedure. As a result, `noisure` may be specified only with `varconstraints()`.
- `dfk` specifies that a small-sample degrees-of-freedom adjustment be used when estimating Σ , the covariance matrix of the VAR disturbances. Specifically, $1/(T - \overline{m})$ is used instead of the large-sample divisor $1/T$, where $\overline{m}$ is the average number of parameters in the functional form for $\mathbf{y}_t$ over the K equations.
- `small` causes `svr` to calculate and report small-sample t and F statistics instead of the large-sample normal and chi-squared statistics.
- `noidencheck` requests that the [Amisano and Giannini \(1997\)](#) check for local identification not be performed. This check is local to the starting values used. Because of this dependence on the starting values, you may wish to suppress this check by specifying the `noidencheck` option. However, be careful in specifying this option. Models that are not structurally identified can still converge, thereby producing meaningless results that only appear to have meaning.
- `nobigf` requests that `svr` not save the estimated parameter vector that incorporates coefficients that have been implicitly constrained to be zero, such as when some lags have been omitted from a model. `e(bf)` is used for computing asymptotic standard errors in the postestimation commands `irf create` and `fcast compute`. Therefore, specifying `nobigf` implies that the asymptotic standard errors will not be available from `irf create` and `fcast compute`. See [Fitting models with some lags excluded](#) in [TS] `var`.

Reporting

`level(#)`; see [R] [estimation options](#).

`full` shows constrained parameters in table.

`var` specifies that the output from `var` also be displayed. By default, the underlying VAR is fit quietly.

`lutstats` specifies that the Lütkepohl versions of the lag-order selection statistics be reported. See [Methods and formulas](#) in [TS] `varsoc` for a discussion of these statistics.

`nocnsreport`; see [R] [estimation options](#).

Maximization

`maximize_options`: `difficult`, `technique(algorithm_spec)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, `nonrtolerance`, `from(init_specs)`; see [R] [maximize](#). These options are seldom used.

The following option is available with `svar` but is not shown in the dialog box:

`coeflegend`; see [R] [estimation options](#).

Remarks

Remarks are presented under the following headings:

[Introduction](#)

[Short-run SVAR models](#)

[Long-run SVAR models](#)

Introduction

This entry assumes that you have already read [TS] [var intro](#) and [TS] [var](#); if not, please do. Here we illustrate how to fit SVARs in Stata subject to short-run and long-run restrictions. For more detailed information on SVARs, see [Amisano and Giannini \(1997\)](#) and [Hamilton \(1994\)](#). For good introductions to VARs, see [Lütkepohl \(2005\)](#), [Hamilton \(1994\)](#), and [Stock and Watson \(2001\)](#).

Short-run SVAR models

A short-run SVAR model without exogenous variables can be written as

$$\mathbf{A}(\mathbf{I}_K - \mathbf{A}_1 L - \mathbf{A}_2 L^2 - \cdots - \mathbf{A}_p L^p) \mathbf{y}_t = \mathbf{A} \boldsymbol{\epsilon}_t = \mathbf{B} \mathbf{e}_t$$

where L is the lag operator, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{A}_1, \dots, \mathbf{A}_p$ are $K \times K$ matrices of parameters, $\boldsymbol{\epsilon}_t$ is a $K \times 1$ vector of innovations with $\boldsymbol{\epsilon}_t \sim N(\mathbf{0}, \boldsymbol{\Sigma})$ and $E[\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_s'] = \mathbf{0}_K$ for all $s \neq t$, and $\mathbf{e}_t$ is a $K \times 1$ vector of orthogonalized disturbances; i.e., $\mathbf{e}_t \sim N(\mathbf{0}, \mathbf{I}_K)$ and $E[\mathbf{e}_t \mathbf{e}_s'] = \mathbf{0}_K$ for all $s \neq t$. These transformations of the innovations allow us to analyze the dynamics of the system in terms of a change to an element of $\mathbf{e}_t$. In a short-run SVAR model, we obtain identification by placing restrictions on $\mathbf{A}$ and $\mathbf{B}$, which are assumed to be nonsingular.

► Example 1: Short-run just-identified SVAR model

Following [Sims \(1980\)](#), the Cholesky decomposition is one method of identifying the impulse-response functions in a VAR; thus, this method corresponds to an SVAR. There are several sets of constraints on $\mathbf{A}$ and $\mathbf{B}$ that are easily manipulated back to the Cholesky decomposition, and the following example illustrates this point.

One way to impose the Cholesky restrictions is to assume an SVAR model of the form

$$\tilde{\mathbf{A}}(\mathbf{I}_K - \mathbf{A}_1 - \mathbf{A}_2 L^2 - \cdots \mathbf{A}_p L^p) \mathbf{y}_t = \tilde{\mathbf{B}} \mathbf{e}_t$$

where $\tilde{\mathbf{A}}$ is a lower triangular matrix with ones on the diagonal and $\tilde{\mathbf{B}}$ is a diagonal matrix. Because the $\mathbf{P}$ matrix for this model is $\mathbf{P}_{\text{sr}} = \tilde{\mathbf{A}}^{-1} \tilde{\mathbf{B}}$, its estimate, $\hat{\mathbf{P}}_{\text{sr}}$, obtained by plugging in estimates of $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$, should equal the Cholesky decomposition of $\hat{\mathbf{\Sigma}}$.

To illustrate, we use the German macroeconomic data discussed in [Lütkepohl \(2005\)](#) and used in [\[TS\] var](#). In this example, $\mathbf{y}_t = (\text{dln\_inv}, \text{dln\_inc}, \text{dln\_consump})$, where `dln_inv` is the first difference of the log of investment, `dln_inc` is the first difference of the log of income, and `dln_consump` is the first difference of the log of consumption. Because the first difference of the natural log of a variable can be treated as an approximation of the percent change in that variable, we will refer to these variables as percent changes in `inv`, `inc`, and `consump`, respectively.

We will impose the Cholesky restrictions on this system by applying equality constraints with the constraint matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ . & 1 & 0 \\ . & . & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} . & 0 & 0 \\ 0 & . & 0 \\ 0 & 0 & . \end{bmatrix}$$

With these structural restrictions, we assume that the percent change in `inv` is not contemporaneously affected by the percent changes in either `inc` or `consump`. We also assume that the percent change of `inc` is affected by contemporaneous changes in `inv` but not `consump`. Finally, we assume that percent changes in `consump` are affected by contemporaneous changes in both `inv` and `inc`.

The following commands fit an SVAR model with these constraints.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. matrix A = (1,0,0\.,1,0\.,.,1)
. matrix B = (.,0,0\0,.,0\0,0,.)
```

(Continued on next page)

```
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), aeq(A) beq(B)
Estimating short-run parameters
(output omitted)
Structural vector autoregression
( 1) [a_1_1]_cons = 1
( 2) [a_1_2]_cons = 0
( 3) [a_1_3]_cons = 0
( 4) [a_2_2]_cons = 1
( 5) [a_2_3]_cons = 0
( 6) [a_3_3]_cons = 1
( 7) [b_1_2]_cons = 0
( 8) [b_1_3]_cons = 0
( 9) [b_2_1]_cons = 0
(10) [b_2_3]_cons = 0
(11) [b_3_1]_cons = 0
(12) [b_3_2]_cons = 0

Sample: 1960q4 - 1978q4                      No. of obs      =          73
Exactly identified model                      Log likelihood =    606.307
```

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
/a_1_1	1	.	.	.	.	.
/a_2_1	-.0336288	.0294605	-1.14	0.254	-.0913702	.0241126
/a_3_1	-.0435846	.0194408	-2.24	0.025	-.0816879	-.0054812
/a_1_2	0	.	.	.	.	.
/a_2_2	1	.	.	.	.	.
/a_3_2	-.424774	.0765548	-5.55	0.000	-.5748187	-.2747293
/a_1_3	0	.	.	.	.	.
/a_2_3	0	.	.	.	.	.
/a_3_3	1	.	.	.	.	.
/b_1_1	.0438796	.0036315	12.08	0.000	.036762	.0509972
/b_2_1	0	.	.	.	.	.
/b_3_1	0	.	.	.	.	.
/b_1_2	0	.	.	.	.	.
/b_2_2	.0110449	.0009141	12.08	0.000	.0092534	.0128365
/b_3_2	0	.	.	.	.	.
/b_1_3	0	.	.	.	.	.
/b_2_3	0	.	.	.	.	.
/b_3_3	.0072243	.0005979	12.08	0.000	.0060525	.0083962

The SVAR output has four parts: an iteration log, a display of the constraints imposed, a header with sample and SVAR log-likelihood information, and a table displaying the estimates of the parameters from the **A** and **B** matrices. From the output above, we can see that the equality constraint matrices supplied to `svar` imposed the intended constraints and that the SVAR header informs us that the model we fit is just identified. The estimates of `a_2_1`, `a_3_1`, and `a_3_2` are all negative. Because the off-diagonal elements of the **A** matrix contain the negative of the actual contemporaneous effects, the estimated effects are positive, as expected.

The estimates $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are stored in `e(A)` and `e(B)`, respectively, allowing us to compute the estimated Cholesky decomposition.

```
. matrix Aest = e(A)
. matrix Best = e(B)
. matrix chol_est = inv(Aest)*Best
```

```
. matrix list chol_est
chol_est[3,3]
      dln_inv      dln_inc      dln_consump
dln_inv      .04387957      0      0
dln_inc      .00147562      .01104494      0
dln_consump   .00253928      .0046916      .00722432
```

svar saves the estimated Σ from the underlying var in `e(Sigma)`. The output below illustrates the computation of the Cholesky decomposition of `e(Sigma)`. It is the same as the output computed from the SVAR estimates.

```
. matrix sig_var = e(Sigma)
. matrix chol_var = cholesky(sig_var)
. matrix list chol_var
chol_var[3,3]
      dln_inv      dln_inc      dln_consump
dln_inv      .04387957      0      0
dln_inc      .00147562      .01104494      0
dln_consump   .00253928      .0046916      .00722432
```

◀

We might now wonder why we bother obtaining parameter estimates via nonlinear estimation if we can obtain them simply by a transform of the estimates produced by `var`. When the model is just identified, as in the previous example, the SVAR parameter estimates can be computed via a transform of the VAR estimates. However, when the model is overidentified, such is not the case.

► Example 2: Short-run overidentified SVAR model

The Cholesky decomposition example above fit a just-identified model. This example considers an overidentified model. In the previous example, the `a_2_1` parameter was not significant, which is consistent with a theory in which changes in our measure of investment affect only changes in income with a lag. We can impose the restriction that `a_2_1` is zero and then test this overidentifying restriction. Our **A** and **B** matrices are now

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ . & . & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} . & 0 & 0 \\ 0 & . & 0 \\ 0 & 0 & . \end{bmatrix}$$

The output below contains the commands and results we obtained by fitting this model on the Lütkepohl data.

```
. matrix B = (.,0,0\0,.,0\0,0,.)
. matrix A = (1,0,0\0,1,0\.,.,1)
```

(Continued on next page)

```
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), aeq(A) beq(B)
Estimating short-run parameters
(output omitted)
```

Structural vector autoregression

```
( 1) [a_1_1]_cons = 1
( 2) [a_1_2]_cons = 0
( 3) [a_1_3]_cons = 0
( 4) [a_2_1]_cons = 0
( 5) [a_2_2]_cons = 1
( 6) [a_2_3]_cons = 0
( 7) [a_3_3]_cons = 1
( 8) [b_1_2]_cons = 0
( 9) [b_1_3]_cons = 0
(10) [b_2_1]_cons = 0
(11) [b_2_3]_cons = 0
(12) [b_3_1]_cons = 0
(13) [b_3_2]_cons = 0
```

Sample: 1960q4 - 1978q4

No. of obs = 73

Overidentified model

Log likelihood = 605.6613

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
/a_1_1	1	.	.	.	.	.
/a_2_1	0	.	.	.	.	.
/a_3_1	-.0435911	.0192696	-2.26	0.024	-.0813589	-.0058233
/a_1_2	0	.	.	.	.	.
/a_2_2	1	.	.	.	.	.
/a_3_2	-.4247741	.0758806	-5.60	0.000	-.5734973	-.2760508
/a_1_3	0	.	.	.	.	.
/a_2_3	0	.	.	.	.	.
/a_3_3	1	.	.	.	.	.
/b_1_1	.0438796	.0036315	12.08	0.000	.036762	.0509972
/b_2_1	0	.	.	.	.	.
/b_3_1	0	.	.	.	.	.
/b_1_2	0	.	.	.	.	.
/b_2_2	.0111431	.0009222	12.08	0.000	.0093356	.0129506
/b_3_2	0	.	.	.	.	.
/b_1_3	0	.	.	.	.	.
/b_2_3	0	.	.	.	.	.
/b_3_3	.0072243	.0005979	12.08	0.000	.0060525	.0083962

LR test of identifying restrictions: chi2(1)= 1.292 Prob > chi2 = 0.256

The footer in this example reports a test of the overidentifying restriction. The null hypothesis of this test is that any overidentifying restrictions are valid. In the case at hand, we cannot reject this null hypothesis at any of the conventional levels.



➤ Example 3: Short-run SVAR model with constraints

svar also allows us to place constraints on the parameters of the underlying VAR. We begin by looking at the underlying VAR for the SVARs that we have used in the previous examples.

```
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4)
```

Vector autoregression

```
Sample: 1960q4 - 1978q4          No. of obs   =          73
Log likelihood = 606.307          AIC         = -16.03581
FPE           = 2.18e-11          HQIC        = -15.77323
Det(Sigma_ml) = 1.23e-11          SBIC        = -15.37691
```

Equation	Parms	RMSE	R-sq	chi2	P>chi2
dln_inv	7	.046148	0.1286	10.76961	0.0958
dln_inc	7	.011719	0.1142	9.410683	0.1518
dln_consump	7	.009445	0.2513	24.50031	0.0004

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
dln_inv						
dln_inv						
L1.	-.3196318	.1192898	-2.68	0.007	-.5534355	-.0858282
L2.	-.1605508	.118767	-1.35	0.176	-.39333	.0722283
dln_inc						
L1.	.1459851	.5188451	0.28	0.778	-.8709326	1.162903
L2.	.1146009	.508295	0.23	0.822	-.881639	1.110841
dln_consump						
L1.	.9612288	.6316557	1.52	0.128	-.2767936	2.199251
L2.	.9344001	.6324034	1.48	0.140	-.3050877	2.173888
_cons	-.0167221	.0163796	-1.02	0.307	-.0488257	.0153814
dln_inc						
dln_inv						
L1.	.0439309	.0302933	1.45	0.147	-.0154427	.1033046
L2.	.0500302	.0301605	1.66	0.097	-.0090833	.1091437
dln_inc						
L1.	-.1527311	.131759	-1.16	0.246	-.4109741	.1055118
L2.	.0191634	.1290799	0.15	0.882	-.2338285	.2721552
dln_consump						
L1.	.2884992	.1604069	1.80	0.072	-.0258926	.6028909
L2.	-.0102	.1605968	-0.06	0.949	-.3249639	.3045639
_cons	.0157672	.0041596	3.79	0.000	.0076146	.0239198
dln_consump						
dln_inv						
L1.	-.002423	.0244142	-0.10	0.921	-.050274	.045428
L2.	.0338806	.0243072	1.39	0.163	-.0137607	.0815219
dln_inc						
L1.	.2248134	.1061884	2.12	0.034	.0166879	.4329389
L2.	.3549135	.1040292	3.41	0.001	.1510199	.558807
dln_consump						
L1.	-.2639695	.1292766	-2.04	0.041	-.517347	-.010592
L2.	-.0222264	.1294296	-0.17	0.864	-.2759039	.231451
_cons	.0129258	.0033523	3.86	0.000	.0063554	.0194962

The equation-level model tests reported in the header indicate that we cannot reject the null hypotheses that all the coefficients in the first equation are zero, nor can we reject the null that all the coefficients in the second equation are zero at the 5% significance level. We use a combination of theory and the p -values from the output above to place some exclusion restrictions on the underlying VAR(2). Specifically, in the equation for the percent change of `inv`, we constrain the coefficients on `L2.dln_inv`, `L.dln_inc`, `L2.dln_inc`, and `L2.dln_consump` to be zero. In the equation for `dln_inc`, we constrain the coefficients on `L2.dln_inv`, `L2.dln_inc`, and `L2.dln_consump` to be zero. Finally, in the equation for `dln_consump`, we constrain `L.dln_inv` and `L2.dln_consump` to be zero. We then refit the SVAR from the previous example.

```
. constraint 1 [dln_inv]L2.dln_inv = 0
. constraint 2 [dln_inv]L.dln_inc = 0
. constraint 3 [dln_inv]L2.dln_inc = 0
. constraint 4 [dln_inv]L2.dln_consump = 0
. constraint 5 [dln_inc]L2.dln_inv = 0
. constraint 6 [dln_inc]L2.dln_inc = 0
. constraint 7 [dln_inc]L2.dln_consump = 0
. constraint 8 [dln_consump]L.dln_inv = 0
. constraint 9 [dln_consump]L2.dln_consump = 0
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), aeq(A) beq(B) varconst(1/9)
> noislog
```

Estimating short-run parameters

(output omitted)

Structural vector autoregression

```
( 1) [a_1_1]_cons = 1
( 2) [a_1_2]_cons = 0
( 3) [a_1_3]_cons = 0
( 4) [a_2_1]_cons = 0
( 5) [a_2_2]_cons = 1
( 6) [a_2_3]_cons = 0
( 7) [a_3_3]_cons = 1
( 8) [b_1_2]_cons = 0
( 9) [b_1_3]_cons = 0
(10) [b_2_1]_cons = 0
(11) [b_2_3]_cons = 0
(12) [b_3_1]_cons = 0
(13) [b_3_2]_cons = 0
```

Sample: 1960q4 - 1978q4
Overidentified model

No. of obs = 73
Log likelihood = 601.8591

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
/a_1_1	1	.	.	.	.	.
/a_2_1	0	.	.	.	.	.
/a_3_1	-.0418708	.0187579	-2.23	0.026	-.0786356	-.0051061
/a_1_2	0	.	.	.	.	.
/a_2_2	1	.	.	.	.	.
/a_3_2	-.4255808	.0745298	-5.71	0.000	-.5716565	-.2795051
/a_1_3	0	.	.	.	.	.
/a_2_3	0	.	.	.	.	.
/a_3_3	1	.	.	.	.	.
/b_1_1	.0451851	.0037395	12.08	0.000	.0378557	.0525145
/b_2_1	0	.	.	.	.	.
/b_3_1	0	.	.	.	.	.
/b_1_2	0	.	.	.	.	.
/b_2_2	.0113723	.0009412	12.08	0.000	.0095276	.013217
/b_3_2	0	.	.	.	.	.
/b_1_3	0	.	.	.	.	.
/b_2_3	0	.	.	.	.	.
/b_3_3	.0072417	.0005993	12.08	0.000	.006067	.0084164

LR test of identifying restrictions: chi2(1)= .8448 Prob > chi2 = 0.358

If we displayed the underlying VAR(2) results by using the `var` option, we would see that most of the unconstrained coefficients are now significant at the 10% level and that none of the equation-level model statistics fail to reject the null hypothesis at the 10% level. The `sva` output reveals that the p -value of the overidentification test rose and that the coefficient on `a_3_1` is still insignificant at the 1% level but not at the 5% level.

◀

Before moving on to models with long-run constraints, consider these limitations. We cannot place constraints on the elements of **A** in terms of the elements of **B**, or vice versa. This limitation is imposed by the form of the check for identification derived by [Amisano and Giannini \(1997\)](#). As noted in [Methods and formulas](#), this test requires separate constraint matrices for the parameters in **A** and **B**. Another limitation is that we cannot mix short-run and long-run constraints.

Long-run SVAR models

As discussed in [\[TS\] var intro](#), a long-run SVAR has the form

$$\mathbf{y}_t = \mathbf{C}\mathbf{e}_t$$

In long-run models, the constraints are placed on the elements of **C**, and the free parameters are estimated. These constraints are often exclusion restrictions. For instance, constraining **C**[1,2] to be zero can be interpreted as setting the long-run response of variable 1 to the structural shocks driving variable 2 to be zero.

Similar to the short-run model, the $\mathbf{P}_{lr}$ matrix such that $\mathbf{P}_{lr}\mathbf{P}'_{lr} = \mathbf{\Sigma}$ identifies the structural impulse-response functions. $\mathbf{P}_{lr} = \mathbf{C}$ is identified by the restrictions placed on the parameters in **C**. There are K^2 parameters in **C**, and the order condition for identification requires that there be at least $K^2 - K(K+1)/2$ restrictions placed on those parameters. As in the short-run model, this order condition is necessary but not sufficient, so the [Amisano and Giannini \(1997\)](#) check for local identification is performed by default.

➤ Example 4: Long-run SVAR model

Suppose that we have a theory in which unexpected changes to the money supply have no long-run effects on changes in output and, similarly, that unexpected changes in output have no long-run effects on changes in the money supply. The $\mathbf{C}$ matrix implied by this theory is

$$\mathbf{C} = \begin{bmatrix} \cdot & 0 \\ 0 & \cdot \end{bmatrix}$$

```
. use http://www.stata-press.com/data/r11/m1gdp
. matrix lr = (.,0\0,.)
. svar d.ln_m1 d.ln_gdp, lreq(lr)
Estimating long-run parameters
(output omitted)
Structural vector autoregression
( 1) [c_1_2]_cons = 0
( 2) [c_2_1]_cons = 0
Sample: 1959q4 - 2002q2
Overidentified model
```

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
/c_1_1	.0301007	.0016277	18.49	0.000	.0269106	.0332909
/c_2_1	0	.	.	.	.	.
/c_1_2	0	.	.	.	.	.
/c_2_2	.0129691	.0007013	18.49	0.000	.0115946	.0143436

```
LR test of identifying restrictions: chi2( 1)= .1368 Prob > chi2 = 0.712
```

We have assumed that the underlying VAR has 2 lags; four of the five selection-order criteria computed by varsoc (see [TS] varsoc) recommended this choice. The test of the overidentifying restrictions provides no indication that it is not valid.



Saved results

`svar` saves the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(N_cns)</code>	number of constraints
<code>e(k_eq)</code>	number of equations
<code>e(k_dv)</code>	number of dependent variables
<code>e(k_aux)</code>	number of auxiliary parameters
<code>e(ll)</code>	log likelihood from <code>svar</code>
<code>e(ll_#)</code>	log likelihood for equation #
<code>e(N_gaps_var)</code>	number of gaps in the sample
<code>e(k_var)</code>	number of coefficients in VAR
<code>e(k_eq_var)</code>	number of equations in underlying VAR
<code>e(k_dv_var)</code>	number of dependent variables in underlying VAR
<code>e(df_eq_var)</code>	average number of parameters in an equation
<code>e(df_m_var)</code>	model degrees of freedom
<code>e(df_r_var)</code>	if <code>small</code> , residual degrees of freedom
<code>e(obs_#_var)</code>	number of observations on equation #
<code>e(k_#_var)</code>	number of coefficients in equation #
<code>e(df_m#_var)</code>	model degrees of freedom for equation #
<code>e(df_r#_var)</code>	residual degrees of freedom for equation # (<code>small</code> only)
<code>e(r2_#_var)</code>	<i>R</i> -squared for equation #
<code>e(ll_#_var)</code>	log likelihood for equation # VAR
<code>e(chi2_#_var)</code>	χ^2 statistic for equation #
<code>e(F_#_var)</code>	<i>F</i> statistic for equation # (<code>small</code> only)
<code>e(rmse_#_var)</code>	root mean squared error for equation #
<code>e(mlag_var)</code>	highest lag in VAR
<code>e(tparms_var)</code>	number of parameters in all equations
<code>e(aic_var)</code>	Akaike information criterion
<code>e(hqic_var)</code>	Hannan–Quinn information criterion
<code>e(sbic_var)</code>	Schwarz–Bayesian information criterion
<code>e(fpe_var)</code>	final prediction error
<code>e(ll_var)</code>	log likelihood from <code>var</code>
<code>e(detsig_var)</code>	determinant of $\mathbf{e}(\Sigma)$
<code>e(detsig_ml_var)</code>	determinant of $\hat{\Sigma}_{ml}$
<code>e(tmin)</code>	first time period in the sample
<code>e(tmax)</code>	maximum time
<code>e(chi2_oid)</code>	overidentification test
<code>e(oid_df)</code>	number of overidentifying restrictions
<code>e(rank)</code>	rank of $\mathbf{e}(V)$
<code>e(ic_ml)</code>	number of iterations
<code>e(rc_ml)</code>	return code from <code>ml</code>

Macros

<code>e(cmd)</code>	<code>svar</code>
<code>e(cmdline)</code>	command as typed
<code>e(lrmodel)</code>	long-run model, if specified
<code>e(lags_var)</code>	lags in model
<code>e(depvar_var)</code>	names of dependent variables
<code>e(endog_var)</code>	names of endogenous variables
<code>e(exog_var)</code>	names of exogenous variables, if specified
<code>e(nocons_var)</code>	noconstant, if <code>noconstant</code> specified
<code>e(cns_lr)</code>	long-run constraints
<code>e(cns_a)</code>	cross-parameter equality constraints on A
<code>e(cns_b)</code>	cross-parameter equality constraints on B
<code>e(dfk_var)</code>	alternate divisor (<code>dfk</code>), if specified
<code>e(eqnames_var)</code>	names of equations
<code>e(lutstats_var)</code>	<code>lutstats</code> , if specified
<code>e(constraints_var)</code>	<code>constraints_var</code> , if there are constraints on VAR
<code>e(small)</code>	<code>small</code> , if specified
<code>e(tsfmt)</code>	format of timevar
<code>e(timevar)</code>	name of timevar
<code>e(title)</code>	title in estimation output
<code>e(properties)</code>	b V
<code>e(predict)</code>	program used to implement <code>predict</code>

Matrices

<code>e(b)</code>	coefficient vector
<code>e(Cns)</code>	constraints matrix
<code>e(Sigma)</code>	Σ matrix
<code>e(V)</code>	variance–covariance matrix of the estimators
<code>e(b_var)</code>	coefficient vector of underlying VAR model
<code>e(V_var)</code>	VCE of underlying VAR model
<code>e(bf_var)</code>	full coefficient vector with zeros in dropped lags
<code>e(G_var)</code>	G matrix saved by <code>var</code> ; see [TS] var Methods and formulas
<code>e(aeq)</code>	<code>aeq(matrix)</code> , if specified
<code>e(acns)</code>	<code>acns(matrix)</code> , if specified
<code>e(beq)</code>	<code>beq(matrix)</code> , if specified
<code>e(bcns)</code>	<code>bcns(matrix)</code> , if specified
<code>e(lreq)</code>	<code>lreq(matrix)</code> , if specified
<code>e(lrcns)</code>	<code>lrcns(matrix)</code> , if specified
<code>e(Cns_var)</code>	constraint matrix from <code>var</code> , if <code>varconstraints()</code> is specified
<code>e(A)</code>	estimated A matrix, if a short-run model
<code>e(B)</code>	estimated B matrix
<code>e(C)</code>	estimated C matrix, if a long-run model
<code>e(A1)</code>	estimated $\bar{A}$ matrix, if a long-run model

Functions

<code>e(sample)</code>	marks estimation sample

Methods and formulas

`sva` is implemented as an ado-file.

The log-likelihood function for models with short-run constraints is

$$L(\mathbf{A}, \mathbf{B}) = -\frac{NK}{2} \ln(2\pi) + \frac{N}{2} \ln(|\mathbf{W}|^2) - \frac{N}{2} \text{tr}(\mathbf{W}'\mathbf{W}\hat{\Sigma})$$

where $\mathbf{W} = \mathbf{B}^{-1}\mathbf{A}$.

When there are long-run constraints, because $\mathbf{C} = \bar{\mathbf{A}}^{-1}\mathbf{B}$ and $\mathbf{A} = \mathbf{I}_K$, $\mathbf{W} = \mathbf{B}^{-1} = \mathbf{C}^{-1}\bar{\mathbf{A}}^{-1} = (\bar{\mathbf{A}}\mathbf{C})^{-1}$. Substituting the last term for $\mathbf{W}$ in the short-run log likelihood produces the long-run log likelihood

$$L(\mathbf{C}) = -\frac{NK}{2} \ln(2\pi) + \frac{N}{2} \ln(|\widetilde{\mathbf{W}}|^2) - \frac{N}{2} \text{tr}(\widetilde{\mathbf{W}}'\widetilde{\mathbf{W}}\hat{\Sigma})$$

where $\widetilde{\mathbf{W}} = (\bar{\mathbf{A}}\mathbf{C})^{-1}$.

For both the short-run and the long-run models, the maximization is performed by the scoring method. See [Harvey \(1990\)](#) for a discussion of this method.

Based on results from [Amisano and Giannini \(1997\)](#), the score vector for the short-run model is

$$\frac{\partial L(\mathbf{A}, \mathbf{B})}{\partial [\text{vec}(\mathbf{A}), \text{vec}(\mathbf{B})]} = N \left[\{\text{vec}(\mathbf{W}'^{-1})\}' - \{\text{vec}(\mathbf{W})\}'(\hat{\Sigma} \otimes \mathbf{I}_K) \right] \times \\ [(\mathbf{I}_K \otimes \mathbf{B}^{-1}), -(\mathbf{A}'\mathbf{B}'^{-1} \otimes \mathbf{B}^{-1})]$$

and the expected information matrix is

$$I [\text{vec}(\mathbf{A}), \text{vec}(\mathbf{B})] = N \begin{bmatrix} (\mathbf{W}^{-1} \otimes \mathbf{B}'^{-1}) \\ -(\mathbf{I}_K \otimes \mathbf{B}'^{-1}) \end{bmatrix} (\mathbf{I}_{K^2} + \oplus) [(\mathbf{W}'^{-1} \otimes \mathbf{B}^{-1}), -(\mathbf{I}_K \otimes \mathbf{B}^{-1})]$$

where $\oplus$ is the commutation matrix defined in [Magnus and Neudecker \(1999, 46–48\)](#).

Using results from [Amisano and Giannini \(1997\)](#), we can derive the score vector and the expected information matrix for the case with long-run restrictions. The score vector is

$$\frac{\partial L(\mathbf{C})}{\partial \text{vec}(\mathbf{C})} = N \left[\{\text{vec}(\mathbf{W}'^{-1})\}' - \{\text{vec}(\mathbf{W})\}'(\hat{\Sigma} \otimes \mathbf{I}_K) \right] [-(\bar{\mathbf{A}}'^{-1}\mathbf{C}'^{-1} \otimes \mathbf{C}^{-1})]$$

and the expected information matrix is

$$I [\text{vec}(\mathbf{C})] = N(\mathbf{I}_K \otimes \mathbf{C}'^{-1})(\mathbf{I}_{K^2} + \oplus)(\mathbf{I}_K \otimes \mathbf{C}'^{-1})$$

Checking for identification

This section describes the methods used to check for identification of models with short-run or long-run constraints. Both methods depend on the starting values. By default, `sva` uses starting values constructed by taking a vector of appropriate dimension and applying the constraints. If there are m parameters in the model, the j th element of the $1 \times m$ vector is $1 + m/100$. `sva` also allows the user to provide starting values.

For the short-run case, the model is identified if the matrix

$$\mathbf{V}_{\text{sr}}^* = \begin{bmatrix} \mathbf{N}_K & \mathbf{N}_K \\ \mathbf{N}_K & \mathbf{N}_K \\ \mathbf{R}_a(\mathbf{W}' \otimes \mathbf{B}) & \mathbf{0}_{K^2} \\ \mathbf{0}_{K^2} & \mathbf{R}_a(\mathbf{I}_K \otimes \mathbf{B}) \end{bmatrix}$$

has full column rank of $2K^2$, where $\mathbf{N}_K = (1/2)(\mathbf{I}_{K^2} + \oplus)$, $\mathbf{R}_a$ is the constraint matrix for the parameters in $\mathbf{A}$ (i.e., $\mathbf{R}_a \text{vec}(\mathbf{A}) = \mathbf{r}_a$), and $\mathbf{R}_b$ is the constraint matrix for the parameters in $\mathbf{B}$ (i.e., $\mathbf{R}_b \text{vec}(\mathbf{B}) = \mathbf{r}_b$).

For the long-run case, based on results from the **C** model in [Amisano and Giannini \(1997\)](#), the model is identified if the matrix

$$\mathbf{V}_{\text{lr}}^* = \begin{bmatrix} (\mathbf{I} \otimes \mathbf{C}'^{-1})(2\mathbf{N}_K)(\mathbf{I} \otimes \mathbf{C}^{-1}) \\ \mathbf{R}_c \end{bmatrix}$$

has full column rank of K^2 , where $\mathbf{R}_c$ is the constraint matrix for the parameters in $\mathbf{C}$; i.e., $\mathbf{R}_c \text{vec}(\mathbf{C}) = \mathbf{r}_c$.

The test of the overidentifying restrictions is computed as

$$LR = 2(LL_{\text{var}} - LL_{\text{svar}})$$

where LR is the value of the test statistic against the null hypothesis that the overidentifying restrictions are valid, LL_{var} is the log likelihood from the underlying $\text{VAR}(p)$ model, and LL_{svar} is the log likelihood from the SVAR model. The test statistic is asymptotically distributed as $\chi^2(q)$, where q is the number of overidentifying restrictions. [Amisano and Giannini \(1997, 38–39\)](#) emphasize that, because this test of the validity of the overidentifying restrictions is an omnibus test, it can be interpreted as a test of the null hypothesis that all the restrictions are valid.

Because constraints might not be independent either by construction or because of the data, the number of restrictions is not necessarily equal to the number of constraints. The rank of $\mathbf{e}(\mathbf{V})$ gives the number of parameters that were independently estimated after applying the constraints. The maximum number of parameters that can be estimated in an identified short-run or long-run SVAR is $K(K+1)/2$. This implies that the number of overidentifying restrictions, q , is equal to $K(K+1)/2$ minus the rank of $\mathbf{e}(\mathbf{V})$.

The number of overidentifying restrictions is also linked to the order condition for each model. In a short-run SVAR model, there are $2K^2$ parameters. Because no more than $K(K+1)/2$ parameters may be estimated, the order condition for a short-run SVAR model is that at least $2K^2 - K(K+1)/2$ restrictions be placed on the model. Similarly, there are K^2 parameters in long-run SVAR model. Because no more than $K(K+1)/2$ parameters may be estimated, the order condition for a long-run SVAR model is that at least $K^2 - K(K+1)/2$ restrictions be placed on the model.

Acknowledgment

We thank Gianni Amisano, Università degli Studi di Brescia, for his helpful comments.

References

- Amisano, G., and C. Giannini. 1997. *Topics in Structural VAR Econometrics*. 2nd ed. Heidelberg: Springer.
- Christiano, L. J., M. Eichenbaum, and C. L. Evans. 1999. Monetary policy shocks: What have we learned and to what end? In *Handbook of Macroeconomics: Volume 1A*, ed. J. B. Taylor and M. Woodford. New York: Elsevier.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Harvey, A. C. 1990. *The Econometric Analysis of Time Series*. 2nd ed. Cambridge, MA: MIT Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Magnus, J. R., and H. Neudecker. 1999. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Rev. ed. New York: Wiley.
- Rothenberg, T. J. 1971. Identification in parametric models. *Econometrica* 39: 577–591.
- Sims, C. A. 1980. Macroeconomics and reality. *Econometrica* 48: 1–48.
- Stock, J. H., and M. W. Watson. 2001. Vector autoregressions. *Journal of Economic Perspectives* 15: 101–115.
- Watson, M. W. 1994. Vector autoregressions and cointegration. In Vol. IV of *Handbook of Econometrics*, ed. R. F. Engle and D. L. McFadden. Amsterdam: Elsevier.

Also see

- [TS] [var svar postestimation](#) — Postestimation tools for svar
- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [var](#) — Vector autoregressive models
- [TS] [varbasic](#) — Fit a simple VAR and graph IRFS or FEVDs
- [TS] [vec](#) — Vector error-correction models
- [U] [20 Estimation and postestimation commands](#)
- [TS] [var intro](#) — Introduction to vector autoregressive models

Description

The following postestimation commands are of special interest after `sva`:

command	description
<code>fcast compute</code>	obtain dynamic forecasts
<code>fcast graph</code>	graph dynamic forecasts obtained from <code>fcast compute</code>
<code>irf</code>	create and analyze IRFs and FEVDs
<code>vargranger</code>	Granger causality tests
<code>varlmar</code>	LM test for autocorrelation in residuals
<code>varnorm</code>	test for normally distributed residuals
<code>varsoc</code>	lag-order selection criteria
<code>varstable</code>	check stability condition of estimates
<code>varwle</code>	Wald lag-exclusion statistics

For information about these commands, see the corresponding entries in this manual.

The following standard postestimation commands are also available:

command	description
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	point estimates, standard errors, testing, and inference for generalized predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

```
predict [type] newvar [if] [in] [, equation(eqno|eqname) statistic]
```

statistic	description
<code>xb</code>	linear prediction; the default
<code>stdp</code>	standard error of the linear prediction
<code>residuals</code>	residuals

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

`equation(eqno|eqname)` specifies the equation to which you are referring.

`equation()` is filled in with one *eqno* or *eqname* for options `xb`, `stdp`, and `residuals`. For example, `equation(#1)` would mean that the calculation is to be made for the first equation, `equation(#2)` would mean the second, and so on. You could also refer to the equation by its name; thus, `equation(income)` would refer to the equation named `income` and `equation(hours)`, to the equation named `hours`.

If you do not specify `equation()`, the results are the same as if you specified `equation(#1)`.

`xb`, the default, calculates the linear prediction for the specified equation.

`stdp` calculates the standard error of the linear prediction for the specified equation.

`residuals` calculates the residuals.

For more information on using `predict` after multiple-equation estimation commands, see [\[R\] predict](#).

Remarks

Remarks are presented under the following headings:

Model selection and inference
Forecasting

Model selection and inference

See the following sections for information on model selection and inference after `var`.

[\[TS\] irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[\[TS\] vargranger](#) — Perform pairwise Granger causality tests after `var` or `svar`

[\[TS\] varlmar](#) — Perform LM test for residual autocorrelation after `var` or `svar`

[\[TS\] varnorm](#) — Test for normally distributed disturbances after `var` or `svar`

[\[TS\] varsoc](#) — Obtain lag-order selection statistics for VARs and VECMs

[\[TS\] varstable](#) — Check the stability condition of VAR or SVAR estimates

[\[TS\] varwle](#) — Obtain Wald lag-exclusion statistics after `var` or `svar`

Forecasting

See the following sections for information on obtaining forecasts after `svar`:

[\[TS\] fcast compute](#) — Compute dynamic forecasts of dependent variables after `var`, `svar`, or `vec`

[\[TS\] fcast graph](#) — Graph forecasts of variables computed by `fcast compute`

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Also see

[TS] [var svar](#) — Structural vector autoregressive models

[U] [20 Estimation and postestimation commands](#)

Syntax

```
varbasic depvarlist [if] [in] [, options]
```

options	description
Main	
<u>lags</u> (numlist)	use lags numlist in the model; default is lags(1 2)
<u>irf</u>	produce matrix graph of IRFs
<u>fevd</u>	produce matrix graph of FEVDs
<u>nograph</u>	do not produce a graph
<u>step</u> (#)	set forecast horizon # for estimating the OIRFs, IRFs, and FEVDs; default is step(8)

You must `tsset` your data before using `varbasic`; see [TS] `tsset`.
`depvarlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.
`rolling`, `statsby`, and `xi` are allowed; see [U] 11.1.10 Prefix commands.
See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

Menu

Statistics > Multivariate time series > Basic VAR

Description

`varbasic` fits a basic vector autoregressive (VAR) model and graphs the impulse–response functions (IRFs), the orthogonalized impulse–response functions (OIRFs), or the forecast-error variance decompositions (FEVDs).

Options

Main
<code>lags(numlist)</code> specifies the lags to be included in the model. The default is <code>lags(1 2)</code> . This option takes a numlist and not simply an integer for the maximum lag. For instance, <code>lags(2)</code> would include only the second lag in the model, whereas <code>lags(1/2)</code> would include both the first and second lags in the model. See [U] 11.1.8 numlist and [U] 11.4.4 Time-series varlists for more discussion of numlists and lags.
<code>irf</code> causes <code>varbasic</code> to produce a matrix graph of the IRFs instead of a matrix graph of the OIRFs, which is produced by default.
<code>fevd</code> causes <code>varbasic</code> to produce a matrix graph of the FEVDs instead of a matrix graph of the OIRFs, which is produced by default.
<code>nograph</code> specifies that no graph be produced. The IRFs, OIRFs, and FEVDs are still estimated and saved in the IRF file <code>_varbasic.irf</code> .

`step(#)` specifies the forecast horizon for estimating the IRFs, OIRFs, and FEVDs. The default is eight periods.

Remarks

`varbasic` simplifies fitting simple VARs and graphing the IRFs, the OIRFs, or the FEVDs. See [\[TS\] var](#) and [\[TS\] var svar](#) for fitting more advanced VAR models and structural vector autoregressive (SVAR) models. All the postestimation commands discussed in [\[TS\] var postestimation](#) work after `varbasic`.

This entry does not discuss the methods for fitting a VAR or the methods surrounding the IRFs, OIRFs, and FEVDs. See [\[TS\] var](#) and [\[TS\] irf create](#) for more on these methods. This entry illustrates how to use `varbasic` to easily obtain results. It also illustrates how `varbasic` serves as an entry point to further analysis.

► Example 1

We fit a three-variable VAR with two lags to the German macro data used by [Lütkepohl \(2005\)](#). The three variables are the first difference of natural log of investment, `dln_inv`; the first difference of the natural log of income, `dln_inc`; and the first difference of the natural log of consumption, `dln_consump`. In addition to fitting the VAR, we want to see the OIRFs. Below we use `varbasic` to fit a VAR(2) model on the data from the second quarter of 1961 through the fourth quarter of 1978. By default, `varbasic` produces graphs of the OIRFs.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. varbasic dln_inv dln_inc dln_consump if qtr<=tq(1978q4)

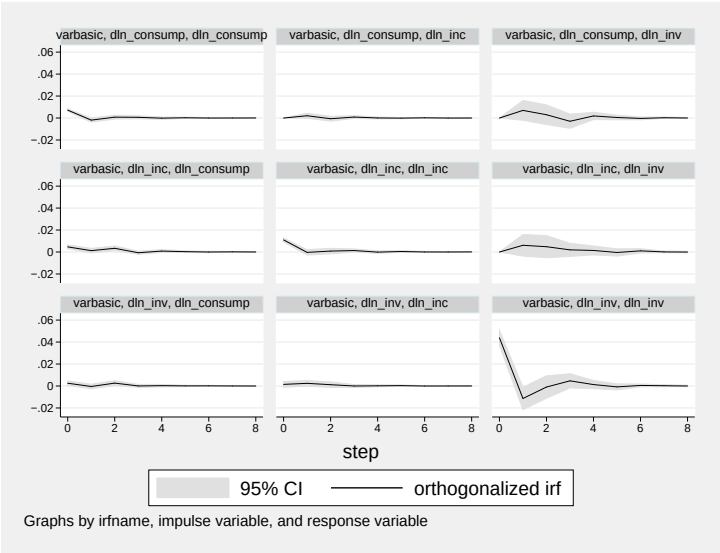
Vector autoregression

Sample: 1960q4 - 1978q4                      No. of obs   =           73
Log likelihood = 606.307                      AIC           = -16.03581
FPE           = 2.18e-11                      HQIC          = -15.77323
Det(Sigma_ml) = 1.23e-11                      SBIC          = -15.37691

Equation      Parms      RMSE      R-sq      chi2      P>chi2
-----
dln_inv       7          .046148   0.1286   10.76961   0.0958
dln_inc       7          .011719   0.1142    9.410683   0.1518
dln_consump   7          .009445   0.2513   24.50031   0.0004
```

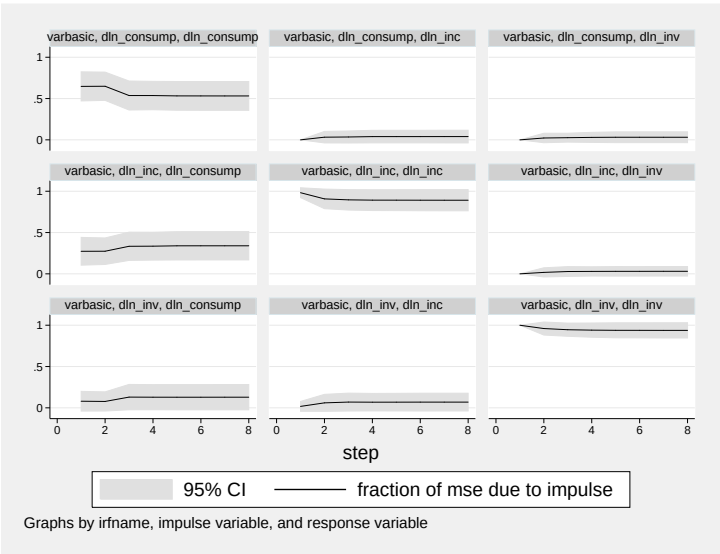
	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
dln_inv						
dln_inv						
L1.	-.3196318	.1192898	-2.68	0.007	-.5534355	-.0858282
L2.	-.1605508	.118767	-1.35	0.176	-.39333	.0722283
dln_inc						
L1.	.1459851	.5188451	0.28	0.778	-.8709326	1.162903
L2.	.1146009	.508295	0.23	0.822	-.881639	1.110841
dln_consump						
L1.	.9612288	.6316557	1.52	0.128	-.2767936	2.199251
L2.	.9344001	.6324034	1.48	0.140	-.3050877	2.173888
_cons	-.0167221	.0163796	-1.02	0.307	-.0488257	.0153814
dln_inc						
dln_inv						
L1.	.0439309	.0302933	1.45	0.147	-.0154427	.1033046
L2.	.0500302	.0301605	1.66	0.097	-.0090833	.1091437
dln_inc						
L1.	-.1527311	.131759	-1.16	0.246	-.4109741	.1055118
L2.	.0191634	.1290799	0.15	0.882	-.2338285	.2721552
dln_consump						
L1.	.2884992	.1604069	1.80	0.072	-.0258926	.6028909
L2.	-.0102	.1605968	-0.06	0.949	-.3249639	.3045639
_cons	.0157672	.0041596	3.79	0.000	.0076146	.0239198
dln_consump						
dln_inv						
L1.	-.002423	.0244142	-0.10	0.921	-.050274	.045428
L2.	.0338806	.0243072	1.39	0.163	-.0137607	.0815219
dln_inc						
L1.	.2248134	.1061884	2.12	0.034	.0166879	.4329389
L2.	.3549135	.1040292	3.41	0.001	.1510199	.558807
dln_consump						
L1.	-.2639695	.1292766	-2.04	0.041	-.517347	-.010592
L2.	-.0222264	.1294296	-0.17	0.864	-.2759039	.231451
_cons	.0129258	.0033523	3.86	0.000	.0063554	.0194962

(Continued on next page)



Because we are also interested in looking at the FEVDs, we can use `irf graph` to obtain the graphs. Although the details are available in [\[TS\] irf](#) and [\[TS\] irf graph](#), the command below produces what we want after the call to `varbasic`.

```
. irf graph fevd, lstep(1)
```



❏ Technical note

Stata stores the estimated IRFs, OIRFs, and FEVDs in a IRF file called `_varbasic.irf` in the current working directory. `varbasic` replaces any `_varbasic.irf` that already exists. Finally, `varbasic` makes `_varbasic.irf` the active IRF file. This means that the graph and table commands `irf graph`,



`irf cgraph`, `irf ograph`, `irf table`, and `irf ctable` will all display results that correspond to the VAR fit by `varbasic`.



Saved results

See *Saved results* in [TS] `var`.

Methods and formulas

`varbasic` is implemented as an ado-file.

`varbasic` uses `var` and `irf graph` to obtain its results. See [TS] `var` and [TS] `irf graph` for a discussion of how those commands obtain their results.

References

- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Also see

- [TS] `varbasic postestimation` — Postestimation tools for `varbasic`
- [TS] `tsset` — Declare data to be time-series data
- [TS] `var` — Vector autoregressive models
- [TS] `var svar` — Structural vector autoregressive models
- [U] **20 Estimation and postestimation commands**
- [TS] `var intro` — Introduction to vector autoregressive models

Description

The following postestimation commands are of special interest after `varbasic`:

command	description
<code>fcast compute</code>	obtain dynamic forecasts
<code>fcast graph</code>	graph dynamic forecasts obtained from <code>fcast compute</code>
<code>irf</code>	create and analyze IRFs and FEVDs
<code>vargranger</code>	Granger causality tests
<code>varlmar</code>	LM test for autocorrelation in residuals
<code>varnorm</code>	test for normally distributed residuals
<code>varsoc</code>	lag-order selection criteria
<code>varstable</code>	check stability condition of estimates
<code>varwle</code>	Wald lag-exclusion statistics

For information about these commands, see the corresponding entries in this manual.

The following standard postestimation commands are also available:

command	description
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	point estimates, standard errors, testing, and inference for generalized predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

`predict` [*type*] *newvar* [*if*] [*in*] [, e*quation*(*eqno*|*eqname*) *statistic*]

<i>statistic</i>	description
<code>xb</code>	linear prediction; the default
<code>stdp</code>	standard error of the linear prediction
<code>residuals</code>	residuals

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

`equation(eqno|eqname)` specifies the equation to which you are referring.

`equation()` is filled in with one *eqno* or *eqname* for the `xb`, `stdp`, and `residuals` options. For example, `equation(#1)` would mean that the calculation is to be made for the first equation, `equation(#2)` would mean the second, and so on. You could also refer to the equation by its name; thus, `equation(income)` would refer to the equation named `income` and `equation(hours)`, to the equation named `hours`.

If you do not specify `equation()`, the results are the same as if you specified `equation(#1)`.

`xb`, the default, calculates the linear prediction for the specified equation.

`stdp` calculates the standard error of the linear prediction for the specified equation.

`residuals` calculates the residuals.

For more information on using `predict` after multiple-equation estimation commands, see [\[R\] predict](#).

Remarks

► Example 1

All the postestimation commands discussed in [\[TS\] var postestimation](#) work after `varbasic`. Suppose that we are interested in testing the hypothesis that there is no autocorrelation in the VAR disturbances. Continuing [example 1](#) from [\[TS\] varbasic](#), we now use `varlmar` to test this hypothesis.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. varbasic dln_inv dln_inc dln_consump if qtr<=tq(1978q4)
(output omitted)
. varlmar
```

Lagrange-multiplier test

lag	chi2	df	Prob > chi2
1	5.5871	9	0.78043
2	6.3189	9	0.70763

H0: no autocorrelation at lag order

Because we cannot reject the null hypothesis of no autocorrelation in the residuals, this test does not indicate any model misspecification.

◀

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Also see

[TS] [varbasic](#) — Fit a simple VAR and graph IRFS or FEVDs

[U] [20 Estimation and postestimation commands](#)

Title

vargranger — Perform pairwise Granger causality tests after var or svar

Syntax

```
vargranger [ , estimates(estname) separator(#) ]
```

vargranger can be used only after **var** or **svar**; see [TS] **var** and [TS] **var svar**.

Menu

Statistics > Multivariate time series > VAR diagnostics and tests > Granger causality tests

Description

vargranger performs a set of Granger causality tests for each equation in a VAR, providing a convenient alternative to **test**.

Options

estimates(*estname*) requests that **vargranger** use the previously obtained set of **var** or **svar** estimates saved as *estname*. By default, **vargranger** uses the active results. See [R] **estimates** for information about saving and restoring estimation results.

separator(#) specifies how often separator lines should be drawn between rows. By default, separator lines appear every *K* lines, where *K* is the number of equations in the VAR under analysis. For example, **separator**(1) would draw a line between each row, **separator**(2) between every other row, and so on. **separator**(0) specifies that lines not appear in the table.

Remarks

After fitting a VAR, we may want to know whether one variable “Granger-causes” another (Granger 1969). A variable *x* is said to Granger-cause a variable *y* if, given the past values of *y*, past values of *x* are useful for predicting *y*. A common method for testing Granger causality is to regress *y* on its own lagged values and on lagged values of *x* and test the null hypothesis that the estimated coefficients on the lagged values of *x* are jointly zero. Failure to reject the null hypothesis is equivalent to failing to reject the hypothesis that *x* does not Granger-cause *y*.

For each equation and each endogenous variable that is not the dependent variable in that equation, **vargranger** computes and reports Wald tests that the coefficients on all the lags of an endogenous variable are jointly zero. For each equation in a VAR, **vargranger** tests the hypotheses that each of the other endogenous variables does not Granger-cause the dependent variable in that equation.

Because it may be interesting to investigate these types of hypotheses by using the VAR that underlies an SVAR, **vargranger** can also produce these tests by using the **e()** results from an **svar**. When **vargranger** uses **svar e()** results, the hypotheses concern the underlying **var** estimates.

See [TS] **var** and [TS] **var svar** for information about fitting VARs and SVARs in Stata. See Lütkepohl (2005), Hamilton (1994), and Amisano and Giannini (1997) for information about Granger causality and on VARs and SVARs in general.

➤ Example 1: After var

Here we refit the model with German data described in [TS] var and then perform Granger causality tests with vargranger.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk small
(output omitted)
. vargranger
Granger causality Wald tests
```

Equation	Excluded	F	df	df_r	Prob > F
dln_inv	dln_inc	.04847	2	66	0.9527
dln_inv	dln_consump	1.5004	2	66	0.2306
dln_inv	ALL	1.5917	4	66	0.1869
dln_inc	dln_inv	1.7683	2	66	0.1786
dln_inc	dln_consump	1.7184	2	66	0.1873
dln_inc	ALL	1.9466	4	66	0.1130
dln_consump	dln_inv	.97147	2	66	0.3839
dln_consump	dln_inc	6.1465	2	66	0.0036
dln_consump	ALL	3.7746	4	66	0.0080

Because the estimates() option was not specified, vargranger used the active e() results. Consider the results of the three tests for the first equation. The first is a Wald test that the coefficients on the two lags of dln_inc that appear in the equation for dln_inv are jointly zero. The null hypothesis that dln_inc does not Granger-cause dln_inv cannot be rejected. Similarly, we cannot reject the null hypothesis that the coefficients on the two lags of dln_consump in the equation for dln_inv are jointly zero, so we cannot reject the hypothesis that dln_consump does not Granger-cause dln_inv. The third test is with respect to the null hypothesis that the coefficients on the two lags of all the other endogenous variables are jointly zero. Because this cannot be rejected, we cannot reject the null hypothesis that dln_inc and dln_consump, jointly, do not Granger-cause dln_inv.

Because we failed to reject most of these null hypotheses, we might be interested in imposing some constraints on the coefficients. See [TS] var for more on fitting VAR models with constraints on the coefficients.



➤ Example 2: Using test instead of vargranger

We could have used test to compute these Wald tests, but vargranger saves a great deal of typing. Still, seeing how to use test to obtain the results reported by vargranger is useful.

```
. test [dln_inv]L.dln_inc [dln_inv]L2.dln_inc
( 1) [dln_inv]L.dln_inc = 0
( 2) [dln_inv]L2.dln_inc = 0
      F( 2, 66) = 0.05
      Prob > F = 0.9527
```

```
. test [dln_inv]L.dln_consump [dln_inv]L2.dln_consump, accumulate
( 1) [dln_inv]L.dln_inc = 0
( 2) [dln_inv]L2.dln_inc = 0
( 3) [dln_inv]L.dln_consump = 0
( 4) [dln_inv]L2.dln_consump = 0
      F( 4, 66) = 1.59
      Prob > F = 0.1869

. test [dln_inv]L.dln_inv [dln_inv]L2.dln_inv, accumulate
( 1) [dln_inv]L.dln_inc = 0
( 2) [dln_inv]L2.dln_inc = 0
( 3) [dln_inv]L.dln_consump = 0
( 4) [dln_inv]L2.dln_consump = 0
( 5) [dln_inv]L.dln_inv = 0
( 6) [dln_inv]L2.dln_inv = 0
      F( 6, 66) = 1.62
      Prob > F = 0.1547
```

The first two calls to `test` show how `vargranger` obtains its results. The first test reproduces the first test reported for the `dln_inv` equation. The second test reproduces the ALL entry for the first equation. The third test reproduces the standard F statistic for the `dln_inv` equation, reported in the header of the `var` output in the previous example. The standard F statistic also includes the lags of the dependent variable, as well as any exogenous variables in the equation. This illustrates that the test performed by `vargranger` of the null hypothesis that the coefficients on all the lags of all the other endogenous variables are jointly zero for a particular equation; i.e., the ALL test is not the same as the standard F statistic for that equation.

◀

► Example 3: After svar

When `vargranger` is run on `svar` estimates, the null hypotheses are with respect to the underlying `var` estimates. We run `vargranger` after using `svar` to fit an SVAR that has the same underlying VAR as our model in the previous example.

```
. matrix A = (., 0,0 \ ., ., 0\ .,.,.)
. matrix B = I(3)
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk small aeq(A) beq(B)
(output omitted)
. vargranger
      Granger causality Wald tests
```

Equation	Excluded	F	df	df_r	Prob > F
dln_inv	dln_inc	.04847	2	66	0.9527
dln_inv	dln_consump	1.5004	2	66	0.2306
dln_inv	ALL	1.5917	4	66	0.1869
dln_inc	dln_inv	1.7683	2	66	0.1786
dln_inc	dln_consump	1.7184	2	66	0.1873
dln_inc	ALL	1.9466	4	66	0.1130
dln_consump	dln_inv	.97147	2	66	0.3839
dln_consump	dln_inc	6.1465	2	66	0.0036
dln_consump	ALL	3.7746	4	66	0.0080

As we expected, the `vargranger` results are identical to those in the first example.

▶

Saved results

vargranger saves the following in `r()`:

Matrices

`r(gstats)` χ^2 , df, and p -values (if `e(small)=""`)

`r(gstats)` F , df, `df_r`, and p -values (if `e(small)!=""`)

Methods and formulas

vargranger is implemented as an ado-file.

vargranger uses `test` to obtain Wald statistics of the hypothesis that all coefficients on the lags of variable x are jointly zero in the equation for variable y . **vargranger** uses the `e()` results saved by **var** or **svar** to determine whether to calculate and report small-sample F statistics or large-sample χ^2 statistics.

Clive William John Granger (1934–2009) was born in Swansea, Wales, and earned degrees at the University of Nottingham in mathematics and statistics. Joining the staff there, he also worked at Princeton on the spectral analysis of economic time series, before moving in 1973 to the University of California, San Diego. He was awarded the 2003 Nobel Prize in Economics for methods of analyzing economic time series with common trends (cointegration). He was knighted in 2005, thus becoming Sir Clive Granger.

References

- Amisano, G., and C. Giannini. 1997. *Topics in Structural VAR Econometrics*. 2nd ed. Heidelberg: Springer.
- Granger, C. W. J. 1969. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica* 37: 424–438.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Phillips, P. C. B. 1997. The ET Interview: Professor Clive Granger. *Econometric Theory* 13: 253–303.

Also see

- [TS] **var** — Vector autoregressive models
- [TS] **var svar** — Structural vector autoregressive models
- [TS] **varbasic** — Fit a simple VAR and graph IRFS or FEVDs
- [TS] **var intro** — Introduction to vector autoregressive models

Title

varlmar — Perform LM test for residual autocorrelation after var or svar

Syntax

`varlmar [ , options ]`

<i>options</i>	description
<code>m^{lag}(#)</code>	use # for the maximum order of autocorrelation; default is <code>m^{lag}(2)</code>
<code>estimates(<i>estname</i>)</code>	use previously saved results <i>estname</i> ; default is to use active results
<code>separator(#)</code>	draw separator line after every # rows

`varlmar` can be used only after `var` or `svar`; see [TS] `var` and [TS] `var svar`.
You must `tsset` your data before using `varlmar`; see [TS] `tsset`.

Menu

Statistics > Multivariate time series > VAR diagnostics and tests > LM test for residual autocorrelation

Description

`varlmar` implements a Lagrange-multiplier (LM) test for autocorrelation in the residuals of VAR models, which was presented in [Johansen \(1995\)](#).

Options

- `mlag(#)` specifies the maximum order of autocorrelation to be tested. The integer specified in `mlag()` must be greater than 0; the default is 2.
- `estimates(estname)` requests that `varlmar` use the previously obtained set of `var` or `svar` estimates saved as *estname*. By default, `varlmar` uses the active results. See [\[R\] estimates](#) for information on saving and restoring estimation results.
- `separator(#)` specifies how often separator lines should be drawn between rows. By default, separator lines do not appear. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on.

Remarks

Most postestimation analyses of VAR models and SVAR models assume that the disturbances are not autocorrelated. `varlmar` implements the LM test for autocorrelation in the residuals of a VAR model discussed in [Johansen \(1995, 21–22\)](#). The test is performed at lags $j = 1, \dots, m\text{lag}()$. For each j , the null hypothesis of the test is that there is no autocorrelation at lag j .

`varlmar` uses the estimation results saved by `var` or `svar`. By default, `varlmar` uses the active estimation results. However, `varlmar` can use any previously saved `var` or `svar` estimation results specified in the `estimates()` option.

➤ Example 1: After var

Here we refit the model with German data described in [TS] var and then call varlmar.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk
(output omitted)
. varlmar, mlag(5)
```

Lagrange-multiplier test

lag	chi2	df	Prob > chi2
1	5.5871	9	0.78043
2	6.3189	9	0.70763
3	8.4022	9	0.49418
4	11.8742	9	0.22049
5	5.2914	9	0.80821

H0: no autocorrelation at lag order

Because we cannot reject the null hypothesis that there is no autocorrelation in the residuals for any of the five orders tested, this test gives no hint of model misspecification. Although we fit the VAR with the dfk option to be consistent with the example in [TS] var, varlmar always uses the ML estimator of Σ . The results obtained from varlmar are the same whether or not dfk is specified.



➤ Example 2: After svar

When varlmar is applied to estimation results produced by svar, the sequence of LM tests is applied to the underlying VAR. See [TS] var svar for a description of how an SVAR model builds on a VAR. In this example, we fit an SVAR that has an underlying VAR with two lags that is identical to the one fit in the previous example.

```
. matrix A = (.,.,0\0,.,.,0\.,.,.)
. matrix B = I(3)
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk aeq(A) beq(B)
(output omitted)
. varlmar, mlag(5)
```

Lagrange-multiplier test

lag	chi2	df	Prob > chi2
1	5.5871	9	0.78043
2	6.3189	9	0.70763
3	8.4022	9	0.49418
4	11.8742	9	0.22049
5	5.2914	9	0.80821

H0: no autocorrelation at lag order

Because the underlying VAR(2) is the same as the previous example (we assure you that this is true), the output from varlmar is also the same.



Saved results

`varlmar` saves the following in `r()`:

Matrices

`r(lm)` χ^2 , df, and p -values

Methods and formulas

`varlmar` is implemented as an ado-file.

The formula for the LM test statistic at lag j is

$$LM_s = (T - d - .5) \ln \left(\frac{|\hat{\Sigma}|}{|\tilde{\Sigma}_s|} \right)$$

where T is the number of observations in the VAR; d is explained below; $\hat{\Sigma}$ is the maximum likelihood estimate of Σ , the variance–covariance matrix of the disturbances from the VAR; and $\tilde{\Sigma}_s$ is the maximum likelihood estimate of Σ from the following augmented VAR.

If there are K equations in the VAR, we can define $\mathbf{e}_t$ to be a $K \times 1$ vector of residuals. After we create the K new variables $\mathbf{e}1$, $\mathbf{e}2$, $\dots$, $\mathbf{e}K$ containing the residuals from the K equations, we can augment the original VAR with lags of these K new variables. For each lag s , we form an augmented regression in which the new residual variables are lagged s times. Per the method of [Davidson and MacKinnon \(1993, 358\)](#), the missing values from these s lags are replaced with zeros. $\tilde{\Sigma}_s$ is the maximum likelihood estimate of Σ from this augmented VAR, and d is the number of coefficients estimated in the augmented VAR. See [\[TS\] var](#) for a discussion of the maximum likelihood estimate of Σ in a VAR.

The asymptotic distribution of LM_s is χ^2 with K^2 degrees of freedom.

References

- Davidson, R., and J. G. MacKinnon. 1993. *Estimation and Inference in Econometrics*. New York: Oxford University Press.
- Johansen, S. 1995. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford: Oxford University Press.

Also see

- [\[TS\] var](#) — Vector autoregressive models
- [\[TS\] var svar](#) — Structural vector autoregressive models
- [\[TS\] varbasic](#) — Fit a simple VAR and graph IRFS or FEVDs
- [\[TS\] var intro](#) — Introduction to vector autoregressive models

Syntax

`varnorm [ , options ]`

<i>options</i>	description
<code>jbera</code>	report Jarque–Bera statistic; default is to report all three statistics
<code>skewness</code>	report skewness statistic; default is to report all three statistics
<code>kurtosis</code>	report kurtosis statistic; default is to report all three statistics
<code>estimates(<i>estname</i>)</code>	use previously saved results <i>estname</i> ; default is to use active results
<code>cholesky</code>	use Cholesky decomposition
<code>separator(#)</code>	draw separator line after every # rows

`varnorm` can be used only after `var` or `svar`; see [TS] [var](#) and [TS] [var svar](#).

You must `tsset` your data before using `varnorm`; see [TS] [tsset](#).

Menu

Statistics > Multivariate time series > VAR diagnostics and tests > Test for normally distributed disturbances

Description

`varnorm` computes and reports a series of statistics against the null hypothesis that the disturbances in a VAR are normally distributed. For each equation, and for all equations jointly, up to three statistics may be computed: a skewness statistic, a kurtosis statistic, and the Jarque–Bera statistic. By default, all three statistics are reported.

Options

`jbera` requests that the Jarque–Bera statistic and any other explicitly requested statistic be reported.

By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.

`skewness` requests that the skewness statistic and any other explicitly requested statistic be reported.

By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.

`kurtosis` requests that the kurtosis statistic and any other explicitly requested statistic be reported.

By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.

`estimates(estname)` specifies that `varnorm` use the previously obtained set of `var` or `svar` estimates saved as *estname*. By default, `varnorm` uses the active results. See [R] [estimates](#) for information on saving and restoring estimation results.

`cholesky` specifies that `varnorm` use the Cholesky decomposition of the estimated variance–covariance matrix of the disturbances, $\widehat{\Sigma}$, to orthogonalize the residuals when `varnorm` is applied to `svar` results. By default, when `varnorm` is applied to `svar` results, it uses the estimated structural decomposition $\widehat{\mathbf{A}}^{-1}\widehat{\mathbf{B}}$ on $\widehat{\mathbf{C}}$ to orthogonalize the residuals. When applied to `var e()` results, `varnorm` always uses the Cholesky decomposition of $\widehat{\Sigma}$. For this reason, the `cholesky` option may not be specified when using `var` results.

`separator(#)` specifies how often separator lines should be drawn between rows. By default, separator lines do not appear. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on.

Remarks

Some of the postestimation statistics for VAR and SVAR assume that the K disturbances have a K -dimensional multivariate normal distribution. `varnorm` uses the estimation results produced by `var` or `svar` to produce a series of statistics against the null hypothesis that the K disturbances in the VAR are normally distributed.

Per the notation in [Lütkepohl \(2005\)](#), call the skewness statistic $\hat{\lambda}_1$, the kurtosis statistic $\hat{\lambda}_2$, and the Jarque–Bera statistic $\hat{\lambda}_3$. The Jarque–Bera statistic is a combination of the other two statistics. The single-equation results are from tests against the null hypothesis that the disturbance for that particular equation is normally distributed. The results for all the equations are from tests against the null hypothesis that the K disturbances follow a K -dimensional multivariate normal distribution. Failure to reject the null hypothesis indicates a lack of model misspecification.

► Example 1: After var

We refit the model with German data described in [\[TS\] var](#) and then call `varnorm`.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk
(output omitted)
. varnorm
Jarque-Bera test
```

Equation	chi2	df	Prob > chi2
dln_inv	2.821	2	0.24397
dln_inc	3.450	2	0.17817
dln_consump	1.566	2	0.45702
ALL	7.838	6	0.25025

Skewness test

Equation	Skewness	chi2	df	Prob > chi2
dln_inv	.11935	0.173	1	0.67718
dln_inc	-.38316	1.786	1	0.18139
dln_consump	-.31275	1.190	1	0.27532
ALL		3.150	3	0.36913

Kurtosis test

Equation	Kurtosis	chi2	df	Prob > chi2
dln_inv	3.9331	2.648	1	0.10367
dln_inc	3.7396	1.664	1	0.19710
dln_consump	2.6484	0.376	1	0.53973
ALL		4.688	3	0.19613

dfk estimator used in computations

In this example, neither the single-equation Jarque–Bera statistics nor the joint Jarque–Bera statistic come close to rejecting the null hypothesis.

The skewness and kurtosis results have similar structures.

The Jarque–Bera results use the sum of the skewness and kurtosis statistics. The skewness and kurtosis results are based on the skewness and kurtosis coefficients, respectively. See [Methods and formulas](#).



➤ Example 2: After svar

The test statistics are computed on the orthogonalized VAR residuals; see [Methods and formulas](#). When `varnorm` is applied to `var` results, `varnorm` uses a Cholesky decomposition of the estimated variance–covariance matrix of the disturbances, $\hat{\Sigma}$, to orthogonalize the residuals.

By default, when `varnorm` is applied to `svar` estimation results, it uses the estimated structural decomposition $\hat{A}^{-1}\hat{B}$ on $\hat{C}$ to orthogonalize the residuals of the underlying VAR. Alternatively, when `varnorm` is applied to `svar` results and the `cholesky` option is specified, `varnorm` uses the Cholesky decomposition of $\hat{\Sigma}$ to orthogonalize the residuals of the underlying VAR.

We fit an SVAR that is based on an underlying VAR with two lags that is the same as the one fit in the previous example. We impose a structural decomposition that is the same as the Cholesky decomposition, as illustrated in [\[TS\] var svar](#).

```
. matrix a = (.,0,0\.,.,0\.,.,.)
. matrix b = I(3)
. svar dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk aeq(a) beq(b)
(output omitted)
. varnorm
Jarque-Bera test
```

Equation	chi2	df	Prob > chi2
dln_inv	2.821	2	0.24397
dln_inc	3.450	2	0.17817
dln_consump	1.566	2	0.45702
ALL	7.838	6	0.25025

Skewness test

Equation	Skewness	chi2	df	Prob > chi2
dln_inv	.11935	0.173	1	0.67718
dln_inc	-.38316	1.786	1	0.18139
dln_consump	-.31275	1.190	1	0.27532
ALL		3.150	3	0.36913

Kurtosis test

Equation	Kurtosis	chi2	df	Prob > chi2
dln_inv	3.9331	2.648	1	0.10367
dln_inc	3.7396	1.664	1	0.19710
dln_consump	2.6484	0.376	1	0.53973
ALL		4.688	3	0.19613

dfk estimator used in computations

Because the estimated structural decomposition is the same as the Cholesky decomposition, the `varnorm` results are the same as those from the previous example.



□ Technical note

The statistics computed by `varnorm` depend on $\hat{\Sigma}$, the estimated variance–covariance matrix of the disturbances. `var` uses the maximum likelihood estimator of this matrix by default, but the `dfk` option produces an estimator that uses a small-sample correction. Thus specifying `dfk` in the call to `var` or `svar` will affect the test results produced by `varnorm`.



Saved results

`varnorm` saves the following in `r()`:

Macros	
<code>r(dfk)</code>	<code>dfk</code> , if specified
Matrices	
<code>r(kurtosis)</code>	kurtosis test, df, and p -values
<code>r(skewness)</code>	skewness test, df, and p -values
<code>r(jb)</code>	Jarque–Bera test, df, and p -values

Methods and formulas

`varnorm` is implemented as an ado-file.

`varnorm` is based on the derivations found in [Lütkepohl \(2005, 174–181\)](#). Let $\hat{\mathbf{u}}_t$ be the $K \times 1$ vector of residuals from the K equations in a previously fitted VAR or the residuals from the K equations of the VAR underlying a previously fitted SVAR. Similarly, let $\hat{\Sigma}$ be the estimated covariance matrix of the disturbances. (Note that $\hat{\Sigma}$ depends on whether the `dfk` option was specified.) The skewness, kurtosis, and Jarque–Bera statistics must be computed using the orthogonalized residuals.

Because

$$\hat{\Sigma} = \hat{\mathbf{P}}\hat{\mathbf{P}}'$$

implies that

$$\hat{\mathbf{P}}^{-1}\hat{\Sigma}\hat{\mathbf{P}}^{-1'} = \mathbf{I}_K$$

premultiplying $\hat{\mathbf{u}}_t$ by $\hat{\mathbf{P}}$ is one way of performing the orthogonalization. When `varnorm` is applied to `var` results, $\hat{\mathbf{P}}$ is defined to be the Cholesky decomposition of $\hat{\Sigma}$. When `varnorm` is applied to `svar` results, $\hat{\mathbf{P}}$ is set, by default, to the estimated structural decomposition; i.e., $\hat{\mathbf{P}} = \hat{\mathbf{A}}^{-1}\hat{\mathbf{B}}$, where $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are the `svar` estimates of the $\mathbf{A}$ and $\mathbf{B}$ matrices, or $\hat{\mathbf{C}}$, where $\hat{\mathbf{C}}$ is the long-run SVAR estimation of $\mathbf{C}$. (See [\[TS\] var svar](#) for more on the origin and estimation of the $\mathbf{A}$ and $\mathbf{B}$ matrices.) When `varnorm` is applied to `svar` results and the `cholesky` option is specified, $\hat{\mathbf{P}}$ is set to the Cholesky decomposition of $\hat{\Sigma}$.

Define $\widehat{\mathbf{w}}_t$ to be the orthogonalized VAR residuals given by

$$\widehat{\mathbf{w}}_t = (\widehat{w}_{1t}, \dots, \widehat{w}_{Kt})' = \widehat{\mathbf{P}}^{-1} \widehat{\mathbf{u}}_t$$

The $K \times 1$ vectors of skewness and kurtosis coefficients are then computed using the orthogonalized residuals by

$$\begin{aligned} \widehat{\mathbf{b}}_1 &= (\widehat{b}_{11}, \dots, \widehat{b}_{K1})'; & \widehat{b}_{k1} &= \frac{1}{T} \sum_{i=1}^T \widehat{w}_{kt}^3 \\ \widehat{\mathbf{b}}_2 &= (\widehat{b}_{12}, \dots, \widehat{b}_{K2})'; & \widehat{b}_{k2} &= \frac{1}{T} \sum_{i=1}^T \widehat{w}_{kt}^4 \end{aligned}$$

Under the null hypothesis of multivariate Gaussian disturbances,

$$\widehat{\lambda}_1 = \frac{T \widehat{\mathbf{b}}_1' \widehat{\mathbf{b}}_1}{6} \xrightarrow{d} \chi^2(K)$$

$$\widehat{\lambda}_2 = \frac{T(\widehat{\mathbf{b}}_2 - 3)'(\widehat{\mathbf{b}}_2 - 3)}{24} \xrightarrow{d} \chi^2(K)$$

and

$$\widehat{\lambda}_3 = \widehat{\lambda}_1 + \widehat{\lambda}_2 \xrightarrow{d} \chi^2(2K)$$

$\widehat{\lambda}_1$ is the skewness statistic, $\widehat{\lambda}_2$ is the kurtosis statistic, and $\widehat{\lambda}_3$ is the Jarque–Bera statistic.

$\widehat{\lambda}_1$, $\widehat{\lambda}_2$, and $\widehat{\lambda}_3$ are for tests of the null hypothesis that the $K \times 1$ vector of disturbances follows a multivariate normal distribution. The corresponding statistics against the null hypothesis that the disturbances from the k th equation come from a univariate normal distribution are

$$\widehat{\lambda}_{1k} = \frac{T \widehat{b}_{k1}^2}{6} \xrightarrow{d} \chi^2(1)$$

$$\widehat{\lambda}_{2k} = \frac{T(\widehat{b}_{k2}^2 - 3)^2}{24} \xrightarrow{d} \chi^2(1)$$

and

$$\widehat{\lambda}_{3k} = \widehat{\lambda}_{1k} + \widehat{\lambda}_{2k} \xrightarrow{d} \chi^2(2)$$

References

- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Jarque, C. M., and A. K. Bera. 1987. A test for normality of observations and regression residuals. *International Statistical Review* 2: 163–172.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Also see

[TS] [var](#) — Vector autoregressive models

[TS] [var svar](#) — Structural vector autoregressive models

[TS] [varbasic](#) — Fit a simple VAR and graph IRFS or FEVDs

[TS] [var intro](#) — Introduction to vector autoregressive models

Syntax

Preestimation syntax

```
varsoc depvarlist [if] [in] [, preestimation_options]
```

Postestimation syntax

```
varsoc [, estimates(estname)]
```

<i>preestimation_options</i>	description

Main

<u>maxlag</u> (#)	set maximum lag order to #; default is maxlag(4)
<u>exog</u> (<i>varlist</i>)	use <i>varlist</i> as exogenous variables
<u>constraints</u> (<i>constraints</i>)	apply constraints to exogenous variables
<u>noconstant</u>	suppress constant term
<u>lutstats</u>	use Lütkepohl’s version of information criteria
<u>level</u> (#)	set confidence level; default is level(95)
<u>separator</u> (#)	draw separator line after every # rows

You must `tsset` your data before using `varsoc`; see [TS] `tsset`.
by is allowed with the preestimation version of `varsoc`; see [U] 11.1.10 Prefix commands.

Menu

Preestimation for VARs

Statistics > Multivariate time series > VAR diagnostics and tests > Lag-order selection statistics (preestimation)

Postestimation for VARs

Statistics > Multivariate time series > VAR diagnostics and tests > Lag-order selection statistics (postestimation)

Preestimation for VECMs

Statistics > Multivariate time series > VEC diagnostics and tests > Lag-order selection statistics (preestimation)

Postestimation for VECMs

Statistics > Multivariate time series > VEC diagnostics and tests > Lag-order selection statistics (postestimation)

Description

`varsoc` reports the final prediction error (FPE), Akaike’s information criterion (AIC), Schwarz’s Bayesian information criterion (SBIC), and the Hannan and Quinn information criterion (HQIC) lag-order selection statistics for a series of vector autoregressions of order 1, . . . , `maxlag()`. A sequence of likelihood-ratio test statistics for all the full VARs of order less than or equal to the highest lag order is also reported. In the postestimation version, the maximum lag and estimation options are based on the model just fit or the model specified in `estimates(estname)`.

The preestimation version of `varsoc` can also be used to select the lag order for a vector error-correction model (VECM). As shown by [Nielsen \(2001\)](#), the lag-order selection statistics discussed here can be used in the presence of I(1) variables.

Preestimation options

Main

`maxlag(#)` specifies the maximum lag order for which the statistics are to be obtained.

`exog(varlist)` specifies exogenous variables to include in the VARs fit by `varsoc`.

`constraints(constraints)` specifies a list of constraints on the exogenous variables to be applied. Do not specify constraints on the lags of the endogenous variables because specifying one would mean that at least one of the VAR models considered by `varsoc` will not contain the lag specified in the constraint. Use [\[TS\] var](#) directly to obtain selection-order criteria with constraints on lags of the endogenous variables.

`noconstant` suppresses the constant terms from the model. By default, constant terms are included.

`lutstats` specifies that the [Lütkepohl \(2005\)](#) versions of the lag-order selection statistics be reported. See [Methods and formulas](#) for a discussion of these statistics.

`level(#)` specifies the confidence level, as a percentage, that is used to identify the first likelihood-ratio test that rejects the null hypothesis that the additional parameters from adding a lag are jointly zero. The default is `level(95)` or as set by `set level`; see [\[U\] 20.7 Specifying the width of confidence intervals](#).

`separator(#)` specifies how often separator lines should be drawn between rows. By default, separator lines do not appear. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on.

Postestimation option

`estimates(estname)` specifies the name of a previously stored set of `var` or `svar` estimates. When no `depvarlist` is specified, `varsoc` uses the *postestimation* syntax and uses the currently active estimation results or the results specified in `estimates(estname)`. See [\[R\] estimates](#) for information about saving and restoring estimation results.

Remarks

Many selection-order statistics have been developed to assist researchers in fitting a VAR of the correct order. Several of these selection-order statistics appear in the [\[TS\] var](#) output. The `varsoc` command computes these statistics over a range of lags p while maintaining a common sample and option specification.

`varsoc` can be used as a preestimation or a postestimation command. When it is used as a preestimation command, a `depvarlist` is required, and the default maximum lag is 4. When it is used as a postestimation command, `varsoc` uses the model specification stored in `estname` or the previously fitted model.

`varsoc` computes four information criteria as well as a sequence of likelihood ratio (LR) tests. The information criteria include the FPE, AIC, the HQIC, and SBIC.

For a given lag p , the LR test compares a VAR with p lags with one with $p - 1$ lags. The null hypothesis is that all the coefficients on the p th lags of the endogenous variables are zero. To use this sequence of LR tests to select a lag order, we start by looking at the results of the test for the model with the most lags, which is at the bottom of the table. Proceeding up the table, the first test that rejects the null hypothesis is the lag order selected by this process. See [Lütkepohl \(2005, 143–144\)](#) for more information on this procedure. An ‘*’ appears next to the LR statistic indicating the optimal lag.

For the remaining statistics, the lag with the smallest value is the order selected by that criterion. An ‘*’ indicates the optimal lag. Strictly speaking, the FPE is not an information criterion, though we include it in this discussion because, as with an information criterion, we select the lag length corresponding to the lowest value; and, naturally, we want to minimize the prediction error. The AIC measures the discrepancy between the given model and the true model, which, of course, we want to minimize. [Amemiya \(1985\)](#) provides an intuitive discussion of the arguments in [Akaike \(1973\)](#). The SBIC and the HQIC can be interpreted similarly to the AIC, though the SBIC and the HQIC have a theoretical advantage over the AIC and the FPE. As [Lütkepohl \(2005, 148–152\)](#) demonstrates, choosing p to minimize the SBIC or the HQIC provides consistent estimates of the true lag order, p . In contrast, minimizing the AIC or the FPE will overestimate the true lag order with positive probability, even with an infinite sample size.

► Example 1: Preestimation

Here we use `varsoc` as a preestimation command.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. varsoc dln_inv dln_inc dln_consump if qtr<=tq(1978q4), lutstats

Selection-order criteria (lutstats)
Sample: 1961q2 - 1978q4                                Number of obs      =           71
```

lag	LL	LR	df	p	FPE	AIC	HQIC	SBIC
0	564.784				2.7e-11	-24.423	-24.423*	-24.423*
1	576.409	23.249	9	0.006	2.5e-11	-24.497	-24.3829	-24.2102
2	588.859	24.901*	9	0.003	2.3e-11*	-24.5942*	-24.3661	-24.0205
3	591.237	4.7566	9	0.855	2.7e-11	-24.4076	-24.0655	-23.5472
4	598.457	14.438	9	0.108	2.9e-11	-24.3575	-23.9012	-23.2102

```
Endogenous:  dln_inv dln_inc dln_consump
Exogenous:   _cons
```

The sample used begins in 1961q2 because all the VARs are fit to the sample defined by any `if` or `in` conditions and the available data for the maximum lag specified. The default maximum number of lags is four. Because we specified the `lutstats` option, the table contains the [Lütkepohl \(2005\)](#) versions of the information criteria, which differ from the standard definitions in that they drop the constant term from the log likelihood. In this example, the likelihood-ratio tests selected a model with two lags. AIC and FPE have also both chosen a model with two lags, whereas SBIC and HQIC have both selected a model with zero lags.



► Example 2: Postestimation

`varsoc` works as a postestimation command when no dependent variables are specified.

```
. var dln_inc dln_consump if qtr<=tq(1978q4), lutstats exog(l.dln_inv)
(output omitted)
. varsoc
Selection-order criteria (lutstats)
Sample: 1960q4 - 1978q4                Number of obs      =          73
```

lag	LL	LR	df	p	FPE	AIC	HQIC	SBIC
0	460.646				1.3e-08	-18.2962	-18.2962	-18.2962*
1	467.606	13.919	4	0.008	1.2e-08	-18.3773	-18.3273	-18.2518
2	477.087	18.962*	4	0.001	1.0e-08*	-18.5275*	-18.4274*	-18.2764

```
Endogenous:  dln_inc dln_consump
Exogenous:   L.dln_inv _cons
```

Because we included one lag of `dln_inv` in our original model, `varsoc` did likewise with each model it fit.



Based on the work of [Tsay \(1984\)](#), [Paulsen \(1984\)](#), and [Nielsen \(2001\)](#), these lag-order selection criteria can be used to determine the lag length of the VAR underlying a VECM. See [\[TS\] vec intro](#) for an example in which we use `varsoc` to choose the lag order for a VECM.

Saved results

`varsoc` saves the following in `r()`:

Scalars			
<code>r(N)</code>	number of observations	<code>r(mlag)</code>	maximum lag order
<code>r(tmax)</code>	last time period in sample	<code>r(N_gaps)</code>	the number of gaps in the sample
<code>r(tmin)</code>	first time period in sample		
Macros			
<code>r(endog)</code>	names of endogenous variables	<code>r(exog)</code>	names of exogenous variables
<code>r(lutstats)</code>	lutstats, if specified	<code>r(rmlutstats)</code>	rmlutstats, if specified
<code>r(cns#)</code>	the #th constraint		
Matrices			
<code>r(stats)</code>	LL, LR, FPE, AIC, HQIC, SBIC, and <i>p</i> -values		

Methods and formulas

`varsoc` is implemented as an ado-file.

As shown by [Hamilton \(1994, 295–296\)](#), the log likelihood for a VAR(*p*) is

$$LL = \left(\frac{T}{2}\right) \left\{ \ln(|\widehat{\Sigma}^{-1}|) - K\ln(2\pi) - K \right\}$$

where T is the number of observations, K is the number of equations, and $\widehat{\Sigma}$ is the maximum likelihood estimate of $E[\mathbf{u}_t \mathbf{u}_t']$, where $\mathbf{u}_t$ is the $K \times 1$ vector of disturbances. Because

$$\ln(|\widehat{\Sigma}^{-1}|) = -\ln(|\widehat{\Sigma}|)$$

the log likelihood can be rewritten as

$$\text{LL} = -\left(\frac{T}{2}\right) \left\{ \ln(|\widehat{\Sigma}|) + K \ln(2\pi) + K \right\}$$

Letting $\text{LL}(j)$ be the value of the log likelihood with j lags yields the LR statistic for lag order j as

$$\text{LR}(j) = 2\{\text{LL}(j) - \text{LL}(j-1)\}$$

Model-order statistics

The formula for the FPE given in [Lütkepohl \(2005, 147\)](#) is

$$\text{FPE} = |\Sigma_u| \left(\frac{T + Kp + 1}{T - Kp - 1} \right)^K$$

This formula, however, assumes that there is a constant in the model and that none of the variables are dropped because of collinearity. To deal with these problems, the FPE is implemented as

$$\text{FPE} = |\Sigma_u| \left(\frac{T + \overline{m}}{T - \overline{m}} \right)^K$$

where $\overline{m}$ is the average number of parameters over the K equations. This implementation accounts for variables dropped because of collinearity.

By default, the AIC, SBIC, and HQIC are computed according to their standard definitions, which include the constant term from the log likelihood. That is,

$$\begin{aligned} \text{AIC} &= -2 \left(\frac{\text{LL}}{T} \right) + \frac{2t_p}{T} \\ \text{SBIC} &= -2 \left(\frac{\text{LL}}{T} \right) + \frac{\ln(T)}{T} t_p \\ \text{HQIC} &= -2 \left(\frac{\text{LL}}{T} \right) + \frac{2\ln\{\ln(T)\}}{T} t_p \end{aligned}$$

where t_p is the total number of parameters in the model and LL is the log likelihood.

Lutstats

[Lütkepohl \(2005\)](#) advocates dropping the constant term from the log likelihood because it does not affect inference. The Lütkepohl versions of the information criteria are

$$\begin{aligned} \text{AIC} &= \ln(|\Sigma_u|) + \frac{2pK^2}{T} \\ \text{SBIC} &= \ln(|\Sigma_u|) + \frac{\ln(T)}{T} pK^2 \\ \text{HQIC} &= \ln(|\Sigma_u|) + \frac{2\ln\{\ln(T)\}}{T} pK^2 \end{aligned}$$

References

- Akaike, H. 1973. Information theory and an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory*, ed. B. N. Petrov and F. Csaki, 267–281. Budapest: Akailseoniai–Kiudo.
- Amemiya, T. 1985. *Advanced Econometrics*. Cambridge, MA: Harvard University Press.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Nielsen, B. 2001. Order determination in general vector autoregressions. Working paper, Department of Economics, University of Oxford and Nuffield College. <http://ideas.repec.org/p/nuf/econwp/0110.html>.
- Paulsen, J. 1984. Order determination of multivariate autoregressive time series with unit roots. *Journal of Time Series Analysis* 5: 115–127.
- Tsay, R. S. 1984. Order selection in nonstationary autoregressive models. *Annals of Statistics* 12: 1425–1433.

Also see

- [TS] **var** — Vector autoregressive models
- [TS] **var svar** — Structural vector autoregressive models
- [TS] **varbasic** — Fit a simple VAR and graph IRFS or FEVDs
- [TS] **vec** — Vector error-correction models
- [TS] **var intro** — Introduction to vector autoregressive models
- [TS] **vec intro** — Introduction to vector error-correction models

Title

varstable — Check the stability condition of VAR or SVAR estimates

Syntax

```
varstable [ , options ]
```

options	description
Main	
<code>estimates(estname)</code>	use previously saved results <i>estname</i> ; default is to use active results
<code>amat(matrix_name)</code>	save the companion matrix as <i>matrix_name</i>
<code>graph</code>	graph eigenvalues of the companion matrix
<code>dlabel</code>	label eigenvalues with the distance from the unit circle
<code>modlabel</code>	label eigenvalues with the modulus
<code>marker_options</code>	change look of markers (color, size, etc.)
<code>rlopts(cline_options)</code>	affect rendition of reference unit circle
<code>nogrid</code>	suppress polar grid circles
<code>pgrid([...])</code>	specify radii and appearance of polar grid circles; see <i>Options</i> for details
Add plots	
<code>addplot(plot)</code>	add other plots to the generated graph
Y axis, X axis, Titles, Legend, Overall	
<code>twoway_options</code>	any options other than <code>by()</code> documented in [G] twoway_options

varstable can be used only after `var` or `svar`; see [\[TS\] var](#) and [\[TS\] var svar](#).

Menu

Statistics > Multivariate time series > VAR diagnostics and tests > Check stability condition of VAR estimates

Description

`varstable` checks the eigenvalue stability condition after estimating the parameters of a vector autoregression using `var` or `svar`.

Options

Main
<code>estimates(estname)</code> requests that <code>varstable</code> use the previously obtained set of <code>var</code> estimates saved as <i>estname</i> . By default, <code>varstable</code> uses the active estimation results. See [R] estimates for information about saving and restoring estimation results.
<code>amat(matrix_name)</code> specifies a valid Stata matrix name by which the companion matrix A can be saved (see Methods and formulas for the definition of the matrix A). The default is not to save the A matrix.

`graph` causes `varstable` to draw a graph of the eigenvalues of the companion matrix.

`dlabel` labels each eigenvalue with its distance from the unit circle. `dlabel` cannot be specified with `modlabel`.

`modlabel` labels the eigenvalues with their moduli. `modlabel` cannot be specified with `dlabel`.

`marker_options` specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G] [marker_options](#).

`rlopts(cline_options)` affect the rendition of the reference unit circle; see [G] [cline_options](#).

`nogrid` suppresses the polar grid circles.

`pgrid([numlist] [,line_options])` determines the radii and appearance of the polar grid circles. By default, the graph includes nine polar grid circles with radii .1, .2, ..., .9 that have the `grid` line style. The `numlist` specifies the radii for the polar grid circles. The `line_options` determine the appearance of the polar grid circles; see [G] [line_options](#). Because the `pgrid()` option can be repeated, circles with different radii can have distinct appearances.

Add plots

`addplot(plot)` adds specified plots to the generated graph. See [G] [addplot_option](#).

Y axis, X axis, Titles, Legend, Overall

`twoway_options` are any of the options documented in [G] [twoway_options](#), except by(). These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

Remarks

Inference after `var` and `svar` requires that variables be covariance stationary. The variables in y_t are covariance stationary if their first two moments exist and are independent of time. More explicitly, a variable y_t is covariance stationary if

1. $E[y_t]$ is finite and independent of t .
2. $\text{Var}[y_t]$ is finite and independent of t
3. $\text{Cov}[y_t, y_s]$ is a finite function of $|t - s|$ but not of t or s alone.

Interpretation of VAR models, however, requires that an even stricter stability condition be met. If a VAR is stable, it is invertible and has an infinite-order vector moving-average representation. If the VAR is stable, impulse–response functions and forecast-error variance decompositions have known interpretations.

Lütkepohl (2005) and Hamilton (1994) both show that if the modulus of each eigenvalue of the matrix $\mathbf{A}$ is strictly less than one, the estimated VAR is stable (see [Methods and formulas](#) for the definition of the matrix $\mathbf{A}$).

► Example 1

After fitting a VAR with `var`, we can use `varstable` to check the stability condition. Using the same VAR model that was used in [TS] `var`, we demonstrate the use of `varstable`.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr>=tq(1961q2) & qtr<=tq(1978q4)
(output omitted)
```

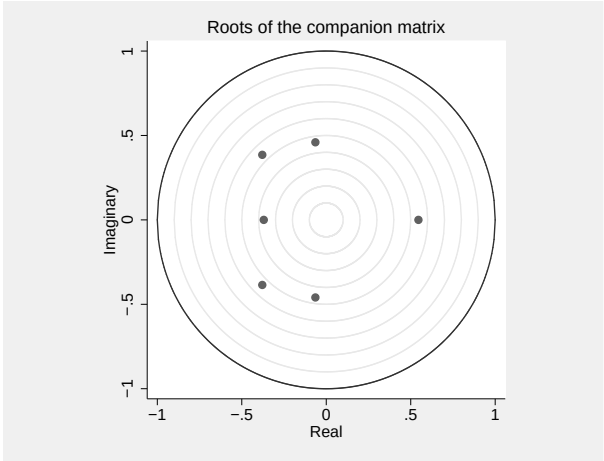
```
. varstable, graph
Eigenvalue stability condition
```

Eigenvalue	Modulus
.5456253	.545625
-.3785754 + .3853982i	.540232
-.3785754 - .3853982i	.540232
-.0643276 + .4595944i	.464074
-.0643276 - .4595944i	.464074
-.3698058	.369806

All the eigenvalues lie inside the unit circle.
VAR satisfies stability condition.

Because the modulus of each eigenvalue is strictly less than 1, the estimates satisfy the eigenvalue stability condition.

Specifying the `graph` option produced a graph of the eigenvalues with the real components on the x axis and the complex components on the y axis. The graph below indicates visually that these eigenvalues are well inside the unit circle.



➤ Example 2

This example illustrates two other features of the `varstable` command. First, `varstable` can check the stability of the estimates of the VAR underlying an SVAR fit by `var svar`. Second, `varstable` can check the stability of any previously stored `var` or `var svar` estimates.

We begin by refitting the previous VAR and storing the results as `var1`. Because this is the same VAR that was fit in the previous example, the stability results should be identical.

```
. var dln_inv dln_inc dln_consump if qtr>=tq(1961q2) & qtr<=tq(1978q4)
(output omitted)
. estimates store var1
```

Now we use `sva` to fit an SVAR with a different underlying VAR and check the estimates of that underlying VAR for stability.

```
. matrix A = (.,0\.,.)
. matrix B = I(2)
. svar d.ln_inc d.ln_consump, aeq(A) beq(B)
(output omitted)
. varstable
Eigenvalue stability condition
```

Eigenvalue	Modulus
.548711	.548711
-.2979493 + .4328013i	.525443
-.2979493 - .4328013i	.525443
-.3570825	.357082

All the eigenvalues lie inside the unit circle.
VAR satisfies stability condition.

The `estimates()` option allows us to check the stability of the `var` results stored as `var1`.

```
. varstable, est(var1)
Eigenvalue stability condition
```

Eigenvalue	Modulus
.5456253	.545625
-.3785754 + .3853982i	.540232
-.3785754 - .3853982i	.540232
-.0643276 + .4595944i	.464074
-.0643276 - .4595944i	.464074
-.3698058	.369806

All the eigenvalues lie inside the unit circle.
VAR satisfies stability condition.

The results are identical to those obtained in the previous example, confirming that we were checking the results in `var1`.

◀

Saved results

`varstable` saves the following in `r()`:

Matrices

`r(Re)` real part of the eigenvalues of A
`r(Im)` imaginary part of the eigenvalues of A
`r(Modulus)` modulus of the eigenvalues of A

Methods and formulas

`varstable` is implemented as an ado-file.

`varstable` forms the companion matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \dots & \mathbf{A}_{p-1} & \mathbf{A}_p \\ \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \mathbf{0} \end{pmatrix}$$

and obtains its eigenvalues by using `matrix eigenvalues`. The modulus of the complex eigenvalue $r + ci$ is $\sqrt{r^2 + c^2}$. As shown by [Lütkepohl \(2005\)](#) and [Hamilton \(1994\)](#), the VAR is stable if the modulus of each eigenvalue of $\mathbf{A}$ is strictly less than 1.

References

- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- . 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Also see

- [TS] [var](#) — Vector autoregressive models
- [TS] [var svar](#) — Structural vector autoregressive models
- [TS] [varbasic](#) — Fit a simple VAR and graph IRFS or FEVDs
- [TS] [var intro](#) — Introduction to vector autoregressive models

Title

varwle — Obtain Wald lag-exclusion statistics after var or svar

Syntax

```
varwle [ , estimates(estname) separator(#) ]
```

`varwle` can be used only after `var` or `svar`; see [\[TS\] var](#) and [\[TS\] var svar](#).

Menu

Statistics > Multivariate time series > VAR diagnostics and tests > Wald lag-exclusion statistics

Description

`varwle` reports Wald tests the hypothesis that the endogenous variables at a given lag are jointly zero for each equation and for all equations jointly.

Options

`estimates(estname)` requests that `varwle` use the previously obtained set of `var` or `svar` estimates saved as *estname*. By default, `varwle` uses the active estimation results. See [\[R\] estimates](#) for information about saving and restoring estimation results.

`separator(#)` specifies how often separator lines should be drawn between rows. By default, separator lines do not appear. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on.

Remarks

After fitting a VAR, one hypothesis of interest is that all the endogenous variables at a given lag are jointly zero. `varwle` reports Wald tests of this hypothesis for each equation and for all equations jointly. `varwle` uses the estimation results from a previously fitted `var` or `svar`. By default, `varwle` uses the active estimation results, but you may also use a stored set of estimates by specifying the `estimates()` option.

If the VAR was fit with the `small` option, `varwle` also presents small-sample F statistics; otherwise, `varwle` presents large-sample chi-squared statistics.

► Example 1: After var

We analyze the model with the German data described in [\[TS\] var](#) using `varwle`.

```
. use http://www.stata-press.com/data/r11/lutkepohl2
(Quarterly SA West German macro data, Bil DM, from Lutkepohl 1993 Table E.1)
. var dln_inv dln_inc dln_consump if qtr<=tq(1978q4), dfk small
(output omitted)
. varwle
```

Equation: dln_inv

lag	F	df	df_r	Prob > F
1	2.64902	3	66	0.0560
2	1.25799	3	66	0.2960

Equation: dln_inc

lag	F	df	df_r	Prob > F
1	2.19276	3	66	0.0971
2	.907499	3	66	0.4423

Equation: dln_consump

lag	F	df	df_r	Prob > F
1	1.80804	3	66	0.1543
2	5.57645	3	66	0.0018

Equation: All

lag	F	df	df_r	Prob > F
1	3.78884	9	66	0.0007
2	2.96811	9	66	0.0050

Because the VAR was fit with the `dfk` and `small` options, `varwle` used the small-sample estimator of $\hat{\Sigma}$ in constructing the VCE, producing an F statistic. The first two equations appear to have a different lag structure from that of the third. In the first two equations, we cannot reject the null hypothesis that all three endogenous variables have zero coefficients at the second lag. The hypothesis that all three endogenous variables have zero coefficients at the first lag can be rejected at the 10% level for both of the first two equations. In contrast, in the third equation, the coefficients on the second lag of the endogenous variables are jointly significant, but not those on the first lag. However, we strongly reject the hypothesis that the coefficients on the first lag of the endogenous variables are zero in all three equations jointly. Similarly, we can also strongly reject the hypothesis that the coefficients on the second lag of the endogenous variables are zero in all three equations jointly.

If we believe these results strongly enough, we might want to refit the original VAR, placing some constraints on the coefficients. See [TS] [var](#) for details on how to fit VAR models with constraints.



➤ Example 2: After svar

Here we fit a simple SVAR and then run `varwle`:

```
. matrix a = (.,0\.,.)
. matrix b = I(2)
```

```
. svar dln_inc dln_consump, aeq(a) beq(b)
Estimating short-run parameters
Iteration 0:  log likelihood = -159.21683
Iteration 1:  log likelihood = 490.92264
Iteration 2:  log likelihood = 528.66126
Iteration 3:  log likelihood = 573.96363
Iteration 4:  log likelihood = 578.05136
Iteration 5:  log likelihood = 578.27633
Iteration 6:  log likelihood = 578.27699
Iteration 7:  log likelihood = 578.27699
```

Structural vector autoregression

```
( 1) [a_1_2]_cons = 0
( 2) [b_1_1]_cons = 1
( 3) [b_1_2]_cons = 0
( 4) [b_2_1]_cons = 0
( 5) [b_2_2]_cons = 1
```

Sample: 1960q4 - 1982q4
Exactly identified model

No. of obs = 89
Log likelihood = 578.277

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
/a_1_1	89.72411	6.725107	13.34	0.000	76.54315	102.9051
/a_2_1	-64.73622	10.67698	-6.06	0.000	-85.66271	-43.80973
/a_1_2	0	.	.	.	.	.
/a_2_2	126.2964	9.466318	13.34	0.000	107.7428	144.8501
/b_1_1	1	.	.	.	.	.
/b_2_1	0	.	.	.	.	.
/b_1_2	0	.	.	.	.	.
/b_2_2	1	.	.	.	.	.

The output table from `var svar` gives information about the estimates of the parameters in the **A** and **B** matrices in the structural VAR. But, as discussed in [TS] [var svar](#), an SVAR model builds on an underlying VAR. When `varwle` uses the estimation results produced by `svar`, it performs Wald lag-exclusion tests on the underlying VAR model. Next we run `varwle` on these `svar` results.

```
. varwle
```

Equation: dln_inc

lag	chi2	df	Prob > chi2
1	6.88775	2	0.032
2	1.873546	2	0.392

Equation: dln_consump

lag	chi2	df	Prob > chi2
1	9.938547	2	0.007
2	13.89996	2	0.001

Equation: All

lag	chi2	df	Prob > chi2
1	34.54276	4	0.000
2	19.44093	4	0.001

Now we fit the underlying VAR with two lags and apply `varwle` to these results.

```
. var dln_inc dln_consump
(output omitted)
. varwle
```

Equation: dln_inc

lag	chi2	df	Prob > chi2
1	6.88775	2	0.032
2	1.873546	2	0.392

Equation: dln_consump

lag	chi2	df	Prob > chi2
1	9.938547	2	0.007
2	13.89996	2	0.001

Equation: All

lag	chi2	df	Prob > chi2
1	34.54276	4	0.000
2	19.44093	4	0.001

Because `varwle` produces the same results in these two cases, we can conclude that when `varwle` is applied to `svar` results, it performs Wald lag-exclusion tests on the underlying VAR.



Saved results

`varwle` saves the following in `r()`:

```
Matrices
if e(small)==" "
    r(chi2)    χ² test statistics
    r(df)      degrees of freedom
    r(p)       p-values
if e(small)!=" "
    r(F)       F test statistics
    r(df_r)    numerator degrees of freedom
    r(df)      denominator degree of freedom
    r(p)       p-values
```

Methods and formulas

`varwle` is implemented as an ado-file.

`varwle` uses `test` to obtain Wald statistics of the hypotheses that all the endogenous variables at a given lag are jointly zero for each equation and for all equations jointly. Like the `test` command, `varwle` uses estimation results saved by `var` or `var svar` to determine whether to calculate and report small-sample F statistics or large-sample chi-squared statistics.

Abraham Wald (1902–1950) was born in Cluj, in what is now Romania. He studied mathematics at the University of Vienna, publishing at first on geometry, but then became interested in economics and econometrics. He moved to the United States in 1938 and later joined the faculty at Columbia. His major contributions to statistics include work in decision theory, optimal sequential sampling, large-sample distributions of likelihood-ratio tests, and nonparametric inference. Wald died in a plane crash in India.

References

- Amisano, G., and C. Giannini. 1997. *Topics in Structural VAR Econometrics*. 2nd ed. Heidelberg: Springer.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Lütkepohl, H. 1993. *Introduction to Multiple Time Series Analysis*. 2nd ed. New York: Springer.
- Mangel, M., and F. J. Samaniego. 1984. Abraham Wald's work on aircraft survivability. *Journal of the American Statistical Association* 79: 259–267.
- Wolfowitz, J. 1952. Abraham Wald, 1902–1950. *Annals of Mathematical Statistics* 23: 1–13 (and other reports in same issue).

Also see

- [TS] [var](#) — Vector autoregressive models
- [TS] [var svar](#) — Structural vector autoregressive models
- [TS] [varbasic](#) — Fit a simple VAR and graph IRFS or FEVDs
- [TS] [var intro](#) — Introduction to vector autoregressive models

Description

Stata has a suite of commands for fitting, forecasting, interpreting, and performing inference on vector error-correction models (VECMs) with cointegrating variables . After fitting a VECM, the `irf` commands can be used to obtain impulse–response functions (IRFs) and forecast-error variance decompositions (FEVDs). The table below describes the available commands.

Fitting a VECM

<code>vec</code>	[TS] vec	Fit vector error-correction models

Model diagnostics and inference

<code>vecrank</code>	[TS] vecrank	Estimate the cointegrating rank of a VECM
<code>veclmar</code>	[TS] veclmar	Perform LM test for residual autocorrelation after <code>vec</code>
<code>vecnorm</code>	[TS] vecnorm	Test for normally distributed disturbances after <code>vec</code>
<code>vecstable</code>	[TS] vecstable	Check the stability condition of VECM estimates
<code>varsoc</code>	[TS] varsoc	Obtain lag-order selection statistics for VARs and VECMs

Forecasting from a VECM

<code>fcast compute</code>	[TS] fcast compute	Compute dynamic forecasts of dependent variables after <code>var</code> , <code>svar</code> , or <code>vec</code>
<code>fcast graph</code>	[TS] fcast graph	Graph forecasts of variables computed by <code>fcast compute</code>

Working with IRFs and FEVDs

<code>irf</code>	[TS] irf	Create and analyze IRFs and FEVDs

This manual entry provides an overview of the commands for VECMs; provides an introduction to integration, cointegration, estimation, inference, and interpretation of VECM models; and gives an example of how to use Stata’s `vec` commands.

Remarks

`vec` estimates the parameters of cointegrating VECMs. You may specify any of the five trend specifications in Johansen (1995, sec. 5.7). By default, identification is obtained via the Johansen normalization, but `vec` allows you to obtain identification by placing your own constraints on the parameters of the cointegrating vectors. You may also put more restrictions on the adjustment coefficients.

`vecrank` is the command for determining the number of cointegrating equations. `vecrank` implements Johansen’s multiple trace test procedure, the maximum eigenvalue test, and a method based on minimizing either of two different information criteria.

Because [Nielsen \(2001\)](#) has shown that the methods implemented in `varsoc` can be used to choose the order of the autoregressive process, no separate `vec` command is needed; you can simply use `varsoc`. `vecldmar` tests that the residuals have no serial correlation, and `vecnorm` tests that they are normally distributed.

All the `irf` routines described in [\[TS\]](#) `irf` are available for estimating, interpreting, and managing estimated IRFs and FEVDs for VECMs.

Remarks are presented under the following headings:

- Introduction to cointegrating VECMs*
 - What is cointegration?*
 - The multivariate VECM specification*
 - Trends in the Johansen VECM framework*
- VECM estimation in Stata*
 - Selecting the number of lags*
 - Testing for cointegration*
 - Fitting a VECM*
 - Fitting VECMs with Johansen's normalization*
 - Postestimation specification testing*
 - Impulse-response functions for VECMs*
 - Forecasting with VECMs*

Introduction to cointegrating VECMs

This section provides a brief introduction to integration, cointegration, and cointegrated vector error-correction models. For more details about these topics, see [Hamilton \(1994\)](#), [Johansen \(1995\)](#), [Lütkepohl \(2005\)](#), and [Watson \(1994\)](#).

What is cointegration?

Standard regression techniques, such as ordinary least squares (OLS), require that the variables be covariance stationary. A variable is covariance stationary if its mean and all its autocovariances are finite and do not change over time. Cointegration analysis provides a framework for estimation, inference, and interpretation when the variables are not covariance stationary.

Instead of being covariance stationary, many economic time series appear to be “first-difference stationary”. This means that the level of a time series is not stationary but its first difference is. First-difference stationary processes are also known as integrated processes of order 1, or $I(1)$ processes. Covariance-stationary processes are $I(0)$. In general, a process whose d th difference is stationary is an integrated process of order d , or $I(d)$.

The canonical example of a first-difference stationary process is the random walk. This is a variable x_t that can be written as

$$x_t = x_{t-1} + \epsilon_t \quad (1)$$

where the ϵ_t are independently and identically distributed (i.i.d.) with mean zero and a finite variance σ^2 . Although $E[x_t] = 0$ for all t , $\text{Var}[x_t] = T\sigma^2$ is not time invariant, so x_t is not covariance stationary. Because $\Delta x_t = x_t - x_{t-1} = \epsilon_t$ and ϵ_t is covariance stationary, x_t is first-difference stationary.

These concepts are important because, although conventional estimators are well behaved when applied to covariance-stationary data, they have nonstandard asymptotic distributions and different rates of convergence when applied to $I(1)$ processes. To illustrate, consider several variants of the model

$$y_t = a + bx_t + e_t \quad (2)$$

Throughout the discussion, we maintain the assumption that $E[e_t] = 0$.

If both y_t and x_t are covariance-stationary processes, e_t must also be covariance stationary. As long as $E[x_t e_t] = 0$, we can consistently estimate the parameters a and b by using OLS. Furthermore, the distribution of the OLS estimator converges to a normal distribution centered at the true value as the sample size grows.

If y_t and x_t are independent random walks and $b = 0$, there is no relationship between y_t and x_t , and (2) is called a spurious regression. Granger and Newbold (1974) performed Monte Carlo experiments and showed that the usual t statistics from OLS regression provide spurious results: given a large enough dataset, we can almost always reject the null hypothesis of the test that $b = 0$ even though b is in fact zero. Here the OLS estimate does not converge to any well-defined population parameter.

Phillips (1986) later provided the asymptotic theory that explained the Granger and Newbold (1974) results. He showed that the random walks y_t and x_t are first-difference stationary processes and that the OLS estimator does not have its usual asymptotic properties when the variables are first-difference stationary.

Because Δy_t and Δx_t are covariance stationary, a simple regression of Δy_t on Δx_t appears to be a viable alternative. However, if y_t and x_t cointegrate, as defined below, the simple regression of Δy_t on Δx_t is misspecified.

If y_t and x_t are $I(1)$ and $b \neq 0$, e_t could be either $I(0)$ or $I(1)$. Phillips and Durlauf (1986) have derived the asymptotic theory for the OLS estimator when e_t is $I(1)$, though it has not been widely used in applied work. More interesting is the case in which $e_t = y_t - a - bx_t$ is $I(0)$. y_t and x_t are then said to be cointegrated. Two variables are cointegrated if each is an $I(1)$ process but a linear combination of them is an $I(0)$ process.

It is not possible for y_t to be a random walk and x_t and e_t to be covariance stationary. As Granger (1981) pointed out, because a random walk cannot be equal to a covariance-stationary process, the equation does not “balance”. An equation balances when the processes on each side of the equal sign are of the same order of integration. Before attacking any applied problem with integrated variables, make sure that the equation balances before proceeding.

An example from Engle and Granger (1987) provides more intuition. Redefine y_t and x_t to be

$$y_t + \beta x_t = \epsilon_t, \quad \epsilon_t = \epsilon_{t-1} + \xi_t \quad (3)$$

$$y_t + \alpha x_t = \nu_t, \quad \nu_t = \rho \nu_{t-1} + \zeta_t, \quad |\rho| < 1 \quad (4)$$

where ξ_t and ζ_t are i.i.d. disturbances over time that are correlated with each other. Because ϵ_t is $I(1)$, (3) and (4) imply that both x_t and y_t are $I(1)$. The condition that $|\rho| < 1$ implies that ν_t and $y_t + \alpha x_t$ are $I(0)$. Thus y_t and x_t cointegrate, and $(1, \alpha)$ is the cointegrating vector.

Using a bit of algebra, we can rewrite (3) and (4) as

$$\Delta y_t = \beta \delta z_{t-1} + \eta_{1t} \quad (5)$$

$$\Delta x_t = -\delta z_{t-1} + \eta_{2t} \quad (6)$$

where $\delta = (1 - \rho)/(\alpha - \beta)$, $z_t = y_t + \alpha x_t$, and η_{1t} and η_{2t} are distinct, stationary, linear combinations of ξ_t and ζ_t . This representation is known as the vector error-correction model (VECM). One can think of $z_t = 0$ as being the point at which y_t and x_t are in equilibrium. The coefficients on z_{t-1} describe how y_t and x_t adjust to z_{t-1} being nonzero, or out of equilibrium. z_t is the “error” in the system, and (5) and (6) describe how system adjusts or corrects back to the equilibrium. As ρ goes to 1, the system degenerates into a pair of correlated random walks. The VECM parameterization highlights this point, because $\delta \rightarrow 0$ as $\rho \rightarrow 1$.

If we knew α , we would know z_t , and we could work with the stationary system of (5) and . Although knowing α seems silly, we can conduct much of the analysis as if we knew α because there is an estimator for the cointegrating parameter α that converges to its true value at a faster rate than the estimator for the adjustment parameters β and δ .

The definition of a bivariate cointegrating relation requires simply that there exist a linear combination of the I(1) variables that is I(0). If y_t and x_t are I(1) and there are two finite real numbers $a \neq 0$ and $b \neq 0$, such that $ay_t + bx_t$ is I(0), then y_t and x_t are cointegrated. Although there are two parameters, a and b , only one will be identifiable because if $ay_t + bx_t$ is I(0), so is $cay_t + cbx_t$ for any finite, nonzero, real number c . Obtaining identification in the bivariate case is relatively simple. The coefficient on y_t in (4) is unity. This natural construction of the model placed the necessary identification restriction on the cointegrating vector. As we discuss below, identification in the multivariate case is more involved.

If $\mathbf{y}_t$ is a $K \times 1$ vector of I(1) variables and there exists a vector β , such that $\beta\mathbf{y}_t$ is a vector of I(0) variables, then $\mathbf{y}_t$ is said to be cointegrating of order (1,0) with cointegrating vector β . We say that the parameters in β are the parameters in the cointegrating equation. For a vector of length K , there may be at most $K - 1$ distinct cointegrating vectors. Engle and Granger (1987) provide a more general definition of cointegration, but this one is sufficient for our purposes.

The multivariate VECM specification

In practice, most empirical applications analyze multivariate systems, so the rest of our discussion focuses on that case. Consider a VAR with p lags

$$\mathbf{y}_t = \mathbf{v} + \mathbf{A}_1\mathbf{y}_{t-1} + \mathbf{A}_2\mathbf{y}_{t-2} + \cdots + \mathbf{A}_p\mathbf{y}_{t-p} + \boldsymbol{\epsilon}_t \quad (7)$$

where $\mathbf{y}_t$ is a $K \times 1$ vector of variables, $\mathbf{v}$ is a $K \times 1$ vector of parameters, $\mathbf{A}_1 - \mathbf{A}_p$ are $K \times K$ matrices of parameters, and $\boldsymbol{\epsilon}_t$ is a $K \times 1$ vector of disturbances. $\boldsymbol{\epsilon}_t$ has mean $\mathbf{0}$, has covariance matrix $\boldsymbol{\Sigma}$, and is i.i.d. normal over time. Any VAR(p) can be rewritten as a VECM. Using some algebra, we can rewrite (7) in VECM form as

$$\Delta\mathbf{y}_t = \mathbf{v} + \boldsymbol{\Pi}\mathbf{y}_{t-1} + \sum_{i=1}^{p-1} \boldsymbol{\Gamma}_i\Delta\mathbf{y}_{t-i} + \boldsymbol{\epsilon}_t \quad (8)$$

where $\boldsymbol{\Pi} = \sum_{j=1}^{j=p} \mathbf{A}_j - \mathbf{I}_K$ and $\boldsymbol{\Gamma}_i = -\sum_{j=i+1}^{j=p} \mathbf{A}_j$. The $\mathbf{v}$ and $\boldsymbol{\epsilon}_t$ in (7) and (8) are identical.

Engle and Granger (1987) show that if the variables $\mathbf{y}_t$ are I(1) the matrix $\boldsymbol{\Pi}$ in (8) has rank $0 \leq r < K$, where r is the number of linearly independent cointegrating vectors. If the variables cointegrate, $0 < r < K$ and (8) shows that a VAR in first differences is misspecified because it omits the lagged level term $\boldsymbol{\Pi}\mathbf{y}_{t-1}$.

Assume that $\boldsymbol{\Pi}$ has reduced rank $0 < r < K$ so that it can be expressed as $\boldsymbol{\Pi} = \boldsymbol{\alpha}\boldsymbol{\beta}'$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are both $K \times r$ matrices of rank r . Without further restrictions, the cointegrating vectors are not identified: the parameters $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ are indistinguishable from the parameters $(\boldsymbol{\alpha}\mathbf{Q}, \boldsymbol{\beta}\mathbf{Q}^{-1'})$ for any $r \times r$ nonsingular matrix $\mathbf{Q}$. Because only the rank of $\boldsymbol{\Pi}$ is identified, the VECM is said to identify the rank of the cointegrating space, or equivalently, the number of cointegrating vectors. In practice, the estimation of the parameters of a VECM requires at least r^2 identification restrictions. Stata's `vec` command can apply the conventional Johansen restrictions discussed below or use constraints that the user supplies.

The VECM in (8) also nests two important special cases. If the variables in $\mathbf{y}_t$ are I(1) but not cointegrated, $\boldsymbol{\Pi}$ is a matrix of zeros and thus has rank 0. If all the variables are I(0), $\boldsymbol{\Pi}$ has full rank K .

There are several different frameworks for estimation and inference in cointegrating systems. Although the methods in Stata are based on the maximum likelihood (ML) methods developed by Johansen (1988, 1991, 1995), other useful frameworks have been developed by Park and Phillips (1988, 1989); Sims, Stock, and Watson (1990); Stock (1987); and Stock and Watson (1988); among others. The ML framework developed by Johansen was independently developed by Ahn and Reinsel (1990). Maddala and Kim (1998) and Watson (1994) survey all these methods. The cointegration methods in Stata are based on Johansen's maximum likelihood framework because it has been found to be particularly useful in several comparative studies, including Gonzalo (1994) and Hubrich, Lütkepohl, and Saikkonen (2001).

Trends in the Johansen VECM framework

Deterministic trends in a cointegrating VECM can stem from two distinct sources; the mean of the cointegrating relationship and the mean of the differenced series. Allowing for a constant and a linear trend and assuming that there are r cointegrating relations, we can rewrite the VECM in (8) as

$$\Delta \mathbf{y}_t = \alpha \beta \mathbf{y}_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \mathbf{v} + \delta t + \epsilon_t \quad (9)$$

where δ is a $K \times 1$ vector of parameters. Because (9) models the differences of the data, the constant implies a linear time trend in the levels, and the time trend δt implies a quadratic time trend in the levels of the data. Often we may want to include a constant or a linear time trend for the differences without allowing for the higher-order trend that is implied for the levels of the data. VECMs exploit the properties of the matrix α to achieve this flexibility.

Because α is a $K \times r$ rank matrix, we can rewrite the deterministic components in (9) as

$$\mathbf{v} = \alpha \boldsymbol{\mu} + \boldsymbol{\gamma} \quad (10a)$$

$$\delta t = \alpha \boldsymbol{\rho} t + \boldsymbol{\tau} t \quad (10b)$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\rho}$ are $r \times 1$ vectors of parameters and $\boldsymbol{\gamma}$ and $\boldsymbol{\tau}$ are $K \times 1$ vectors of parameters. $\boldsymbol{\gamma}$ is orthogonal to $\alpha \boldsymbol{\mu}$, and $\boldsymbol{\tau}$ is orthogonal to $\alpha \boldsymbol{\rho}$; i.e., $\boldsymbol{\gamma}' \alpha \boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\tau}' \alpha \boldsymbol{\rho} = \mathbf{0}$, allowing us to rewrite (9) as

$$\Delta \mathbf{y}_t = \alpha (\beta \mathbf{y}_{t-1} + \boldsymbol{\mu} + \boldsymbol{\rho} t) + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \boldsymbol{\gamma} + \boldsymbol{\tau} t + \epsilon_t \quad (11)$$

Placing restrictions on the trend terms in (11) yields five cases.

CASE 1: Unrestricted trend

If no restrictions are placed on the trend parameters, (11) implies that there are quadratic trends in the levels of the variables and that the cointegrating equations are stationary around time trends (trend stationary).

CASE 2: Restricted trend, $\boldsymbol{\tau} = \mathbf{0}$

By setting $\boldsymbol{\tau} = \mathbf{0}$, we assume that the trends in the levels of the data are linear but not quadratic. This specification allows the cointegrating equations to be trend stationary.

CASE 3: Unrestricted constant, $\boldsymbol{\tau} = \mathbf{0}$ and $\boldsymbol{\rho} = \mathbf{0}$

By setting $\boldsymbol{\tau} = \mathbf{0}$ and $\boldsymbol{\rho} = \mathbf{0}$, we exclude the possibility that the levels of the data have quadratic trends, and we restrict the cointegrating equations to be stationary around constant means. Because $\boldsymbol{\gamma}$ is not restricted to zero, this specification still puts a linear time trend in the levels of the data.

CASE 4: Restricted constant, $\tau = \mathbf{0}$, $\rho = \mathbf{0}$, and $\gamma = \mathbf{0}$

By adding the restriction that $\gamma = \mathbf{0}$, we assume there are no linear time trends in the levels of the data. This specification allows the cointegrating equations to be stationary around a constant mean, but it allows no other trends or constant terms.

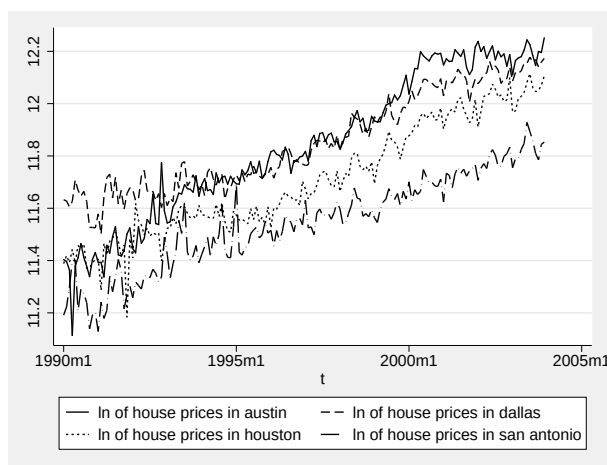
CASE 5: No trend, $\tau = \mathbf{0}$, $\rho = \mathbf{0}$, $\gamma = \mathbf{0}$, and $\mu = \mathbf{0}$

This specification assumes that there are no nonzero means or trends. It also assumes that the cointegrating equations are stationary with means of zero and that the differences and the levels of the data have means of zero.

This flexibility does come at a price. Below we discuss testing procedures for determining the number of cointegrating equations. The asymptotic distribution of the LR for hypotheses about r changes with the trend specification, so we must first specify a trend specification. A combination of theory and graphical analysis will aid in specifying the trend before proceeding with the analysis.

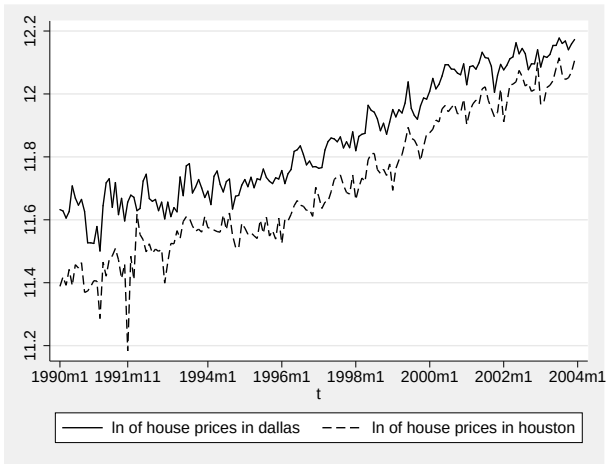
VECM estimation in Stata

We provide an overview of the `vec` commands in Stata through an extended example. We have monthly data on the average selling prices of houses in four cities in Texas: Austin, Dallas, Houston, and San Antonio. In the dataset, these average housing prices are contained in the variables `austin`, `dallas`, `houston`, and `sa`. The series begin in January of 1990 and go through December 2003, for a total of 168 observations. The following graph depicts our data.



The plots on the graph indicate that all the series are trending and potential $I(1)$ processes. In a competitive market, the current and past prices contain all the information available, so tomorrow's price will be a random walk from today's price. Some researchers may opt to use [TS] `dfgls` to investigate the presence of a unit root in each series, but the test for cointegration we use includes the case in which all the variables are stationary, so we defer formal testing until we test for cointegration. The time trends in the data appear to be approximately linear, so we will specify `trend(constant)` when modeling these series, which is the default with `vec`.

The next graph shows just Dallas' and Houston's data, so we can more carefully examine their relationship.



Except for the crash at the end of 1991, housing prices in Dallas and Houston appear closely related. Although average prices in the two cities will differ because of resource variations and other factors, if the housing markets become too dissimilar, people and businesses will migrate, bringing the average housing prices back toward each other. We therefore expect the series of average housing prices in Houston to be cointegrated with the series of average housing prices in Dallas.

Selecting the number of lags

To test for cointegration or fit cointegrating VECMs, we must specify how many lags to include. Building on the work of Tsay (1984) and Paulsen (1984), Nielsen (2001) has shown that the methods implemented in varsoc can be used to determine the lag order for a VAR model with I(1) variables. As can be seen from (9), the order of the corresponding VECM is always one less than the VAR. vec makes this adjustment automatically, so we will always refer to the order of the underlying VAR. The output below uses varsoc to determine the lag order of the VAR of the average housing prices in Dallas and Houston.

```
. use http://www.stata-press.com/data/r11/txhprice
. varsoc dallas houston
Selection-order criteria
Sample: 1990m5 - 2003m12
Number of obs = 164
```

lag	LL	LR	df	p	FPE	AIC	HQIC	SBIC
0	299.525				.000091	-3.62835	-3.61301	-3.59055
1	577.483	555.92	4	0.000	3.2e-06	-6.9693	-6.92326	-6.85589
2	590.978	26.991*	4	0.000	2.9e-06*	-7.0851*	-7.00837*	-6.89608*
3	593.437	4.918	4	0.296	2.9e-06	-7.06631	-6.95888	-6.80168
4	596.364	5.8532	4	0.210	3.0e-06	-7.05322	-6.9151	-6.71299

```
Endogenous: dallas houston
Exogenous: _cons
```

We will use two lags for this bivariate model because the Hannan–Quinn information criterion (HQIC) method, Schwarz Bayesian information criterion (SBIC) method, and sequential likelihood-ratio (LR) test all chose two lags, as indicated by the “*” in the output.

The reader can verify that when all four cities’ data are used, the LR test selects three lags, the HQIC method selects two lags, and the SBIC method selects one lag. We will use three lags in our four-variable model.

Testing for cointegration

The tests for cointegration implemented in `vecrank` are based on Johansen's method. If the log likelihood of the unconstrained model that includes the cointegrating equations is significantly different from the log likelihood of the constrained model that does not include the cointegrating equations, we reject the null hypothesis of no cointegration.

Here we use `vecrank` to determine the number of cointegrating equations:

```
. vecrank dallas houston
```

Johansen tests for cointegration						
Trend: constant			Number of obs =		166	
Sample: 1990m3 - 2003m12			Lags =		2	

maximum				trace	5%
rank	parms	LL	eigenvalue	statistic	critical
0	6	576.26444	.	46.8252	15.41
1	9	599.58781	0.24498	0.1785*	3.76
2	10	599.67706	0.00107		

Besides presenting information about the sample size and time span, the header indicates that test statistics are based on a model with two lags and a constant trend. The body of the table presents test statistics and their critical values of the null hypotheses of no cointegration (line 1) and one or fewer cointegrating equations (line 2). The eigenvalue shown on the last line is used to compute the trace statistic in the line above it. Johansen's testing procedure starts with the test for zero cointegrating equations (a maximum rank of zero) and then accepts the first null hypothesis that is not rejected.

In the output above, we strongly reject the null hypothesis of no cointegration and fail to reject the null hypothesis of at most one cointegrating equation. Thus we accept the null hypothesis that there is one cointegrating equation in the bivariate model.

Using all four series and a model with three lags, we find that there are two cointegrating relationships.

```
. vecrank austin dallas houston sa, lag(3)
```

Johansen tests for cointegration						
Trend: constant			Number of obs =		165	
Sample: 1990m4 - 2003m12			Lags =		3	

maximum				trace	5%
rank	parms	LL	eigenvalue	statistic	critical
0	36	1107.7833	.	101.6070	47.21
1	43	1137.7484	0.30456	41.6768	29.68
2	48	1153.6435	0.17524	9.8865*	15.41
3	51	1158.4191	0.05624	0.3354	3.76
4	52	1158.5868	0.00203		

Fitting a VECM

`vec` estimates the parameters of cointegrating VECMs. There are four types of parameters of interest:

1. The parameters in the cointegrating equations β
2. The adjustment coefficients α
3. The short-run coefficients
4. Some standard functions of β and α that have useful interpretations

Although all four types are discussed in [TS] `vec`, here we discuss only types 1–3 and how they appear in the output of `vec`.

Having determined that there is a cointegrating equation between the Dallas and Houston series, we now want to estimate the parameters of a bivariate cointegrating VECM for these two series by using `vec`.

```
. vec dallas houston
Vector error-correction model
Sample: 1990m3 - 2003m12
No. of obs      =      166
AIC             = -7.115516
Log likelihood = 599.5878
HQIC           = -7.04703
Det(Sigma_ml)  = 2.50e-06
SBIC          = -6.946794

Equation      Parms    RMSE      R-sq      chi2      P>chi2
-----
D_dallas      4        .038546  0.1692    32.98959  0.0000
D_houston     4        .045348  0.3737    96.66399  0.0000
```

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
D_dallas						
_cel						
L1.	-.3038799	.0908504	-3.34	0.001	-.4819434	-.1258165
dallas						
LD.	-.1647304	.0879356	-1.87	0.061	-.337081	.0076202
houston						
LD.	-.0998368	.0650838	-1.53	0.125	-.2273988	.0277251
_cons	.0056128	.0030341	1.85	0.064	-.0003339	.0115595
D_houston						
_cel						
L1.	.5027143	.1068838	4.70	0.000	.2932258	.7122028
dallas						
LD.	-.0619653	.1034547	-0.60	0.549	-.2647327	.1408022
houston						
LD.	-.3328437	.07657	-4.35	0.000	-.4829181	-.1827693
_cons	.0033928	.0035695	0.95	0.342	-.0036034	.010389

```
Cointegrating equations
Equation      Parms    chi2      P>chi2
-----
_cel          1    1640.088  0.0000
```

Identification: beta is exactly identified
Johansen normalization restriction imposed

beta	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
_cel						
dallas	1	.	.	.	.	.
houston	-.8675936	.0214231	-40.50	0.000	-.9095821	-.825605
_cons	-1.688897	.	.	.	.	.

The header contains information about the sample, the fit of each equation, and overall model fit statistics. The first estimation table contains the estimates of the short-run parameters, along with their standard errors, z statistics, and confidence intervals. The two coefficients on `L._cel` are the parameters in the adjustment matrix α for this model. The second estimation table contains the

estimated parameters of the cointegrating vector for this model, along with their standard errors, z statistics, and confidence intervals.

Using our previous notation, we have estimated

$$\hat{\alpha} = (-0.304, 0.503) \quad \hat{\beta} = (1, -0.868) \quad \hat{\nu} = (0.0056, 0.0034)$$

and

$$\hat{\Gamma} = \begin{pmatrix} -0.165 & -0.0998 \\ -0.062 & -0.333 \end{pmatrix}$$

Overall, the output indicates that the model fits well. The coefficient on `houston` in the cointegrating equation is statistically significant, as are the adjustment parameters. The adjustment parameters in this bivariate example are easy to interpret, and we can see that the estimates have the correct signs and imply rapid adjustment toward equilibrium. When the predictions from the cointegrating equation are positive, `dallas` is above its equilibrium value because the coefficient on `dallas` in the cointegrating equation is positive. The estimate of the coefficient `[D_dallas]L._ce1` is $-.3$. Thus when the average housing price in Dallas is too high, it quickly falls back toward the Houston level. The estimated coefficient `[D_houston]L._ce1` of $.5$ implies that when the average housing price in Dallas is too high, the average price in Houston quickly adjusts toward the Dallas level at the same time that the Dallas prices are adjusting.

Fitting VECMs with Johansen's normalization

As discussed by [Johansen \(1995\)](#), if there are r cointegrating equations, then at least r^2 restrictions are required to identify the free parameters in β . Johansen proposed a default identification scheme that has become the conventional method of identifying models in the absence of theoretically justified restrictions. Johansen's identification scheme is

$$\beta' = (\mathbf{I}_r, \tilde{\beta}')$$

where $\mathbf{I}_r$ is the $r \times r$ identity matrix and $\tilde{\beta}$ is an $(K - r) \times r$ matrix of identified parameters. `vec` applies Johansen's normalization by default.

To illustrate, we fit a VECM with two cointegrating equations and three lags on all four series. We are interested only in the estimates of the parameters in the cointegrating equations, so we can specify the `noetable` option to suppress the estimation table for the adjustment and short-run parameters.

(Continued on next page)

```
. vec austin dallas houston sa, lags(3) rank(2) noetable
Vector error-correction model
Sample: 1990m4 - 2003m12                                No. of obs      =          165
                                                         AIC              = -13.40174
                                                         HQIC             = -13.03496
                                                         SBIC             = -12.49819
Cointegrating equations
Equation          Parms      chi2      P>chi2
-----
 _ce1              2      586.3044  0.0000
 _ce2              2      2169.826  0.0000
```

Identification: beta is exactly identified
Johansen normalization restrictions imposed

beta		Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
_ce1	austin	1	.	.	.	.	.
	dallas	-3.48e-17	.	.	.	.	.
	houston	-.2623782	.1893625	-1.39	0.166	-.6335219	.1087655
	sa	-1.241805	.229643	-5.41	0.000	-1.691897	-.7917128
	_cons	5.577099	.	.	.	.	.
_ce2	austin	-1.26e-17	.	.	.	.	.
	dallas	1	.	.	.	.	.
	houston	-1.095652	.0669898	-16.36	0.000	-1.22695	-.9643545
	sa	.2883986	.0812396	3.55	0.000	.1291718	.4476253
	_cons	-2.351372	.	.	.	.	.

The Johansen identification scheme has placed four constraints on the parameters in β : `[_ce1]austin=1`, `[_ce1]dallas=0`, `[_ce2]austin=0`, and `[_ce2]dallas=1`. (The computational method used imposes zero restrictions that are numerical rather than exact. The values `-3.48e-17` and `-1.26e-17` are indistinguishable from zero.) We interpret the results of the first equation as indicating the existence of an equilibrium relationship between the average housing price in Austin and the average prices of houses in Houston and San Antonio.

The Johansen normalization restricted the coefficient on `dallas` to be unity in the second cointegrating equation, but we could instead constrain the coefficient on `houston`. Both sets of restrictions define just-identified models, so fitting the model with the latter set of restrictions will yield the same maximized log likelihood. To impose the alternative set of constraints, we use the `constraint` command.

```
. constraint 1 [_ce1]austin = 1
. constraint 2 [_ce1]dallas = 0
. constraint 3 [_ce2]austin = 0
. constraint 4 [_ce2]houston = 1
```



```
. vec austin dallas houston sa, lags(3) rank(2) noetable bconstraints(1/4)
Iteration 1:      log likelihood = 1148.8745
(output omitted)
Iteration 25:     log likelihood = 1153.6435
Vector error-correction model
Sample: 1990m4 - 2003m12
                                No. of obs   =      165
                                AIC           = -13.40174
                                HQIC          = -13.03496
                                SBIC          = -12.49819
Log likelihood = 1153.644
Det(Sigma_ml) = 9.93e-12
Cointegrating equations
Equation      Parms      chi2      P>chi2
-----
 _ce1          2      586.3392   0.0000
 _ce2          2     3455.469   0.0000

Identification:  beta is exactly identified
Identifying constraints:
( 1)  [_ce1]austin = 1
( 2)  [_ce1]dallas = 0
( 3)  [_ce2]austin = 0
( 4)  [_ce2]houston = 1
```

beta	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
_ce1						
austin	1	.	.	.	.	.
dallas	(omitted)					
houston	-.2623784	.1876727	-1.40	0.162	-.6302102	.1054534
sa	-1.241805	.2277537	-5.45	0.000	-1.688194	-.7954157
_cons	5.577099	.	.	.	.	.
_ce2						
austin	(omitted)					
dallas	-.9126985	.0595804	-15.32	0.000	-1.029474	-.7959231
houston	1	.	.	.	.	.
sa	-.2632209	.0628791	-4.19	0.000	-.3864617	-.1399802
_cons	2.146094	.	.	.	.	.

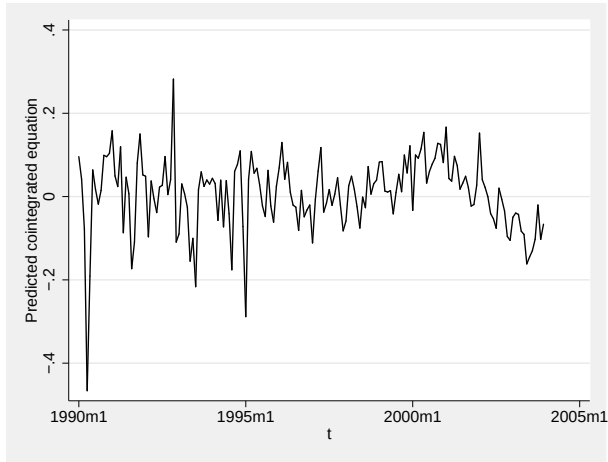
Only the estimates of the parameters in the second cointegrating equation have changed, and the new estimates are simply the old estimates divided by -1.095652 because the new constraints are just an alternative normalization of the same just-identified model. With the new normalization, we can interpret the estimates of the parameters in the second cointegrating equation as indicating an equilibrium relationship between the average house price in Houston and the average prices of houses in Dallas and San Antonio.

Postestimation specification testing

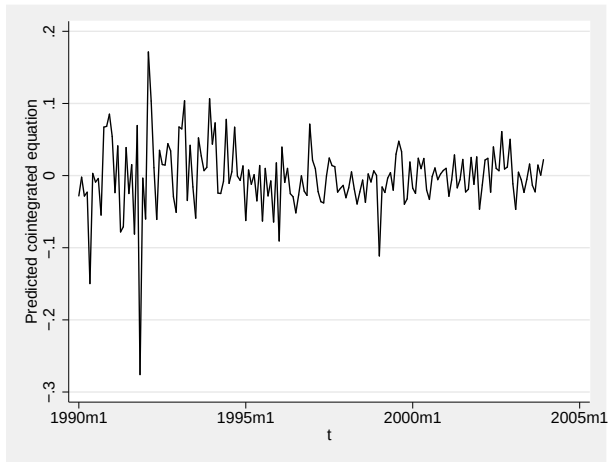
Inference on the parameters in α depends crucially on the stationarity of the cointegrating equations, so we should check the specification of the model. As a first check, we can predict the cointegrating equations and graph them over time.

```
. predict ce1, ce equ(#1)
. predict ce2, ce equ(#2)
```

```
. twoway line ce1 t
```



```
. twoway line ce2 t
```



Although the large shocks apparent in the graph of the levels have clear effects on the predictions from the cointegrating equations, our only concern is the negative trend in the first cointegrating equation since the end of 2000. The graph of the levels shows that something put a significant brake on the growth of housing prices after 2000 and that the growth of housing prices in San Antonio slowed during 2000 but then recuperated while Austin maintained slower growth. We suspect that this indicates that the end of the high-tech boom affected Austin more severely than San Antonio. This difference is what causes the trend in the first cointegrating equation. Although we could try to account for this effect with a more formal analysis, we will proceed as if the cointegrating equations are stationary.

We can use `vecstable` to check whether we have correctly specified the number of cointegrating equations. As discussed in [TS] [vecstable](#), the companion matrix of a VECM with K endogenous variables and r cointegrating equations has $K - r$ unit eigenvalues. If the process is stable, the moduli of the remaining r eigenvalues are strictly less than one. Because there is no general distribution

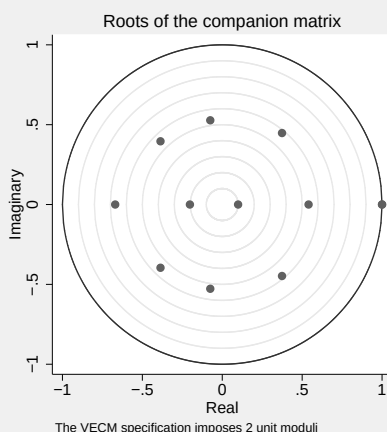
theory for the moduli of the eigenvalues, ascertaining whether the moduli are too close to one can be difficult.

```
. vecstable, graph
```

Eigenvalue stability condition

Eigenvalue	Modulus
1	1
1	1
-.6698661	.669866
.3740191 + .4475996i	.583297
.3740191 - .4475996i	.583297
-.386377 + .395972i	.553246
-.386377 - .395972i	.553246
.540117	.540117
-.0749239 + .5274203i	.532715
-.0749239 - .5274203i	.532715
-.2023955	.202395
.09923966	.09924

The VECM specification imposes 2 unit moduli.



Because we specified the `graph` option, `vecstable` plotted the eigenvalues of the companion matrix. The graph of the eigenvalues shows that none of the remaining eigenvalues appears close to the unit circle. The stability check does not indicate that our model is misspecified.

Here we use `veclmar` to test for serial correlation in the residuals.

```
. veclmar, mlag(4)
```

Lagrange-multiplier test

lag	chi2	df	Prob > chi2
1	56.8757	16	0.00000
2	31.1970	16	0.01270
3	30.6818	16	0.01477
4	14.6493	16	0.55046

H0: no autocorrelation at lag order

The results clearly indicate serial correlation in the residuals. The results in [Gonzalo \(1994\)](#) indicate that underspecifying the number of lags in a VECM can significantly increase the finite-sample bias in the parameter estimates and lead to serial correlation. For this reason, we refit the model with five lags instead of three.

```
. vec austin dallas houston sa, lags(5) rank(2) noetable bconstraints(1/4)
Iteration 1:      log likelihood = 1200.5402
(output omitted)
Iteration 20:     log likelihood = 1203.9465
Vector error-correction model
Sample: 1990m6 - 2003m12
Log likelihood = 1203.946
Det(Sigma_ml) = 4.51e-12
Cointegrating equations
Equation      Parms      chi2      P>chi2
-----
 _ce1          2      498.4682   0.0000
 _ce2          2      4125.926   0.0000

Identification:  beta is exactly identified
Identifying constraints:
( 1) [_ce1]austin = 1
( 2) [_ce1]dallas = 0
( 3) [_ce2]austin = 0
( 4) [_ce2]houston = 1
```

beta	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
_ce1	austin	1	.	.	.	.
	dallas	(omitted)	.	.	.	.
	houston	-.6525574	.2047061	-3.19	0.001	-1.053774 - .2513407
	sa	-.6960166	.2494167	-2.79	0.005	-1.184864 - .2071688
	_cons	3.846275	.	.	.	.
_ce2	austin	(omitted)	.	.	.	.
	dallas	-.932048	.0564332	-16.52	0.000	-1.042655 - .8214409
	houston	1	.	.	.	.
	sa	-.2363915	.0599348	-3.94	0.000	-.3538615 - .1189215
	_cons	2.065719	.	.	.	.

Comparing these results with those from the previous model reveals that

- 1. there is now evidence that the coefficient [_ce1]houston is not equal to zero,
- 2. the two sets of estimated coefficients for the first cointegrating equation are different, and
- 3. the two sets of estimated coefficients for the second cointegrating equation are similar.

The assumption that the errors are independently, identically, and normally distributed with zero mean and finite variance allows us to derive the likelihood function. If the errors do not come from a normal distribution but are just independently and identically distributed with zero mean and finite variance, the parameter estimates are still consistent, but they are not efficient.

We use `vecnorm` to test the null hypothesis that the errors are normally distributed.

```
. qui vec austin dallas houston sa, lags(5) rank(2) bconstraints(1/4)
. vecnorm
Jarque-Bera test
```

Equation	chi2	df	Prob > chi2
D_austin	74.324	2	0.00000
D_dallas	3.501	2	0.17370
D_houston	245.032	2	0.00000
D_sa	8.426	2	0.01481
ALL	331.283	8	0.00000

Skewness test

Equation	Skewness	chi2	df	Prob > chi2
D_austin	.60265	9.867	1	0.00168
D_dallas	.09996	0.271	1	0.60236
D_houston	-1.0444	29.635	1	0.00000
D_sa	.38019	3.927	1	0.04752
ALL		43.699	4	0.00000

Kurtosis test

Equation	Kurtosis	chi2	df	Prob > chi2
D_austin	6.0807	64.458	1	0.00000
D_dallas	3.6896	3.229	1	0.07232
D_houston	8.6316	215.397	1	0.00000
D_sa	3.8139	4.499	1	0.03392
ALL		287.583	4	0.00000

The results indicate that we can strongly reject the null hypothesis of normally distributed errors. Most of the errors are both skewed and kurtotic.

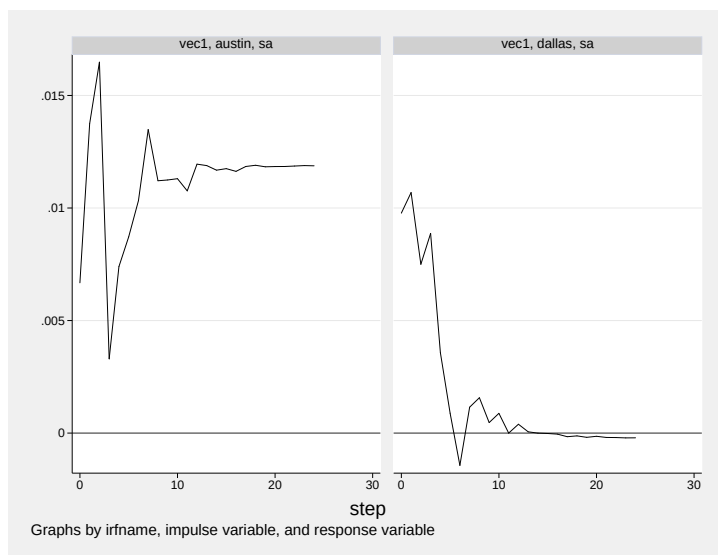
Impulse–response functions for VECMs

With a model that we now consider acceptably well specified, we can use the `irf` commands to estimate and interpret the IRFs. Whereas IRFs from a stationary VAR die out over time, IRFs from a cointegrating VECM do not always die out. Because each variable in a stationary VAR has a time-invariant mean and finite, time-invariant variance, the effect of a shock to any one of these variables must die out so that the variable can revert to its mean. In contrast, the $I(1)$ variables modeled in a cointegrating VECM are not mean reverting, and the unit moduli in the companion matrix imply that the effects of some shocks will not die out over time.

These two possibilities gave rise to new terms. When the effect of a shock dies out over time, the shock is said to be transitory. When the effect of a shock does not die out over time, the shock is said to be permanent.

Below we use `irf create` to estimate the IRFs and `irf graph` to graph two of the orthogonalized IRFs.

```
. irf create vec1, set(vecintro, replace) step(24)
(file vecintro.irf created)
(file vecintro.irf now active)
(file vecintro.irf updated)
. irf graph oirf, impulse(austin dallas) response(sa) yline(0)
```



The graphs indicate that an orthogonalized shock to the average housing price in Austin has a permanent effect on the average housing price in San Antonio but that an orthogonalized shock to the average price of housing in Dallas has a transitory effect. According to this model, unexpected shocks that are local to the Austin housing market will have a permanent effect on the housing market in San Antonio, but unexpected shocks that are local to the Dallas housing market will have only a transitory effect on the housing market in San Antonio.

Forecasting with VECMs

Cointegrating VECMs are also used to produce forecasts of both the first-differenced variables and the levels of the variables. Comparing the variances of the forecast errors of stationary VARs with those from a cointegrating VECM reveals a fundamental difference between the two models. Whereas the variances of the forecast errors for a stationary VAR converge to a constant as the prediction horizon grows, the variances of the forecast errors for the levels of a cointegrating VECM diverge with the forecast horizon. (See sec. 6.5 of [Lütkepohl \[2005\]](#) for more about this result.) Because all the variables in the model for the first differences are stationary, the forecast errors for the dynamic forecasts of the first differences remain finite. In contrast, the forecast errors for the dynamic forecasts of the levels diverge to infinity.

We use `fcast compute` to obtain dynamic forecasts of the levels and `fcast graph` to graph these dynamic forecasts, along with their asymptotic confidence intervals.

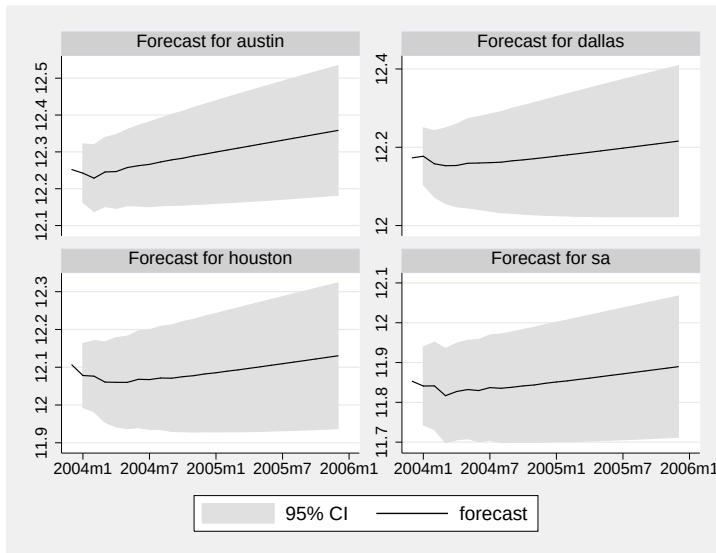
```

. tsset
    time variable: t, 1990m1 to 2003m12
      delta: 1 month

. fcast compute m1_, step(24)

. fcast graph m1_austin m1_dallas m1_houston m1_sa

```



As expected, the widths of the confidence intervals grow with the forecast horizon.

References

- Ahn, S. K., and G. C. Reinsel. 1990. Estimation for partially nonstationary multivariate autoregressive models. *Journal of the American Statistical Association* 85: 813–823.
- Engle, R. F., and C. W. J. Granger. 1987. Co-integration and error correction: Representation, estimation, and testing. *Econometrica* 55: 251–276.
- Gonzalo, J. 1994. Five alternative methods of estimating long-run equilibrium relationships. *Journal of Econometrics* 60: 203–233.
- Granger, C. W. J. 1981. Some properties of time series data and their use in econometric model specification. *Journal of Econometrics* 16: 121–130.
- Granger, C. W. J., and P. Newbold. 1974. Spurious regressions in econometrics. *Journal of Econometrics* 2: 111–120.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Hubrich, K., H. Lütkepohl, and P. Saikkonen. 2001. A review of systems cointegration tests. *Econometric Reviews* 20: 247–318.
- Johansen, S. 1988. Statistical analysis of cointegration vectors. *Journal of Economic Dynamics and Control* 12: 231–254.
- . 1991. Estimation and hypothesis testing of cointegration vectors in Gaussian vector autoregressive models. *Econometrica* 59: 1551–1580.
- . 1995. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford: Oxford University Press.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Maddala, G. S., and I.-M. Kim. 1998. *Unit Roots, Cointegration, and Structural Change*. Cambridge: Cambridge University Press.

- Nielsen, B. 2001. Order determination in general vector autoregressions. Working paper, Department of Economics, University of Oxford and Nuffield College. <http://ideas.repec.org/p/nuf/econwp/0110.html>.
- Park, J. Y., and P. C. B. Phillips. 1988. Statistical inference in regressions with integrated processes: Part I. *Econometric Theory* 4: 468–497.
- . 1989. Statistical inference in regressions with integrated processes: Part II. *Econometric Theory* 5: 95–131.
- Paulsen, J. 1984. Order determination of multivariate autoregressive time series with unit roots. *Journal of Time Series Analysis* 5: 115–127.
- Phillips, P. C. B. 1986. Understanding spurious regressions in econometrics. *Journal of Econometrics* 33: 311–340.
- Phillips, P. C. B., and S. N. Durlauf. 1986. Multiple time series regressions with integrated processes. *Review of Economic Studies* 53: 473–495.
- Sims, C. A., J. H. Stock, and M. W. Watson. 1990. Inference in linear time series models with some unit roots. *Econometrica* 58: 113–144.
- Stock, J. H. 1987. Asymptotic properties of least squares estimators of cointegrating vectors. *Econometrica* 55: 1035–1056.
- Stock, J. H., and M. W. Watson. 1988. Testing for common trends. *Journal of the American Statistical Association* 83: 1097–1107.
- Tsay, R. S. 1984. Order selection in nonstationary autoregressive models. *Annals of Statistics* 12: 1425–1433.
- Watson, M. W. 1994. Vector autoregressions and cointegration. In Vol. IV of *Handbook of Econometrics*, ed. R. F. Engle and D. L. McFadden. Amsterdam: Elsevier.

Also see

[TS] [vec](#) — Vector error-correction models

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

Syntax

vec *varlist* [*if*] [*in*] [, *options*]

<i>options</i>	description
Model	
<u>rank</u> (#)	use # cointegrating equations; default is rank(1)
<u>lags</u> (#)	use # for the maximum lag in underlying VAR model
<u>trend</u> (constant)	include an unrestricted constant in model; the default
<u>trend</u> (<u>r</u> constant)	include a restricted constant in model
<u>trend</u> (<u>trend</u>)	include a linear trend in the cointegrating equations and a quadratic trend in the undifferenced data
<u>trend</u> (<u>r</u> trend)	include a restricted trend in model
<u>trend</u> (<u>n</u> one)	do not include a trend or a constant
<u>b</u> constraints(<i>constraints</i> _{b_c})	place <i>constraints</i> _{b_c} on cointegrating vectors
<u>a</u> constraints(<i>constraints</i> _{a_c})	place <i>constraints</i> _{a_c} on adjustment parameters
Adv. model	
<u>s</u> indicators(<i>varlist</i> _{si})	include normalized seasonal indicator variables <i>varlist</i> _{si}
<u>n</u> oreduce	do not perform checks and corrections for collinearity among lags of dependent variables
Reporting	
<u>l</u> evel(#)	set confidence level; default is level(95)
<u>n</u> o table	do not report parameters in the cointegrating equations
<u>n</u> o id test	do not report the likelihood-ratio test of overidentifying restrictions
<u>a</u> lpha	report adjustment parameters in separate table
<u>p</u> i	report parameters in $\Pi = \alpha\beta'$
<u>n</u> o ptable	do not report elements of Π matrix
<u>m</u> ai	report parameters in the moving-average impact matrix
<u>n</u> o e table	do not report adjustment and short-run parameters
<u>d</u> force	force reporting of short-run, beta, and alpha parameters when the parameters in beta are not identified; advanced option
<u>n</u> o cns report	do not display constraints
<i>display_options</i>	control spacing
Maximization	
<i>maximize_options</i>	control the maximization process; seldom used
[†] <u>c</u> oef l egend	display coefficients' legend instead of coefficient table

[†]`coeflegend` does not appear in the dialog box.

You must `tsset` your data before using `vec`; see [TS] [tsset](#).

`varlist` must contain at least two variables and may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

`by`, `rolling`, `statsby`, and `xi` are allowed; see [U] [11.1.10 Prefix commands](#).

See [U] [20 Estimation and postestimation commands](#) for more capabilities of estimation commands.

`vec` does not allow gaps in the data.

Menu

Statistics > Multivariate time series > Vector error-correction model (VECM)

Description

`vec` fits a type of vector autoregression in which some of the variables are cointegrated by using Johansen's (1995) maximum likelihood method. Constraints may be placed on the parameters in the cointegrating equations or on the adjustment terms. See [TS] [vec intro](#) for a list of commands that are used in conjunction with `vec`.

Options

Model

`rank(#)` specifies the number of cointegrating equations; `rank(1)` is the default.

`lags(#)` specifies the maximum lag to be included in the underlying VAR model. The maximum lag in a VECM is one smaller than the maximum lag in the corresponding VAR in levels; the number of lags must be greater than zero but small enough so that the degrees of freedom used up by the model are fewer than the number of observations. The default is `lags(2)`.

`trend(trend_spec)` specifies which of Johansen's five trend specifications to include in the model. These specifications are discussed in [Specification of constants and trends](#) below. The default is `trend(constant)`.

`bconstraints(constraintsbc)` specifies the constraints to be placed on the parameters of the cointegrating equations. When no constraints are placed on the adjustment parameters—that is, when the `aconstraints()` option is not specified—the default is to place the constraints defined by Johansen's normalization on the parameters of the cointegrating equations. When constraints are placed on the adjustment parameters, the default is not to place constraints on the parameters in the cointegrating equations.

`aconstraints(constraintsac)` specifies the constraints to be placed on the adjustment parameters. By default, no constraints are placed on the adjustment parameters.

Adv. model

`sindicators(varlistsi)` specifies the normalized seasonal indicator variables to include in the model. The indicator variables specified in this option must be normalized as discussed in [Johansen \(1995\)](#). If the indicators are not properly normalized, the estimator of the cointegrating vector does not converge to the asymptotic distribution derived by [Johansen \(1995\)](#). More details about how these variables are handled are provided in [Methods and formulas](#). `sindicators()` cannot be specified with `trend(none)` or with `trend(rconstant)`.

`noreduce` causes `vec` to skip the checks and corrections for collinearity among the lags of the dependent variables. By default, `vec` checks to see whether the current lag specification causes some of the regressions performed by `vec` to contain perfectly collinear variables; if so, it reduces the maximum lag until the perfect collinearity is removed.

Reporting

`level(#)`; see [R] [estimation options](#).

`nobtable` suppresses the estimation table for the parameters in the cointegrating equations. By default, `vec` displays the estimation table for the parameters in the cointegrating equations.

`noidtest` suppresses the likelihood-ratio test of the overidentifying restrictions, which is reported by default when the model is overidentified.

`alpha` displays a separate estimation table for the adjustment parameters, which is not displayed by default.

`pi` displays a separate estimation table for the parameters in $\Pi = \alpha\beta'$, which is not displayed by default.

`noptable` suppresses the estimation table for the elements of the Π matrix, which is displayed by default when the parameters in the cointegrating equations are not identified.

`mai` displays a separate estimation table for the parameters in the moving-average impact matrix, which is not displayed by default.

`noetable` suppresses the main estimation table that contains information about the estimated adjustment parameters and the short-run parameters, which is displayed by default.

`dforce` displays the estimation tables for the short-run parameters and α and β —if the last two are requested—when the parameters in β are not identified. By default, when the specified constraints do not identify the parameters in the cointegrating equations, estimation tables are displayed only for Π and the MAI.

`nocnsreport`; see [R] [estimation options](#).

`display_options`: `vsquish`; see [R] [estimation options](#).

Maximization

`maximize_options`: `iterate(#)`, `nolog`, `trace`, `toltrace`, `tolerance(#)`, `ltolerance(#)`, `afrom(matrixa)`, `bfrom(matrixb)`; see [R] [maximize](#).

`toltrace` displays the relative differences for the log likelihood and the coefficient vector at every iteration. This option cannot be specified if no constraints are defined or if `nolog` is specified.

`afrom(matrixa)` specifies a $1 \times (K * r)$ row vector with starting values for the adjustment parameters, where K is the number of endogenous variables and r is the number of cointegrating equations specified in the `rank()` option. The starting values should be ordered as they are reported in `e(alpha)`. This option cannot be specified if no constraints are defined.

`bfrom(matrixb)` specifies a $1 \times (m_1 * r)$ row vector with starting values for the parameters of the cointegrating equations, where m_1 is the number of variables in the trend-augmented system and r is the number of cointegrating equations specified in the `rank()` option. (See [Methods and formulas](#) for more details about m_1 .) The starting values should be ordered as they are reported in `e(betavec)`. As discussed in [Methods and formulas](#), for some trend specifications, `e(beta)` contains parameter estimates that are not obtained directly from the optimization algorithm. `bfrom()` should specify only starting values for the parameters reported in `e(betavec)`. This option cannot be specified if no constraints are defined.

The following option is available with `vec` but is not shown in the dialog box:
`coeflegend`; see [R] [estimation options](#).

Remarks

Remarks are presented under the following headings:

Introduction
Specification of constants and trends
Collinearity

Introduction

VECMs are used to model the stationary relationships between multiple time series that contain unit roots. `vec` implements Johansen’s approach for estimating the parameters of a VECM.

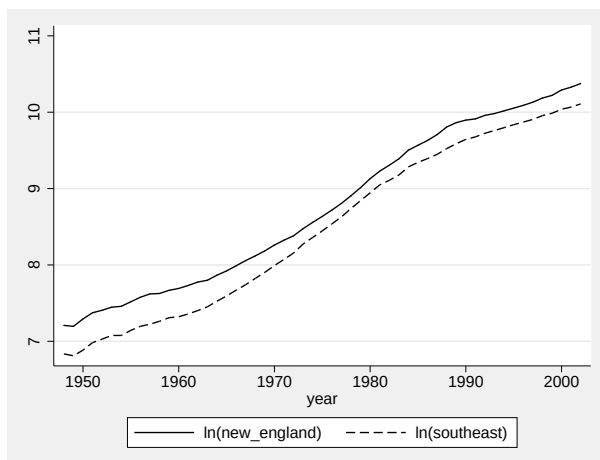
[TS] [vec intro](#) reviews the basics of integration and cointegration and highlights why we need special methods for modeling the relationships between processes that contain unit roots. This manual entry assumes familiarity with the material in [TS] [vec intro](#) and provides examples illustrating how to use the `vec` command. See [Johansen \(1995\)](#) and [Hamilton \(1994\)](#) for more in-depth introductions to cointegration analysis.

► Example 1

This example uses annual data on the average per-capita disposable personal income in the eight U.S. Bureau of Economic Analysis (BEA) regions of the United States. We use data from 1948–2002 in logarithms. Unit-root tests on these series fail to reject the null hypothesis that per-capita disposable income in each region contains a unit root. Because capital and labor can move easily between the different regions of the United States, we would expect that no one series will diverge from all the remaining series and that cointegrating relationships exist.

Below we graph the natural logs of average disposal income in the New England and the Southeast regions.

```
. use http://www.stata-press.com/data/r11/rdinc
. line ln_ne ln_se year
```



The graph indicates a differential between the two series that shrinks between 1960 and about 1980 and then grows until it stabilizes around 1990. We next estimate the parameters of a bivariate VECM with one cointegrating relationship.

```
. vec ln_ne ln_se
```

Vector error-correction model

Sample: 1950 - 2002

No. of obs = 53

AIC = -11.00462

HQIC = -10.87595

SBIC = -10.67004

Log likelihood = 300.6224

Det(Sigma_ml) = 4.06e-08

Equation	Parms	RMSE	R-sq	chi2	P>chi2
D_ln_ne	4	.017896	0.9313	664.4668	0.0000
D_ln_se	4	.018723	0.9292	642.7179	0.0000

	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
D_ln_ne						
_cel						
L1.	-.4337524	.0721365	-6.01	0.000	-.5751373	-.2923675
ln_ne						
LD.	.7168658	.1889085	3.79	0.000	.3466119	1.08712
ln_se						
LD.	-.6748754	.2117975	-3.19	0.001	-1.089991	-.2597599
_cons	-.0019846	.0080291	-0.25	0.805	-.0177214	.0137521
D_ln_se						
_cel						
L1.	-.3543935	.0754725	-4.70	0.000	-.5023168	-.2064701
ln_ne						
LD.	.3366786	.1976448	1.70	0.088	-.050698	.7240553
ln_se						
LD.	-.1605811	.2215922	-0.72	0.469	-.5948939	.2737317
_cons	.002429	.0084004	0.29	0.772	-.0140355	.0188936

Cointegrating equations

Equation	Parms	chi2	P>chi2
_cel	1	29805.02	0.0000

Identification: beta is exactly identified

Johansen normalization restriction imposed

beta	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
_cel						
ln_ne	1	.	.	.	.	.
ln_se	-.9433708	.0054643	-172.64	0.000	-.9540807	-.9326609
_cons	-.8964065	.	.	.	.	.

The default output has three parts. The header provides information about the sample, the model fit, and the identification of the parameters in the cointegrating equation. The main estimation table

contains the estimates of the short-run parameters, along with their standard errors and confidence intervals. The second estimation table reports the estimates of the parameters in the cointegrating equation, along with their standard errors and confidence intervals.

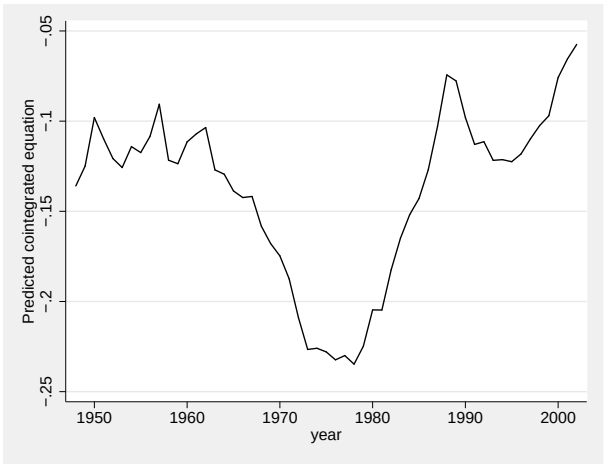
The results indicate strong support for a cointegrating equation such that

$$\ln\_ne - .943 \ln\_se - .896$$

should be a stationary series. Identification of the parameters in the cointegrating equation is achieved by constraining some of them to be fixed, and fixed parameters do not have standard errors. In this example, the coefficient on `ln_ne` has been normalized to 1, so its standard error is missing. As discussed in [Methods and formulas](#), the constant term in the cointegrating equation is not directly estimated in this trend specification but rather is backed out from other estimates. Not all the elements of the VCE that correspond to this parameter are readily available, so the standard error for the `_cons` parameter is missing.

To get a better idea of how our model fits, we predict the cointegrating equation and graph it over time:

```
. predict ce, ce
. line ce year
```



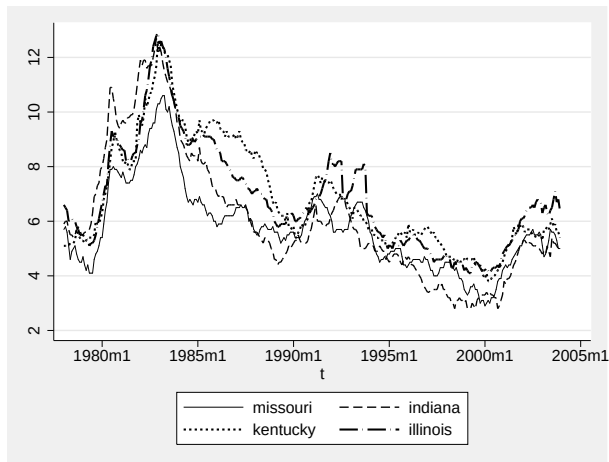
Although the predicted cointegrating equation has the right appearance for the time before the mid-1960s, afterward the predicted cointegrating equation does not look like a stationary series. A better model would account for the trends in the size of the differential.

As discussed in [\[TS\] vec intro](#), simply normalizing one of the coefficients to be one is sufficient to identify the parameters of the single cointegrating vector. When there is more than one cointegrating equation, more restrictions are required.

➤ Example 2

We have data on monthly unemployment rates in Indiana, Illinois, Kentucky, and Missouri from January 1978 through December 2003. We suspect that factor mobility will keep the unemployment rates in equilibrium. The following graph plots the data.

```
. use http://www.stata-press.com/data/r11/urates, clear  
. line missouri indiana kentucky illinois t
```



The graph shows that although the series do appear to move together, the relationship is not as clear as in the previous example. There are periods when Indiana has the highest rate and others when Indiana has the lowest rate. Although the Kentucky rate moves closely with the other series for most of the sample, there is a period in the mid-1980s when the unemployment rate in Kentucky does not fall at the same rate as the other series.

We will model the series with two cointegrating equations and no linear or quadratic time trends in the original series. Because we are focusing on the cointegrating vectors, we use the `noetable` option to suppress displaying the short-run estimation table.

(Continued on next page)

```
. vec missouri indiana kentucky illinois, trend(rconstant) rank(2) lags(4)
> noetable

Vector error-correction model

Sample: 1978m5 - 2003m12                                No. of obs      =          308
                                                           AIC              = -2.306048
Log likelihood = 417.1314                                HQIC            = -2.005818
Det(Sigma_ml)  = 7.83e-07                                SBIC            = -1.555184
```

Cointegrating equations

Equation	Parms	chi2	P>chi2
_ce1	2	133.3885	0.0000
_ce2	2	195.6324	0.0000

Identification: beta is exactly identified
 Johansen normalization restrictions imposed

beta	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
_ce1						
missouri	1	.	.	.	.	.
indiana	-2.19e-18	.	.	.	.	.
kentucky	.3493902	.2005537	1.74	0.081	-.0436879	.7424683
illinois	-1.135152	.2069063	-5.49	0.000	-1.540681	-.7296235
_cons	-.3880707	.4974323	-0.78	0.435	-1.36302	.5868787
_ce2						
missouri	1.09e-16	.	.	.	.	.
indiana	1	.	.	.	.	.
kentucky	.2059473	.2718678	0.76	0.449	-.3269038	.7387985
illinois	-1.51962	.2804792	-5.42	0.000	-2.069349	-.9698907
_cons	2.92857	.6743122	4.34	0.000	1.606942	4.250197

Except for the coefficients on `kentucky` in the two cointegrating equations and the constant term in the first, all the parameters are significant at the 5% level. We can refit the model with the Johansen normalization and the overidentifying constraint that the coefficient on `kentucky` in the second cointegrating equation is zero.

```
. constraint 1 [_ce1]missouri = 1
. constraint 2 [_ce1]indiana  = 0
. constraint 3 [_ce2]missouri = 0
. constraint 4 [_ce2]indiana  = 1
. constraint 5 [_ce2]kentucky = 0
```



```
. vec missouri indiana kentucky illinois, trend(rconstant) rank(2) lags(4)
> noetable bconstraints(1/5)
Iteration 1:      log likelihood = 416.97177
(output omitted)
Iteration 20:     log likelihood = 416.9744
Vector error-correction model
Sample: 1978m5 - 2003m12
Log likelihood = 416.9744
Det(Sigma_ml) = 7.84e-07
No. of obs      =      308
AIC             = -2.311522
HQIC           = -2.016134
SBIC           = -1.572769
```

Cointegrating equations

Equation	Parms	chi2	P>chi2
_ce1	2	145.233	0.0000
_ce2	1	209.9344	0.0000

Identification: beta is overidentified

Identifying constraints:

- (1) [_ce1]missouri = 1
- (2) [_ce1]indiana = 0
- (3) [_ce2]missouri = 0
- (4) [_ce2]indiana = 1
- (5) [_ce2]kentucky = 0

beta	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
_ce1	missouri	1	.	.	.	.
	indiana	(omitted)				
	kentucky	.2521685	.1649653	1.53	0.126	-.0711576 .5754946
	illinois	-1.037453	.1734165	-5.98	0.000	-1.377343 - .6975626
	_cons	-.3891102	.4726968	-0.82	0.410	-1.315579 .5373586
_ce2	missouri	(omitted)				
	indiana	1	.	.	.	.
	kentucky	(omitted)				
	illinois	-1.314265	.0907071	-14.49	0.000	-1.492048 -1.136483
	_cons	2.937016	.6448924	4.55	0.000	1.67305 4.200982

LR test of identifying restrictions: chi2(1) = .3139 Prob > chi2 = 0.575

The test of the overidentifying restriction does not reject the null hypothesis that the restriction is valid, and the p -value on the coefficient on *kentucky* in the first cointegrating equation indicates that it is not significant. We will leave the variable in the model and attribute the lack of significance to whatever caused the *kentucky* series to temporarily rise above the others from 1985 until 1990, though we could instead consider removing *kentucky* from the model.

Next, we look at the estimates of the adjustment parameters. In the output below, we `replay` the previous results. We specify the `alpha` option so that `vec` will display an estimation table for the estimates of the adjustment parameters, and we specify `nobtable` to suppress the table for the parameters of the cointegrating equations because we have already looked at those.

```
. vec, alpha nobtable noetable
Vector error-correction model
Sample: 1978m5 - 2003m12
Log likelihood = 416.9744
Det(Sigma_ml) = 7.84e-07
Adjustment parameters
```

Equation	Parms	chi2	P>chi2
D_missouri	2	19.39607	0.0001
D_indiana	2	6.426086	0.0402
D_kentucky	2	8.524901	0.0141
D_illinois	2	22.32893	0.0000

alpha	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
D_missouri						
_ce1						
L1.	-.0683152	.0185763	-3.68	0.000	-.1047242	-.0319063
_ce2						
L1.	.0405613	.0112417	3.61	0.000	.018528	.0625946
D_indiana						
_ce1						
L1.	-.0342096	.0220955	-1.55	0.122	-.0775159	.0090967
_ce2						
L1.	.0325804	.0133713	2.44	0.015	.0063732	.0587877
D_kentucky						
_ce1						
L1.	-.0482012	.0231633	-2.08	0.037	-.0936004	-.0028021
_ce2						
L1.	.0374395	.0140175	2.67	0.008	.0099657	.0649133
D_illinois						
_ce1						
L1.	.0138224	.0227041	0.61	0.543	-.0306768	.0583215
_ce2						
L1.	.0567664	.0137396	4.13	0.000	.0298373	.0836955

LR test of identifying restrictions: chi2(1) = .3139 Prob > chi2 = 0.575

All the coefficients are significant at the 5% level, except those on Indiana and Illinois in the first cointegrating equation. From an economic perspective, the issue is whether the unemployment rates in Indiana and Illinois adjust when the first cointegrating equation is out of equilibrium. We could impose restrictions on one or both of those parameters and refit the model, or we could just decide to use the current results.



□ Technical note

`vec` can be used to fit models in which the parameters in β are not identified, in which case only the parameters in Π and the moving-average impact matrix C are identified. When the parameters in β are not identified, the values of $\hat{\beta}$ and $\hat{\alpha}$ can vary depending on the starting values. However, the estimates of Π and C are identified and have known asymptotic distributions. This method is valid because these additional normalization restrictions impose no restriction on Π or C .

□

Specification of constants and trends

As discussed in [TS] [vec intro](#), allowing for a constant term and linear time trend allow us to write the VECM as

$$\Delta y_t = \alpha(\beta y_{t-1} + \mu + \rho t) + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \gamma + \tau t + \epsilon_t$$

Five different trend specifications are available:

Option in <code>trend()</code>	Parameter restrictions	Johansen (1995) notation
<code>trend</code>	none	$H(r)$
<code>rtrend</code>	$\tau = 0$	$H^*(r)$
<code>constant</code>	$\rho = 0$, and $\tau = 0$	$H_1(r)$
<code>rconstant</code>	$\rho = 0$, $\gamma = 0$ and $\tau = 0$	$H_1^*(r)$
<code>none</code>	$\mu = 0$, $\rho = 0$, $\gamma = 0$, and $\tau = 0$	$H_2(r)$

`trend(trend)` allows for a linear trend in the cointegrating equations and a quadratic trend in the undifferenced data. A linear trend in the cointegrating equations implies that the cointegrating equations are assumed to be trend stationary.

`trend(rtrend)` defines a restricted trend model that excludes linear trends in the differenced data but allows for linear trends in the cointegrating equations. As in the previous case, a linear trend in a cointegrating equation implies that the cointegrating equation is trend stationary.

`trend(constant)` defines a model with an unrestricted constant. This allows for a linear trend in the undifferenced data and cointegrating equations that are stationary around a nonzero mean. This is the default.

`trend(rconstant)` defines a model with a restricted constant in which there is no linear or quadratic trend in the undifferenced data. A nonzero μ allows for the cointegrating equations to be stationary around nonzero means, which provide the only intercepts for differenced data. Seasonal indicators are not allowed with this specification.

`trend(none)` defines a model that does not include a trend or a constant. When there is no trend or constant, the cointegrating equations are restricted to being stationary with zero means. Also, after adjusting for the effects of lagged endogenous variables, the differenced data are modeled as having mean zero. Seasonal indicators are not allowed with this specification.

□ Technical note

`vec` uses a switching algorithm developed by [Boswijk \(1995\)](#) to maximize the log-likelihood function when constraints are placed on the parameters. The starting values affect both the ability of the algorithm to find a maximum and its speed in finding that maximum. By default, `vec` uses the parameter estimates that correspond to Johansen’s normalization. Sometimes, other starting values will cause the algorithm to find a maximum faster.

To specify starting values for the parameters in α , we specify a $1 \times (K * r)$ matrix in the `afrom()` option. Specifying starting values for the parameters in β is slightly more complicated. As explained in [Methods and formulas](#), specifying `trend(constant)`, `trend(rtrend)`, or `trend(trend)` causes some of the estimates of the trend parameters appearing in $\hat{\beta}$ to be “backed out”. The switching algorithm estimates only the parameters of the cointegrating equations whose estimates are saved in `e(betavec)`. For this reason, only the parameters saved in `e(betavec)` can have their initial values set via `bfrom()`.

The table below describes which trend parameters in the cointegrating equations are estimated by the switching algorithm for each of the five specifications.

Trend specification	Trend parameters in cointegrating equations	Trend parameter estimated via switching algorithm
none	none	none
rconstant	_cons	_cons
constant	_cons	none
rtrend	_cons, _trend	_trend
trend	_cons, _trend	none

□

Collinearity

As expected, collinearity among variables causes some parameters to be unidentified numerically. If `vec` encounters perfect collinearity among the dependent variables, it exits with an error.

In contrast, if `vec` encounters perfect collinearity that appears to be due to too many lags in the model, `vec` displays a warning message and reduces the maximum lag included in the model in an effort to find a model with fewer lags in which all the parameters are identified by the data. Specifying the `noreduce` option causes `vec` to skip over these additional checks and corrections for collinearity. Thus the `noreduce` option can be used to force the estimation to proceed when not all the parameters are identified by the data. When some parameters are not identified because of collinearity, the results cannot be interpreted but can be used to find the source of the collinearity.

Saved results

vec saves the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(k_rank)</code>	number of unconstrained parameters
<code>e(k_eq)</code>	number of equations
<code>e(k_dv)</code>	number of dependent variables
<code>e(k_ce)</code>	number of cointegrating equations
<code>e(n_lags)</code>	number of lags
<code>e(df_m)</code>	model degrees of freedom
<code>e(ll)</code>	log likelihood
<code>e(chi2_res)</code>	value of test of overidentifying restrictions
<code>e(df_lr)</code>	degrees of freedom of the test of overidentifying restrictions
<code>e(beta_iden)</code>	1 if the parameters in β are identified and 0 otherwise
<code>e(beta_icnt)</code>	number of independent restrictions placed on β
<code>e(k_#)</code>	number of variables in equation #
<code>e(df_m#)</code>	model degrees of freedom in equation #
<code>e(r2_#)</code>	R^2 of equation #
<code>e(chi2_#)</code>	χ^2 statistic for equation #
<code>e(rmse_#)</code>	RMSE of equation #
<code>e(aic)</code>	value of AIC
<code>e(hqic)</code>	value of HQIC
<code>e(sbic)</code>	value of SBIC
<code>e(tmin)</code>	minimum time
<code>e(tmax)</code>	maximum time
<code>e(detsig_ml)</code>	determinant of the estimated covariance matrix
<code>e(rank)</code>	rank of $e(V)$
<code>e(converge)</code>	1 if the switching algorithm converged, 0 if it did not converge

Macros

<code>e(cmd)</code>	vec
<code>e(cmdline)</code>	command as typed
<code>e(trend)</code>	trend specified
<code>e(tsfmt)</code>	format of the time variable
<code>e(tvar)</code>	name of the time variable
<code>e(endog)</code>	endogenous variables
<code>e(eqnames)</code>	equation names
<code>e(cenames)</code>	names of cointegrating equations
<code>e(reduce_opt)</code>	noreduce, if noreduce is specified
<code>e(reduce_lags)</code>	list of maximum lags to which the model has been reduced
<code>e(title)</code>	title in estimation output
<code>e(aconstraints)</code>	constraints placed on α
<code>e(bconstraints)</code>	constraints placed on β
<code>e(sindicators)</code>	sindicators, if specified
<code>e(properties)</code>	b V
<code>e(predict)</code>	program used to implement predict
<code>e(marginsok)</code>	predictions allowed by margins
<code>e(marginsnotok)</code>	predictions disallowed by margins

Matrices	
e(b)	estimates of short-run parameters
e(V)	VCE of short-run parameter estimates
e(beta)	estimates of β
e(V_beta)	VCE of $\hat{\beta}$
e(betavec)	directly obtained estimates of β
e(pi)	estimates of $\hat{\Pi}$
e(V_pi)	VCE of $\hat{\Pi}$
e(alpha)	estimates of α
e(V_alpha)	VCE of $\hat{\alpha}$
e(omega)	estimates of $\hat{\Omega}$
e(mai)	estimates of $\mathbf{C}$
e(V_mai)	VCE of $\hat{\mathbf{C}}$
Functions	
e(sample)	marks estimation sample

Methods and formulas

vec is implemented as an ado-file. *Methods and formulas* are presented under the following headings:

- General specification of the VECM
- The log-likelihood function
 - Unrestricted trend
 - Restricted trend
 - Unrestricted constant
 - Restricted constant
 - No trend
- Estimation with Johansen identification
- Estimation with constraints: β identified
- Estimation with constraints: β not identified
- Formulas for the information criteria
- Formulas for predict

General specification of the VECM

vec estimates the parameters of a VECM that can be written as

$$\Delta \mathbf{y}_t = \alpha \beta' \mathbf{y}_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \mathbf{v} + \delta t + \mathbf{w}_1 s_1 + \cdots + \mathbf{w}_m s_m + \boldsymbol{\epsilon}_t \tag{1}$$

where

- $\mathbf{y}_t$ is a $K \times 1$ vector of endogenous variables,
- α is a $K \times r$ matrix of parameters,
- β is a $K \times r$ matrix of parameters,
- $\Gamma_1, \dots, \Gamma_{p-1}$ are $K \times K$ matrices of parameters,
- $\mathbf{v}$ is a $K \times 1$ vector of parameters,
- δ is a $K \times 1$ vector of trend coefficients,

t is a linear time trend,

$s_1, \dots, s_m$ are orthogonalized seasonal indicators specified in the `sindicators()` option, and

$\mathbf{w}_1, \dots, \mathbf{w}_m$ are $K \times 1$ vectors of coefficients on the orthogonalized seasonal indicators.

There are two types of deterministic elements in (1): the trend, $\mathbf{v} + \delta t$, and the orthogonalized seasonal terms, $\mathbf{w}_1 s_1 + \dots + \mathbf{w}_m s_m$. Johansen (1995, chap. 11) shows that inference about the number of cointegrating equations is based on nonstandard distributions and that the addition of any term that generalizes the deterministic specification in (1) changes the asymptotic distributions of the statistics used for inference on the number of cointegrating equations and the asymptotic distribution of the ML estimator of the cointegrating equations. In fact, Johansen (1995, 84) notes that including event indicators causes the statistics used for inference on the number of cointegrating equations to have asymptotic distributions that must be computed case by case. For this reason, event indicators may not be specified in the present version of `vec`.

If seasonal indicators are included in the model, they cannot be collinear with a constant term. If they are collinear with a constant term, one of the indicator variables is omitted.

As discussed in [Specification of constants and trends](#), we can reparametrize the model as

$$\Delta \mathbf{y}_t = \alpha(\beta \mathbf{y}_{t-1} + \boldsymbol{\mu} + \rho t) + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \boldsymbol{\gamma} + \boldsymbol{\tau} t + \boldsymbol{\epsilon}_t \quad (2)$$

The log-likelihood function

We can maximize the log-likelihood function much more easily by writing it in concentrated form. In fact, as discussed below, in the simple case with the Johansen normalization on β and no constraints on α , concentrating the log-likelihood function produces an analytical solution for the parameter estimates.

To concentrate the log likelihood, rewrite (2) as

$$\mathbf{Z}_{0t} = \alpha \tilde{\beta}' \mathbf{Z}_{1t} + \Psi \mathbf{Z}_{2t} + \boldsymbol{\epsilon}_t \quad (3)$$

where $\mathbf{Z}_{0t}$ is a $K \times 1$ vector of variables $\Delta \mathbf{y}_t$, α is the $K \times r$ matrix of adjustment coefficients, and $\boldsymbol{\epsilon}_t$ is a $K \times 1$ vector of independently and identically distributed normal vectors with mean 0 and contemporaneous covariance matrix Ω . $\mathbf{Z}_{1t}$, $\mathbf{Z}_{2t}$, $\tilde{\beta}$, and Ψ depend on the trend specification and are defined below.

The log-likelihood function for the model in (3) is

$$\begin{aligned} L = & -\frac{1}{2} \left\{ TK \ln(2\pi) + T \ln(|\Omega|) \right. \\ & \left. + \sum_{t=1}^T (\mathbf{Z}_{0t} - \alpha \tilde{\beta}' \mathbf{Z}_{1t} - \Psi \mathbf{Z}_{2t})' \Omega^{-1} (\mathbf{Z}_{0t} - \alpha \tilde{\beta}' \mathbf{Z}_{1t} - \Psi \mathbf{Z}_{2t}) \right\} \end{aligned} \quad (4)$$

with the constraints that α and $\tilde{\beta}$ have rank r .

Johansen (1995, chap. 6), building on Anderson (1951), shows how the Ψ parameters can be expressed as analytic functions of α , β , and the data, yielding the concentrated log-likelihood function

$$L_c = -\frac{1}{2} \left\{ TK \ln(2\pi) + T \ln(|\Omega|) + \sum_{t=1}^T (\mathbf{R}_{0t} - \alpha \tilde{\beta}' \mathbf{R}_{1t})' \Omega^{-1} (\mathbf{R}_{0t} - \alpha \tilde{\beta}' \mathbf{R}_{1t}) \right\} \quad (5)$$

where

$$\mathbf{M}_{ij} = T^{-1} \sum_{t=1}^T \mathbf{Z}_{it} \mathbf{Z}_{jt}', \quad i, j \in \{0, 1, 2\};$$

$$\mathbf{R}_{0t} = \mathbf{Z}_{0t} - \mathbf{M}_{02} \mathbf{M}_{22}^{-1} \mathbf{Z}_{2t}; \text{ and}$$

$$\mathbf{R}_{1t} = \mathbf{Z}_{1t} - \mathbf{M}_{12} \mathbf{M}_{22}^{-1} \mathbf{Z}_{2t}.$$

The definitions of $\mathbf{Z}_{1t}$, $\mathbf{Z}_{2t}$, $\tilde{\beta}$, and Ψ change with the trend specifications, although some of their components stay the same.

Unrestricted trend

When the trend in the VECM is unrestricted, we can define the variables in (3) directly in terms of the variables in (1):

$$\mathbf{Z}_{1t} = \mathbf{y}_{t-1} \text{ is } K \times 1$$

$$\mathbf{Z}_{2t} = (\Delta \mathbf{y}'_{t-1}, \dots, \Delta \mathbf{y}'_{t-p+1}, 1, t, s_1, \dots, s_m)' \text{ is } \{K(p-1) + 2 + m\} \times 1;$$

$$\Psi = (\Gamma_1, \dots, \Gamma_{p-1}, \mathbf{v}, \delta, \mathbf{w}_1, \dots, \mathbf{w}_m) \text{ is } K \times \{K(p-1) + 2 + m\}$$

$$\tilde{\beta} = \beta \text{ is the } K \times r \text{ matrix composed of the } r \text{ cointegrating vectors.}$$

In the unrestricted trend specification, $m_1 = K$, $m_2 = K(p-1) + 2 + m$, and there are $n_{\text{parms}} = Kr + Kr + K\{K(p-1) + 2 + m\}$ parameters in (3).

Restricted trend

When there is a restricted trend in the VECM in (2), $\tau = 0$, but the intercept $\mathbf{v} = \alpha\mu + \gamma$ is unrestricted. The VECM with the restricted trend can be written as

$$\Delta \mathbf{y}_t = \alpha(\beta', \rho) \begin{pmatrix} \mathbf{y}_{t-1} \\ t \end{pmatrix} + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \mathbf{v} + \mathbf{w}_1 s_1 + \dots + \mathbf{w}_m s_m + \epsilon_t$$

This equation can be written in the form of (3) by defining

$$\mathbf{Z}_{1t} = (\mathbf{y}'_{t-1}, t)' \text{ is } (K+1) \times 1$$

$$\mathbf{Z}_{2t} = (\Delta \mathbf{y}'_{t-1}, \dots, \Delta \mathbf{y}'_{t-p+1}, 1, s_1, \dots, s_m)' \text{ is } \{K(p-1) + 1 + m\} \times 1$$

$$\Psi = (\Gamma_1, \dots, \Gamma_{p-1}, \mathbf{v}, \mathbf{w}_1, \dots, \mathbf{w}_m) \text{ is } K \times \{K(p-1) + 1 + m\}$$

$$\tilde{\beta} = (\beta', \rho)' \text{ is the } (K+1) \times r \text{ matrix composed of the } r \text{ cointegrating vectors and the } r \text{ trend coefficients } \rho$$

In the restricted trend specification, $m_1 = K+1$, $m_2 = \{K(p-1) + 1 + m\}$, and there are $n_{\text{parms}} = Kr + (K+1)r + K\{K(p-1) + 1 + m\}$ parameters in (3).

Unrestricted constant

An unrestricted constant in the VECM in (2) is equivalent to setting $\delta = 0$ in (1), which can be written in the form of (3) by defining

$$\mathbf{Z}_{1t} = \mathbf{y}_{t-1} \text{ is } (K \times 1)$$

$$\mathbf{Z}_{2t} = (\Delta \mathbf{y}'_{t-1}, \dots, \Delta \mathbf{y}'_{t-p+1}, 1, s_1, \dots, s_m)' \text{ is } \{K(p-1) + 1 + m\} \times 1;$$

$$\Psi = (\Gamma_1, \dots, \Gamma_{p-1}, \mathbf{v}, \mathbf{w}_1, \dots, \mathbf{w}_m) \text{ is } K \times \{K(p-1) + 1 + m\}$$

$$\tilde{\beta} = \beta \text{ is the } K \times r \text{ matrix composed of the } r \text{ cointegrating vectors}$$

In the unrestricted constant specification, $m_1 = K$, $m_2 = \{K(p-1) + 1 + m\}$, and there are $n_{\text{parms}} = Kr + Kr + K\{K(p-1) + 1 + m\}$ parameters in (3).

Restricted constant

When there is a restricted constant in the VECM in (2), it can be written in the form of (3) by defining

$$\mathbf{Z}_{1t} = (\mathbf{y}'_{t-1}, 1)' \text{ is } (K+1) \times 1$$

$$\mathbf{Z}_{2t} = (\Delta \mathbf{y}'_{t-1}, \dots, \Delta \mathbf{y}'_{t-p+1})' \text{ is } K(p-1) \times 1$$

$$\Psi = (\Gamma_1, \dots, \Gamma_{p-1}) \text{ is } K \times K(p-1)$$

$$\tilde{\beta} = (\beta', \mu')' \text{ is the } (K+1) \times r \text{ matrix composed of the } r \text{ cointegrating vectors and the } r \text{ constants in the cointegrating relations.}$$

In the restricted trend specification, $m_1 = K+1$, $m_2 = K(p-1)$, and there are $n_{\text{parms}} = Kr + (K+1)r + K\{K(p-1)\}$ parameters in (3).

No trend

When there is no trend in the VECM in (2), it can be written in the form of (3) by defining

$$\mathbf{Z}_{1t} = \mathbf{y}_{t-1} \text{ is } K \times 1$$

$$\mathbf{Z}_{2t} = (\Delta \mathbf{y}'_{t-1}, \dots, \Delta \mathbf{y}'_{t-p+1})' \text{ is } K(p-1) + m \times 1$$

$$\Psi = (\Gamma_1, \dots, \Gamma_{p-1}) \text{ is } K \times K(p-1)$$

$$\tilde{\beta} = \beta \text{ is } K \times r \text{ matrix of } r \text{ cointegrating vectors}$$

In the no-trend specification, $m_1 = K$, $m_2 = K(p-1)$, and there are $n_{\text{parms}} = Kr + Kr + K\{K(p-1)\}$ parameters in (3).

Estimation with Johansen identification

Not all the parameters in α and $\tilde{\beta}$ are identified. Consider the simple case in which $\tilde{\beta}$ is $K \times r$ and let $\mathbf{Q}$ be a nonsingular $r \times r$ matrix. Then

$$\alpha \tilde{\beta}' = \alpha \mathbf{Q} \mathbf{Q}^{-1} \tilde{\beta}' = \alpha \mathbf{Q} (\tilde{\beta} \mathbf{Q}'^{-1})' = \alpha \dot{\beta}'$$

Substituting $\alpha \dot{\beta}'$ into the log likelihood in (5) for $\alpha \tilde{\beta}'$ would not change the value of the log likelihood, so some a priori identification restrictions must be found to identify α and $\tilde{\beta}$. As discussed in Johansen (1995, chap. 5 and 6) and Boswijk (1995), if the restrictions exactly identify or overidentify

$\tilde{\beta}$, the estimates of the unconstrained parameters in $\tilde{\beta}$ will be superconsistent, meaning that the estimates of the free parameters in $\tilde{\beta}$ will converge at a faster rate than estimates of the short-run parameters in α and Γ_i . This allows the distribution of the estimator of the short-run parameters to be derived conditional on the estimated $\tilde{\beta}$.

Johansen (1995, chap. 6) has proposed a normalization method for use when theory does not provide sufficient a priori restrictions to identify the cointegrating vector. This method has become widely adopted by researchers. Johansen's identification scheme is

$$\tilde{\beta}' = (\mathbf{I}_r, \check{\beta}') \quad (6)$$

where $\mathbf{I}_r$ is the $r \times r$ identity matrix and $\check{\beta}$ is a $(m_1 - r) \times r$ matrix of identified parameters.

Johansen's identification method places r^2 linearly independent constraints on the parameters in $\tilde{\beta}$, thereby defining an exactly identified model. The total number of freely estimated parameters is $n_{\text{parms}} - r^2 = \{K + m_2 + (K + m_1 - r)r\}$, and the degrees of freedom d is calculated as the integer part of $(n_{\text{parms}} - r^2)/K$.

When only the rank and the Johansen identification restrictions are placed on the model, we can further manipulate the log likelihood in (5) to obtain analytic formulas for the parameters in $\tilde{\beta}$, α , and Ω . For a given value of $\tilde{\beta}$, α and Ω can be found by regressing $\mathbf{R}_{0t}$ on $\tilde{\beta}'\mathbf{R}_{1t}$. This allows a further simplification of the problem in which

$$\begin{aligned} \alpha(\tilde{\beta}) &= \mathbf{S}_{01}\tilde{\beta}(\tilde{\beta}'\mathbf{S}_{11}\tilde{\beta})^{-1} \\ \Omega(\tilde{\beta}) &= \mathbf{S}_{00} - \mathbf{S}_{01}\tilde{\beta}(\tilde{\beta}'\mathbf{S}_{11}\tilde{\beta})^{-1}\tilde{\beta}'\mathbf{S}_{10} \\ \mathbf{S}_{ij} &= (1/T) \sum_{t=1}^T R_{it}R'_{jt} \quad i, j \in \{0, 1\} \end{aligned}$$

Johansen (1995) shows that by inserting these solutions into equation (5), $\hat{\beta}$ is given by the r eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_r$ corresponding to the r largest eigenvalues $\lambda_1, \dots, \lambda_r$ that solve the generalized eigenvalue problem

$$|\lambda_i \mathbf{S}_{11} - \mathbf{S}_{10}\mathbf{S}_{00}^{-1}\mathbf{S}_{01}| = 0 \quad (7)$$

The eigenvectors corresponding to $\lambda_1, \dots, \lambda_r$ that solve (7) are the unidentified parameter estimates. To impose the identification restrictions in (6), we normalize the eigenvectors such that

$$\lambda_i \mathbf{S}_{11} \mathbf{v}_i = \mathbf{S}_{01}\mathbf{S}_{00}^{-1}\mathbf{S}_{01} \mathbf{v}_i \quad (8)$$

and

$$\mathbf{v}_i' \mathbf{S}_{11} \mathbf{v}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

At the optimum the log-likelihood function with the Johansen identification restrictions can be expressed in terms of T , K , $\mathbf{S}_{00}$, and the r largest eigenvalues

$$L_c = -\frac{1}{2}T \left\{ K \ln(2\pi) + K + \ln(|\mathbf{S}_{00}|) + \sum_{i=1}^r \ln(1 - \hat{\lambda}_i) \right\}$$

where the $\hat{\lambda}_i$ are the eigenvalues that solve (7), (8), and (9).

Using the normalized $\hat{\beta}$, we can then obtain the estimates

$$\hat{\alpha} = \mathbf{S}_{01}\hat{\beta}(\hat{\beta}'\mathbf{S}_{11}\hat{\beta})^{-1} \quad (10)$$

and

$$\hat{\Omega} = \mathbf{S}_{00} - \hat{\alpha}\hat{\beta}'\mathbf{S}_{10}$$

Let $\hat{\beta}_y$ be a $K \times r$ matrix that contains the estimates of the parameters in β in (1). $\hat{\beta}_y$ differs from $\hat{\beta}$ in that any trend parameter estimates are omitted from $\hat{\beta}$. We can then use $\hat{\beta}_y$ to obtain predicted values for the r nondemeaned cointegrating equations

$$\hat{\mathbf{E}}_t = \hat{\beta}_y' \mathbf{y}_t$$

The r series in $\hat{\mathbf{E}}_t$ are called the predicted, nondemeaned cointegrating equations because they still contain the terms μ and ρ . We want to work with the predicted, demeaned cointegrating equations. Thus we need estimates of μ and ρ . In the `trend(rconstant)` specification, the algorithm directly produces the estimator $\hat{\mu}$. Similarly, in the `trend(rtrend)` specification, the algorithm directly produces the estimator $\hat{\rho}$. In the remaining cases, to back out estimates of μ and ρ , we need estimates of $\mathbf{v}$ and δ , which we can obtain by estimating the parameters of the following VAR:

$$\Delta \mathbf{y}_t = \alpha \hat{\mathbf{E}}_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \mathbf{v} + \delta t + \mathbf{w}_1 s_1 + \cdots + \mathbf{w}_m s_m + \epsilon_t \quad (11)$$

Depending on the trend specification, we use $\hat{\alpha}$ to back out the estimates of

$$\hat{\mu} = (\hat{\alpha}'\hat{\alpha})^{-1}\hat{\alpha}'\hat{\mathbf{v}} \quad (12)$$

$$\hat{\rho} = (\hat{\alpha}'\hat{\alpha})^{-1}\hat{\alpha}'\hat{\delta} \quad (13)$$

if they are not already in $\hat{\beta}$ and are included in the trend specification.

We then augment $\hat{\beta}_y$ to

$$\hat{\beta}_f' = (\hat{\beta}_y', \hat{\mu}, \hat{\rho})$$

where the estimates of $\hat{\mu}$ and $\hat{\rho}$ are either obtained from $\hat{\beta}$ or backed out using (12) and (13). We next use $\hat{\beta}_f$ to obtain the r predicted, demeaned cointegrating equations, $\hat{\mathbf{E}}_t$, via

$$\hat{\mathbf{E}}_t = \hat{\beta}_f' (\mathbf{y}_t', 1, t)'$$

We last obtain estimates of all the short-run parameters from the VAR:

$$\Delta \mathbf{y}_t = \alpha \hat{\mathbf{E}}_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \gamma + \tau t + \mathbf{w}_1 s_1 + \cdots + \mathbf{w}_m s_m + \epsilon_t \quad (14)$$

Because the estimator $\hat{\beta}_f$ converges in probability to its true value at a rate faster than $T^{-\frac{1}{2}}$, we can take our estimated $\hat{\mathbf{E}}_{t-1}$ as given data in (14). This allows us to estimate the variance–covariance (VCE) matrix of the estimates of the parameters in (14) by using the standard VAR VCE estimator. Equation (11) can be used to obtain consistent estimates of all the parameters and of the VCE of all the parameters, except $\mathbf{v}$ and δ . The standard VAR VCE of $\hat{\mathbf{v}}$ and $\hat{\delta}$ is incorrect because these estimates converge at a faster rate. This is why it is important to use the predicted, demeaned cointegrating equations, $\hat{\mathbf{E}}_{t-1}$, when estimating the short-run parameters and trend terms. In keeping with the cointegration literature, `vec` makes a small-sample adjustment to the VCE estimator so that the divisor is $(T - d)$ instead of T , where d represents the degrees of freedom of the model. d is calculated as the integer part of n_{parms}/K , where n_{parms} is the total number of freely estimated parameters in the model.

In the `trend(rconstant)` specification, the estimation procedure directly estimates μ . For `trend(constant)`, `trend(rtrend)`, and `trend(trend)`, the estimates of μ are backed out using (12). In the `trend(rtrend)` specification, the estimation procedure directly estimates ρ . In the `trend(trend)` specification, the estimates of ρ are backed out using (13). Because the elements of the estimated VCE are readily available only when the estimates are obtained directly, when the trend parameter estimates are backed out, their elements in the VCE for $\hat{\beta}_f$ are missing.

Under the Johansen identification restrictions, `vec` obtains $\hat{\beta}$, the estimates of the parameters in the $r \times m_1$ matrix $\hat{\beta}'$ in (5). The VCE of `vec`($\hat{\beta}$) is $rm_1 \times rm_1$. Per Johansen (1995), the asymptotic distribution of $\hat{\beta}$ is mixed Gaussian, and its VCE is consistently estimated by

$$\left(\frac{1}{T-d} \right) (\mathbf{I}_r \otimes \mathbf{H}_J) \{ (\hat{\alpha}' \Omega^{-1} \hat{\alpha}) \otimes (\mathbf{H}_J' \mathbf{S}_{11} \mathbf{H}_J) \}^{-1} (\mathbf{I}_r \otimes \mathbf{H}_J)' \quad (15)$$

where $\mathbf{H}_J$ is the $m_1 \times (m_1 - r)$ matrix given by $\mathbf{H}_J = (\mathbf{0}'_{r \times (m_1 - r)}, \mathbf{I}_{m_1 - r})'$. The VCE reported in `e(V_beta)` is the estimated VCE in (15) augmented with missing values to account for any backed-out estimates of μ or ρ .

The parameter estimates $\hat{\alpha}$ can be found either as a function of $\hat{\beta}$, using (10) or from the VAR in (14). The estimated VCE of $\hat{\alpha}$ reported in `e(V_alpha)` is given by

$$\frac{1}{(T-d)} \hat{\Omega} \otimes \hat{\Sigma}_B$$

where $\hat{\Sigma}_B = (\hat{\beta}' \mathbf{S}_{11} \hat{\beta})^{-1}$.

As we would expect, the estimator of $\Pi = \alpha \beta'$ is

$$\hat{\Pi} = \hat{\alpha} \hat{\beta}'$$

and its estimated VCE is given by

$$\frac{1}{(T-d)} \hat{\Omega} \otimes (\hat{\beta} \hat{\Sigma}_B \hat{\beta}')$$

The moving-average impact matrix $\mathbf{C}$ is estimated by

$$\hat{\mathbf{C}} = \hat{\beta}_\perp (\hat{\alpha}_\perp \hat{\Gamma} \hat{\beta}_\perp)^{-1} \hat{\alpha}'_\perp$$

where $\hat{\beta}_\perp$ is the orthogonal complement of $\hat{\beta}_y$, $\hat{\alpha}_\perp$ is the orthogonal complement of $\hat{\alpha}$, and $\hat{\Gamma} = \mathbf{I}_K - \sum_{i=1}^{p-1} \Gamma_i$. The orthogonal complement of a $K \times r$ matrix $\mathbf{Q}$ that has rank r is a matrix $\mathbf{Q}_\perp$ of rank $K - r$, such that $\mathbf{Q}' \mathbf{Q}_\perp = \mathbf{0}$. Although this operation is not uniquely defined, the results used by `vec` do not depend on the method of obtaining the orthogonal complement. `vec` uses the following method: the orthogonal complement of $\mathbf{Q}$ is given by the r eigenvectors with the highest eigenvalues from the matrix $\mathbf{Q}'(\mathbf{Q}'\mathbf{Q})^{-1}\mathbf{Q}'$.

Per Johansen (1995, chap. 13) and Drukker (2004), the VCE of $\hat{\mathbf{C}}$ is estimated by

$$\frac{T-d}{T} \hat{\mathbf{S}}_q \hat{\mathbf{V}}_{\hat{\mathbf{C}}} \hat{\mathbf{S}}_q' \quad (16)$$

where

$$\widehat{\mathbf{S}}_q = \widehat{\mathbf{C}} \otimes \widehat{\boldsymbol{\xi}}$$

$$\widehat{\boldsymbol{\xi}} = \begin{cases} (\widehat{\boldsymbol{\xi}}_1, \widehat{\boldsymbol{\xi}}_2) & \text{if } p > 1 \\ \widehat{\boldsymbol{\xi}}_1 & \text{if } p = 1 \end{cases}$$

$$\widehat{\boldsymbol{\xi}}_1 = (\widehat{\mathbf{C}}' \widehat{\boldsymbol{\Gamma}}' - \mathbf{I}_K) \bar{\boldsymbol{\alpha}}$$

$$\bar{\boldsymbol{\alpha}} = \widehat{\boldsymbol{\alpha}} (\widehat{\boldsymbol{\alpha}}' \widehat{\boldsymbol{\alpha}})^{-1}$$

$$\widehat{\boldsymbol{\xi}}_2 = \boldsymbol{\iota}_{p-1} \otimes \widehat{\mathbf{C}}$$

$\boldsymbol{\iota}_{p-1}$ is a $(p-1) \times 1$ vector of ones

$\widehat{\mathbf{V}}_{\widehat{\boldsymbol{\nu}}}$ is the estimated VCE of $\widehat{\boldsymbol{\nu}} = (\widehat{\boldsymbol{\alpha}}, \widehat{\boldsymbol{\Gamma}}_1, \dots, \widehat{\boldsymbol{\Gamma}}_{p-1})$

Estimation with constraints: β identified

vec can also fit models in which the adjustment parameters are subject to homogeneous linear constraints and the cointegrating vectors are subject to general linear restrictions. Mathematically, vec allows for constraints of the form

$$\mathbf{R}'_{\boldsymbol{\alpha}} \text{vec}(\boldsymbol{\alpha}) = \mathbf{0} \quad (17)$$

where $\mathbf{R}_{\boldsymbol{\alpha}}$ is a known $Kr \times n_{\boldsymbol{\alpha}}$ constraint matrix, and

$$\mathbf{R}'_{\widetilde{\boldsymbol{\beta}}} \text{vec}(\widetilde{\boldsymbol{\beta}}) = \mathbf{b} \quad (18)$$

where $\mathbf{R}_{\widetilde{\boldsymbol{\beta}}}$ is a known $m_1 r \times n_{\boldsymbol{\beta}}$ constraint matrix and $\mathbf{b}$ is a known $n_{\boldsymbol{\beta}} \times 1$ vector of constants. Although (17) and (18) are intuitive, they can be rewritten in a form to facilitate computation. Specifically, (17) can be written as

$$\text{vec}(\boldsymbol{\alpha}') = \mathbf{G} \mathbf{a} \quad (19)$$

where $\mathbf{G}$ is $Kr \times n_{\boldsymbol{\alpha}}$ and $\mathbf{a}$ is $n_{\boldsymbol{\alpha}} \times 1$. Equation (18) can be rewritten as

$$\text{vec}(\widetilde{\boldsymbol{\beta}}) = \mathbf{H} \mathbf{b} + \mathbf{h}_0 \quad (20)$$

where $\mathbf{H}$ is a known $n_1 r \times n_{\boldsymbol{\beta}}$ matrix, $\mathbf{b}$ is an $n_{\boldsymbol{\beta}} \times 1$ matrix of parameters, and $\mathbf{h}_0$ is a known $n_1 r \times 1$ matrix. See [P] [makecns](#) for a discussion of the different ways of specifying the constraints.

When constraints are specified via the `aconstraints()` and `bconstraints()` options, the [Boswijk \(1995\)](#) rank method determines whether the parameters in $\widetilde{\boldsymbol{\beta}}$ are underidentified, exactly identified, or overidentified.

[Boswijk \(1995\)](#) uses the [Rothenberg \(1971\)](#) method to determine whether the parameters in $\widetilde{\boldsymbol{\beta}}$ are identified. Thus the parameters in $\widetilde{\boldsymbol{\beta}}$ are exactly identified if $\rho_{\boldsymbol{\beta}} = r^2$, and the parameters in $\widetilde{\boldsymbol{\beta}}$ are overidentified if $\rho_{\boldsymbol{\beta}} > r^2$, where

$$\rho_{\boldsymbol{\beta}} = \text{rank} \left\{ \mathbf{R}_{\widetilde{\boldsymbol{\beta}}} (\mathbf{I}_r \otimes \widetilde{\boldsymbol{\beta}}) \right\}$$

and $\widetilde{\boldsymbol{\beta}}$ is a full-rank matrix with the same dimensions as $\widetilde{\boldsymbol{\beta}}$. The computed $\rho_{\boldsymbol{\beta}}$ is saved in `e(beta_icnt)`.

Similarly, the number of freely estimated parameters in α and $\tilde{\beta}$ is given by ρ_{jacob} , where

$$\rho_{\text{jacob}} = \text{rank} \left\{ (\hat{\alpha} \otimes \mathbf{I}_{m_1})\mathbf{H}, (\mathbf{I}_K \otimes \hat{\beta})\mathbf{G} \right\}$$

Using ρ_{jacob} , we can calculate several other parameter counts of interest. In particular, the degrees of freedom of the overidentifying test are given by $(K + m_1 - r)r - \rho_{\text{jacob}}$, and the number of freely estimated parameters in the model is $n_{\text{parms}} = Km_2 + \rho_{\text{jacob}}$.

Although the problem of maximizing the log-likelihood function in (4), subject to the constraints in (17) and (18), could be handled by the algorithms in [R] **ml**, the switching algorithm of Boswijk (1995) has proven to be more convergent. For this reason, **vec** uses the Boswijk (1995) switching algorithm to perform the optimization.

Given starting values $(\hat{\mathbf{b}}_0, \hat{\mathbf{a}}_0, \hat{\Omega}_0)$, the algorithm iteratively updates the estimates until convergence is achieved, as follows:

$\hat{\alpha}_j$ is constructed from (19) and $\hat{\mathbf{a}}_j$

$\hat{\beta}_j$ is constructed from (20) and $\hat{\mathbf{b}}_j$

$$\hat{\mathbf{b}}_{j+1} = \{\mathbf{H}'(\hat{\alpha}'_j \hat{\Omega}_j^{-1} \hat{\alpha}_j \otimes \mathbf{S}_{11})\mathbf{H}\}^{-1} \mathbf{H}'(\hat{\alpha}_j \hat{\Omega}_j^{-1} \otimes \mathbf{S}_{11})\{\text{vec}(\hat{\mathbf{P}}) - (\hat{\alpha}_j \otimes \mathbf{I}_{n_{z1}})\mathbf{h}_0\}$$

$$\hat{\mathbf{a}}_{j+1} = \{\mathbf{G}(\hat{\Omega}_j^{-1} \otimes \hat{\beta}_j \mathbf{S}_{11} \hat{\beta}_j)\mathbf{G}\}^{-1} \mathbf{G}'(\hat{\Omega}_j^{-1} \otimes \hat{\beta}_j \mathbf{S}_{11})\text{vec}(\hat{\mathbf{P}})$$

$$\hat{\Omega}_{j+1} = \mathbf{S}_{00} - \mathbf{S}_{01}\hat{\beta}_j\hat{\alpha}'_j - \hat{\alpha}_j\hat{\beta}'_j\mathbf{S}_{10} + \hat{\alpha}_j\hat{\beta}'_j\mathbf{S}_{11}\hat{\beta}_j\hat{\alpha}'_j$$

The estimated VCE of $\hat{\beta}$ is given by

$$\frac{1}{(T-d)} \mathbf{H}\{\mathbf{H}'(\mathbf{W} \otimes \mathbf{S}_{11})\mathbf{H}\}^{-1} \mathbf{H}'$$

where $\mathbf{W}$ is $\hat{\alpha}'\hat{\Omega}^{-1}\hat{\alpha}$. As in the case without constraints, the estimated VCE of $\hat{\alpha}$ can be obtained either from the VCE of the short-run parameters, as described below, or via the formula

$$\hat{V}_{\hat{\alpha}} = \frac{1}{(T-d)} \mathbf{G} \left[\mathbf{G}' \left\{ \hat{\Omega}^{-1} \otimes (\hat{\beta}' \mathbf{S}_{11} \hat{\beta}) \mathbf{G} \right\}^{-1} \right] \mathbf{G}'$$

Boswijk (1995) notes that, as long as the parameters of the cointegrating equations are exactly identified or overidentified, the constrained ML estimator produces superconsistent estimates of $\hat{\beta}$. This implies that the method of estimating the short-run parameters described above applies in the presence of constraints, as well, albeit with a caveat: when there are constraints placed on α , the VARs must be estimated subject to these constraints.

With these estimates and the estimated VCE of the short-run parameter matrix $\hat{\mathbf{V}}_{\hat{\nu}}$, Drukker (2004) shows that the estimated VCE for $\hat{\Pi}$ is given by

$$(\hat{\beta} \otimes \mathbf{I}_K) \hat{V}_{\hat{\alpha}} (\hat{\beta} \otimes \mathbf{I}_K)'$$

Drukker (2004) also shows that the estimated VCE of $\hat{\mathbf{C}}$ can be obtained from (16) with the extension that $\hat{V}_{\hat{\nu}}$ is the estimated VCE of $\hat{\nu}$ that takes into account any constraints on $\hat{\alpha}$.

Estimation with constraints: β not identified

When the parameters in β are not identified, only the parameters in $\Pi = \alpha\beta$ and C are identified. The estimates of Π and C would not change if more identification restrictions were imposed to achieve exact identification. Thus the VCE matrices for $\hat{\Pi}$ and $\hat{C}$ can be derived as if the model exactly identified β .

Formulas for the information criteria

The AIC, SBIC, and HQIC are calculated according to their standard definitions, which include the constant term from the log likelihood; that is,

$$\begin{aligned}\text{AIC} &= -2\left(\frac{L}{T}\right) + \frac{2n_{\text{parms}}}{T} \\ \text{SBIC} &= -2\left(\frac{L}{T}\right) + \frac{\ln(T)}{T}n_{\text{parms}} \\ \text{HQIC} &= -2\left(\frac{L}{T}\right) + \frac{2\ln\{\ln(T)\}}{T}n_{\text{parms}}\end{aligned}$$

where n_{parms} is the total number of parameters in the model and L is the value of the log likelihood at the optimum.

Formulas for predict

`xb`, `residuals` and `stdp` are standard and are documented in [R] [predict](#). `ce` causes `predict` to compute $\hat{E}_t = \hat{\beta}_f y_t$ for the requested cointegrating equation.

`levels` causes `predict` to compute the predictions for the levels of the data. Let $\hat{y}_t^d$ be the predicted value of Δy_t . Because the computations are performed for a given equation, y_t is a scalar. Using $\hat{y}_t^d$, we can predict the level by $\hat{y}_t = \hat{y}_t^d + y_{t-1}$.

Because the residuals from the VECM for the differences and the residuals from the corresponding VAR in levels are identical, there is no need for an option for predicting the residuals in levels.

References

- Anderson, T. W. 1951. Estimating linear restrictions on regression coefficients for multivariate normal distributions. *Annals of Mathematical Statistics* 22: 327–351.
- Boswijk, H. P. 1995. Identifiability of cointegrated systems. Discussion Paper #95-78, Tinbergen Institute. <http://www1.fee.uva.nl/pp/bin/258fulltext.pdf>.
- Boswijk, H. P., and J. A. Doornik. 2004. Identifying, estimating and testing restricted cointegrating systems: An overview. *Statistica Neerlandica* 58: 440–465.
- Drukker, D. M. 2004. Some further results on estimation and inference in the presence of constraints on alpha in a cointegrating VECM. Working paper, StataCorp.
- Engle, R. F., and C. W. J. Granger. 1987. Co-integration and error correction: Representation, estimation, and testing. *Econometrica* 55: 251–276.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Johansen, S. 1988. Statistical analysis of cointegration vectors. *Journal of Economic Dynamics and Control* 12: 231–254.

- . 1991. Estimation and hypothesis testing of cointegration vectors in Gaussian vector autoregressive models. *Econometrica* 59: 1551–1580.
- . 1995. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford: Oxford University Press.
- Maddala, G. S., and I.-M. Kim. 1998. *Unit Roots, Cointegration, and Structural Change*. Cambridge: Cambridge University Press.
- Park, J. Y., and P. C. B. Phillips. 1988. Statistical inference in regressions with integrated processes: Part I. *Econometric Theory* 4: 468–497.
- . 1989. Statistical inference in regressions with integrated processes: Part II. *Econometric Theory* 5: 95–131.
- Phillips, P. C. B. 1986. Understanding spurious regressions in econometrics. *Journal of Econometrics* 33: 311–340.
- Phillips, P. C. B., and S. N. Durlauf. 1986. Multiple time series regressions with integrated processes. *Review of Economic Studies* 53: 473–495.
- Rothenberg, T. J. 1971. Identification in parametric models. *Econometrica* 39: 577–591.
- Sims, C. A., J. H. Stock, and M. W. Watson. 1990. Inference in linear time series models with some unit roots. *Econometrica* 58: 113–144.
- Stock, J. H. 1987. Asymptotic properties of least squares estimators of cointegrating vectors. *Econometrica* 55: 1035–1056.
- Stock, J. H., and M. W. Watson. 1988. Testing for common trends. *Journal of the American Statistical Association* 83: 1097–1107.
- Watson, M. W. 1994. Vector autoregressions and cointegration. In Vol. IV of *Handbook of Econometrics*, ed. R. F. Engle and D. L. McFadden. Amsterdam: Elsevier.

Also see

- [TS] **vec postestimation** — Postestimation tools for **vec**
- [TS] **tsset** — Declare data to be time-series data
- [TS] **var** — Vector autoregressive models
- [TS] **var svar** — Structural vector autoregressive models
- [U] **20 Estimation and postestimation commands**
- [TS] **vec intro** — Introduction to vector error-correction models

Description

The following postestimation commands are of special interest after `vec`:

command	description
<code>fcast compute</code>	obtain dynamic forecasts
<code>fcast graph</code>	graph dynamic forecasts obtained from <code>fcast compute</code>
<code>irf</code>	create and analyze IRFs and FEVDs
<code>veclmar</code>	LM test for autocorrelation in residuals
<code>vecnorm</code>	test for normally distributed residuals
<code>vecstable</code>	check stability condition of estimates

For information about these commands, see the corresponding entries in this manual.

The following standard postestimation commands are also available:

command	description
<code>estat</code>	AIC, BIC, VCE, and estimation sample summary
<code>estimates</code>	cataloging estimation results
<code>lincom</code>	point estimates, standard errors, testing, and inference for linear combinations of coefficients
<code>lrtest</code>	likelihood-ratio test
<code>margins</code>	marginal means, predictive margins, marginal effects, and average marginal effects
<code>nlcom</code>	point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients
<code>predict</code>	predictions, residuals, influence statistics, and other diagnostic measures
<code>predictnl</code>	point estimates, standard errors, testing, and inference for generalized predictions
<code>test</code>	Wald tests of simple and composite linear hypotheses
<code>testnl</code>	Wald tests of nonlinear hypotheses

See the corresponding entries in the *Base Reference Manual* for details.

Syntax for predict

```
predict [type] newvar [if] [in] [, equation(eqno|eqname) statistic]
```

statistic	description
<code>xb</code>	linear prediction; the default
<code>stdp</code>	standard error of the linear prediction
<code>residuals</code>	residuals
<code>ce</code>	the predicted value of specified cointegrating equation
<code>levels</code>	one-step prediction of the level of the endogenous variable
<code>usece(varlist_{ce})</code>	compute the predictions using previously predicted cointegrating equations

These statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample.

Menu

Statistics > Postestimation > Predictions, residuals, etc.

Options for predict

`equation(eqno|eqname)` specifies to which equation you are referring.

`equation()` is filled in with one *eqno* or *eqname* for `xb`, `residuals`, `stdp`, `ce`, and `levels` options. `equation(#1)` would mean that the calculation is to be made for the first equation, `equation(#2)` would mean the second, and so on. You could also refer to the equation by its name. `equation(D_income)` would refer to the equation named `D_income` and `equation(_ce1)`, to the first cointegrating equation, which is named `_ce1` by `vec`.

If you do not specify `equation()`, the results are as if you specified `equation(#1)`.

`xb`, the default, calculates the linear prediction for the specified equation. The form of the VECM implies that these fitted values are the one-step predictions for the first-differenced variables.

`stdp` calculates the standard error of the linear prediction for the specified equation.

`residuals` calculates the residuals from the specified equation of the VECM.

`ce` calculates the predicted value of the specified cointegrating equation.

`levels` calculates the one-step prediction of the level of the endogenous variable in the requested equation.

`usece(varlistce)` specifies that previously predicted cointegrating equations saved under the names in *varlist_{ce}* be used to compute the predictions. The number of variables in the *varlist_{ce}* must equal the number of cointegrating equations specified in the model.

For more information on using `predict` after multiple-equation estimation commands, see [\[R\] predict](#).

Remarks

Remarks are presented under the following headings:

Model selection and inference
Forecasting

Model selection and inference

See the following sections for information on model selection and inference after `vec`.

[TS] [irf](#) — Create and analyze IRFs, dynamic-multiplier functions, and FEVDs

[TS] [varsoc](#) — Obtain lag-order selection statistics for VARs and VECMs

[TS] [veclmar](#) — Perform LM test for residual autocorrelation after `vec`

[TS] [vecnorm](#) — Test for normally distributed disturbances after `vec`

[TS] [vecrank](#) — Estimate the cointegrating rank of a VECM

[TS] [vecstable](#) — Check the stability condition of VECM estimates

Forecasting

See the following sections for information on obtaining forecasts after `vec`:

[TS] [fcast compute](#) — Compute dynamic forecasts of dependent variables after `var`, `svar`, or `vec`

[TS] [fcast graph](#) — Graph forecasts of variables computed by `fcast compute`

Methods and formulas

All postestimation commands listed above are implemented as ado-files.

Also see

[TS] [vec](#) — Vector error-correction models

[U] [20 Estimation and postestimation commands](#)

[TS] [vec intro](#) — Introduction to vector error-correction models

Syntax

```
veclmar [ , options ]
```

options	description
<code>m</code> lag(<i>#</i>)	use <i>#</i> for the maximum order of autocorrelation; default is <code>m</code> lag(2)
<code>e</code> stimates(<i>estname</i>)	use previously saved results <i>estname</i> ; default is to use active results
<code>s</code> eparator(<i>#</i>)	draw separator line after every <i>#</i> rows

veclmar can be used only after `vec`; see [TS] `vec`.
You must `tsset` your data before using `veclmar`; see [TS] `tsset`.

Menu

Statistics > Multivariate time series > VEC diagnostics and tests > LM test for residual autocorrelation

Description

`veclmar` implements a Lagrange-multiplier (LM) test for autocorrelation in the residuals of vector error-correction models (VECMs).

Options

- `m`lag(*#*) specifies the maximum order of autocorrelation to be tested. The integer specified in `m`lag() must be greater than 0; the default is 2.
- `e`stimates(*estname*) requests that `veclmar` use the previously obtained set of `vec` estimates saved as *estname*. By default, `veclmar` uses the active results. See [R] `estimates` for information on saving and restoring estimation results.
- `s`eparator(*#*) specifies how many rows should appear in the table between separator lines. By default, separator lines do not appear. For example, `separator`(1) would draw a line between each row, `separator`(2) between every other row, and so on.

Remarks

Estimation, inference, and postestimation analysis of VECMs is predicated on the errors' not being autocorrelated. `veclmar` implements the LM test for autocorrelation in the residuals of a VECM discussed in Johansen (1995, 21–22). The test is performed at lags $j = 1, \dots, \text{m}lag()$. For each j , the null hypothesis of the test is that there is no autocorrelation at lag j .

► Example 1

We fit a VECM using the regional income data described in [TS] [vec](#) and then call `vec1mar` to test for autocorrelation.

```
. use http://www.stata-press.com/data/r11/rdinc
. vec ln_ne ln_se
(output omitted)
. vec1mar, mlag(4)
Lagrange-multiplier test
```

lag	chi2	df	Prob > chi2
1	8.9586	4	0.06214
2	4.9809	4	0.28926
3	4.8519	4	0.30284
4	0.3270	4	0.98801

H0: no autocorrelation at lag order

At the 5% level, we cannot reject the null hypothesis that there is no autocorrelation in the residuals for any of the orders tested. Thus this test finds no evidence of model misspecification.

◀

Saved results

`vec1mar` saves the following in `r()`:

Matrices

`r(1m)` χ^2 , df, and p -values

Methods and formulas

`vec1mar` is implemented as an ado-file.

Consider a VECM without any trend:

$$\Delta \mathbf{y}_t = \alpha \beta \mathbf{y}_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \epsilon_t$$

As discussed in [TS] [vec](#), as long as the parameters in the cointegrating vectors, β , are exactly identified or overidentified, the estimates of these parameters are superconsistent. This implies that the $r \times 1$ vector of estimated cointegrating relations

$$\hat{\mathbf{E}}_t = \hat{\beta} \mathbf{y}_t \tag{1}$$

can be used as data with standard estimation and inference methods. When the parameters of the cointegrating equations are not identified, (1) does not provide consistent estimates of $\hat{\mathbf{E}}_t$; in these cases, `vec1mar` exits with an error message.

The VECM above can be rewritten as

$$\Delta \mathbf{y}_t = \boldsymbol{\alpha} \widehat{\mathbf{E}}_t + \sum_{i=1}^{p-1} \boldsymbol{\Gamma}_i \Delta \mathbf{y}_{t-i} + \boldsymbol{\epsilon}_t$$

which is just a VAR with $p - 1$ lags where the endogenous variables have been first-differenced and is augmented with the exogenous variables $\widehat{\mathbf{E}}$. **veclmar** fits this VAR and then calls **varlmar** to compute the LM test for autocorrelation.

The above discussion assumes no trend and implicitly ignores constraints on the parameters in $\boldsymbol{\alpha}$. As discussed in **vec**, the other four trend specifications considered by [Johansen \(1995, sec. 5.7\)](#) complicate the estimation of the free parameters in $\boldsymbol{\beta}$ but do not alter the basic result that the $\widehat{\mathbf{E}}_t$ can be used as data in the subsequent VAR. Similarly, constraints on the parameters in $\boldsymbol{\alpha}$ imply that the subsequent VAR must be estimated with these constraints applied, but $\widehat{\mathbf{E}}_t$ can still be used as data in the VAR.

See [\[TS\] varlmar](#) for more information on the Johansen LM test.

Reference

Johansen, S. 1995. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford: Oxford University Press.

Also see

[\[TS\] vec](#) — Vector error-correction models

[\[TS\] varlmar](#) — Perform LM test for residual autocorrelation after var or svar

[\[TS\] vec intro](#) — Introduction to vector error-correction models

Syntax

vecnorm [, options]

options	description
jbera	report Jarque–Bera statistic; default is to report all three statistics
skewness	report skewness statistic; default is to report all three statistics
kurtosis	report kurtosis statistic; default is to report all three statistics
estimates(estname)	use previously saved results <i>estname</i> ; default is to use active results
dfk	make small-sample adjustment when computing the estimated variance–covariance matrix of the disturbances
separator(#)	draw separator line after every # rows

vecnorm can be used only after `vec`; see [\[TS\]](#) `vec`.

Menu

Statistics > Multivariate time series > VEC diagnostics and tests > Test for normally distributed disturbances

Description

`vecnorm` computes and reports a series of statistics against the null hypothesis that the disturbances in a VECM are normally distributed.

Options

- `jbera` requests that the Jarque–Bera statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.
- `skewness` requests that the skewness statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.
- `kurtosis` requests that the kurtosis statistic and any other explicitly requested statistic be reported. By default, the Jarque–Bera, skewness, and kurtosis statistics are reported.
- `estimates(estname)` requests that `vecnorm` use the previously obtained set of `vec` estimates saved as *estname*. By default, `vecnorm` uses the active results. See [\[R\]](#) `estimates` for information about saving and restoring estimation results.
- `dfk` requests that a small-sample adjustment be made when computing the estimated variance–covariance matrix of the disturbances.
- `separator(#)` specifies how many rows should appear in the table between separator lines. By default, separator lines do not appear. For example, `separator(1)` would draw a line between each row, `separator(2)` between every other row, and so on.

Remarks

vecnorm computes a series of test statistics of the null hypothesis that the disturbances in a VECM are normally distributed. For each equation and all equations jointly, up to three statistics may be computed: a skewness statistic, a kurtosis statistic, and the Jarque–Bera statistic. By default, all three statistics are reported; if you specify only one statistic, the others are not reported. The Jarque–Bera statistic tests skewness and kurtosis jointly. The single-equation results are against the null hypothesis that the disturbance for that particular equation is normally distributed. The results for all the equations are against the null that all K disturbances have a K -dimensional multivariate normal distribution. Failure to reject the null hypothesis indicates lack of model misspecification.

As noted by Johansen (1995, 141), the log likelihood for the VECM is derived assuming the errors are independently and identically distributed (i.i.d.) normal, though many of the asymptotic properties can be derived under the weaker assumption that the errors are merely i.i.d. Many researchers still prefer to test for normality. vecnorm uses the results from vec to produce a series of statistics against the null hypothesis that the K disturbances in the VECM are normally distributed.

► Example 1

This example uses vecnorm to test for normality after estimating the parameters of a VECM using the regional income data.

```
. use http://www.stata-press.com/data/r11/rdinc
. vec ln_ne ln_se
(output omitted)
. vecnorm
Jarque-Bera test
```

Equation	chi2	df	Prob > chi2
D_ln_ne	0.094	2	0.95417
D_ln_se	0.586	2	0.74608
ALL	0.680	4	0.95381

Skewness test

Equation	Skewness	chi2	df	Prob > chi2
D_ln_ne	.05982	0.032	1	0.85890
D_ln_se	.243	0.522	1	0.47016
ALL		0.553	2	0.75835

Kurtosis test

Equation	Kurtosis	chi2	df	Prob > chi2
D_ln_ne	3.1679	0.062	1	0.80302
D_ln_se	2.8294	0.064	1	0.79992
ALL		0.126	2	0.93873

The Jarque–Bera results present test statistics for each equation and for all equations jointly against the null hypothesis of normality. For the individual equations, the null hypothesis is that the disturbance term in that equation has a univariate normal distribution. For all equations jointly, the null hypothesis is that the K disturbances come from a K -dimensional normal distribution. In this example, the single-equation and overall Jarque–Bera statistics do not reject the null of normality.

The single-equation skewness test statistics are of the null hypotheses that the disturbance term in each equation has zero skewness, which is the skewness of a normally distributed variable. The row marked ALL shows the results for a test that the disturbances in all equations jointly have zero skewness. The skewness results shown above do not suggest nonnormality.

The kurtosis of a normally distributed variable is three, and the kurtosis statistics presented in the table test the null hypothesis that the disturbance terms have kurtosis consistent with normality. The results in this example do not reject the null hypothesis.



The statistics computed by `vecnorm` are based on the estimated variance–covariance matrix of the disturbances. `vec` saves the ML estimate of this matrix, which `vecnorm` uses by default. Specifying the `dfk` option instructs `vecnorm` to make a small-sample adjustment to the variance–covariance matrix before computing the test statistics.

Saved results

`vecnorm` saves the following in `r()`:

Macros	
<code>r(dfk)</code>	dfk, if specified
Matrices	
<code>r(jb)</code>	Jarque–Bera χ^2 , df, and p -values
<code>r(skewness)</code>	skewness χ^2 , df, and p -values
<code>r(kurtosis)</code>	kurtosis χ^2 , df, and p -values

Methods and formulas

`vecnorm` is implemented as an ado-file.

As discussed in [Methods and formulas](#) of [TS] `vec`, a cointegrating VECM can be rewritten as a VAR in first differences that includes the predicted cointegrating equations as exogenous variables. `vecnorm` computes the tests discussed in [TS] `varnorm` for the corresponding augmented VAR in first differences. See [Methods and formulas](#) of [TS] `veclmar` for more information on this approach.

When the parameters of the cointegrating equations are not identified, the consistent estimates of the cointegrating equations are not available, and, in these cases, `vecnorm` exits with an error message.

References

Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.

Jarque, C. M., and A. K. Bera. 1987. A test for normality of observations and regression residuals. *International Statistical Review* 2: 163–172.

Johansen, S. 1995. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford: Oxford University Press.

Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Also see

[TS] [vec](#) — Vector error-correction models

[TS] [varnorm](#) — Test for normally distributed disturbances after var or svar

[TS] [vec intro](#) — Introduction to vector error-correction models

Syntax

```
vecrank depvar [if] [in] [, options]
```

<i>options</i>	description
Model	
<u>lags</u> (#)	use # for the maximum lag in underlying VAR model
<u>trend</u> (constant)	include an unrestricted constant in model; the default
<u>trend</u> (rconstant)	include a restricted constant in model
<u>trend</u> (trend)	include a linear trend in the cointegrating equations and a quadratic trend in the undifferenced data
<u>trend</u> (rtrend)	include a restricted trend in model
<u>trend</u> (none)	do not include a trend or a constant
Adv. model	
<u>s</u> indicators(<i>varlist</i> _{si})	include normalized seasonal indicator variables <i>varlist</i> _{si}
noreduce	do not perform checks and corrections for collinearity among lags of dependent variables
Reporting	
<u>not</u> race	do not report the trace statistic
<u>max</u>	report maximum-eigenvalue statistic
<u>i</u> c	report information criteria
level199	report 1% critical values instead of 5% critical values
levela	report both 1% and 5% critical values

You must `tsset` your data before using `vecrank`; see [\[TS\] `tsset`](#).
depvar may contain time-series operators; see [\[U\] 11.4.4 Time-series varlists](#).
`by`, `rolling`, and `statsby` are allowed; see [\[U\] 11.1.10 Prefix commands](#).
`vecrank` does not allow gaps in the data.

Menu

Statistics > Multivariate time series > Cointegrating rank of a VECM

Description

`vecrank` produces statistics used to determine the number of cointegrating equations in a vector error-correction model (VECM).

Options

Model

`lags(#)` specifies the number of lags in the VAR representation of the model. The VECM will include one fewer lag of the first differences. The number of lags must be greater than zero but small enough so that the degrees of freedom used by the model are less than the number of observations.

`trend(trend_spec)` specifies one of five trend specifications to include in the model. See [TS] [vec intro](#) and [TS] [vec](#) for descriptions. The default is `trend(constant)`.

Adv. model

`sindicators(varlistsi)` specifies normalized seasonal indicator variables to be included in the model. The indicator variables specified in this option must be normalized as discussed in [Johansen \(1995, 84\)](#). If the indicators are not properly normalized, the likelihood-ratio–based tests for the number of cointegrating equations do not converge to the asymptotic distributions derived by Johansen. For details, see *Methods and formulas* of [TS] [vec](#). `sindicators()` cannot be specified with `trend(none)` or `trend(rconstant)`

`noreduce` causes `vecrank` to skip the checks and corrections for collinearity among the lags of the dependent variables. By default, `vecrank` checks whether the current lag specification causes some of the regressions performed by `vecrank` to contain perfectly collinear variables and reduces the maximum lag until the perfect collinearity is removed. See *Collinearity* in [TS] [vec](#) for more information.

Reporting

`notrace` requests that the output for the trace statistic not be displayed. The default is to display the trace statistic.

`max` requests that the output for the maximum-eigenvalue statistic be displayed. The default is to not display this output.

`ic` causes the output for the information criteria to be displayed. The default is to not display this output.

`level99` causes the 1% critical values to be displayed instead of the default 5% critical values.

`levela` causes both the 1% and the 5% critical values to be displayed.

Remarks

Remarks are presented under the following headings:

[Introduction](#)

[The trace statistic](#)

[The maximum-eigenvalue statistic](#)

[Minimizing an information criterion](#)

Introduction

Before estimating the parameters of a VECM models, you must choose the number of lags in the underlying VAR, the trend specification, and the number of cointegrating equations. `vecrank` offers several ways of determining the number of cointegrating vectors conditional on a trend specification and lag order.

vecrank implements three types of methods for determining r , the number of cointegrating equations in a VECM. The first is Johansen’s “trace” statistic method. The second is his “maximum eigenvalue” statistic method. The third method chooses r to minimize an information criterion.

All three methods are based on Johansen’s maximum likelihood (ML) estimator of the parameters of a cointegrating VECM. The basic VECM is

$$\Delta \mathbf{y}_t = \alpha \beta' \mathbf{y}_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta \mathbf{y}_{t-i} + \epsilon_t$$

where $\mathbf{y}$ is a $(K \times 1)$ vector of $I(1)$ variables, α and β are $(K \times r)$ parameter matrices with rank $r < K$, $\Gamma_1, \dots, \Gamma_{p-1}$ are $(K \times K)$ matrices of parameters, and ϵ_t is a $(K \times 1)$ vector of normally distributed errors that is serially uncorrelated but has contemporaneous covariance matrix Ω .

Building on the work of [Anderson \(1951\)](#), [Johansen \(1995\)](#) derives an ML estimator for the parameters and two likelihood-ratio (LR) tests for inference on r . These LR tests are known as the trace statistic and the maximum-eigenvalue statistic because the log likelihood can be written as the log of the determinant of a matrix plus a simple function of the eigenvalues of another matrix.

Let $\lambda_1, \dots, \lambda_K$ be the K eigenvalues used in computing the log likelihood at the optimum. Furthermore, assume that these eigenvalues are sorted from the largest λ_1 to the smallest λ_K . If there are $r < K$ cointegrating equations, α and β have rank r and the eigenvalues $\lambda_{r+1}, \dots, \lambda_K$ are zero.

The trace statistic

The null hypothesis of the trace statistic is that there are no more than r cointegrating relations. Restricting the number of cointegrating equations to be r or less implies that the remaining $K - r$ eigenvalues are zero. [Johansen \(1995, chap. 11 and 12\)](#) derives the distribution of the trace statistic

$$-T \sum_{i=r+1}^K \ln(1 - \hat{\lambda}_i)$$

where T is the number of observations and the $\hat{\lambda}_i$ are the estimated eigenvalues. For any given value of r , large values of the trace statistic are evidence against the null hypothesis that there are r or fewer cointegrating relations in the VECM.

One of the problems in determining the number of cointegrating equations is that the process involves more than one statistical test. [Johansen \(1995, chap. 6, 11, and 12\)](#) derives a method based on the trace statistic that has nominal coverage despite evaluating multiple tests. This method can be interpreted as being an estimator $\hat{r}$ of the true number of cointegrating equations r_0 . The method starts testing at $r = 0$ and accepts as $\hat{r}$ the first value of r for which the trace statistic fails to reject the null.

► Example 1

We have quarterly data on the natural logs of aggregate consumption, investment, and GDP in the United States from the first quarter of 1959 through the fourth quarter of 1982. As discussed in [King et al. \(1991\)](#), the balanced-growth hypothesis in economics implies that we would expect to find two cointegrating equations among these three variables. In the output below, we use **vecrank** to determine the number of cointegrating equations using Johansen’s multiple-trace test method.

```
. use http://www.stata-press.com/data/r11/balance2
(macro data for VECM/balance study)
. vecrank y i c, lags(5)
```

Johansen tests for cointegration

Trend: constant

Sample: 1960q2 - 1982q4

Number of obs = 91

Lags = 5

maximum				trace	5%
rank	parms	LL	eigenvalue	statistic	critical
0	39	1231.1041	.	46.1492	29.68
1	44	1245.3882	0.26943	17.5810	15.41
2	47	1252.5055	0.14480	3.3465*	3.76
3	48	1254.1787	0.03611		

The header produces information about the sample, the trend specification, and the number of lags included in the model. The main table contains a separate row for each possible value of r , the number of cointegrating equations. When $r = 3$, all three variables in this model are stationary.

In this example, because the trace statistic at $r = 0$ of 46.1492 exceeds its critical value of 29.68, we reject the null hypothesis of no cointegrating equations. Similarly, because the trace statistic at $r = 1$ of 17.581 exceeds its critical value of 15.41, we reject the null hypothesis that there is one or fewer cointegrating equation. In contrast, because the trace statistic at $r = 2$ of 3.3465 is less than its critical value of 3.76, we cannot reject the null hypothesis that there are two or fewer cointegrating equations. Because Johansen’s method for estimating r is to accept as $\hat{r}$ the first r for which the null hypothesis is not rejected, we accept $r = 2$ as our estimate of the number of cointegrating equations between these three variables. The “*” by the trace statistic at $r = 2$ indicates that this is the value of r selected by Johansen’s multiple-trace test procedure. The eigenvalue shown in the last line of output computes the trace statistic in the preceding line.

➤ Example 2

In the previous example, we used the default 5% critical values. We can estimate r with 1% critical values instead by specifying the `level199` option.

```
. vecrank y i c, lags(5) level199
```

Johansen tests for cointegration

Trend: constant

Sample: 1960q2 - 1982q4

Number of obs = 91

Lags = 5

maximum				trace	1%
rank	parms	LL	eigenvalue	statistic	critical
0	39	1231.1041	.	46.1492	35.65
1	44	1245.3882	0.26943	17.5810*	20.04
2	47	1252.5055	0.14480	3.3465	6.65
3	48	1254.1787	0.03611		

The output indicates that switching from the 5% to the 1% level changes the resulting estimate from $r = 2$ to $r = 1$.

The maximum-eigenvalue statistic

The alternative hypothesis of the trace statistic is that the number of cointegrating equations is strictly larger than the number r assumed under the null hypothesis. Instead, we could assume a given r under the null hypothesis and test this against the alternative that there are $r + 1$ cointegrating equations. [Johansen \(1995, chap. 6, 11, and 12\)](#) derives an LR test of the null of r cointegrating relations against the alternative of $r + 1$ cointegrating relations. Because the part of the log likelihood that changes with r is a simple function of the eigenvalues of a $(K \times K)$ matrix, this test is known as the maximum-eigenvalue statistic. This method is used less often than the trace statistic method because no solution to the multiple-testing problem has yet been found.

► Example 3

In the output below, we reexamine the balanced-growth hypothesis. We use the `levela` option to obtain both the 5% and 1% critical values, and we use the `notrace` option to suppress the table of trace statistics.

```
. vecrank y i c, lags(5) max levela notrace
```

Johansen tests for cointegration

Trend: constant	Number of obs =	91
Sample: 1960q2 - 1982q4	Lags =	5

maximum				max	5% critical	1% critical
rank	parms	LL	eigenvalue	statistic	value	value
0	39	1231.1041		28.5682	20.97	25.52
1	44	1245.3882	0.26943	14.2346	14.07	18.63
2	47	1252.5055	0.14480	3.3465	3.76	6.65
3	48	1254.1787	0.03611			

We can reject $r = 1$ in favor of $r = 2$ at the 5% level but not at the 1% level. As with the trace statistic method, whether we choose to specify one or two cointegrating equations in our VECM will depend on the significance level we use here.



Minimizing an information criterion

Many multiple-testing problems in the time-series literature have been solved by defining an estimator that minimizes an information criterion with known asymptotic properties. Selecting the lag length in an autoregressive model is probably the best-known example. [Gonzalo and Pitarakis \(1998\)](#) and [Aznar and Salvador \(2002\)](#) have shown that this approach can be applied to determining the number of cointegrating equations in a VECM. As in the lag-length selection problem, choosing the number of cointegrating equations that minimizes either the Schwarz Bayesian information criterion (SBIC) or the Hannan and Quinn information criterion (HQIC) provides a consistent estimator of the number of cointegrating equations.

► Example 4

We use these information-criteria methods to estimate the number of cointegrating equations in our balanced-growth data.

```
. vecrank y i c, lags(5) ic notrace
                                Johansen tests for cointegration
Trend: constant
Sample: 1960q2 - 1982q4
                                Number of obs = 91
                                Lags = 5
```

maximum							
rank	parms	LL	eigenvalue	SBIC	HQIC	AIC	
0	39	1231.1041		-25.12401	-25.76596	-26.20009	
1	44	1245.3882	0.26943	-25.19009	-25.91435	-26.40414	
2	47	1252.5055	0.14480	-25.19781*	-25.97144*	-26.49463	
3	48	1254.1787	0.03611	-25.18501	-25.97511	-26.50942	

Both the SBIC and the HQIC estimators suggest that there are two cointegrating equations in the balanced-growth data.



Saved results

vecrank saves the following in `e()`:

- Scalars
- `e(N)` number of observations
 - `e(k_eq)` number of equations
 - `e(k_dv)` number of dependent variables
 - `e(tmin)` minimum time
 - `e(tmax)` maximum time
 - `e(n_lags)` number of lags
 - `e(k_ce95)` number of cointegrating equations chosen by multiple trace tests with `level(95)`
 - `e(k_ce99)` number of cointegrating equations chosen by multiple trace tests with `level(99)`
 - `e(k_cesbic)` number of cointegrating equations chosen by minimizing SBIC
 - `e(k_cehqic)` number of cointegrating equations chosen by minimizing HQIC
- Macros
- `e(cmd)` `vecrank`
 - `e(cmdline)` command as typed
 - `e(tmins)` formatted minimum time
 - `e(tmaxs)` formatted maximum time
 - `e(trend)` trend specified
 - `e(reduced_lags)` list of maximum lags to which the model has been reduced
 - `e(reduce_opt)` `noreduce`, if `noreduce` is specified
 - `e(tsfmt)` format for current time variable
- Matrices
- `e(max)` vector of maximum-eigenvalue statistics
 - `e(trace)` vector of trace statistics
 - `e(lambda)` vector of eigenvalues
 - `e(k_rank)` vector of numbers of unconstrained parameters
 - `e(hqic)` vector of HQIC values
 - `e(sbic)` vector of SBIC values
 - `e(aic)` vector of AIC values
 - `e(ll)` vector of log-likelihood values

Methods and formulas

`vecrank` is implemented as an ado-file.

As shown in [Methods and formulas](#) of [TS] `vec`, given a lag, trend, and seasonal specification when there are $0 \leq r \leq K$ cointegrating equations, the log likelihood with the Johansen identification restrictions can be written as

$$L = -\frac{1}{2}T \left[K \{ \ln(2\pi) + 1 \} + \ln(|S_{00}|) + \sum_{i=1}^r \ln(1 - \hat{\lambda}_i) \right] \quad (1)$$

where the $(K \times K)$ matrix S_{00} and the eigenvalues $\hat{\lambda}_i$ are defined in [Methods and formulas](#) of [TS] `vec`.

The trace statistic compares the null hypothesis that there are r or fewer cointegrating relations with the alternative hypothesis that there are more than r cointegrating equations. Under the alternative hypothesis, the log likelihood is

$$L_A = -\frac{1}{2}T \left[K \{ \ln(2\pi) + 1 \} + \ln(|S_{00}|) + \sum_{i=1}^K \ln(1 - \hat{\lambda}_i) \right] \quad (2)$$

Thus the LR test that compares the unrestricted model in (2) with the restricted model in (1) is given by

$$LR_{\text{trace}} = -T \sum_{i=r+1}^K \ln(1 - \hat{\lambda}_i)$$

As discussed by [Johansen \(1995\)](#), the trace statistic has a nonstandard distribution under the null hypothesis because the null hypothesis places restrictions on the coefficients on $\mathbf{y}_{t-1}$, which is assumed to have $K - r$ random-walk components. `vecrank` reports the [Osterwald-Lenum \(1992\)](#) critical values.

The maximum-eigenvalue statistic compares the null model containing r cointegrating relations with the alternative model that has $r + 1$ cointegrating relations. Thus using these two values for r in (1) and a few lines of algebra implies that the LR test of this hypothesis is

$$LR_{\text{max}} = -T \ln(1 - \hat{\lambda}_{r+1})$$

As for the trace statistic, because this test involves restrictions on the coefficients on a vector of $I(1)$ variables, the test statistic's distribution will be nonstandard. `vecrank` reports the [Osterwald-Lenum \(1992\)](#) critical values.

The formulas for the AIC, SBIC, and HQIC are given in [Methods and formulas](#) of [TS] `vec`.

Søren Johansen (1939–) earned degrees in mathematical statistics at the University of Copenhagen, where he is now based. In addition to making contributions to mathematical statistics, probability theory, and medical statistics, he has worked mostly in econometrics—in particular, on the theory of cointegration.

References

- Anderson, T. W. 1951. Estimating linear restrictions on regression coefficients for multivariate normal distributions. *Annals of Mathematical Statistics* 22: 327–351.
- Aznar, A., and M. Salvador. 2002. Selecting the rank of the cointegration space and the form of the intercept using an information criterion. *Econometric Theory* 18: 926–947.
- Engle, R. F., and C. W. J. Granger. 1987. Co-integration and error correction: Representation, estimation, and testing. *Econometrica* 55: 251–276.
- Gonzalo, J., and J.-Y. Pitarakis. 1998. Specification via model selection in vector error correction models. *Economics Letters* 60: 321–328.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Hubrich, K., H. Lütkepohl, and P. Saikkonen. 2001. A review of systems cointegration tests. *Econometric Reviews* 20: 247–318.
- Johansen, S. 1988. Statistical analysis of cointegration vectors. *Journal of Economic Dynamics and Control* 12: 231–254.
- . 1991. Estimation and hypothesis testing of cointegration vectors in Gaussian vector autoregressive models. *Econometrica* 59: 1551–1580.
- . 1995. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford: Oxford University Press.
- King, R. G., C. I. Plosser, J. H. Stock, and M. W. Watson. 1991. Stochastic trends and economic fluctuations. *American Economic Review* 81: 819–840.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.
- Maddala, G. S., and I.-M. Kim. 1998. *Unit Roots, Cointegration, and Structural Change*. Cambridge: Cambridge University Press.
- Osterwald-Lenum, M. 1992. A note with quantiles of the asymptotic distribution of the maximum likelihood cointegration rank test statistics. *Oxford Bulletin of Economics and Statistics* 54: 461–472.
- Park, J. Y., and P. C. B. Phillips. 1988. Statistical inference in regressions with integrated processes: Part I. *Econometric Theory* 4: 468–497.
- . 1989. Statistical inference in regressions with integrated processes: Part II. *Econometric Theory* 5: 95–131.
- Phillips, P. C. B. 1986. Understanding spurious regressions in econometrics. *Journal of Econometrics* 33: 311–340.
- Phillips, P. C. B., and S. N. Durlauf. 1986. Multiple time series regressions with integrated processes. *Review of Economic Studies* 53: 473–495.
- Sims, C. A., J. H. Stock, and M. W. Watson. 1990. Inference in linear time series models with some unit roots. *Econometrica* 58: 113–144.
- Stock, J. H. 1987. Asymptotic properties of least squares estimators of cointegrating vectors. *Econometrica* 55: 1035–1056.
- Stock, J. H., and M. W. Watson. 1988. Testing for common trends. *Journal of the American Statistical Association* 83: 1097–1107.
- Watson, M. W. 1994. Vector autoregressions and cointegration. In Vol. IV of *Handbook of Econometrics*, ed. R. F. Engle and D. L. McFadden. Amsterdam: Elsevier.

Also see

[TS] [tsset](#) — Declare data to be time-series data

[TS] [vec](#) — Vector error-correction models

[TS] [vec intro](#) — Introduction to vector error-correction models

Syntax

```
vecstable [ , options ]
```

options	description
Main	
<code>estimates(estname)</code>	use previously saved results <i>estname</i> ; default is to use active results
<code>amat(matrix_name)</code>	save the companion matrix as <i>matrix_name</i>
<code>graph</code>	graph eigenvalues of the companion matrix
<code>dlabel</code>	label eigenvalues with the distance from the unit circle
<code>modlabel</code>	label eigenvalues with the modulus
<code>marker_options</code>	change look of markers (color, size, etc.)
<code>rlopts(cline_options)</code>	affect rendition of reference unit circle
<code>nogrid</code>	suppress polar grid circles
<code>pgrid([...])</code>	specify radii and appearance of polar grid circles; see <i>Options</i> for details
Add plots	
<code>addplot(plot)</code>	add other plots to the generated graph
Y axis, X axis, Titles, Legend, Overall	
<code>twoway_options</code>	any options other than <code>by()</code> documented in [G] twoway_options
vecstable can be used only after <code>vec</code> ; see [TS] vec .	

Menu

Statistics > Multivariate time series > VEC diagnostics and tests > Check stability condition of VEC estimates

Description

`vecstable` checks the eigenvalue stability condition in a vector error-correction model (VECM) fit using `vec`.

Options

Main
<code>estimates(estname)</code> requests that <code>vecstable</code> use the previously obtained set of <code>vec</code> estimates saved as <i>estname</i> . By default, <code>vecstable</code> uses the active results. See [R] estimates for information about saving and restoring estimation results.
<code>amat(matrix_name)</code> specifies a valid Stata matrix name by which the companion matrix can be saved. The companion matrix is referred to as the A matrix in Lütkepohl (2005) and [TS] varstable . The default is not to save the companion matrix.

graph causes `vecstable` to draw a graph of the eigenvalues of the companion matrix.

`dlabel` labels the eigenvalues with their distances from the unit circle. `dlabel` cannot be specified with `modlabel`.

`modlabel` labels the eigenvalues with their moduli. `modlabel` cannot be specified with `dlabel`.

`marker_options` specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G] [marker_options](#).

`rlopts(cline_options)` affects the rendition of the reference unit circle; see [G] [cline_options](#).

`nogrid` suppresses the polar grid circles.

`pgrid([numlist][, line_options]) [pgrid([numlist][, line_options]) ...`

`pgrid([numlist][, line_options])` determines the radii and appearance of the polar grid circles. By default, the graph includes nine polar grid circles with radii .1, .2, ..., .9 that have the `grid` linestyle. The `numlist` specifies the radii for the polar grid circles. The `line_options` determine the appearance of the polar grid circles; see [G] [line_options](#). Because the `pgrid()` option can be repeated, circles with different radii can have distinct appearances.

Add plots

`addplot(plot)` adds specified plots to the generated graph; see [G] [addplot_option](#).

Y axis, X axis, Titles, Legend, Overall

`twoway_options` are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

Remarks

Inference after `vec` requires that the cointegrating equations be stationary and that the number of cointegrating equations be correctly specified. Although the methods implemented in `vecrank` identify the number of stationary cointegrating equations, they assume that the individual variables are $I(1)$. `vecstable` provides indicators of whether the number of cointegrating equations is misspecified or whether the cointegrating equations, which are assumed to be stationary, are not stationary.

`vecstable` is analogous to `varstable`. `vecstable` uses the coefficient estimates from the previously fitted VECM to back out estimates of the coefficients of the corresponding VAR and then compute the eigenvalues of the companion matrix. See [TS] [varstable](#) for details about how the companion matrix is formed and about how to interpret the resulting eigenvalues for covariance-stationary VAR models.

If a VECM has K endogenous variables and r cointegrating vectors, there will be $K - r$ unit moduli in the companion matrix. If any of the remaining moduli computed by `vecrank` are too close to one, either the cointegrating equations are not stationary or there is another common trend and the `rank()` specified in the `vec` command is too high. Unfortunately, there is no general distribution theory that allows you to determine whether an estimated root is too close to one for all the cases that commonly arise in practice.

► Example 1

In [example 1](#) of [\[TS\] vec](#), we estimated the parameters of a bivariate VECM of the natural logs of the average disposable incomes in two of the economic regions created by the U.S. Bureau of Economic Analysis. In that example, we concluded that the predicted cointegrating equation was probably not stationary. Here we continue that example by refitting that model and using `vecstable` to analyze the eigenvalues of the companion matrix of the corresponding VAR.

```
. use http://www.stata-press.com/data/r11/rdinc
. vec ln_ne ln_se
  (output omitted)
. vecstable
  Eigenvalue stability condition
```

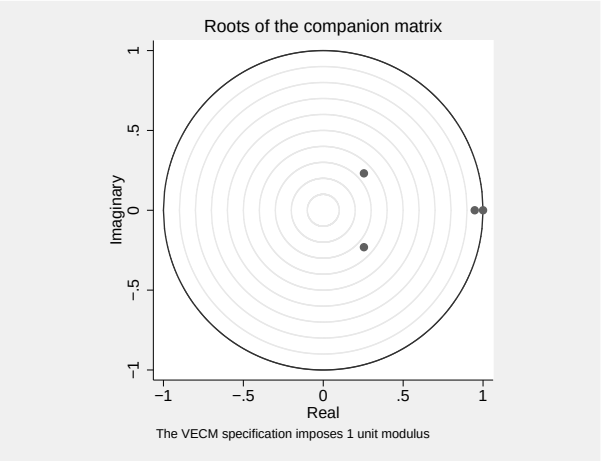
Eigenvalue	Modulus
1	1
.9477854	.947785
.2545357 + .2312756i	.343914
.2545357 - .2312756i	.343914

The VECM specification imposes a unit modulus.

The output contains a table showing the eigenvalues of the companion matrix and their associated moduli. The table shows that one of the roots is 1. The table footer reminds us that the specified VECM imposes one unit modulus on the companion matrix.

The output indicates that there is a real root at about 0.95. Although there is no distribution theory to measure how close this root is to one, per other discussions in the literature (e.g., [Johansen \[1995, 137–138\]](#)), we conclude that the root of 0.95 supports our earlier analysis, in which we concluded that the predicted cointegrating equation is probably not stationary.

If we had included the `graph` option with `vecstable`, the following graph would have been displayed:



The graph plots the eigenvalues of the companion matrix with the real component on the x axis and the imaginary component on the y axis. Although the information is the same as in the table, the graph shows visually how close the root with modulus 0.95 is to the unit circle.



Saved results

`vecstable` saves the following in `r()`:

Scalars

`r(unitmod)` number of unit moduli imposed on the companion matrix

Matrices

`r(Re)` real part of the eigenvalues of **A**

`r(Im)` imaginary part of the eigenvalues of **A**

`r(Modulus)` moduli of the eigenvalues of **A**

where **A** is the companion matrix of the VAR that corresponds to the VECM.

Methods and formulas

`vecstable` is implemented as an ado-file.

`vecstable` uses the formulas given in [Methods and formulas](#) of [\[TS\] irf create](#) to obtain estimates of the parameters in the corresponding VAR from the `vec` estimates. With these estimates, the calculations are identical to those discussed in [\[TS\] varstable](#). In particular, the derivation of the companion matrix, **A**, from the VAR point estimates is given in [\[TS\] varstable](#).

References

- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Johansen, S. 1995. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford: Oxford University Press.
- Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. New York: Springer.

Also see

[\[TS\] vec](#) — Vector error-correction models

[\[TS\] varstable](#) — Check the stability condition of VAR or SVAR estimates

[\[TS\] vec intro](#) — Introduction to vector error-correction models

Syntax

```
wntestb varname [if] [in] [, options]
```

<i>options</i>	description
Main	
<code>table</code>	display a table instead of graphical output
<code>level(#)</code>	set confidence level; default is <code>level(95)</code>
Plot	
<code>marker_options</code>	change look of markers (color, size, etc.)
<code>marker_label_options</code>	add marker labels; change look or position
<code>cline_options</code>	add connecting lines; change look
Add plots	
<code>addplot(plot)</code>	add other plots to the generated graph
Y axis, X axis, Titles, Legend, Overall	
<code>twoway_options</code>	any options other than <code>by()</code> documented in [G] twoway_options

You must `tsset` your data before using `wntestb`; see [TS] [tsset](#). In addition, the time series must be dense (nonmissing with no gaps in the time variable) in the specified sample.
varname may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Tests > Bartlett’s periodogram-based white-noise test

Description

`wntestb` performs Bartlett’s periodogram-based test for white noise. The result is presented graphically by default but optionally may be presented as text in a table.

Options

Main

`table` displays the test results as a table instead of as the default graph.
`level(#)` specifies the confidence level, as a percentage, for the confidence bands included on the graph. The default is `level(95)` or as set by `set level`; see [U] [20.7 Specifying the width of confidence intervals](#).

Plot

marker_options specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G] [marker_options](#).

marker_label_options specify if and how the markers are to be labeled; see [G] [marker_label_options](#).

cline_options specify if the points are to be connected with lines and the rendition of those lines; see [G] [cline_options](#).

Add plots

`addplot(plot)` adds specified plots to the generated graph; see [G] [addplot_option](#).

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

Remarks

Bartlett's test is a test of the null hypothesis that the data come from a white-noise process of uncorrelated random variables having a constant mean and a constant variance.

For a discussion of this test, see [Bartlett \(1955, 92–94\)](#), [Newton \(1988, 172\)](#), or [Newton \(1996\)](#).

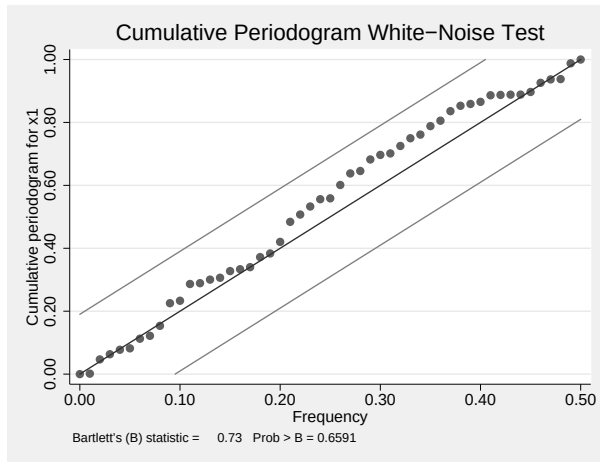
► Example 1

In this example, we generate two time series and show the graphical and statistical tests that can be obtained from this command. The first time series is a white-noise process, and the second is a white-noise process with an embedded deterministic cosine curve.

```
. drop _all
. set seed 12393
. set obs 100
obs was 0, now 100
. generate x1 = rnormal()
. generate x2 = rnormal() + cos(2*_pi*( _n-1)/10)
. generate time = _n
. tsset time
      time variable:  time, 1 to 100
                delta:  1 unit
```

We can then submit the white-noise data to the `wntestb` command by typing

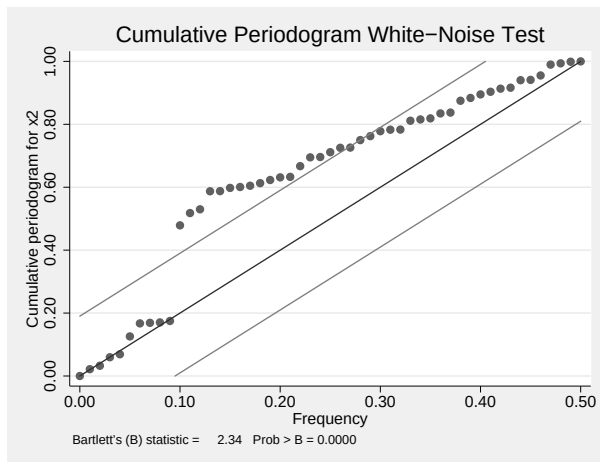

```
. wntestb x1
```



We can see in the graph that the values never appear outside the confidence bands. The test statistic has a p -value of 0.91, so we conclude that the process is not different from white noise. If we had wanted only the statistic without the plot, we could have used the `table` option.

Turning our attention to the other series (`x2`), we type

```
. wntestb x2
```



Here the process does appear outside of the bands. In fact, it steps out of the bands at a frequency of 0.1 (exactly as we synthesized this process). We also have confirmation from the test statistic, at a p -value of 0.001, that the process is significantly different from white noise.

Saved results

wntestb saves the following in `r()`:

Scalars			
<code>r(stat)</code>	Bartlett’s statistic	<code>r(p)</code>	probability value

Methods and formulas

wntestb is implemented as an ado-file.

If $x(1), \dots, x(T)$ is a realization from a white-noise process with variance σ^2 , the spectral distribution would be given by $F(\omega) = \sigma^2\omega$ for $\omega \in [0, 1]$, and we would expect the cumulative periodogram (see [TS] [cumsp](#)) of the data to be close to the points $S_k = k/q$ for $q = \lfloor n/2 \rfloor + 1, k = 1, \dots, q$. $\lfloor n/2 \rfloor$ is the greatest integer less than or equal to $n/2$.

Except for $\omega = 0$ and $\omega = .5$, the random variables $2\hat{f}(\omega_k)/\sigma^2$ are asymptotically independently and identically distributed as χ^2_2 . Because χ^2_2 is the same as twice a random variable distributed exponentially with mean 1, the cumulative periodogram has approximately the same distribution as the ordered values from a uniform (on the unit interval) distribution. [Feller \(1948\)](#) shows that this results in

$$\lim_{q \rightarrow \infty} \Pr \left(\max_{1 \leq k \leq q} \sqrt{q} \left| U_k - \frac{k}{q} \right| \leq a \right) = \sum_{j=-\infty}^{\infty} (-1)^j e^{-2a^2 j^2} = G(a)$$

where U_k is the ordered uniform quantile. The Bartlett statistic is computed as

$$B = \max_{1 \leq k \leq q} \sqrt{\frac{n}{2}} \left| \hat{F}_k - \frac{k}{q} \right|$$

where $\hat{F}_k$ is the cumulative periodogram defined in terms of the sample spectral density $\hat{f}$ (see [TS] [pergram](#)) as

$$\hat{F}_k = \frac{\sum_{j=1}^k \hat{f}(\omega_j)}{\sum_{j=1}^q \hat{f}(\omega_j)}$$

The associated p -value for the Bartlett statistic and the confidence bands on the graph are computed as $1 - G(B)$ using Feller’s result.

Maurice Stevenson Bartlett (1910–2002) was a British statistician. Apart from a short period in industry, he spent his career teaching and researching at the universities of Cambridge, Manchester, London (University College), and Oxford. His many contributions include work on the statistical analysis of multivariate data (especially factor analysis) and time series and on stochastic models of population growth, epidemics, and spatial processes.

Acknowledgment

wntestb is based on the `wntestf` command by H. Joseph Newton ([1996](#)), Department of Statistics, Texas A&M University.

References

- Bartlett, M. S. 1955. *An Introduction to Stochastic Processes with Special Reference to Methods and Applications*. Cambridge: Cambridge University Press.
- Feller, W. 1948. On the Kolmogorov–Smirnov limit theorems for empirical distributions. *Annals of Mathematical Statistics* 19: 177–189.
- Gani, J. 2002. Professor M. S. Bartlett FRS, 1910–2002. *Statistician* 51: 399–402.
- Newton, H. J. 1988. *TIMESLAB: A Time Series Analysis Laboratory*. Belmont, CA: Wadsworth.
- . 1996. [sts12: A periodogram-based test for white noise](#). *Stata Technical Bulletin* 34: 36–39. Reprinted in *Stata Technical Bulletin Reprints*, vol. 6, pp. 203–207. College Station, TX: Stata Press.
- Olkin, I. 1989. A conversation with Maurice Bartlett. *Statistical Science* 4: 151–163.

Also see

- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [corrgram](#) — Tabulate and graph autocorrelations
- [TS] [cumsp](#) — Cumulative spectral distribution
- [TS] [pergram](#) — Periodogram
- [TS] [wntestq](#) — Portmanteau (Q) test for white noise

Title

wntestq — Portmanteau (Q) test for white noise

Syntax

```
wntestq varname [if] [in] [, lags(#)]
```

You must `tsset` your data before using `wntestq`; see [TS] [tsset](#). Also the time series must be dense (nonmissing with no gaps in the time variable) in the specified sample.

varname may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Tests > Portmanteau white-noise test

Description

`wntestq` performs the portmanteau (or Q) test for white noise.

Option

`lags(#)` specifies the number of autocorrelations to calculate. The default is to use $\min(\lfloor n/2 \rfloor - 2, 40)$, where $\lfloor n/2 \rfloor$ is the greatest integer less than or equal to $n/2$.

Remarks

[Box and Pierce \(1970\)](#) developed a portmanteau test of white noise that was refined by [Ljung and Box \(1978\)](#). See also [Diggle \(1990, sec. 2.5\)](#).

► Example 1

In the [example](#) shown in [TS] [wntestb](#), we generated two time series. One (`x1`) was a white-noise process, and the other (`x2`) was a white-noise process with an embedded cosine curve. Here we compare the output of the two tests.

```
. drop _all
. set seed 12393
. set obs 100
obs was 0, now 100
. generate x1 = rnormal()
. generate x2 = rnormal() + cos(2*_pi*(_n-1)/10)
. generate time = _n
. tsset time
    time variable:  time, 1 to 100
                delta:  1 unit
```

```
. wntestb x1, table
Cumulative periodogram white-noise test
-----
Bartlett's (B) statistic =    0.5665
Prob > B                =    0.9053

. wntestq x1
Portmanteau test for white noise
-----
Portmanteau (Q) statistic =    27.0818
Prob > chi2(40)          =    0.9407

. wntestb x2, table
Cumulative periodogram white-noise test
-----
Bartlett's (B) statistic =    1.9464
Prob > B                =    0.0010

. wntestq x2
Portmanteau test for white noise
-----
Portmanteau (Q) statistic =    96.2401
Prob > chi2(40)          =    0.0000
```

This example shows that both tests agree. For the first process, the Bartlett and portmanteau tests result in nonsignificant test statistics: a p -value of 0.9053 for `wntestb` and one of 0.9407 for `wntestq`. For the second process, each test has a significant result to 0.0010.



Saved results

`wntestq` saves the following in `r()`:

Scalars		
<code>r(stat)</code>	Q statistic	<code>r(p)</code> probability value
<code>r(df)</code>	degrees of freedom	

Methods and formulas

`wntestq` is implemented as an ado-file.

The portmanteau test relies on the fact that if $x(1), \dots, x(n)$ is a realization from a white-noise process. Then

$$Q = n(n + 2) \sum_{j=1}^m \frac{1}{n - j} \hat{\rho}^2(j) \longrightarrow \chi_m^2$$

where m is the number of autocorrelations calculated (equal to the number of lags specified) and $\longrightarrow$ indicates convergence in distribution to a χ^2 distribution with m degrees of freedom. $\hat{\rho}_j$ is the estimated autocorrelation for lag j ; see [TS] [corrgram](#) for details.

References

- Box, G. E. P., and D. A. Pierce. 1970. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association* 65: 1509–1526.
- Diggle, P. J. 1990. *Time Series: A Biostatistical Introduction*. Oxford: Oxford University Press.
- Ljung, G. M., and G. E. P. Box. 1978. On a measure of lack of fit in time series models. *Biometrika* 65: 297–303.
- Sperling, R., and C. F. Baum. 2001. [sts19: Multivariate portmanteau \(Q\) test for white noise](#). *Stata Technical Bulletin* 60: 39–41. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 373–375. College Station, TX: Stata Press.

Also see

- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [corrgram](#) — Tabulate and graph autocorrelations
- [TS] [cumsp](#) — Cumulative spectral distribution
- [TS] [pergram](#) — Periodogram
- [TS] [wntestb](#) — Bartlett’s periodogram-based test for white noise

Syntax

```
xcorr varname1 varname2 [if] [in] [, options]
```

options	description
Main	
generate(<i>newvar</i>)	create <i>newvar</i> containing cross-correlation values
table	display a table instead of graphical output
noplot	do not include the character-based plot in tabular output
lags(#)	include # lags and leads in graph
Plot	
base(#)	value to drop to; default is 0
marker_options	change look of markers (color, size, etc.)
marker_label_options	add marker labels; change look or position
line_options	change look of dropped lines
Add plots	
addplot(<i>plot</i>)	add other plots to the generated graph
Y axis, X axis, Titles, Legend, Overall	
twoway_options	any options other than by() documented in [G] twoway_options

You must `tsset` your data before using `xcorr`; see [TS] [tsset](#).
varname₁ and *varname₂* may contain time-series operators; see [U] [11.4.4 Time-series varlists](#).

Menu

Statistics > Time series > Graphs > Cross-correlogram for bivariate time series

Description

`xcorr` plots the sample cross-correlation function.

Options

Main

- `generate(newvar)` specifies a new variable to contain the cross-correlation values.
- `table` requests that the results be presented as a table rather than the default graph.
- `noplot` requests that the table not include the character-based plot of the cross-correlations.
- `lags(#)` indicates the number of lags and leads to include in the graph. The default is to use $\min(\lfloor n/2 \rfloor - 2, 20)$.

Plot

`base(#)` specifies the value from which the lines should extend. The default is `base(0)`.

`marker_options`, `marker_label_options`, and `line_options` affect the rendition of the plotted cross-correlations.

`marker_options` specify the look of markers. This look includes the marker symbol, the marker size, and its color and outline; see [G] [marker_options](#).

`marker_label_options` specify if and how the markers are to be labeled; see [G] [marker_label_options](#).

`line_options` specify the look of the dropped lines, including pattern, width, and color; see [G] [line_options](#).

Add plots

`addplot(plot)` provides a way to add other plots to the generated graph. See [G] [addplot_option](#).

Y axis, X axis, Titles, Legend, Overall

`twoway_options` are any of the options documented in [G] [twoway_options](#), excluding `by()`. These include options for titling the graph (see [G] [title_options](#)) and for saving the graph to disk (see [G] [saving_option](#)).

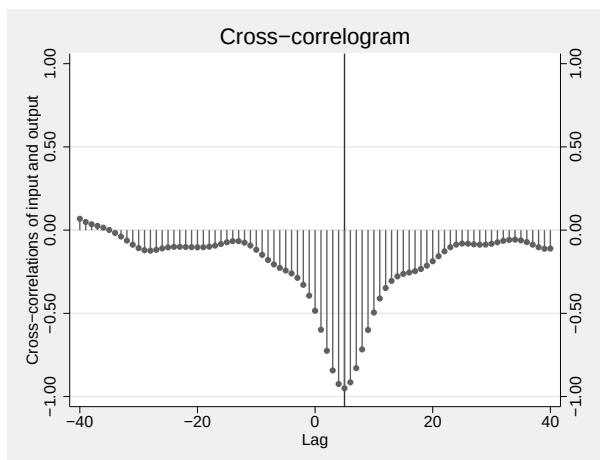
Remarks

► Example 1

We have a bivariate time series (Box, Jenkins, and Reinsel 1994, Series J) on the input and output of a gas furnace, where 296 paired observations on the input (gas rate) and output (% CO₂) were recorded every 9 seconds. The cross-correlation function is given by

```
. use http://www.stata-press.com/data/r11/furnace
(TIMESLAB: Gas furnace)

. xcorr input output, xline(5) lags(40)
```



We included a vertical line at lag 5, because there is a well-defined peak at this value. This peak indicates that the output lags the input by five periods. Further, the fact that the correlations are negative indicates that as input (coded gas rate) is increased, output (% CO₂) decreases.

We may obtain the table of autocorrelations and the character-based plot of the cross-correlations (analogous to the univariate time-series command `corrgram`) by specifying the `table` option.

```
. xcorr input output, table
```

LAG	CORR	-1	0	1
		[Cross-correlation]		
-20	-0.1033			
-19	-0.1027			
-18	-0.0998			
-17	-0.0932			
-16	-0.0832			
-15	-0.0727			
-14	-0.0660			
-13	-0.0662			
-12	-0.0751			
-11	-0.0927			
-10	-0.1180			
-9	-0.1484			
-8	-0.1793			
-7	-0.2059			
-6	-0.2266			
-5	-0.2429			
-4	-0.2604			
-3	-0.2865			
-2	-0.3287			
-1	-0.3936			
0	-0.4845			
1	-0.5985			
2	-0.7251			
3	-0.8429			
4	-0.9246			
5	-0.9503			
6	-0.9146			
7	-0.8294			
8	-0.7166			
9	-0.5998			
10	-0.4952			
11	-0.4107			
12	-0.3479			
13	-0.3049			
14	-0.2779			
15	-0.2632			
16	-0.2548			
17	-0.2463			
18	-0.2332			
19	-0.2135			
20	-0.1869			

Once again, the well-defined peak is apparent in the plot.

Methods and formulas

`xcorr` is implemented as an ado-file.

The cross-covariance function of lag k for time series x_1 and x_2 is given by

$$\text{Cov}\{x_1(t), x_2(t+k)\} = R_{12}(k)$$

This function is not symmetric about lag zero; that is,

$$R_{12}(k) \neq R_{12}(-k)$$

We define the cross-correlation function as

$$\rho_{ij}(k) = \text{Corr}\{x_i(t), x_j(t+k)\} = \frac{R_{ij}(k)}{\sqrt{R_{ii}(0)R_{jj}(0)}}$$

where ρ_{11} and ρ_{22} are the autocorrelation functions for x_1 and x_2 , respectively. The sequence $\rho_{12}(k)$ is the cross-correlation function and is drawn for lags $k \in (-Q, -Q+1, \dots, -1, 0, 1, \dots, Q-1, Q)$.

If $\rho_{12}(k) = 0$ for all lags, x_1 and x_2 are not cross-correlated.

References

- Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Newton, H. J. 1988. *TIMESLAB: A Time Series Analysis Laboratory*. Belmont, CA: Wadsworth.

Also see

- [TS] [tsset](#) — Declare data to be time-series data
- [TS] [corrgram](#) — Tabulate and graph autocorrelations
- [TS] [pergram](#) — Periodogram

Glossary

ARCH model. An autoregressive conditional heteroskedasticity (ARCH) model is a regression model in which the conditional variance is modeled as an autoregressive (AR) process. The ARCH(m) model is

$$y_t = \mathbf{x}_t\beta + \epsilon_t$$

$$E(\epsilon_t^2 | \epsilon_{t-1}^2, \epsilon_{t-2}^2, \dots) = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \dots + \alpha_m \epsilon_{t-m}^2$$

where ϵ_t is a white-noise error term. The equation for y_t represents the conditional mean of the process, and the equation for $E(\epsilon_t^2 | \epsilon_{t-1}^2, \epsilon_{t-2}^2, \dots)$ specifies the conditional variance as an autoregressive function of its past realizations. Although the conditional variance changes over time, the unconditional variance is time invariant because y_t is a stationary process. Modeling the conditional variance as an AR process raises the implied unconditional variance, making this model particularly appealing to researchers modeling fat-tailed data, such as financial data.

ARIMA model. An autoregressive integrated moving-average (ARIMA) model is a time-series model suitable for use with integrated processes. In an ARIMA(p, d, q) model, the data is differenced d times to obtain a stationary series, and then an ARMA(p, q) model is fit to this differenced data. ARIMA models that include exogenous explanatory variables are known as ARMAX models.

ARMA model. An autoregressive moving-average (ARMA) model is a time-series model in which the current period's realization is the sum of an autoregressive (AR) process and a moving-average (MA) process. An ARMA(p, q) model includes p AR terms and q MA terms. ARMA models with just a few lags are often able to fit data, as well as pure AR or MA models with many more lags.

ARMAX model. An ARMAX model is a time-series model in which the current period's realization is an ARMA process plus a linear function of a set of a exogenous variables. Equivalently, an ARMAX model is a linear regression model in which the error term is specified to follow an ARMA process.

autocorrelation function. The autocorrelation function (ACF) expresses the correlation between periods t and $t - k$ of a time series as function of the time t and the lag k . For a stationary time series, the ACF does not depend on t and is symmetric about $k = 0$, meaning that the correlation between periods t and $t - k$ is equal to the correlation between periods t and $t + k$.

autoregressive process. An autoregressive process is a time-series model in which the current value of a variable is a linear function of its own past values and a white-noise error term. A first-order autoregressive process, denoted as an AR(1) process, is $y_t = \rho y_{t-1} + \epsilon_t$. An AR(p) model contains p lagged values of the dependent variable.

Cholesky ordering. Cholesky ordering is a method used to orthogonalize the error term in a VAR or VECM to impose a recursive structure on the dynamic model, so that the resulting impulse-response functions can be given a causal interpretation. The method is so named because it uses the Cholesky decomposition of the error covariance matrix.

Cochrane–Orcutt estimator. This estimation is a linear regression estimator that can be used when the error term exhibits first-order autocorrelation. An initial estimate of the autocorrelation parameter ρ is obtained from OLS residuals, and then OLS is performed on the transformed data $\tilde{y}_t = y_t - \rho y_{t-1}$ and $\tilde{\mathbf{x}}_t = \mathbf{x}_t - \rho \mathbf{x}_{t-1}$.

cointegrating vector. A cointegrating vector specifies a stationary linear combination of nonstationary variables. Specifically, if each of the variables $x_1, x_2, \dots, x_k$ is integrated of order one and there exists a set of parameters $\beta_1, \beta_2, \dots, \beta_k$ such that $z_t = \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k$ is a stationary process, the variables $x_1, x_2, \dots, x_k$ are said to be cointegrated, and the vector β is known as a cointegrating vector.

conditional variance. Although the conditional variance is simply the variance of a conditional distribution, in time-series analysis the conditional variance is often modeled as an autoregressive process, giving rise to ARCH models.

correlogram. A correlogram is a table or graph showing the sample autocorrelations or partial autocorrelations of a time series.

covariance stationarity. A process is covariance stationary if the mean of the process is finite and independent of t , the unconditional variance of the process is finite and independent of t , and the covariance between periods t and $t - s$ is finite and depends on $t - s$ but not on t or s themselves. Covariance-stationary processes are also known as weakly stationary processes.

cross-correlation function. The cross-correlation function expresses the correlation between one series at time t and another series at time $t - k$ as a function of the time t and lag k . If both series are stationary, the function does not depend on t . The function is not symmetric about $k = 0$: $\rho_{12}(k) \neq \rho_{12}(-k)$.

difference operator. The difference operator Δ denotes the change in the value of a variable from period $t - 1$ to period t . Formally, $\Delta y_t = y_t - y_{t-1}$, and $\Delta^2 y_t = \Delta(y_t - y_{t-1}) = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2}$.

dynamic forecast. A dynamic forecast is one in which the current period's forecast is calculated using forecasted values for prior periods.

dynamic-multiplier function. A dynamic-multiplier function measures the effect of a shock to an exogenous variable on an endogenous variable. The k th dynamic-multiplier function of variable i on variable j measures the effect on variable j in period $t + k$ in response to a one-unit shock to variable i in period t , holding everything else constant.

endogenous variable. An endogenous variable is a regressor that is correlated with the unobservable error term. Equivalently, an endogenous variable is one whose values are determined by the equilibrium or outcome of a structural model.

exogenous variable. An exogenous variable is one that is correlated with none of the unobservable error terms in the model. Equivalently, an exogenous variable is one whose values change independently of the other variables in a structural model.

exponential smoothing. Exponential smoothing is a method of smoothing a time series in which the smoothed value at period t is equal to a fraction α of the series value at time t plus a fraction $1 - \alpha$ of the previous period's smoothed value. The fraction α is known as the smoothing parameter.

forecast-error variance decomposition. Forecast-error variance decompositions measure the fraction of the error in forecasting variable i after h periods that is attributable to the orthogonalized shocks to variable j .

forward operator. The forward operator F denotes the value of a variable at time $t + 1$. Formally, $Fy_t = y_{t+1}$, and $F^2 y_t = Fy_{t+1} = y_{t+2}$.

frequency-domain analysis. Frequency-domain analysis is analysis of time-series data by considering its frequency properties. The spectral density and distribution functions are key components of frequency-domain analysis, so it is often called spectral analysis. In Stata, the `cumsp` and `pergram` commands are used to analyze the sample spectral distribution and density functions, respectively.

GARCH model. A generalized autoregressive conditional heteroskedasticity (GARCH) model is a regression model in which the conditional variance is modeled as an ARMA process. The GARCH(m, k) model is

$$y_t = \mathbf{x}_t\boldsymbol{\beta} + \epsilon_t$$

$$\sigma_t^2 = \gamma_0 + \gamma_1\epsilon_{t-1}^2 + \cdots + \gamma_m\epsilon_{t-m}^2 + \delta_1\sigma_{t-1}^2 + \cdots + \delta_k\sigma_{t-k}^2$$

where the equation for y_t represents the conditional mean of the process and σ_t represents the conditional variance. See [TS] [arch](#) or [Hamilton \(1994, chap. 21\)](#) for details on how the conditional variance equation can be viewed as an ARMA process. GARCH models are often used because the ARMA specification often allows the conditional variance to be modeled with fewer parameters than are required by a pure ARCH model. Many extensions to the basic GARCH model exist; see [TS] [arch](#) for those that are implemented in Stata. See also [ARCH model](#).

generalized least-squares estimator. A generalized least-squares (GLS) estimator is used to estimate the parameters of a regression function when the error term is heteroskedastic or autocorrelated. In the linear case, GLS is sometimes described as “OLS on transformed data” because the GLS estimator can be implemented by applying an appropriate transformation to the dataset and then using OLS.

Granger causality. The variable x is said to Granger-cause variable y if, given the past values of y , past values of x are useful for predicting y .

Holt–Winters smoothing. A set of methods for smoothing time-series data that assume that the value of a time series at time t can be approximated as the sum of a mean term that drifts over time, as well as a time trend whose strength also drifts over time. Variations of the basic method allow for seasonal patterns in data, as well.

impulse–response function. An impulse–response function (IRF) measures the effect of a shock to an endogenous variable on itself or another endogenous variable. The k th impulse–response function of variable i on variable j measures the effect on variable j in period $t + k$ in response to a one-unit shock to variable i in period t , holding everything else constant.

independent and identically distributed. A series of observations is independently and identically distributed (i.i.d.) if each observation is an independent realization from the same underlying distribution. In some contexts, the definition is relaxed to mean only that the observations are independent and have identical means and variances; see [Davidson and MacKinnon \(1993, 42\)](#).

integrated process. A nonstationary process is integrated of order d , written $I(d)$, if the process must be differenced d times to produce a stationary series. An $I(1)$ process y_t is one in which Δy_t is stationary.

Kalman filter. The Kalman filter is a recursive procedure for predicting the state vector in a state-space model.

lag operator. The lag operator L denotes the value of a variable at time $t - 1$. Formally, $Ly_t = y_{t-1}$, and $L^2y_t = Ly_{t-1} = y_{t-2}$.

moving-average process. A moving-average process is a time-series process in which the current value of a variable is modeled as a weighted average of current and past realizations of a white-noise process and, optionally, a time-invariant constant. By convention, the weight on the current realization of the white-noise process is equal to one, and the weights on the past realizations are known as the moving-average (MA) coefficients. A first-order moving-average process, denoted as an MA(1) process, is $y_t = \theta\epsilon_{t-1} + \epsilon_t$.

Newey–West covariance matrix. The Newey–West covariance matrix is a member of the class of heteroskedasticity- and autocorrelation-consistent (HAC) covariance matrix estimators used with

time-series data that produces covariance estimates that are robust to both arbitrary heteroskedasticity and autocorrelation up to a prespecified lag.

orthogonalized impulse–response function. An orthogonalized impulse–response function (OIRF) measures the effect of an orthogonalized shock to an endogenous variable on itself or another endogenous variable. An orthogonalized shock is one that affects one variable at time t but no other variables. See [TS] [irf create](#) for a discussion of the difference between IRFs and OIRFs.

partial autocorrelation function. The partial autocorrelation function (PACF) expresses the correlation between periods t and $t - k$ of a time series as a function of the time t and lag k , after controlling for the effects of intervening lags. For a stationary time series, the PACF does not depend on t . The PACF is not symmetric about $k = 0$: the partial autocorrelation between y_t and y_{t-k} is not equal to the partial autocorrelation between y_t and y_{t+k} .

periodogram. A periodogram is a graph of the spectral density function of a time series as a function of frequency. The `pergram` command first standardizes the amplitude of the density by the sample variance of the time series, and then plots the logarithm of that standardized density. Peaks in the periodogram represent cyclical behavior in the data.

Portmanteau statistic. The portmanteau, or Q , statistic is used to test for white noise and is calculated using the first m autocorrelations of the series, where m is chosen by the user. Under the null hypothesis that the series is a white-noise process, the portmanteau statistic has a χ^2 distribution with m degrees of freedom.

Prais–Winsten estimator. A Prais–Winsten estimator is a linear regression estimator that is used when the error term exhibits first-order autocorrelation; see also [Cochrane–Orcutt estimator](#). Here the first observation in the dataset is transformed as $\tilde{y}_1 = \sqrt{1 - \rho^2} y_1$ and $\tilde{x}_1 = \sqrt{1 - \rho^2} x_1$, so that the first observation is not lost. The Prais–Winsten estimator is a generalized least-squares estimator.

priming values. Priming values are the initial, preestimation values used to begin a recursive process.

random walk. A random walk is a time-series process in which the current period's realization is equal to the previous period's realization plus a white-noise error term: $y_t = y_{t-1} + \epsilon_t$. A *random walk with drift* also contains a nonzero time-invariant constant: $y_t = \delta + y_{t-1} + \epsilon_t$. The constant term δ is known as the drift parameter. An important property of random-walk processes is that the best predictor of the value at time $t + 1$ is the value at time t plus the value of the drift parameter.

recursive regression analysis. A recursive regression analysis involves performing a regression at time t by using all available observations from some starting time t_0 through time t , performing another regression at time $t + 1$ by using all observations from time t_0 through time $t + 1$, and so on. Unlike a rolling regression analysis, the first period used for all regressions is held fixed.

rolling regression analysis. A rolling, or moving window, regression analysis involves performing regressions for each period by using the most recent m periods' data, where m is known as the window size. At time t the regression is fit using observations for times $t - 19$ through time t ; at time $t + 1$ the regression is fit using the observations for time $t - 18$ through $t + 1$; and so on.

seasonal difference operator. The period- s seasonal difference operator Δ_s denotes the difference in the value of a variable at time t and time $t - s$. Formally, $\Delta_s y_t = y_t - y_{t-s}$, and $\Delta_s^2 y_t = \Delta_s(y_t - y_{t-s}) = (y_t - y_{t-s}) - (y_{t-s} - y_{t-2s}) = y_t - 2y_{t-s} + y_{t-2s}$.

serial correlation. Serial correlation refers to regression errors that are correlated over time. If a regression model does not contain lagged dependent variables as regressors, the OLS estimates are consistent in the presence of mild serial correlation, but the covariance matrix is incorrect. When the model includes lagged dependent variables and the residuals are serially correlated, the

OLS estimates are biased and inconsistent. See, for example, [Davidson and MacKinnon \(1993, chap. 10\)](#) for more information.

serial correlation tests. Because OLS estimates are at least inefficient and potentially biased in the presence of serial correlation, econometricians have developed many tests to detect it. Popular ones include the Durbin–Watson (1950, 1951, 1971) test, the Breusch–Pagan (1980) test, and Durbin’s (1970) alternative test. See [\[R\] regress postestimation time series](#).

smoothing. Smoothing a time series refers to the process of extracting an overall trend in the data. The motivation behind smoothing is the belief that a time series exhibits a trend component as well as an irregular component and that the analyst is interested only in the trend component. Some smoothers also account for seasonal or other cyclical patterns.

spectral analysis. See [frequency-domain analysis](#).

spectral density function. The spectral density function is the derivative of the spectral distribution function. Intuitively, the spectral density function $f(\omega)$ indicates the amount of variance in a time series that is attributable to sinusoidal components with frequency ω . See also [spectral distribution function](#). The spectral density function is sometimes called the *spectrum*.

spectral distribution function. The (normalized) spectral distribution function $F(\omega)$ of a process describes the proportion of variance that can be explained by sinusoids with frequencies in the range $(0, \omega)$, where $0 \leq \omega \leq \pi$. The spectral distribution and density functions used in frequency-domain analysis are closely related to the autocorrelation function used in time-domain analysis; see [Chatfield \(2004, chap. 6\)](#).

spectrum. See [spectral density function](#).

state-space model. A state-space model describes the relationship between an observed time series and an unobservable state vector that represents the “state” of the world. The measurement equation expresses the observed series as a function of the state vector, and the transition equation describes how the unobserved state vector evolves over time. By defining the parameters of the measurement and transition equations appropriately, one can write a wide variety of time-series models in the state-space form.

steady-state equilibrium. The steady-state equilibrium is the predicted value of a variable in a dynamic model, ignoring the effects of past shocks, or, equivalently, the value of a variable, assuming that the effects of past shocks have fully died out and no longer affect the variable of interest.

strict stationarity. A process is strictly stationary if the joint distribution of $y_1, \dots, y_k$ is the same as the joint distribution of $y_{1+\tau}, \dots, y_{k+\tau}$ for all k and τ . Intuitively, shifting the origin of the series by τ units has no effect on the joint distributions.

structural model. In time-series analysis, a structural model is one that describes the relationship among a set of variables, based on underlying theoretical considerations. Structural models may contain both endogenous and exogenous variables.

SVAR. A structural vector autoregressive (SVAR) model is a type of VAR in which short- or long-run constraints are placed on the resulting impulse–response functions. The constraints are usually motivated by economic theory and therefore allow causal interpretations of the IRFs to be made.

time-domain analysis. Time-domain analysis is analysis of data viewed as a sequence of observations observed over time. The autocorrelation function, linear regression, ARCH models, and ARIMA models are common tools used in time-domain analysis.

unit-root process. A unit-root process is one that is integrated of order one, meaning that the process is nonstationary but that first-differencing the process produces a stationary series. The simplest example of a unit-root process is the random walk. See [Hamilton \(1994, chap. 15\)](#) for a discussion of when general ARMA processes may contain a unit root.

unit-root tests. Whether a process has a unit root has both important statistical and economic ramifications, so a variety of tests have been developed to test for them. Among the earliest tests proposed is the one by [Dickey and Fuller \(1979\)](#), though most researchers now use an improved variant called the augmented Dickey–Fuller test instead of the original version. Other common unit-root tests implemented in Stata include the DF–GLS test of Elliott, Rothenberg, and Stock (1996) and the Phillips–Perron (1988) test. See [\[TS\] dfuller](#), [\[TS\] dfgl](#)s, and [\[TS\] pperron](#).

Variants of unit-root tests suitable for panel data have also been developed; see [\[XT\] xtunitroot](#).

VAR. A vector autoregressive (VAR) model is a multivariate regression technique in which each dependent variable is regressed on lags of itself and on lags of all the other dependent variables in the model. Occasionally, exogenous variables are also included in the model.

VECM. A vector error-correction model (VECM) is a type of VAR that is used with variables that are cointegrated. Although first-differencing variables that are integrated of order one makes them stationary, fitting a VAR to such first-differenced variables results in misspecification error if the variables are cointegrated. See [The multivariate VECM specification](#) in [\[TS\] vec intro](#) for more on this point.

white noise. A variable u_t represents a white-noise process if the mean of u_t is zero, the variance of u_t is σ^2 , and the covariance between u_t and u_s is zero for all $s \neq t$. Gaussian white noise refers to white noise in which u_t is normally distributed.

Yule–Walker equations. The Yule–Walker equations are a set of difference equations that describe the relationship among the autocovariances and autocorrelations of an autoregressive moving-average (ARMA) process.

References

- Breusch, T. S., and A. R. Pagan. 1980. The Lagrange multiplier test and its applications to model specification in econometrics. *Review of Economic Studies* 47: 239–253.
- Chatfield, C. 2004. *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/CRC.
- Davidson, R., and J. G. MacKinnon. 1993. *Estimation and Inference in Econometrics*. New York: Oxford University Press.
- Dickey, D. A., and W. A. Fuller. 1979. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association* 74: 427–431.
- Durbin, J. 1970. Testing for serial correlation in least-squares regressions when some of the regressors are lagged dependent variables. *Econometrica* 38: 410–421.
- Durbin, J., and G. S. Watson. 1950. Testing for serial correlation in least squares regression. I. *Biometrika* 37: 409–428.
- . 1951. Testing for serial correlation in least squares regression. II. *Biometrika* 38: 159–177.
- . 1971. Testing for serial correlation in least squares regression. III. *Biometrika* 58: 1–19.
- Elliott, G., T. J. Rothenberg, and J. H. Stock. 1996. Efficient tests for an autoregressive unit root. *Econometrica* 64: 813–836.
- Hamilton, J. D. 1994. *Time Series Analysis*. Princeton: Princeton University Press.
- Phillips, P. C. B., and P. Perron. 1988. Testing for a unit root in time series regression. *Biometrika* 75: 335–346.

Subject and author index

This is the subject and author index for the *Time-Series Reference Manual*. Readers interested in topics other than time series should see the [combined subject index](#) (and the [combined author index](#)) in the *Quick Reference and Index*. The combined index indexes the *Getting Started* manuals, the *User's Guide*, and all the reference manuals except the *Mata Reference Manual*.

Semicolons set off the most important entries from the rest. Sometimes no entry will be set off with semicolons, meaning that all entries are equally important.

A

Abraham, B., [TS] [tssmooth](#), [TS] [tssmooth dexpontential](#), [TS] [tssmooth exponential](#), [TS] [tssmooth hwinters](#), [TS] [tssmooth shwinters](#)

ac command, [TS] [corrgram](#)

add, [irf](#) subcommand, [TS] [irf add](#)

Adkins, L. C., [TS] [arch](#)

Ahn, S. K., [TS] [vec intro](#)

Akaike, H., [TS] [varsoc](#)

Amemiya, T., [TS] [varsoc](#)

Amisano, G., [TS] [irf create](#), [TS] [var intro](#), [TS] [var svar](#), [TS] [vargranger](#), [TS] [varwle](#)

Anderson, B. D. O., [TS] [sspace](#)

Anderson, T. W., [TS] [vec](#), [TS] [vecrank](#)

Ansley, C. F., [TS] [arima](#)

A-PARCH, [TS] [arch](#)

AR, [TS] [dfactor](#), [TS] [sspace](#)

ARCH

- effects, estimation, [TS] [arch](#)
- model, [TS] [Glossary](#)
- postestimation, [TS] [arch postestimation](#)
- regression, [TS] [arch](#)

[arch](#) command, [TS] [arch](#), [TS] [arch postestimation](#)

ARCH models, [TS] [dvech](#)

ARIMA

- postestimation, [TS] [arima postestimation](#)
- regression, [TS] [arima](#)

[arima](#) command, [TS] [arima](#), [TS] [arima postestimation](#)

ARMA, [TS] [arch](#), [TS] [arima](#), [TS] [dfactor](#), [TS] [Glossary](#), [TS] [sspace](#)

ARMAX, [TS] [dfactor](#), [TS] [sspace](#)

ARMAX model, [TS] [arima](#), [TS] [Glossary](#)

autocorrelation, [TS] [arch](#), [TS] [arima](#), [TS] [corrgram](#), [TS] [dfactor](#), [TS] [Glossary](#), [TS] [newey](#), [TS] [prais](#), [TS] [sspace](#), [TS] [var](#), [TS] [varlmar](#)

autoregressive

- conditional heteroskedasticity, [TS] [arch](#)
- integrated moving average, [TS] [arch](#), [TS] [arima](#)
- moving average, [TS] [arch](#), [TS] [arima](#)
- process, [TS] [Glossary](#)

autoregressive conditional heteroskedasticity, [TS] [dvech](#)

autoregressive model, [TS] [dfactor](#), [TS] [sspace](#)

autoregressive moving average model, [TS] [dfactor](#), [TS] [sspace](#)

Aznar, A., [TS] [vecrank](#)

B

Bartlett, M. S., [TS] [wntestb](#)

Bartlett's

- bands, [TS] [corrgram](#)
- periodogram test, [TS] [wntestb](#)

Baum, C. F., [TS] [arch](#), [TS] [arima](#), [TS] [dfgls](#), [TS] [rolling](#), [TS] [time series](#), [TS] [tsset](#), [TS] [var](#), [TS] [wntestq](#)

Bauwens, L., [TS] [dvech](#)

Beckett, S., [TS] [corrgram](#)

Bera, A. K., [TS] [arch](#), [TS] [varnorm](#), [TS] [vecnorm](#)

Berkes, I., [TS] [dvech](#)

Berndt, E. K., [TS] [arch](#), [TS] [arima](#)

Black, F., [TS] [arch](#)

block exogeneity, [TS] [vargranger](#)

Bollerslev, T., [TS] [arch](#), [TS] [arima](#), [TS] [dvech](#)

Boswijk, H. P., [TS] [vec](#)

Bowerman, B. L., [TS] [tssmooth](#), [TS] [tssmooth dexpontential](#), [TS] [tssmooth exponential](#), [TS] [tssmooth hwinters](#), [TS] [tssmooth shwinters](#)

Box, G. E. P., [TS] [arima](#), [TS] [corrgram](#), [TS] [cumsp](#), [TS] [dfuller](#), [TS] [pergram](#), [TS] [pperron](#), [TS] [wntestq](#), [TS] [xcorr](#)

Breusch, T. S., [TS] [Glossary](#)

Brockwell, P. J., [TS] [corrgram](#), [TS] [sspace](#)

C

Caines, P. E., [TS] [sspace](#)

Casals, J., [TS] [sspace](#)

[cgraph](#), [irf](#) subcommand, [TS] [irf cgraph](#)

Chang, Y., [TS] [sspace](#)

Chatfield, C., [TS] [arima](#), [TS] [corrgram](#), [TS] [Glossary](#), [TS] [pergram](#), [TS] [tssmooth dexpontential](#), [TS] [tssmooth exponential](#), [TS] [tssmooth hwinters](#), [TS] [tssmooth ma](#), [TS] [tssmooth shwinters](#), [TS] [tssmooth](#)

Chatterjee, S., [TS] [prais](#)

Cheung, Y.-W., [TS] [dfgls](#)

Cholesky ordering, [TS] [Glossary](#)

Chou, R. Y., [TS] [arch](#)

Christiano, L. J., [TS] [irf create](#), [TS] [var svar](#)

Chu-Chun-Lin, S., [TS] [sspace](#)

clock time, [TS] [tsset](#)

cluster estimator of variance, Prais–Winsten and Cochrane–Orcutt regression, [TS] [prais](#)

Cochrane, D., [TS] [prais](#)

Cochrane–Orcutt regression, [TS] [Glossary](#), [TS] [prais](#)

cointegration, [TS] [fcst compute](#), [TS] [fcst graph](#), [TS] [Glossary](#), [TS] [vec](#), [TS] [vec intro](#), [TS] [veclmar](#), [TS] [vecnorm](#), [TS] [vecrank](#), [TS] [vecstable](#)

compute, **fcast** subcommand, [TS] **fcast compute**
 Comte, F., [TS] **dvech**
 conditional variance, [TS] **arch**, [TS] **Glossary**
 constrained estimation
 ARCH, [TS] **arch**
 ARIMA and ARMAX, [TS] **arima**
 structural vector autoregressive models, [TS] **var svar**
 vector autoregressive models, [TS] **var**
 vector error-correction models, [TS] **vec**
 constrained optimization, [TS] **arch**, [TS] **arima**,
 [TS] **dfactor**, [TS] **dvech**, [TS] **sspace**, [TS] **var**
 correlogram, [TS] **corrgram**, [TS] **Glossary**
 corrgram command, [TS] **corrgram**
 covariance stationarity, [TS] **Glossary**
 Cox, N. J., [TS] **tssmooth hwinters**, [TS] **tssmooth shwinters**
 create, **irf** subcommand, [TS] **irf create**
 cross-correlation function, [TS] **Glossary**, [TS] **xcorr**
 cross-correlogram, [TS] **xcorr**
 ctable, **irf** subcommand, [TS] **irf ctable**
 cumsp command, [TS] **cumsp**
 cumulative spectral distribution, empirical, [TS] **cumsp**

D

data manipulation, [TS] **tsappend**, [TS] **tsfill**,
 [TS] **tsreport**, [TS] **tsrevar**, [TS] **tsset**
 David, J. S., [TS] **arima**
 Davidson, R., [TS] **arch**, [TS] **arima**, [TS] **Glossary**,
 [TS] **prais**, [TS] **sspace**, [TS] **varlmar**
 Davis, R. A., [TS] **corrgram**, [TS] **sspace**
 De Jong, P., [TS] **dfactor**, [TS] **sspace**, [TS] **sspace**
 postestimation
 DeGroot, M. H., [TS] **arima**
 Deistler, M., [TS] **sspace**
 describe, **irf** subcommand, [TS] **irf describe**
 dexponential, **tssmooth** subcommand,
 [TS] **tssmooth dexponential**
 dfactor command, [TS] **dfactor**
 dfgls command, [TS] **dfgl**
 dfuller command, [TS] **dfuller**
 diagonal vech GARCH model, [TS] **dvech**
 Dickens, R., [TS] **prais**
 Dickey, D. A., [TS] **dfuller**, [TS] **Glossary**,
 [TS] **pperron**
 Dickey–Fuller test, [TS] **dfgl**, [TS] **dfuller**
 Diebold, F. X., [TS] **arch**
 difference operator, [TS] **Glossary**
 Diggle, P. J., [TS] **arima**, [TS] **wntestb**
 Ding, Z., [TS] **arch**
 Doornik, J. A., [TS] **vec**
 double-exponential smoothing, [TS] **tssmooth**
 dexponential
 drop, **irf** subcommand, [TS] **irf drop**
 Drukker, D. M., [TS] **sspace**, [TS] **vec**
 Durbin, J., [TS] **Glossary**, [TS] **prais**
 Durbin–Watson statistic, [TS] **prais**

Durlauf, S. N., [TS] **vec**, [TS] **vec intro**, [TS] **vecrank**
 dvech command, [TS] **dvech**
 dynamic-factor model, [TS] **dfactor**, [TS] **dfactor**
 postestimation, [TS] **sspace**
 dynamic forecast, [TS] **fcast compute**, [TS] **fcast**
 graph, [TS] **Glossary**
 dynamic-multiplier function, [TS] **Glossary**, [TS] **irf**,
 [TS] **irf cgraph**, [TS] **irf create**, [TS] **irf ctable**,
 [TS] **irf ograph**, [TS] **irf table**, [TS] **var intro**
 dynamic regression model, [TS] **arima**, [TS] **var**
 dynamic structural simultaneous equations, [TS] **var**
 svar

E

EGARCH, [TS] **arch**
 Eichenbaum, M., [TS] **irf create**, [TS] **var svar**
 eigenvalue stability condition, [TS] **varstable**,
 [TS] **vecstable**
 Elliott, G., [TS] **dfgl**, [TS] **Glossary**
 Enders, W., [TS] **arch**, [TS] **arima**, [TS] **corrgram**
 endogenous variable, [TS] **Glossary**
 Engle, R. F., [TS] **arch**, [TS] **arima**, [TS] **dfactor**,
 [TS] **dvech**, [TS] **vec**, [TS] **vec intro**,
 [TS] **vecrank**
 Evans, C. L., [TS] **irf create**, [TS] **var svar**
 exogenous variable, [TS] **Glossary**
 exp_list, [TS] **rolling**
 exponential smoothing, [TS] **Glossary**, [TS] **tssmooth**,
 [TS] **tssmooth exponential**
 exponential, **tssmooth** subcommand, [TS] **tssmooth**
 exponential

F

factor model, [TS] **dfactor**
 fcast compute command, [TS] **fcast compute**
 fcast graph command, [TS] **fcast graph**
 feasible generalized least squares, see FGLS
 Feller, W., [TS] **wntestb**
 FEVD, [TS] **Glossary**, [TS] **irf**, [TS] **irf create**,
 [TS] **irf ograph**, [TS] **irf table**, [TS] **var intro**,
 [TS] **varbasic**, [TS] **vec intro**
 FGLS, [TS] **dfgl**, [TS] **prais**, [TS] **var**
 filters, [TS] **tssmooth**, [TS] **tssmooth dexponential**,
 [TS] **tssmooth exponential**, [TS] **tssmooth**
 hwinters, [TS] **tssmooth ma**, [TS] **tssmooth nl**,
 [TS] **tssmooth shwinters**
 Flannery, B. P., [TS] **arch**, [TS] **arima**
 forecast, [TS] **dfactor postestimation**, [TS] **dvech**
 postestimation, [TS] **sspace postestimation**, see
 forecasting
 forecast-error variance decomposition, see FEVD
 forecasting, [TS] **arch**, [TS] **arima**, [TS] **fcast**
 compute, [TS] **fcast graph**, [TS] **irf create**,
 [TS] **tsappend**, [TS] **tssmooth**, [TS] **tssmooth**
 dexponential, [TS] **tssmooth exponential**,
 [TS] **tssmooth hwinters**, [TS] **tssmooth ma**,
 [TS] **tssmooth shwinters**, [TS] **var**, [TS] **var**
 intro, [TS] **vec**, [TS] **vec intro**

forward operator, [TS] **Glossary**
 frequency-domain analysis, [TS] **cumsp**, [TS] **Glossary**,
 [TS] **pergram**
 Friedman, M., [TS] **arima**
 Fuller, W. A., [TS] **dfuller**, [TS] **Glossary**,
 [TS] **pperron**

G

Gani, J., [TS] **wntestb**
 GARCH, [TS] **arch**, [TS] **Glossary**
 garch command, [TS] **arch**
 GARCH Models, [TS] **dvech**
 Gardiner, J. S., [TS] **tssmooth**, [TS] **tssmooth**
dexponential, [TS] **tssmooth exponential**,
 [TS] **tssmooth hwinters**, [TS] **tssmooth**
shwinters
 Gardner Jr., E. S., [TS] **tssmooth dexponential**,
 [TS] **tssmooth hwinters**
 generalized autoregressive conditional heteroskedasticity,
 [TS] **arch**, [TS] **dvech**
 generalized least-squares estimator, [TS] **Glossary**
 Geweke, J., [TS] **dfactor**
 Giannini, C., [TS] **irf create**, [TS] **var intro**, [TS] **var**
svar, [TS] **vargranger**, [TS] **varwle**
 Giles, D. E. A., [TS] **prais**
 GJR, [TS] **arch**
 Glosten, L. R., [TS] **arch**
 Gonzalo, J., [TS] **vec intro**, [TS] **vecrank**
 Gouriéroux, C., [TS] **arima**
 Granger, C. W. J., [TS] **arch**, [TS] **vargranger**,
 [TS] **vec**, [TS] **vec intro**, [TS] **vecrank**
 Granger causality, [TS] **Glossary**, [TS] **vargranger**
 graph, **fcast** subcommand, [TS] **fcast graph**
 graph, **irf** subcommand, [TS] **irf graph**
 graphs,
 autocorrelations, [TS] **corrgram**
 correlogram, [TS] **corrgram**
 cross-correlogram, [TS] **xcorr**
 cumulative spectral density, [TS] **cumsp**
 forecasts, [TS] **fcast graph**
 impulse–response functions, [TS] **irf**, [TS] **irf**
 cgraph, [TS] **irf graph**, [TS] **irf ograph**
 partial correlogram, [TS] **corrgram**
 periodogram, [TS] **pergram**
 white-noise test, [TS] **wntestb**
 Greene, W. H., [TS] **arch**, [TS] **arima**, [TS] **corrgram**,
 [TS] **var**
 Griffiths, W. E., [TS] **arch**, [TS] **prais**

H

Hadi, A. S., [TS] **prais**
 Hall, B. H., [TS] **arch**, [TS] **arima**
 Hall, R. E., [TS] **arch**, [TS] **arima**

Hamilton, J. D., [TS] **arch**, [TS] **arima**,
 [TS] **corrgram**, [TS] **dfuller**, [TS] **fcast**
compute, [TS] **Glossary**, [TS] **irf**, [TS] **irf**
create, [TS] **pergram**, [TS] **pperron**,
 [TS] **sspace postestimation**, [TS] **sspace**,
 [TS] **time series**, [TS] **var**, [TS] **var intro**,
 [TS] **var svar**, [TS] **vargranger**, [TS] **varnorm**,
 [TS] **varsoc**, [TS] **varstable**, [TS] **varwle**,
 [TS] **vec**, [TS] **vec intro**, [TS] **vecnorm**,
 [TS] **vecrank**, [TS] **vecstable**, [TS] **xcorr**
 Hannan, E. J., [TS] **sspace**
 Hardin, J. W., [TS] **newey**, [TS] **prais**
 Harvey, A. C., [TS] **arch**, [TS] **arima**, [TS] **prais**,
 [TS] **sspace**, [TS] **sspace postestimation**,
 [TS] **var svar**
 Hausman, J. A., [TS] **arch**, [TS] **arima**
 Haver Analytics, [TS] **haver**
 haver command, [TS] **haver**
 heteroskedasticity
 conditional, [TS] **arch**
 multiplicative heteroskedastic regression, [TS] **arch**
 Higgins, M. L., [TS] **arch**
 Hildreth, C., [TS] **prais**
 Hildreth–Lu regression, [TS] **prais**
 Hill, R. C., [TS] **arch**, [TS] **prais**
 Hipel, K. W., [TS] **arima**
 Holt, C. C., [TS] **tssmooth**, [TS] **tssmooth**
dexponential, [TS] **tssmooth exponential**,
 [TS] **tssmooth hwinters**, [TS] **tssmooth**
shwinters
 Holt–Winters smoothing, [TS] **Glossary**,
 [TS] **tssmooth**, [TS] **tssmooth dexponential**,
 [TS] **tssmooth exponential**, [TS] **tssmooth**
hwinters, [TS] **tssmooth shwinters**
 Horváth, L., [TS] **dvech**
 Hubrich, K., [TS] **vec intro**, [TS] **vecrank**
 hwinters, **tssmooth** subcommand, [TS] **tssmooth**
hwinters

I

impulse–response functions, see **IRF**
 independent and identically distributed, [TS] **Glossary**
 information criterion, [TS] **varsoc**
 innovation accounting, [TS] **irf**
 integrated process, [TS] **Glossary**
 IRF, [TS] **Glossary**, [TS] **irf**, [TS] **irf add**, [TS] **irf**
cgraph, [TS] **irf create**, [TS] **irf etable**, [TS] **irf**
describe, [TS] **irf drop**, [TS] **irf graph**, [TS] **irf**
ograph, [TS] **irf rename**, [TS] **irf set**, [TS] **irf**
table, [TS] **var intro**, [TS] **varbasic**, [TS] **vec**
intro
 cumulative impulse–response functions, [TS] **irf**
 create
 cumulative orthogonalized impulse–response
 functions, [TS] **irf create**
 orthogonalized impulse–response functions, [TS] **irf**
 create
 irf add command, [TS] **irf add**
 irf cgraph command, [TS] **irf cgraph**

irf commands, introduction, [TS] **irf**
 irf create command, [TS] **irf create**
 irf ctable command, [TS] **irf ctable**
 irf describe command, [TS] **irf describe**
 irf drop command, [TS] **irf drop**
 irf graph command, [TS] **irf graph**
 irf ograph command, [TS] **irf ograph**
 irf rename command, [TS] **irf rename**
 irf set command, [TS] **irf set**
 irf table command, [TS] **irf table**

J

Jagannathan, R., [TS] **arch**
 Jarque, C. M., [TS] **varnorm**, [TS] **vecnorm**
 Jarque–Bera statistic, [TS] **varnorm**, [TS] **vecnorm**
 Jenkins, G. M., [TS] **arima**, [TS] **corrgram**,
 [TS] **cumsp**, [TS] **dfuller**, [TS] **pergram**,
 [TS] **pperron**, [TS] **xcorr**
 Jerez, M., [TS] **sspace**
 Johansen, S., [TS] **irf create**, [TS] **varlmar**, [TS] **vec**,
 [TS] **vec intro**, [TS] **veclmar**, [TS] **vecnorm**,
 [TS] **vecrank**, [TS] **vecstable**
 Johnson, L. A., [TS] **tssmooth**, [TS] **tssmooth**
dexponential, [TS] **tssmooth exponential**,
 [TS] **tssmooth hwinters**, [TS] **tssmooth**
shwinters
 Judge, G. G., [TS] **arch**, [TS] **prais**

K

Kalman, R. E., [TS] **arima**
 Kalman
 filter, [TS] **arima**, [TS] **dfactor**, [TS] **dfactor**
postestimation, [TS] **Glossary**, [TS] **sspace**,
 [TS] **sspace postestimation**
 forecast, [TS] **dfactor postestimation**, [TS] **sspace**
postestimation
 smoothing, [TS] **dfactor postestimation**,
 [TS] **sspace postestimation**
 Kim, I.-M., [TS] **vec**, [TS] **vec intro**, [TS] **vecrank**
 King, M. L., [TS] **prais**
 King, R. G., [TS] **vecrank**
 Kmenta, J., [TS] **arch**, [TS] **prais**, [TS] **rolling**
 Kohn, R., [TS] **arima**
 Kroner, K. F., [TS] **arch**
 kurtosis, [TS] **varnorm**, [TS] **vecnorm**

L

lag-exclusion statistics, [TS] **varwle**
 lag operator, [TS] **Glossary**
 lag-order selection statistics, [TS] **varsoc**; [TS] **var**,
 [TS] **var intro**, [TS] **var svar**, [TS] **vec intro**
 Lagrange-multiplier test, [TS] **varlmar**, [TS] **veclmar**
 Lai, K. S., [TS] **dfgl**
 Laurent, S., [TS] **dvech**
 leap seconds, [TS] **tsset**

Ledolter, J., [TS] **tssmooth**, [TS] **tssmooth**
dexponential, [TS] **tssmooth exponential**,
 [TS] **tssmooth hwinters**, [TS] **tssmooth**
shwinters
 Lee, T.-C., [TS] **arch**, [TS] **prais**
 Lieberman, O., [TS] **dvech**
 Lilien, D. M., [TS] **arch**
 Lim, G. C., [TS] **arch**
 linear regression, [TS] **newey**, [TS] **prais**, [TS] **var**,
 [TS] **var intro**, [TS] **var svar**, [TS] **varbasic**
 Ljung, G. M., [TS] **wntestq**
 Lu, J. Y., [TS] **prais**
 Lütkepohl, H., [TS] **arch**, [TS] **dfactor**, [TS] **dvech**,
 [TS] **fcast compute**, [TS] **irf**, [TS] **irf**
create, [TS] **prais**, [TS] **sspace**, [TS] **sspace**
postestimation, [TS] **time series**, [TS] **var**,
 [TS] **var intro**, [TS] **var svar**, [TS] **varbasic**,
 [TS] **vargranger**, [TS] **varnorm**, [TS] **varsoc**,
 [TS] **varstable**, [TS] **varwle**, [TS] **vec intro**,
 [TS] **vecnorm**, [TS] **vecrank**, [TS] **vecstable**

M

MA, [TS] **arch**, [TS] **arima**, [TS] **dfactor**, [TS] **sspace**
ma, **tssmooth** subcommand, [TS] **tssmooth ma**
 MacKinnon, J. G., [TS] **arch**, [TS] **arima**, [TS] **dfuller**,
 [TS] **Glossary**, [TS] **pperron**, [TS] **prais**,
 [TS] **sspace**, [TS] **varlmar**
 Maddala, G. S., [TS] **vec**, [TS] **vec intro**, [TS] **vecrank**
 Magnus, J. R., [TS] **var svar**
 Mandelbrot, B., [TS] **arch**
 Mangel, M., [TS] **varwle**
 maximum likelihood, [TS] **var**, [TS] **var intro**,
 [TS] **var svar**, [TS] **varbasic**
 McCullough, B. D., [TS] **corrgram**
 McDowell, A. W., [TS] **arima**
 McLeod, A. I., [TS] **arima**
 Meiselman, D., [TS] **arima**
 Miller, J. I., [TS] **sspace**
 Monfort, A., [TS] **arima**
 Montgomery, D. C., [TS] **tssmooth**, [TS] **tssmooth**
dexponential, [TS] **tssmooth exponential**,
 [TS] **tssmooth hwinters**, [TS] **tssmooth**
shwinters
 Moore, J. B., [TS] **sspace**
 moving average, [TS] **arch**, [TS] **arima**, [TS] **Glossary**,
 [TS] **tssmooth**, [TS] **tssmooth ma**
 multiplicative heteroskedasticity, [TS] **arch**
 multivariate ARCH, [TS] **dvech postestimation**
 multivariate GARCH, [TS] **dvech**, [TS] **dvech**
postestimation
 multivariate time series, [TS] **dfactor**, [TS] **dvech**,
 [TS] **sspace**, [TS] **var**, [TS] **var intro**, [TS] **var**
svar, [TS] **varbasic**, [TS] **vec**, [TS] **vec intro**,
 [TS] **xcorr**

N

NARCH, [TS] **arch**

NARCHK, [TS] **arch**
 Nelson, D. B., [TS] **arch**, [TS] **arima**, [TS] **dvech**
 Neudecker, H., [TS] **var svar**
 Newbold, P., [TS] **arima**, [TS] **vec intro**
 Newey, W. K., [TS] **newey**, [TS] **pperron**
newey command, [TS] **newey**, [TS] **newey**
postestimation
 Newey–West
 covariance matrix, [TS] **Glossary**
 postestimation, [TS] **newey postestimation**
 regression, [TS] **newey**
 Newton, H. J., [TS] **arima**, [TS] **corrgram**,
 [TS] **cumsp**, [TS] **dfuller**, [TS] **pergram**,
 [TS] **wntestb**, [TS] **xcorr**
 Ng, S., [TS] **dfgls**
 Nielsen, B., [TS] **varsoc**, [TS] **vec intro**
nl, **tssmooth** subcommand, [TS] **tssmooth nl**
 nonlinear
 estimation, [TS] **arch**
 smoothing, [TS] **tssmooth nl**
 nonstationary time series, [TS] **dfgls**, [TS] **dfuller**,
 [TS] **pperron**, [TS] **vec intro**, [TS] **vec**
 normality test
 after VAR or SVAR, [TS] **varnorm**
 after VEC, [TS] **vecnorm**
 NPARCH, [TS] **arch**

O

O’Connell, R. T., [TS] **tssmooth**, [TS] **tssmooth**
dexponential, [TS] **tssmooth exponential**,
 [TS] **tssmooth hwinters**, [TS] **tssmooth**
shwinters
ograph, **irf** subcommand, [TS] **irf ograph**
 Olkin, I., [TS] **wntestb**
 Orcutt, G. H., [TS] **prais**
 orthogonalized impulse–response function,
 [TS] **Glossary**, [TS] **irf**, [TS] **var intro**,
 [TS] **vec**, [TS] **vec intro**
 Osterwald-Lenum, M., [TS] **vecrank**

P

pac command, [TS] **corrgram**
 Pagan, A. R., [TS] **Glossary**
PARCH, [TS] **arch**
 Park, J. Y., [TS] **sspace**, [TS] **vec**, [TS] **vec intro**,
 [TS] **vecrank**
 partial autocorrelation function, [TS] **corrgram**,
 [TS] **Glossary**
 Paulsen, J., [TS] **varsoc**, [TS] **vec intro**
pergram command, [TS] **pergram**
 periodogram, [TS] **Glossary**, [TS] **pergram**
 Perron, P., [TS] **dfgls**, [TS] **Glossary**, [TS] **pperron**
 Phillips, P. C. B., [TS] **Glossary**, [TS] **pperron**,
 [TS] **vargranger**, [TS] **vec**, [TS] **vec intro**,
 [TS] **vecrank**
 Phillips–Perron test, [TS] **pperron**

Pierce, D. A., [TS] **wntestq**
 Pisati, M., [TS] **time series**
 Pitarakis, J.-Y., [TS] **vecrank**
 Plosser, C. I., [TS] **vecrank**
 portmanteau statistic, [TS] **Glossary**
 portmanteau test, [TS] **corrgram**, [TS] **wntestq**
 postestimation command, [TS] **fcst compute**,
 [TS] **fcst graph**, [TS] **irf**, [TS] **vargranger**,
 [TS] **varlmar**, [TS] **varnorm**, [TS] **varsoc**,
 [TS] **varstable**, [TS] **varwle**, [TS] **veclmar**,
 [TS] **vecnorm**, [TS] **vecstable**
pperron command, [TS] **pperron**
 Prais, S. J., [TS] **prais**
prais command, [TS] **Glossary**, [TS] **prais**, [TS] **prais**
postestimation
 Prais–Winsten regression, see **prais** command
 predict, [TS] **dvech postestimation**, [TS] **sspace**
postestimation
 predict command=predict command, [TS] **dfactor**
postestimation
 Press, W. H., [TS] **arch**, [TS] **arima**
 priming values, [TS] **Glossary**

Q

Q statistic, [TS] **wntestq**

R

random walk, [TS] **Glossary**
 recursive estimation, [TS] **rolling**
 recursive regression analysis, [TS] **Glossary**
 Reinsel, G. C., [TS] **arima**, [TS] **corrgram**,
 [TS] **cumsp**, [TS] **dfuller**, [TS] **pergram**,
 [TS] **pperron**, [TS] **vec intro**, [TS] **xcorr**
rename, **irf** subcommand, [TS] **irf rename**
 Robins, R. P., [TS] **arch**
 robust, Huber/White/sandwich estimator of variance
 ARCH, [TS] **arch**
 ARIMA and ARMAX, [TS] **arima**
 Newey–West regression, [TS] **newey**
 Prais–Winsten and Cochrane–Orcutt regression,
 [TS] **prais**
rolling command, [TS] **rolling**
 rolling regression, [TS] **Glossary**, [TS] **rolling**
 Rombouts, J. V. K., [TS] **dvech**
 Room, T., [TS] **arima**
 Rothenberg, T. J., [TS] **dfgls**, [TS] **Glossary**,
 [TS] **sspace**, [TS] **var svar**, [TS] **vec**
 Runkle, D. E., [TS] **arch**

S

SAARCH, [TS] **arch**
 Saikkonen, P., [TS] **vec intro**, [TS] **vecrank**
 Salvador, M., [TS] **vecrank**
 Samaniego, F. J., [TS] **varwle**
 Sargan, J. D., [TS] **prais**

Sargent, T. J., [TS] **dfactor**
 Schneider, W., [TS] **sspace**
 Schwert, G. W., [TS] **dfgls**
 seasonal
 ARIMA, [TS] **tssmooth**, [TS] **tssmooth shwinters**
 smoothing, [TS] **tssmooth**, [TS] **tssmooth shwinters**
 seasonal difference operator, [TS] **Glossary**
 selection-order statistics, [TS] **varsoc**
 Serfling, R. J., [TS] **irf create**
 serial correlation, see autocorrelation
set, **irf** subcommand, [TS] **irf set**
 Shumway, R. H., [TS] **arima**
shwinters, **tssmooth** subcommand, [TS] **tssmooth shwinters**
 Silvennoinen, A., [TS] **dvech**
 Sims, C. A., [TS] **dfactor**, [TS] **irf create**, [TS] **var svar**, [TS] **vec**, [TS] **vec intro**, [TS] **vecrank**
 skewness, [TS] **varnorm**
 smoothers, [TS] **Glossary**, [TS] **tssmooth**,
 [TS] **tssmooth dexpontential**, [TS] **tssmooth exponential**, [TS] **tssmooth hwinters**,
 [TS] **tssmooth ma**, [TS] **tssmooth nl**,
 [TS] **tssmooth shwinters**
 Sotoca, S., [TS] **sspace**
 spectral distribution, [TS] **cumsp**, [TS] **Glossary**,
 [TS] **pergram**
 spectrum, [TS] **Glossary**
 Sperling, R., [TS] **arch**, [TS] **arima**, [TS] **dfgls**,
 [TS] **wntestq**
sspace command, [TS] **sspace**
 stability, [TS] **var**, [TS] **var intro**, [TS] **var svar**,
 [TS] **vecstable**
 after VAR or SVAR, [TS] **varstable**
 after VEC, [TS] **vec**, [TS] **vec intro**
 state-space model, [TS] **arima**, [TS] **dfactor**,
 [TS] **dfactor postestimation**, [TS] **Glossary**,
 [TS] **sspace**, [TS] **sspace postestimation**
 stationary time series, [TS] **dfgls**, [TS] **dfuller**,
 [TS] **pperron**, [TS] **var**, [TS] **var intro**,
 [TS] **vec**, [TS] **vec intro**
 steady-state equilibrium, [TS] **Glossary**
 Stock, J. H., [TS] **arch**, [TS] **dfactor**, [TS] **dfgls**,
 [TS] **Glossary**, [TS] **irf create**, [TS] **rolling**,
 [TS] **sspace**, [TS] **time series**, [TS] **var**, [TS] **var intro**, [TS] **var svar**, [TS] **vec**, [TS] **vec intro**,
 [TS] **vecrank**
 strict stationarity, [TS] **Glossary**
 structural time-series model, [TS] **sspace**
 structural vector autoregression, see SVAR
SUR, [TS] **dfactor**
SVAR, [TS] **Glossary**, [TS] **var intro**, [TS] **var svar**
 postestimation, [TS] **var svar postestimation**;
 [TS] **fcst compute**, [TS] **fcst graph**, [TS] **irf**,
 [TS] **irf create**, [TS] **vargranger**, [TS] **varlmar**,
 [TS] **varnorm**, [TS] **varsoc**, [TS] **varstable**,
 [TS] **varwle**
svr command, [TS] **var svar**, [TS] **var svar**
 postestimation

T

tables, [TS] **irf ctable**, [TS] **irf table**
TARCH, [TS] **arch**
 Teräsvirta, T., [TS] **dvech**
 test,
 Dickey–Fuller, [TS] **dfgls**, [TS] **dfuller**
 granger causality, [TS] **vargranger**
 Lagrange-multiplier, [TS] **varlmar**, [TS] **veclmar**
 normality, [TS] **varnorm**, [TS] **vecnorm**
 Wald, [TS] **vargranger**, [TS] **varwle**
 Teukolsky, S. A., [TS] **arch**, [TS] **arima**
 Theil, H., [TS] **prais**
 time-domain analysis, [TS] **arch**, [TS] **arima**,
 [TS] **Glossary**
 time-series
 analysis, [TS] **pergram**
 operators, [TS] **tsset**
 time-varying variance, [TS] **arch**
 tsappend command, [TS] **tsappend**
 Tsay, R. S., [TS] **varsoc**, [TS] **vec intro**
tsfill command, [TS] **tsfill**
tsline command, [TS] **tsline**
tsreport command, [TS] **tsreport**
tsrevar command, [TS] **tsrevar**
tsrline command, [TS] **tsline**
tsset command, [TS] **tsset**
tssmooth commands, introduction, [TS] **tssmooth**
tssmooth dexpontential command, [TS] **tssmooth**
 dexpontential
tssmooth exponential command, [TS] **tssmooth**
 exponential
tssmooth hwinters command, [TS] **tssmooth**
 hwinters
tssmooth ma command, [TS] **tssmooth ma**
tssmooth nl command, [TS] **tssmooth nl**
tssmooth shwinters command, [TS] **tssmooth**
 shwinters

U

unit-root
 models, [TS] **vec**, [TS] **vec intro**
 process, [TS] **Glossary**
 test, [TS] **dfgls**, [TS] **dfuller**, [TS] **Glossary**,
 [TS] **pperron**
 univariate time series, [TS] **arch**, [TS] **arima**,
 [TS] **newey**, [TS] **prais**
 unobserved-component model, [TS] **dfactor**,
 [TS] **sspace**

V

VAR, [TS] **dfactor**, [TS] **Glossary**, [TS] **sspace**,
 [TS] **var**, [TS] **var intro**, [TS] **var svar**,
 [TS] **varbasic**

VAR, *continued*

postestimation, [TS] [fcast compute](#), [TS] [fcast graph](#), [TS] [irf](#), [TS] [irf create](#), [TS] [var postestimation](#), [TS] [vargranger](#), [TS] [varlmar](#), [TS] [varnorm](#), [TS] [varsoc](#), [TS] [varstable](#), [TS] [varwle](#)

var command, [TS] [var](#), [TS] [var postestimation](#)

varbasic command, [TS] [varbasic](#), [TS] [varbasic postestimation](#)

vargranger command, [TS] [vargranger](#)

variance decompositions, see [FEVD](#)

varlmar command, [TS] [varlmar](#)

varnorm command, [TS] [varnorm](#)

varsoc command, [TS] [varsoc](#)

varstable command, [TS] [varstable](#)

varwle command, [TS] [varwle](#)

vec command, [TS] [vec](#), [TS] [vec postestimation](#)

veclmar command, [TS] [veclmar](#)

VECM, [TS] [dvech](#), [TS] [Glossary](#), [TS] [vec](#), [TS] [vec intro](#)

postestimation, [TS] [fcast compute](#), [TS] [fcast graph](#), [TS] [irf](#), [TS] [irf create](#), [TS] [varsoc](#), [TS] [vec postestimation](#), [TS] [veclmar](#), [TS] [vecnorm](#), [TS] [vecrank](#), [TS] [vecstable](#)

vecnorm command, [TS] [vecnorm](#)

vecrank command, [TS] [vecrank](#)

vecstable command, [TS] [vecstable](#)

vector autoregression, see [VAR](#)

vector autoregressive forecast, [TS] [fcast compute](#), [TS] [fcast graph](#)

vector autoregressive moving average model, [TS] [dfactor](#), [TS] [sspace](#)

vector error-correction model, see [VECM](#)

Vetterling, W. T., [TS] [arch](#), [TS] [arima](#)

W

Wald, A., [TS] [varwle](#)

Wald test, [TS] [vargranger](#), [TS] [varwle](#)

Watson, G. S., [TS] [Glossary](#), [TS] [prais](#)

Watson, M. W., [TS] [arch](#), [TS] [dfactor](#), [TS] [dfgls](#), [TS] [irf create](#), [TS] [rolling](#), [TS] [sspace](#), [TS] [time series](#), [TS] [var](#), [TS] [var intro](#), [TS] [var svar](#), [TS] [vec](#), [TS] [vec intro](#), [TS] [vecrank](#)

weighted moving average, [TS] [tssmooth](#), [TS] [tssmooth ma](#)

West, K. D., [TS] [newey](#), [TS] [pperron](#)

White, H., [TS] [newey](#), [TS] [prais](#)

white noise, [TS] [Glossary](#)

white-noise test, [TS] [wntestb](#), [TS] [wntestq](#)

Wiggins, V. L., [TS] [arch](#), [TS] [arima](#), [TS] [sspace](#)

Winsten, C. B., [TS] [prais](#)

Winters, P. R., [TS] [tssmooth](#), [TS] [tssmooth dexponential](#), [TS] [tssmooth exponential](#), [TS] [tssmooth hwinters](#), [TS] [tssmooth shwinters](#)

wntestb command, [TS] [wntestb](#)

wntestq command, [TS] [wntestq](#)

Wolfowitz, J., [TS] [varwle](#)

Wooldridge, J. M., [TS] [arch](#), [TS] [dvech](#), [TS] [prais](#)

X

xcorr command, [TS] [xcorr](#)

Y

Yar, M., [TS] [tssmooth](#), [TS] [tssmooth dexponential](#), [TS] [tssmooth exponential](#), [TS] [tssmooth hwinters](#), [TS] [tssmooth shwinters](#)

Yule–Walker equations, [TS] [corrgram](#), [TS] [Glossary](#)

Z

Zakoian, J. M., [TS] [arch](#)

Zellner, A., [TS] [prais](#)

