

GARCH Toolbox

For Use with MATLAB®

- Computation
- Visualization
- Programming

How to Contact The MathWorks:



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

GARCH Toolbox User's Guide

© COPYRIGHT 1999–2006 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

July 1999	First printing	New for Version 1.0 (Release 11)
November 2000	Online only	Revised for Version 1.0.1 (Release 12)
July 2002	Online only	Revised for Version 1.0.2 (Release 13)
November 2002	Second printing	Revised for Version 2.0 (Release 13+)
June 2004	Online only	Minor revision for Version 2.0.1 (Release 14)
August 2004	Third printing	Revised for Version 2.0.1
September 2005	Online only	Revised for Version 2.1 (Release 14SP3)
March 2006	Fourth printing	Revised for Version 2.2 (Release 2006a)

Getting Started

1

What Is the GARCH Toolbox?	1-2
GARCH Overview	1-3
What Is GARCH?	1-3
Why Use GARCH?	1-3
GARCH Limitations	1-4
Software Requirements and Compatibility	1-5
Expected Background	1-6
Technical Conventions	1-7
Data Sets	1-11
DEM2GBP	1-11
NASDAQ	1-12
NYSE	1-12

GARCH Overview

2

Modeling of Financial Time Series	2-2
Characteristics of Financial Time Series	2-2
Correlation and Forecasting of Financial Time Series	2-4
Serial Dependence in Innovations	2-5
Conditional Mean and Variance Models	2-6
Conditional Mean Model	2-6
Conditional Variance Models	2-7
Comments on the Models	2-10

The Default Model	2-13
Primary Toolbox Functions	2-14
Analysis and Estimation Example	
Using the Default Model	2-16
Preestimation Analysis	2-16
Parameter Estimation	2-25
Postestimation Analysis	2-28

GARCH Specification Structure

3

Introduction	3-2
Equation Variables and Parameter Names	3-4
Conditional Mean Model	3-4
Conditional Variance Models	3-4
Examples of Specification Structures	3-5
Reading and Writing Specification Structures	3-8
Creating and Modifying a Specification Structure	3-8
Retrieving Specification Structure Values	3-11

Simulation

4

Simulating Sample Paths	4-2
Introduction	4-2
Simulating a Single Path	4-4
Simulating Multiple Paths	4-6

Presample Data	4-7
Automatically Generated Presample Data	4-7
User-Specified Presample Data	4-13

Estimation

5

Maximum Likelihood Estimation	5-2
 Initial Parameter Estimates	 5-4
User-Specified Initial Estimates	5-4
Automatically Generated Initial Estimates	5-5
Parameter Bounds	5-9
 Presample Observations	 5-11
User-Specified Presample Observations	5-11
Automatically Generated Presample Observations	5-11
 Termination Criteria and Optimization Results	 5-13
MaxIter and MaxFunEvals	5-13
TolCon, TolFun, and TolX	5-14
Convergence	5-14
Optimization Results	5-15
Constraint Violation Tolerance	5-16
 Examples	 5-19
Specifying Presample Data	5-19
Presample Data and Transient Effects	5-23
Alternative Technique for Estimating ARMA(R,M) Parameters	5-29
Active Lower Bound Constraint	5-30
Determining Convergence Status	5-34

6

Minimum Mean Square Error Forecasting	6-2
Conditional Standard Deviations of Future Innovations	6-2
Conditional Mean Forecasts of the Return Series	6-3
MMSE Volatility Forecasts of Returns	6-3
RMSE Associated with Conditional Mean Forecasts	6-4
Presample Observations	6-5
Asymptotic Behavior for Long-Range Forecast Horizons .	6-6
Examples	6-8
Computing a Forecast	6-8
Volatility Forecasts over Multiple Periods	6-11
Computing a Forecast with Multiple Realizations	6-14

**Regression Components
in Conditional Mean Models**

7

Introduction	7-2
Incorporating a Regression Model in an Estimation	7-3
Fitting a Model to a Simulated Return Series	7-3
Fitting a Regression Model to the Same Return Series	7-5
Simulation and Inference Using a Regression Component	7-9
Forecasting Using a Regression Component	7-10
Forecasted Explanatory Data	7-10
Generating Forecasted Explanatory Data	7-11
Ordinary Least Squares Regression	7-12
Regression in a Monte Carlo Framework	7-14

Univariate Unit Root Tests

8

Introduction	8-2
Critical Values	8-2
Serial Dependence	8-2
Dickey-Fuller Tests	8-3
dfARTest	8-3
dfARDTest	8-4
dfTSTest	8-4
Phillips-Perron Tests	8-5
ppARTest	8-5
ppARDTest	8-6
ppTSTest	8-6
How to Test for Unit Roots: Inputs and Outputs	8-7
Lags	8-7
Significance Level	8-7
TestType	8-8
Outputs	8-8
Interpretation of Results	8-9
Examples	8-10
Test GDP by OLS Regression with a Stationary Component .	8-11
Test T-Bill Rate by OLS Regression with a Drift Component .	8-16

Model Selection and Analysis

9

Likelihood Ratio Tests	9-2
Akaike and Bayesian Information Criteria	9-5
Equality Constraints and Parameter Significance	9-7

The Specification Structure Fix Fields	9-7
The GARCH(2,1) Model as an Example	9-8
Equality Constraints and Initial Parameter Estimates ...	9-12
Complete Model Specification	9-12
Empty Fix Fields	9-13
Limiting Use of Equality Constraints	9-14
Simplicity and Parsimony	9-15

Advanced Example

10

Estimating the Model	10-2
Forecasting	10-4
Monte Carlo Simulation	10-6
Comparing Forecasts with Simulation Results	10-8

Function Reference

11

Functions — By Category	11-2
GARCH Modeling	11-2
GARCH Innovations Inference	11-2
Statistics and Tests	11-2
GARCH Specification Structure Interface Functions	11-3
Helpers and Utilities	11-3
Graphics	11-4

Functions — Alphabetical List 11-5

Bibliography

A

Glossary

Index

Getting Started

What Is the GARCH Toolbox? (p. 1-2)	Introduces the GARCH Toolbox, and describes its intended use and its capabilities.
GARCH Overview (p. 1-3)	Introduces GARCH and the characteristics of GARCH models that are commonly associated with financial time series.
Software Requirements and Compatibility (p. 1-5)	Lists other MathWorks toolboxes and version compatibility required by the GARCH Toolbox.
Expected Background (p. 1-6)	Describes the intended audience for this product.
Technical Conventions (p. 1-7)	Describes the use of common mathematical terms in this guide. See the “Glossary” for definitions of GARCH-specific terms.
Data Sets (p. 1-11)	Introduces the data sets that are used in examples throughout this manual.

What Is the GARCH Toolbox?

The GARCH Toolbox, combined with MATLAB® and the Optimization and Statistics Toolboxes, provides an integrated computing environment for modeling the volatility of univariate economic time series. The GARCH Toolbox uses a general ARMAX conditional mean model combined with a conditional variance model of GARCH, GJR, or EGARCH form to perform simulation, forecasting, and parameter estimation of univariate time series in the presence of conditional heteroscedasticity. Supporting functions perform tasks such as pre- and postestimation diagnostic testing, hypothesis testing of residuals, model order selection, and time-series transformations. Graphics capabilities let you plot correlation functions and visually compare matched innovations, volatility, and return series.

More specifically, you can

- Perform Monte Carlo simulation of univariate returns, innovations, and conditional volatilities
- Specify general ARMAX conditional mean models combined with conditional variance models of GARCH, GJR, or EGARCH form for univariate asset returns
- Estimate parameters of general ARMAX conditional mean models combined with conditional variance models of GARCH, GJR, or EGARCH form
- Generate minimum mean square error forecasts of the conditional mean and conditional variance of univariate return series
- Perform pre- and postestimation diagnostic and hypothesis testing, such as Engle's ARCH test, Ljung-Box Q-statistic test, likelihood ratio tests, and AIC/BIC model order selection
- Perform graphical correlation analysis, including autocorrelation, cross correlation, and partial autocorrelation
- Convert price/return series to return/price series, and transform finite-order ARMA models to infinite-order AR and MA models

GARCH Overview

This section discusses

- “What Is GARCH?” on page 1-3
- “Why Use GARCH?” on page 1-3
- “GARCH Limitations” on page 1-4

What Is GARCH?

GARCH stands for Generalized Autoregressive Conditional Heteroscedasticity. Loosely speaking, you can think of heteroscedasticity as time-varying variance (i.e., volatility). Conditional implies a dependence on the observations of the immediate past, and autoregressive describes a feedback mechanism that incorporates past observations into the present. GARCH then is a mechanism that includes past variances in the explanation of future variances. More specifically, GARCH is a time-series technique that allows users to model the serial dependence of volatility.

In this manual, whenever a time series is said to have GARCH effects, the series is heteroscedastic, i.e., its variances vary with time. If its variances remain constant with time, the series is homoscedastic.

Why Use GARCH?

GARCH modeling builds on advances in the understanding and modeling of volatility in the last decade. It takes into account excess kurtosis (i.e., fat tail behavior) and volatility clustering, two important characteristics of financial time series. It provides accurate forecasts of variances and covariances of asset returns through its ability to model time-varying conditional variances. As a consequence, you can apply GARCH models to such diverse fields as risk management, portfolio management and asset allocation, option pricing, foreign exchange, and the term structure of interest rates.

You can find highly significant GARCH effects in equity markets, not only for individual stocks, but for stock portfolios and indices, and equity futures markets as well [5]. These effects are important in such areas as value-at-risk (VaR) and other risk management applications that concern the efficient allocation of capital. You can use GARCH models to examine the relationship between long- and short-term interest rates. As the uncertainty for rates over various horizons changes through time, you can also apply GARCH models in

the analysis of time-varying risk premiums [5]. Foreign exchange markets, which couple highly persistent periods of volatility and tranquility with significant fat-tail behavior [5], are particularly well-suited for GARCH modeling.

Note Bollerslev [4] developed GARCH as a generalization of Engle's [12] original ARCH volatility modeling technique. Bollerslev designed GARCH to offer a more parsimonious model (i.e., using fewer parameters) that lessens the computational burden.

GARCH Limitations

Although GARCH models are useful across a wide range of applications, they do have limitations:

- GARCH models are only part of a solution. Although GARCH models are usually applied to return series, financial decisions are rarely based solely on expected returns and volatilities.
- GARCH models are parametric specifications that operate best under relatively stable market conditions [15]. Although GARCH is explicitly designed to model time-varying conditional variances, GARCH models often fail to capture highly irregular phenomena, including wild market fluctuations (e.g., crashes and subsequent rebounds), and other highly unanticipated events that can lead to significant structural change.
- GARCH models often fail to fully capture the fat tails observed in asset return series. Heteroscedasticity explains some of the fat-tail behavior, but typically not all of it. To compensate for this limitation, fat-tailed distributions such as Student's t have been applied to GARCH modeling.

Software Requirements and Compatibility

The GARCH Toolbox requires the Statistics and Optimization Toolboxes. However, you need not read those manuals before reading this one.

The GARCH Toolbox Version 2.1 is compatible with Release 14 with Service Pack 3, including MATLAB 7.1, Statistics Toolbox 5.1, and Optimization Toolbox 3.0.3.

Expected Background

This guide is a practical introduction to the GARCH Toolbox. In general, it assumes you are familiar with the basic concepts of General Autoregressive Conditional Heteroscedasticity (GARCH) modeling.

In designing the GARCH Toolbox and this manual, we assume your title is similar to one of these:

- Analyst, quantitative analyst
- Risk manager
- Portfolio manager
- Fund manager, asset manager
- Economist
- Financial engineer
- Trader
- Student, professor, or other academic

We also assume your background, education, training, and responsibilities match some aspects of this profile:

- Finance, economics, perhaps accounting
- Engineering, mathematics, physics, other quantitative sciences
- Bachelor's degree minimum; MS or MBA likely; Ph.D. perhaps; CFA
- Comfortable with probability theory, statistics, and algebra
- Understand linear or matrix algebra, calculus, and differential equations
- Previously done traditional programming (C, Fortran, etc.)
- Responsible for instruments or analyses involving large sums of money
- Perhaps new to MATLAB

Technical Conventions

This user's guide uses the following definitions and descriptions. See the "Glossary" for general term definitions.

Array and Vector Size

The *size* of an array describes the dimensions of the array. If a matrix has m rows and n columns, its size is m -by- n . If two arrays are the same size, their dimensions are the same.

If two vectors are of the same size, then they not only have the same length, but they also have the same orientation.

Vector Length

The *length* of a vector indicates only the number of elements in the vector. If the length of a vector is n , it could be a 1-by- n (row) vector or an n -by-1 (column) vector. Two vectors of length n , one a row vector and the other a column vector, do not have the same size.

Time-Series Arrays

The concept of a time series, an ordered set of observations stored in a MATLAB array, is used throughout this User's Guide. The rows of a time-series array correspond to time-tagged indices, or observations, and the columns correspond to sample paths, independent realizations, or individual time series. In any given column, the first row contains the oldest observation and the last row contains the most recent observation. In this representation, a time-series array is said to be column-oriented.

Note Although some GARCH Toolbox functions can process univariate time-series arrays formatted as either row or column vectors, many functions now strictly enforce the column-oriented representation of a time series. Because of this and to avoid ambiguity, you should format single realizations of univariate time series as column vectors. Representing a time series in column-oriented format will avoid misinterpretation of the arguments, and will also make it easier for you to display data in the command window.

Conditional vs. Unconditional

The term *conditional* implies explicit dependence on a past sequence of observations. The term *unconditional* is more concerned with long-term behavior of a time series and assumes no explicit knowledge of the past.

Precision

The GARCH Toolbox performs all its calculations in double precision. Select **File -> Preferences -> Command Window -> Text display** to set the numeric format for your displays. The default is **short**.

Prices, Returns, and Compounding

The GARCH Toolbox assumes that time-series vectors and matrices are time-tagged series of observations. If you have a price series, the toolbox lets you convert it to a return series using either continuous compounding or simple periodic compounding.

If you denote successive price observations made at times t and $t + 1$ as P_t and P_{t+1} , respectively, continuous compounding transforms a price series $\{P_t\}$ into a return series $\{y_t\}$ as

$$y_t = \log \frac{P_{t+1}}{P_t} = \log P_{t+1} - \log P_t \quad (1-1)$$

Simple periodic compounding defines the transformation as

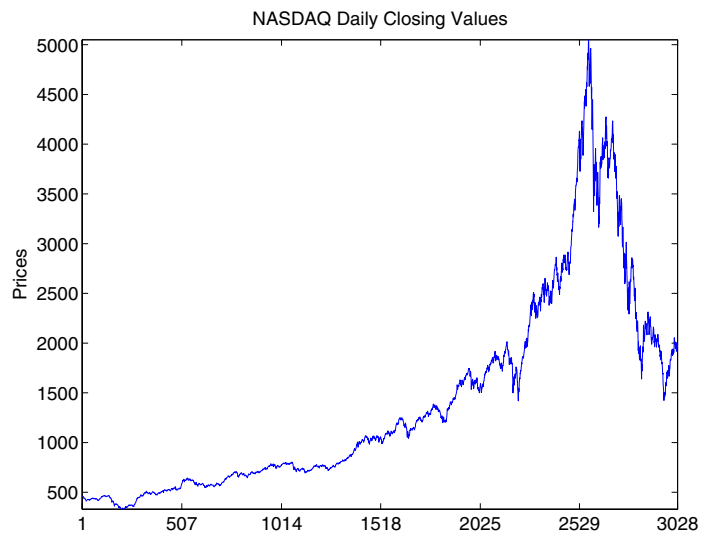
$$y_t = \frac{P_{t+1} - P_t}{P_t} = \frac{P_{t+1}}{P_t} - 1 \quad (1-2)$$

Continuous compounding is the default compounding method of the GARCH Toolbox, and is the preferred method for most of continuous-time finance. Since GARCH modeling is typically based on relatively high frequency data (i.e., daily or weekly observations), the difference between the two methods is usually small. However, there are some toolbox functions whose results are approximations for simple periodic compounding, but exact for continuous compounding. If you adopt the continuous compounding default convention when moving between prices and returns, all toolbox functions produce exact results.

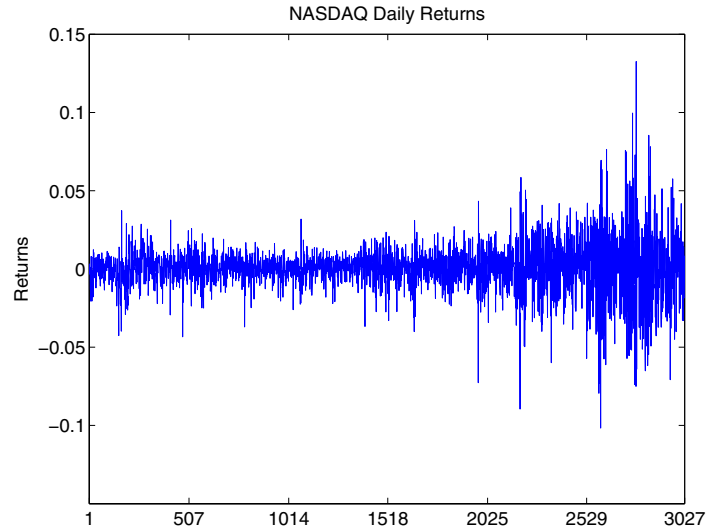
Stationary and Nonstationary Time Series

The GARCH Toolbox assumes that return series are stationary processes. The price-to-return transformation generally guarantees a stable data set for GARCH modeling.

This figure illustrates an equity price series. In this case, it shows daily closing values of the Nasdaq™ Composite Index (see “Data Sets” on page 1-11). Notice that there appears to be no long-run average level about which the series evolves. This is evidence of a nonstationary time series.



The following figure, however, illustrates the continuously compounded returns associated with the same price series. In contrast, the returns appear to be quite stable over time, and the transformation from prices to returns has produced a stationary time series.



The GARCH Toolbox assumes that return series are stationary processes. This may seem limiting, but the price-to-return transformation is common and generally guarantees a stable data set for GARCH modeling.

Data Sets

The GARCH Toolbox documentation uses the following financial time series. You can find them in the MAT-file `garchdata.mat`.

- “DEM2GBP” on page 1-11
- “NASDAQ” on page 1-12
- “NYSE” on page 1-12

DEM2GBP

The DEM2GBP series contains daily observations of the Deutschmark/British Pound foreign exchange rate, i.e., an FX price series. The sample period is from January 2, 1984, to December 31, 1991, for a total of 1975 daily observations of FX exchange rates.

The DEM2GBP price series is derived from the corresponding daily percentage nominal returns for the Deutschemark/British Pound exchange rate computed as

$$y_t = 100\ln(P_{t+1}/P_t) = 100[\ln(P_{t+1}) - \ln(P_t)]$$

where P_t is the bilateral Deutschmark/British Pound FX rate constructed from the corresponding U.S. dollar rates. The original nominal returns, expressed in percent, were originally published in Bollerslev and Ghysels [7].

You can also obtain the percentage returns data from the *Journal of Business and Economic Statistics* (JBES) FTP site, ftp://www.amstat.org/JBES_View/96-2-APR/bollerslev_ghysels/bollerslev.sec41.dat.

The sample period discussed in the Bollerslev and Ghysels article is from January 3, 1984, to December 31, 1991, for a total of 1974 observations of daily percentage nominal returns. These returns, combined with an approximate closing exchange rate from January 2, 1984, obtained from OANDA.com, The Currency Site™ (<http://www.oanda.com>), allow an approximate reconstruction of the corresponding FX closing price series.

This particular FX price series is included in the GARCH Toolbox documentation because it has been promoted as an informal benchmark for GARCH time-series software validation. See McCullough & Renfro [22], and Brooks, Burke, & Persaud [9] for details. Note that the estimation results

published in these references are based on the original percentage returns. The GARCH Toolbox presents the data as a price series merely to maintain consistency with the other two datasets highlighted throughout this manual.

NASDAQ

The NASDAQ series contains daily closing values of the Nasdaq™ Composite Index. The sample period is from January 2, 1990, to December 31, 2001, for a total of 3028 daily equity index observations.

The Nasdaq Composite closing index values were downloaded directly from the Market Data section of the Nasdaq™ web page.

NYSE

The NYSE series contains daily closing values of the New York Stock Exchange™ Composite Index. The sample period is from January 2, 1990, to December 31, 2001, for a total of 3028 daily equity index observations of the NYSE Composite Index.

The NYSE Composite Index daily closing values were downloaded directly from the Market Information section of the NYSE™ web page.

GARCH Overview

Modeling of Financial Time Series (p. 2-2)	Discusses some general concepts related to the modeling of financial time series.
Conditional Mean and Variance Models (p. 2-6)	Introduces the models you can use to describe conditional mean and variance to the GARCH Toolbox.
The Default Model (p. 2-13)	Describes the GARCH Toolbox default conditional mean and variance models.
Primary Toolbox Functions (p. 2-14)	Introduces the core functions you use to perform estimation, simulation, and forecasting.
Analysis and Estimation Example Using the Default Model (p. 2-16)	Uses the default model to examine the Deutschmark/British Pound foreign exchange rate series.

Modeling of Financial Time Series

This section discusses

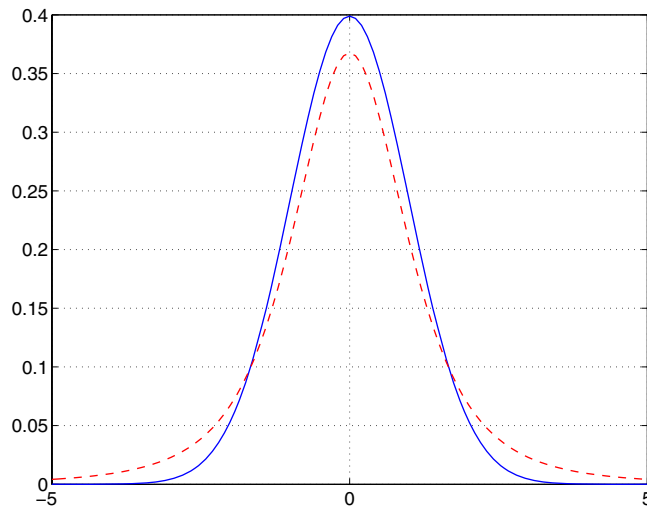
- “Characteristics of Financial Time Series” on page 2-2
- “Correlation and Forecasting of Financial Time Series” on page 2-4
- “Serial Dependence in Innovations” on page 2-5

Characteristics of Financial Time Series

GARCH models are designed to capture certain characteristics that are commonly associated with financial time series:

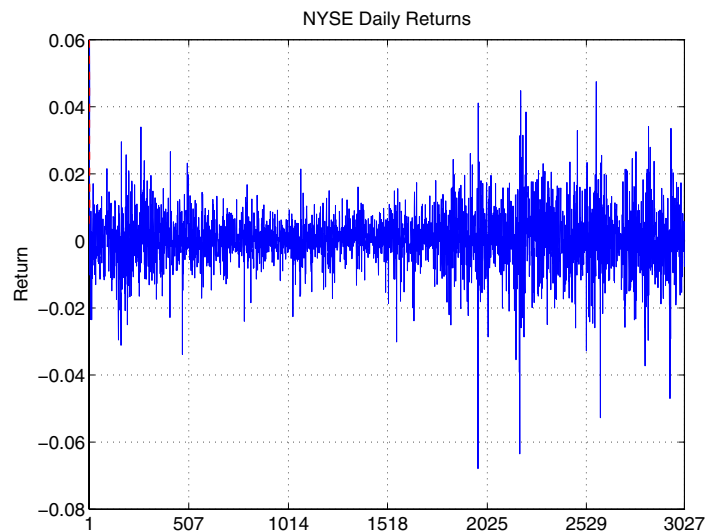
- Fat tails
- Volatility clustering
- Leverage effects

Probability distributions for asset returns often exhibit fatter tails than the standard normal, or Gaussian, distribution. The fat tail phenomenon is known as excess kurtosis. Time series that exhibit a fat tail distribution are often referred to as leptokurtic. The red (dashed) line in the following figure illustrates excess kurtosis. The blue (solid) line shows a Gaussian distribution.



In addition, financial time series usually exhibit a characteristic known as volatility clustering, in which large changes tend to follow large changes, and small changes tend to follow small changes. In either case, the changes from one period to the next are typically of unpredictable sign. Large disturbances, positive or negative, become part of the information set used to construct the variance forecast of the next period's disturbance. In this manner, large shocks of either sign are allowed to persist, and can influence the volatility forecasts for several periods.

Volatility clustering, or persistence, suggests a time-series model in which successive disturbances, although uncorrelated, are nonetheless serially dependent. The following figure illustrates this characteristic. It shows the daily returns of the New York Stock Exchange™ Composite Index (see “Data Sets” on page 1-11).



Volatility clustering (a type of heteroscedasticity) accounts for some but not all of the fat tail effect (or excess kurtosis) typically observed in financial data. A part of the fat tail effect can also result from the presence of non-Gaussian asset return distributions that just happen to have fat tails, such as Student's t .

Finally, certain classes of asymmetric GARCH models are also capable of capturing the so-called leverage effect, in which asset returns are often observed to be negatively correlated with changes in volatility. That is, for certain asset classes, most notably equities but excluding foreign exchange, volatility tends to rise in response to lower than expected returns and to fall in response to higher than expected returns. Such an effect suggests GARCH models that include an asymmetric response to positive and negative surprises.

Correlation and Forecasting of Financial Time Series

If you treat a financial time series as a sequence of random observations, this random sequence, or stochastic process, may exhibit some degree of correlation from one observation to the next. You can use this correlation structure to predict future values of the process based on the past history of observations. Exploiting the correlation structure, if any, allows you to decompose the time series into a deterministic component (i.e., the forecast), and a random component (i.e., the error, or uncertainty, associated with the forecast).

The following equation uses these components to represent a univariate model of an observed time series y_t .

$$y_t = f(t - 1, X) + \varepsilon_t$$

In this equation,

- $f(t - 1, X)$ represents the forecast, or deterministic component, of the current return as a function of any information known at time $t - 1$, including past innovations $\{\varepsilon_{t-1}, \varepsilon_{t-2}, \dots\}$, past observations $\{y_{t-1}, y_{t-2}, \dots\}$, and any other relevant explanatory time-series data, X .
- ε_t is the random component. It represents the innovation in the mean of y_t . Note that you can also interpret the random disturbance, or shock, ε_t , as the single-period-ahead forecast error.

Serial Dependence in Innovations

A common assumption when modeling financial time series is that the forecast errors (i.e., the innovations) are zero-mean random disturbances uncorrelated from one period to the next.

$$E\{\varepsilon_t\} = 0$$

$$E\{\varepsilon_t\varepsilon_T\} = 0 \quad t \neq T$$

Although successive innovations are uncorrelated, they are not independent. In fact, an explicit generating mechanism for a GARCH innovations process, $\{\varepsilon_t\}$, is

$$\varepsilon_t = \sigma_t z_t \tag{2-1}$$

where σ_t is the conditional standard deviation derived from one of the conditional variance equations shown in “Conditional Variance Models” on page 2-7.

z_t is a standardized, independent, identically distributed (i.i.d.) random draw from some specified probability distribution. The GARCH Toolbox provides two distributions for modeling GARCH processes: Gaussian and Student's t . Eq. (2-1) illustrates that a GARCH innovations process $\{\varepsilon_t\}$ simply rescales an i.i.d process $\{z_t\}$ such that the conditional standard deviation incorporates the serial dependence of the conditional variance equation. Equivalently, Eq. (2-1) also states that a standardized GARCH disturbance, ε_t/σ_t , is itself an i.i.d. random variable z_t .

Notice that GARCH models are consistent with various forms of efficient market theory, which state that asset returns observed in the past cannot improve the forecasts of asset returns in the future. Since GARCH innovations $\{\varepsilon_t\}$ are serially uncorrelated, GARCH modeling does not violate efficient market theory.

Conditional Mean and Variance Models

This section describes the conditional mean and variance models that the GARCH Toolbox supports and offers some comments to help clarify their descriptions.

- “Conditional Mean Model” on page 2-6
- “Conditional Variance Models” on page 2-7
- “Comments on the Models” on page 2-10

Conditional Mean Model

This general ARMAX(R,M,Nx) model for the conditional mean applies to all variance models.

$$y_t = C + \sum_{i=1}^R \phi_i y_{t-i} + \varepsilon_t + \sum_{j=1}^M \theta_j \varepsilon_{t-j} + \sum_{k=1}^{Nx} \beta_k X(t, k) \quad (2-2)$$

with autoregressive coefficients $\{\phi_i\}$, moving average coefficients $\{\theta_j\}$, innovations $\{\varepsilon_t\}$, and returns $\{y_t\}$. X is an explanatory regression matrix in which each column is a time series and $X(t, k)$ denotes the t th row and k th column.

The eigenvalues $\{\lambda_i\}$ associated with the characteristic AR polynomial

$$\lambda^R - \phi_1 \lambda^{R-1} - \phi_2 \lambda^{R-2} - \dots - \phi_R$$

must lie inside the unit circle to ensure stationarity. Similarly, the eigenvalues associated with the characteristic MA polynomial

$$\lambda^M + \theta_1 \lambda^{M-1} + \theta_2 \lambda^{M-2} + \dots + \theta_M$$

must lie inside the unit circle to ensure invertibility.

Conditional Variance Models

The conditional variance of the innovations, σ_t^2 , is by definition

$$\text{Var}_{t-1}(y_t) = E_{t-1}(\varepsilon_t^2) = \sigma_t^2 \quad (2-3)$$

The key insight of GARCH lies in the distinction between conditional and unconditional variances of the innovations process $\{\varepsilon_t\}$. The term *conditional* implies explicit dependence on a past sequence of observations. The term *unconditional* is more concerned with long-term behavior of a time series and assumes no explicit knowledge of the past.

The various GARCH models characterize the conditional distribution of ε_t by imposing alternative parameterizations to capture serial dependence on the conditional variance of the innovations. “Comments on the Models” on page 2-10 further defines the conditional variance models.

GARCH(P,Q) Conditional Variance

The general GARCH(P,Q) model for the conditional variance of innovations is

$$\sigma_t^2 = \kappa + \sum_{i=1}^P G_i \sigma_{t-i}^2 + \sum_{j=1}^Q A_j \varepsilon_{t-j}^2 \quad (2-4)$$

with constraints

$$\begin{aligned} \sum_{i=1}^P G_i + \sum_{j=1}^Q A_j &< 1 \\ \kappa &> 0 \\ G_i &\geq 0 \quad i = 1, 2, \dots, P \\ A_j &\geq 0 \quad j = 1, 2, \dots, Q \end{aligned}$$

Note that the basic GARCH(P,Q) model is a symmetric variance process, in that the sign of the disturbance is ignored.

GJR(P,Q) Conditional Variance

The general GJR(P,Q) model for the conditional variance of the innovations with leverage terms is

$$\sigma_t^2 = \kappa + \sum_{i=1}^P G_i \sigma_{t-1}^2 + \sum_{j=1}^Q A_j \varepsilon_{t-j}^2 + \sum_{j=1}^Q L_j S_{t-j}^- \varepsilon_{t-j}^2 \quad (2-5)$$

where

$$S_{t-j}^- = \begin{cases} 1 & \varepsilon_{t-j} < 0 \\ 0 & \text{otherwise} \end{cases}$$

and

$$\sum_{i=1}^P G_i + \sum_{j=1}^Q A_j + \frac{1}{2} \sum_{j=1}^Q L_j < 1$$

$$\begin{aligned} \kappa &> 0 \\ G_i &\geq 0 && i = 1, 2, \dots, P \\ A_j &\geq 0 && j = 1, 2, \dots, Q \\ A_j + L_j &\geq 0 && j = 1, 2, \dots, Q \end{aligned}$$

EGARCH(P,Q) Conditional Variance

The general EGARCH(P,Q) model for the conditional variance of the innovations with leverage terms and an explicit probability distribution assumption is

$$\log \sigma_t^2 = \kappa + \sum_{i=1}^P G_i \log \sigma_{t-i}^2 + \sum_{j=1}^Q A_j \left[\frac{|\varepsilon_{t-j}|}{\sigma_{t-j}} - E \left\{ \frac{|\varepsilon_{t-j}|}{\sigma_{t-j}} \right\} \right] + \sum_{j=1}^Q L_j \left(\frac{\varepsilon_{t-j}}{\sigma_{t-j}} \right) \quad (2-6)$$

where

$$E \{ |z_{t-j}| \} = E \left(\frac{|\varepsilon_{t-j}|}{\sigma_{t-j}} \right) = \begin{cases} \sqrt{2/\pi} & \text{Gaussian} \\ \sqrt{\frac{\nu-2}{\pi}} \cdot \frac{\Gamma(\frac{\nu-1}{2})}{\Gamma(\frac{\nu}{2})} & \text{Student's } t \end{cases}$$

with degrees of freedom $\nu > 2$.

EGARCH(P,Q) models are treated as ARMA(P,Q) models for $\log \sigma_t^2$. Thus, the stationarity constraint for EGARCH(P,Q) models is included by ensuring that the eigenvalues of the characteristic polynomial

$$\lambda^P - G_1 \lambda^{P-1} - G_2 \lambda^{P-2} - \dots - G_P$$

are inside the unit circle.

Note that EGARCH models are fundamentally different from GARCH and GJR models in that the standardized innovation, z_t , serves as the forcing variable for both the conditional variance and the error. GARCH and GJR models allow for volatility clustering (i.e., persistence) by a combination of the G_i and A_j terms, whereas persistence in EGARCH models is entirely captured by the G_i terms.

Comments on the Models

The econometrics literature is often vague and lacks consensus regarding the exact definition of any particular class of GARCH model. As a result, there are often discrepancies among software vendors, researchers, and references as to the exact functional form, or parameter constraints, or both, of almost all GARCH models. To help you reconcile some of these discrepancies, a few comments are useful:

- Although the functional form of a GARCH(P,Q) model (Eq. (2-4)) is quite standard, alternative positivity constraints exist. However, these alternatives involve additional nonlinear inequalities that are difficult to impose in practice, and do not affect the GARCH(1,1) model, which is by far the most common. In contrast, the standard linear positivity constraints imposed by the GARCH Toolbox are commonly used, and are straightforward to implement.
- Many references and software vendors refer to the GJR(P,Q) model (Eq. (2-5)) as a TGARCH, or Threshold GARCH, model. However, others make a very clear distinction between GJR(P,Q) and TGARCH(P,Q) models: a GJR(P,Q) model is a recursive equation for the conditional variance, whereas a TGARCH(P,Q) model is the identical recursive equation for the conditional standard deviation (see, for example, Hamilton [19] page 669, Bollerslev, et. al. [6] page 2970). Furthermore, additional discrepancies exist regarding whether or not to allow both negative and positive innovations to affect the conditional variance process. The GJR(P,Q) model included in the GARCH Toolbox is relatively standard.
- The manner in which the GARCH Toolbox parameterizes GARCH(P,Q) and GJR(P,Q) models, Eq. (2-4) and Eq. (2-5), including constraints, allows you to interpret a GJR(P,Q) model as a straightforward extension of a GARCH(P,Q) model. Equivalently, you can interpret a GARCH(P,Q) model as a restricted GJR(P,Q) model with zero leverage terms. This latter interpretation is convenient for, among other things, estimation and hypothesis testing via likelihood ratios.
- For GARCH(P,Q) and GJR(P,Q) models, the lag lengths P and Q , as well as the magnitudes of the coefficients G_i and A_j , determine the extent to which disturbances persist. These values then determine the minimum amount of presample data needed to initiate the simulation and estimation processes. Note that persistence in EGARCH models is entirely captured by the G_i terms.

- Although the functional form of an EGARCH(P,Q) model (Eq. (2-6)) is relatively standard, it is not the same as Nelson's original (see Nelson [23]). Many forms of EGARCH(P,Q) models exist. Another popular form is

$$\log \sigma_t^2 = \kappa + \sum_{i=1}^P G_i \log \sigma_{t-i}^2 + \sum_{j=1}^Q A_j \left[\frac{|\varepsilon_{t-j}| + L_j \varepsilon_{t-j}}{\sigma_{t-j}} \right]$$

This EGARCH(P,Q) model form appears to offer an advantage in that it does not explicitly make any assumptions about the conditional probability distribution (i.e., whether the distribution of $z_t = (\varepsilon_t/\sigma_t)$ is Gaussian or Student's t). However, this is not entirely true. Although no distribution is explicitly assumed in the above equation, generally such an assumption is required for forecasting as well as Monte Carlo simulation in the absence of user-specified presample data. In fact, the above equation can easily be rearranged to highlight the probability distribution.

The particular form of the EGARCH(P,Q) model, Eq. (2-6), implemented in the GARCH Toolbox is selected because it closely resembles Nelson's original form and is widely used.

- Although EGARCH(P,Q) models require no parameter constraints to ensure positive conditional variances, stationarity constraints are necessary. Since an EGARCH(P,Q) model is treated as an ARMA(P,Q) model for the logarithm of the conditional variance, the GARCH Toolbox imposes non-linear constraints on the G_i coefficients to ensure that the eigenvalues of the characteristic polynomial are all inside the unit circle (see, for example, page 2969 of Bollerslev, Engle, and Nelson [6], and page 12 of Bollerslev, Chou, and Kroner [5]).
- The EGARCH(P,Q) and GJR(P,Q) models, Eq. (2-6) and Eq. (2-5), are both asymmetric models designed to capture the leverage effect, or negative correlation, between asset returns and volatility. Both the EGARCH(P,Q) and GJR(P,Q) models include leverage terms that explicitly take into account the sign as well as the magnitude of the innovation noise term. Although both models are designed to capture the leverage effect, the manner in which they do so is markedly different.

For EGARCH(P,Q) models, the leverage coefficients L_i are applied to the actual innovations ε_{t-i} . For GJR(P,Q) models, the leverage coefficients enter the model through a Boolean indicator, or dummy, variable. For this

reason, if the leverage effect does indeed hold, the leverage coefficients L_i should be negative for EGARCH(P,Q) models and positive for GJR(P,Q) models. This is in contrast to GARCH(P,Q) models, in which the sign of the innovation is ignored.

- Although GARCH(P,Q) and GJR(P,Q) models, Eq. (2-4) and Eq. (2-5), include terms related to the model innovations, $\varepsilon_t = z_t \sigma_t$, EGARCH(P,Q) models, Eq. (2-6), include terms related to the standardized innovations, $z_t = \varepsilon_t / \sigma_t$, such that z_t acts as the forcing variable for both the conditional variance and the error. In this respect, EGARCH(P,Q) models are fundamentally unique.
- Generally, there are no asymmetries in foreign exchange rates, and therefore asymmetric EGARCH(P,Q) and GJR(P,Q) conditional variance models are often inappropriate for modeling such return series.

The Default Model

The GARCH Toolbox default model is the simple (yet common) constant mean model with GARCH(1,1) Gaussian innovations, based on Eq. (2-2) and Eq. (2-4).

$$y_t = C + \varepsilon_t \quad (2-7)$$

$$\sigma_t^2 = \kappa + G_1 \sigma_{t-1}^2 + A_1 \varepsilon_{t-1}^2 \quad (2-8)$$

In the conditional mean model, Eq. (2-7), the returns, y_t , consist of a simple constant, plus an uncorrelated, white noise disturbance, ε_t . This model is often sufficient to describe the conditional mean in a financial return series. Most financial return series do not require the comprehensiveness that an ARMAX model provides.

In the conditional variance model, Eq. (2-8), the variance forecast, σ_t^2 , consists of a constant plus a weighted average of last period's forecast, σ_{t-1}^2 , and last period's squared disturbance, ε_{t-1}^2 . Although financial return series, as defined in Eq. (1-1) and Eq. (1-2), typically exhibit little correlation, the squared returns often indicate significant correlation and persistence. This implies correlation in the variance process, and is an indication that the data is a candidate for GARCH modeling.

Although simplistic, the default model shown in Eq. (2-7) and Eq. (2-8) has several benefits:

- It represents a parsimonious model that requires you to estimate only four parameters (C , κ , G_1 , and A_1). According to Box and Jenkins [8], the fewer parameters to estimate, the less that can go wrong. Elaborate models often fail to offer real benefits when forecasting (see Hamilton [19], page 109).
- The simple GARCH(1,1) model captures most of the variability in most return series. Small lags for P and Q are common in empirical applications. Typically, GARCH(1,1), GARCH(2,1), or GARCH(1,2) models are adequate for modeling volatilities even over long sample periods (see Bollerslev, Chou, and Kroner [5], pages 10 and 22).

Primary Toolbox Functions

Use of the GARCH Toolbox focuses on three high-level processing functions: `garchfit`, `garchpred`, and `garchsim`, for model estimation, forecasting, and Monte Carlo simulation, respectively. A fourth function, `garchinfer`, infers the innovations and conditional standard deviations via inverse filtering, and is closely related to `garchfit` in that they both call the appropriate objective function.

These functions use a *GARCH specification structure* to share information about the specified model. The specification structure contains the model orders for the chosen conditional mean and variance models, and the parameters for those models. All these functions accept a specification structure as input, but only `garchfit` can update the structure and provide it as an output. (See “GARCH Specification Structure” on page 3-1 for detailed information about the structure.)

An analysis process using real-world data might involve calling these processing functions:

`garchfit` Estimates the model parameters. `garchfit` can accept a specification structure as an input. If you provide only the model orders for the chosen conditional mean and variance model, `garchfit` populates it with the coefficients resulting from the estimation process. If you provide, in addition, valid coefficients, `garchfit` uses them as initial estimates that are refined by the estimation process. If you provide no specification structure, `garchfit` assumes the default model (see “The Default Model” on page 2-13).

In all cases, `garchfit` returns an updated specification structure, which encapsulates parameter estimates. This output structure is of the same form as the input structure, and you can use it as an input for further modeling.

- `garchpred` Forecasts returns and conditional standard deviations. It accepts as input the specification structure provided by the estimation engine `garchfit`. You can also use `garchpred` to forecast volatility of asset returns over multiperiod holding intervals, or to forecast the standard errors of conditional mean forecasts.
- `garchsim` Simulates one or more sample paths for the return series, innovations, and conditional standard deviation processes, for the specified conditional mean and variance model. You can use these sample paths to perform Monte Carlo simulation of a given process.

Analysis and Estimation Example Using the Default Model

The example in this section uses the GARCH Toolbox default model to model a foreign exchange series. Specifically, the example explores

- “Prestimation Analysis” on page 2-16
- “Parameter Estimation” on page 2-25
- “Postestimation Analysis” on page 2-28

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

For more information see “Model Selection and Analysis” on page 9-1.

Prestimation Analysis

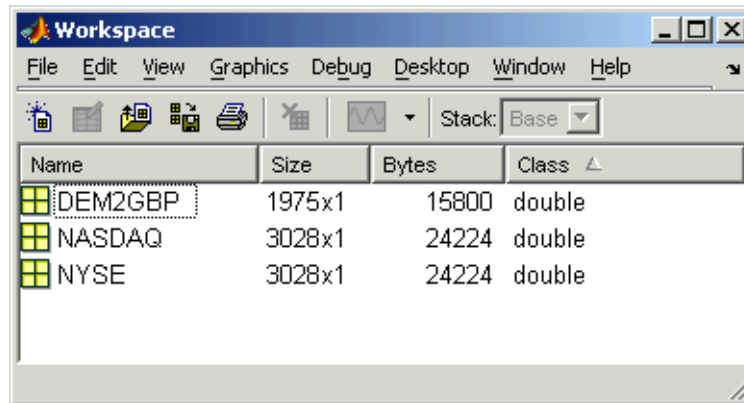
When estimating the parameters of a composite conditional mean/variance model, you may occasionally encounter convergence problems. For example, the estimation may appear to stall, showing little or no progress. It may terminate prematurely prior to convergence. Or, it may converge to an unexpected, suboptimal solution.

You can avoid many of these difficulties by performing a prefit analysis. This section uses an example to show techniques such as plotting the return series, and examining the ACF and PACF, as well as some preliminary tests, including Engle’s ARCH test and the Q-test. The goal is to avoid convergence problems by selecting the simplest model that adequately describes your data.

The following preestimation analysis example loads the data in the form of a price series, then converts the price series to a return series. It checks the return series for correlation, and then quantifies the correlation.

- 1 Load the raw data: daily exchange rate.** Start by loading the MATLAB binary file `garchdata.mat`, and examining its contents using the Workspace Browser.

```
load garchdata
```



The data consists of three single-column vectors of different lengths, DEM2GBP, NASDAQ, and NYSE. Each vector is a separate price series for the named group. (See “Data Sets” on page 1-11 for more information about these data sets.) You can also use the `whos` command to see all the variables in the current workspace.

```
whos
```

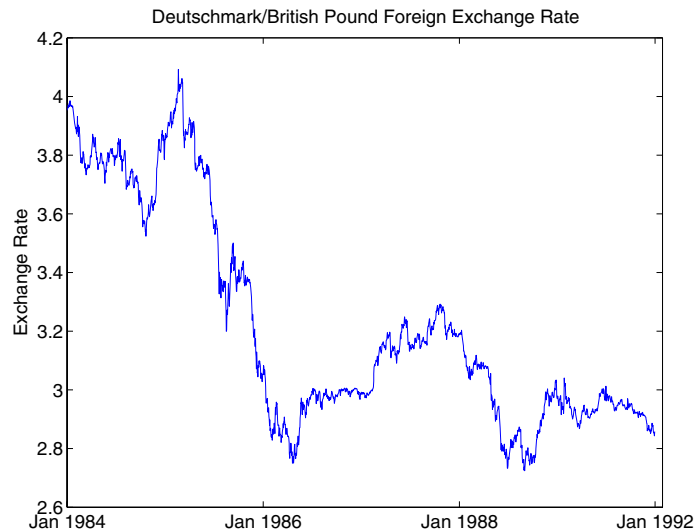
Name	Size	Bytes	Class
DEM2GBP	1975x1	15800	double array
NASDAQ	3028x1	24224	double array
NYSE	3028x1	24224	double array

```
Grand total is 8031 elements using 64248 bytes
```

This example uses DEM2GBP, which contains daily price observations of the Deutschmark/British Pound foreign exchange rate. Use the MATLAB `plot` function to examine the data.

```
plot([0:1974],DEM2GBP)
set(gca,'XTick',[1 659 1318 1975])
set(gca,'XTickLabel',{'Jan 1984' 'Jan 1986' 'Jan 1988' ...
    'Jan 1992'})
ylabel('Exchange Rate')
title('Deutschmark/British Pound Foreign Exchange Rate')
```

Note The `set` command allows you to set object properties. This example uses it to set the position of and relabel the x -axis ticks of the current figure.



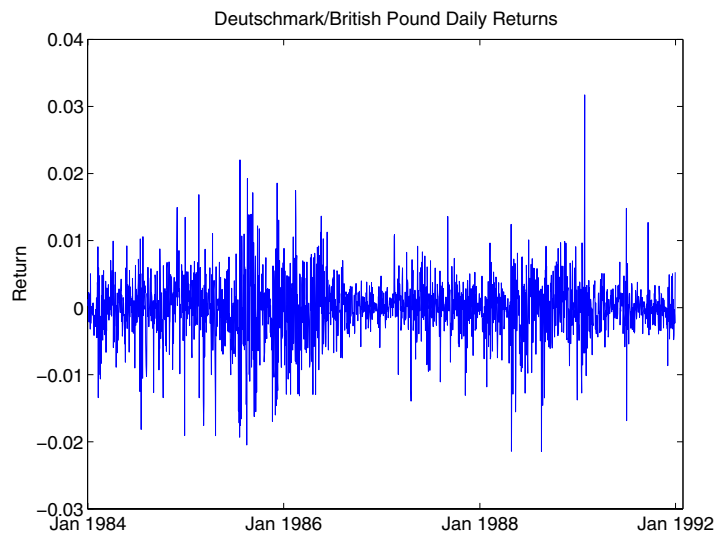
2 Convert the prices to a return series. Because GARCH modeling assumes a return series, you need to convert the prices to returns. Use the utility function `price2ret`, and then examine the result.

```
dem2gbp = price2ret(DEM2GBP);
```

The workspace information shows both the 1975-point price series and the 1974-point return series derived from it.

Now, use the MATLAB `plot` function to see the return series. Notice the presence of volatility clustering in the raw return series.

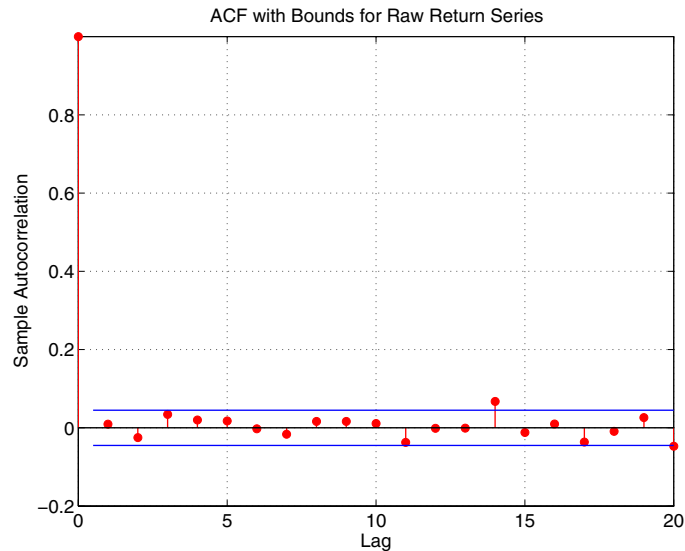
```
plot(dem2gbp)
set(gca,'XTick',[1 659 1318 1975])
set(gca,'XTickLabel',{'Jan 1984' 'Jan 1986' 'Jan 1988' ...
    'Jan 1992'})
ylabel('Return')
title('Deutschmark/British Pound Daily Returns')
```



3 Check for correlation in the return series. You can check qualitatively for correlation in the raw return series by calling the functions `autocorr` and `parcorr` to examine the sample autocorrelation function (ACF) and partial-autocorrelation (PACF) function, respectively.

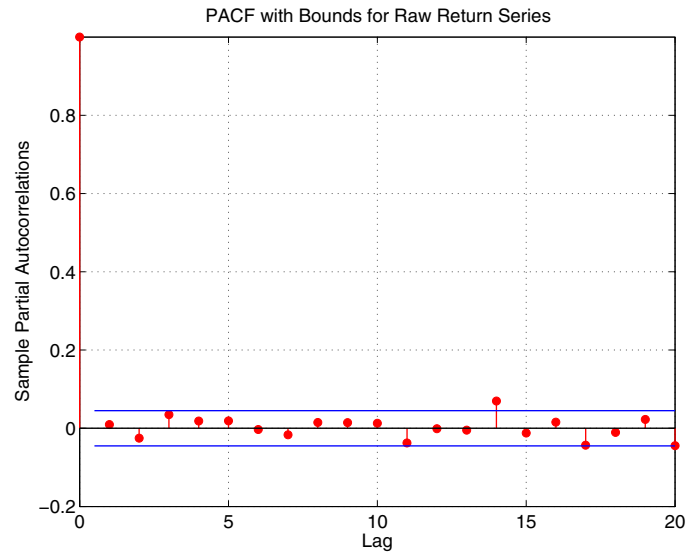
The `autocorr` function computes and displays the sample ACF of the returns, along with the upper and lower standard deviation confidence bounds, based on the assumption that all autocorrelations are zero beyond lag zero.

```
autocorr(dem2gbp)  
title('ACF with Bounds for Raw Return Series')
```



Similarly, the `parcorr` function displays the sample PACF with upper and lower confidence bounds.

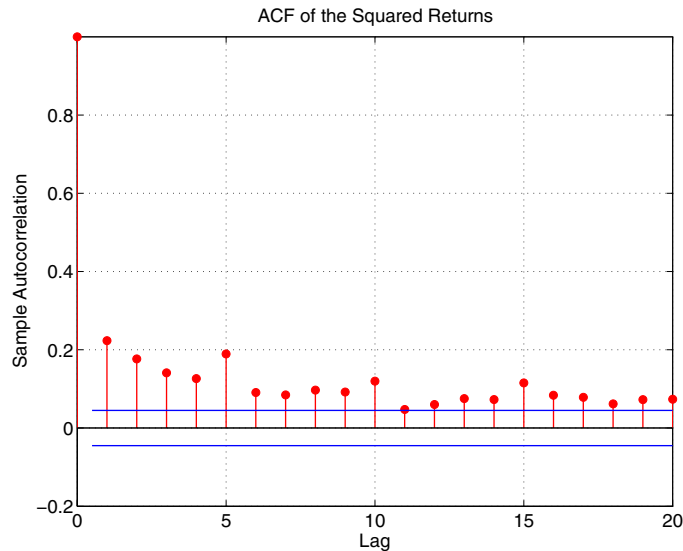
```
parcorr(dem2gbp)  
title('PACF with Bounds for Raw Return Series')
```



Since the individual ACF values can have large variances and can also be autocorrelated, you should view the sample ACF and PACF with care (see Box, Jenkins, Reinsel [8], pages 34 and 186). However, as preliminary identification tools, the ACF and PACF provide some indication of the broad correlation characteristics of the returns. From these figures for the ACF and PACF, there is very little indication that you need to use any correlation structure in the conditional mean. Also, notice the similarity between the graphs.

- 4 Check for correlation in the squared returns.** Although the ACF of the observed returns exhibits little correlation, the ACF of the squared returns may still indicate significant correlation and persistence in the second-order moments. Check this by plotting the ACF of the squared returns.

```
autocorr(dem2gbp.^2)  
title('ACF of the Squared Returns')
```



This figure shows that, although the returns themselves are largely uncorrelated, the variance process exhibits some correlation. This is consistent with the earlier discussion in the section, “The Default Model” on page 2-13. Note that the ACF shown in this figure appears to die out slowly, indicating the possibility of a variance process close to being nonstationary.

Note The syntax in the preceding command, an operator preceded by the dot operator (`.`), indicates that the operation is performed on an element-by-element basis. In the preceding command, `dem2gbp.^2` indicates that each element of the vector `dem2gbp` is squared.

5 Quantify the correlation. You can quantify the preceding qualitative checks for correlation using formal hypothesis tests, such as the Ljung-Box-Pierce Q-test and Engle's ARCH test.

The function `lbqtest` implements the Ljung-Box-Pierce Q-test for a departure from randomness based on the ACF of the data. The Q-test is most often used as a postestimation lack-of-fit test applied to the fitted innovations (i.e., residuals). In this case, however, you can also use it as part of the prefit analysis because the default model assumes that returns are just a simple constant plus a pure innovations process. Under the null hypothesis of no serial correlation, the Q-test statistic is asymptotically Chi-Square distributed (see Box, Jenkins, Reinsel [8], page 314).

The function `archtest` implements Engle's test for the presence of ARCH effects. Under the null hypothesis that a time series is a random sequence of Gaussian disturbances (i.e., no ARCH effects exist), this test statistic is also asymptotically Chi-Square distributed (see Engle [12], pages 999-1000).

Both functions return identical outputs. The first output, `H`, is a Boolean decision flag. `H = 0` implies that no significant correlation exists (i.e., do not reject the null hypothesis). `H = 1` means that significant correlation exists (i.e., reject the null hypothesis). The remaining outputs are the P-value (`pValue`), the test statistic (`Stat`), and the critical value of the Chi-Square distribution (`CriticalValue`).

Ljung-Box-Pierce Q-Test. Using `lbqtest`, you can verify, at least approximately, that no significant correlation is present in the raw returns when tested for up to 10, 15, and 20 lags of the ACF at the 0.05 level of significance.

```
[H,pValue,Stat,CriticalValue] = ...
    lbqtest(dem2gbp-mean(dem2gbp),[10 15 20]',0.05);
[H pValue Stat CriticalValue]
```

```
ans =  
      0    0.7278    6.9747   18.3070  
      0    0.2109   19.0628   24.9958  
      0    0.1131   27.8445   31.4104
```

However, there is significant serial correlation in the squared returns when you test them with the same inputs.

```
[H,pValue,Stat,CriticalValue] = ...  
    lbqtest((dem2gbp-mean(dem2gbp)).^2,[10 15 20]',0.05);  
[H pValue Stat CriticalValue]
```

```
ans =  
  1.0000         0  392.9790   18.3070  
  1.0000         0  452.8923   24.9958  
  1.0000         0  507.5858   31.4104
```

Engle's ARCH Test. You can also perform Engle's ARCH test using the function `archtest`. This test also shows significant evidence in support of GARCH effects (i.e., heteroscedasticity).

```
[H,pValue,Stat,CriticalValue] = ...  
    archtest(dem2gbp-mean(dem2gbp),[10 15 20]',0.05);  
[H pValue Stat CriticalValue]
```

```
ans =  
  1.0000         0  192.3783   18.3070  
  1.0000         0  201.4652   24.9958  
  1.0000         0  203.3018   31.4104
```

Each of these examples extracts the sample mean from the actual returns. This is consistent with the definition of the conditional mean equation of the default model, in which the innovations process is $\varepsilon_t = y_t - C$, and C is the mean of y_t .

Parameter Estimation

This section continues the example begun in “Preestimation Analysis” on page 2-16. It estimates model parameters, then examines the estimated GARCH model.

1 Estimate the Model Parameters. The presence of heteroscedasticity, shown in the previous analysis, indicates that GARCH modeling is appropriate. Use the estimation function `garchfit` to estimate the model parameters. Assume the default GARCH model described in the section “The Default Model” on page 2-13. This only requires that you specify the return series of interest as an argument to the function `garchfit`.

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

Note Because the default value of the `Display` parameter in the specification structure is `'on'`, `garchfit` prints diagnostic optimization and summary information to the command window in the example below. (See `fmincon` in the Optimization Toolbox for information about the output of the `Display` parameter.)

```
[coeff,errors,LLF,innovations,sigmas,summary] = ...
garchfit(dem2gbp);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Diagnostic Information

Number of variables: 4

Functions

```
Objective:          garchllfn
Gradient:           finite-differencing
Hessian:            finite-differencing (or Quasi-Newton)
Nonlinear constraints: armanlc
Gradient of nonlinear constraints: finite-differencing
```

Constraints

```
Number of nonlinear inequality constraints: 0
Number of nonlinear equality constraints: 0

Number of linear inequality constraints: 1
Number of linear equality constraints: 0
Number of lower bound constraints: 4
Number of upper bound constraints: 4
```

Algorithm selected

medium-scale

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

End diagnostic information

Iter	F-count	f(x)	max constraint	Step-size	Directional derivative	First-order optimality	Procedure
1	28	-7916.01	-2.01e-006	7.63e-006	857	1.42e+005	
2	36	-7959.65	-1.508e-006	0.25	389	9.8e+007	
3	45	-7963.98	-3.113e-006	0.125	131	5.29e+006	
4	52	-7965.59	-1.586e-006	0.5	55.9	4.45e+007	
5	65	-7966.9	-1.574e-006	0.00781	101	1.46e+007	
6	74	-7969.46	-2.201e-006	0.125	14.9	2.77e+007	
7	83	-7973.56	-2.663e-006	0.125	36.6	1.45e+007	
8	90	-7982.09	-1.332e-006	0.5	-6.39	5.59e+006	
9	103	-7982.13	-1.399e-006	0.00781	6.49	1.32e+006	
10	111	-7982.53	-1.049e-006	0.25	12.5	1.87e+007	
11	120	-7982.56	-1.186e-006	0.125	3.72	3.8e+006	
12	128	-7983.69	-1.11e-006	0.25	0.184	4.91e+006	
13	134	-7983.91	-7.813e-007	1	0.732	1.22e+006	
14	140	-7983.98	-9.265e-007	1	0.186	1.17e+006	
15	146	-7984	-8.723e-007	1	0.0427	9.52e+005	

```

16 154 -7984 -8.775e-007 0.25 0.0152 6.33e+005
17 160 -7984 -8.75e-007 1 0.00197 6.98e+005
18 166 -7984 -8.763e-007 1 0.000931 7.38e+005
19 173 -7984 -8.759e-007 0.5 0.000469 7.37e+005
20 179 -7984 -8.761e-007 1 0.00012 7.22e+005
21 199 -7984 -8.761e-007 -6.1e-005 0.0167 7.37e+005 Hessian modified twice
22 213 -7984 -8.761e-007 0.00391 0.00582 7.26e+005 Hessian modified twice

```

Optimization terminated successfully:
Search direction less than 2*options.TolX and
maximum constraint violation is less than options.TolCon
No Active Constraints

2 Examine the Estimated GARCH Model. Now that the estimation is complete, you can display the parameter estimates and their standard errors using the function `garchdisp`,

```
garchdisp(coeff,errors)
```

```
Mean: ARMAX(0,0,0); Variance: GARCH(1,1)
```

```
Conditional Probability Distribution: Gaussian
```

```
Number of Parameters Estimated: 4
```

Parameter	Value	Standard Error	T Statistic
C	-6.1919e-005	8.4331e-005	-0.7342
K	1.0761e-006	1.323e-007	8.1341
GARCH(1)	0.80598	0.016561	48.6685
ARCH(1)	0.15313	0.013974	10.9586

If you substitute these estimates in the definition of the default model, Eq. (2-7) and Eq. (2-8), the estimation process implies that the constant conditional mean/GARCH(1,1) conditional variance model that best fits the observed data is

$$y_t = -6.1919e-005 + \varepsilon_t$$

$$\sigma_t^2 = 1.0761e-006 + 0.80598\sigma_{t-1}^2 + 0.15313\varepsilon_{t-1}^2$$

where $G_1 = \text{GARCH}(1) = 0.80598$ and $A_1 = \text{ARCH}(1) = 0.15313$. In addition, $C = C = -6.1919e-005$ and $K = K = 1.0761e-006$.

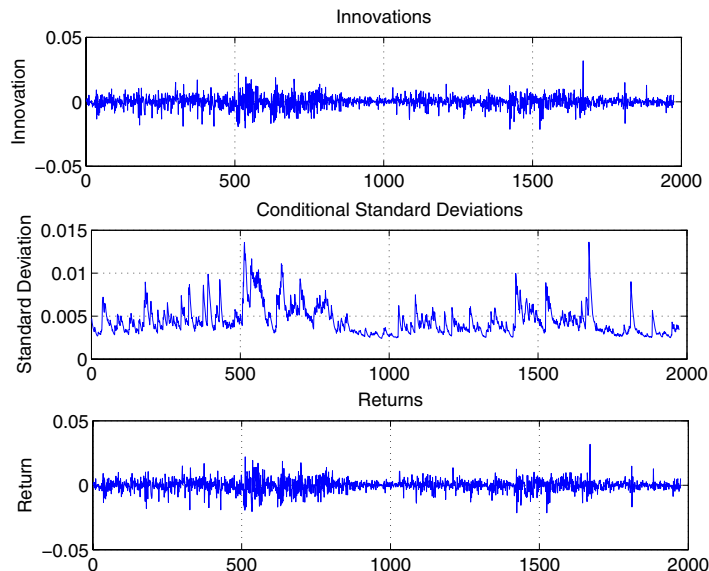
Postestimation Analysis

The postestimation analysis continues the example begun in “Preestimation Analysis” on page 2-16 and continued in “Parameter Estimation” on page 2-25. This part of the example starts by comparing the residuals, conditional standard deviations, and returns. It then uses plots and quantitative techniques to compare correlation of the standardized innovations.

1 Compare the Residuals, Conditional Standard Deviations, and Returns.

In addition to the parameter estimates and standard errors, `garchfit` also returns the optimized log-likelihood function value (LLF), the residuals (innovations), and conditional standard deviations (sigmas). Use the function `garchplot` to inspect the relationship between the innovations (i.e., residuals) derived from the fitted model, the corresponding conditional standard deviations, and the observed returns.

```
garchplot(innovations,sigmas,dem2gbp)
```



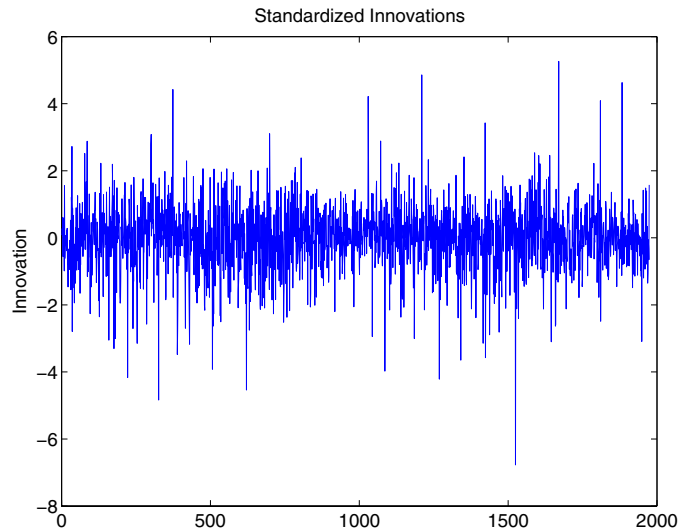
Notice that both the innovations (top plot) and the returns (bottom plot) exhibit volatility clustering. Also, notice that the sum,

$G_1 + A_1 = 0.80598 + 0.15313$, is 0.95911, which is close to the integrated, nonstationary boundary given by the constraints associated with Eq. (2-4).

2 Plot and Compare the Correlation of the Standardized Innovations.

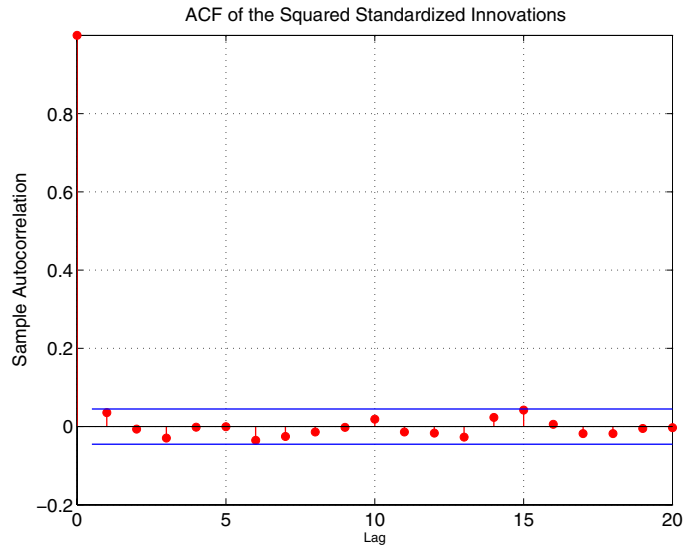
Although the figure in step 1 shows that the fitted innovations exhibit volatility clustering, if you plot the standardized innovations (the innovations divided by their conditional standard deviation), they appear generally stable with little clustering.

```
plot(innovations./sigmas)
ylabel('Innovation')
title('Standardized Innovations')
```



If you plot the ACF of the squared standardized innovations, they also show no correlation.

```
autocorr((innovations./sigmas).^2)  
title('ACF of the Squared Standardized Innovations')
```



Now compare the ACF of the squared standardized innovations in this figure to the ACF of the squared returns prior to fitting the default model (See “Prestimation Analysis” on page 2-16, step 4). The comparison shows that the default model sufficiently explains the heteroscedasticity in the raw returns.

3 Quantify and Compare Correlation of the Standardized Innovations.

Compare the results below of the Q-test and the ARCH test with the results of these same tests in the preestimation analysis. In the preestimation analysis, both the Q-test and the ARCH test indicate rejection ($H = 1$ with $pValue = 0$) of their respective null hypotheses, showing significant evidence in support of GARCH effects. In the postestimate analysis, using standardized innovations based on the estimated model, these same tests indicate acceptance ($H = 0$ with highly significant $pValues$) of their respective null hypotheses and confirm the explanatory power of the default model.

```
[H, pValue, Stat, CriticalValue] = ...
    lbqtest((innovations./sigmas).^2,[10 15 20]',0.05);
[H pValue Stat CriticalValue]
```

```
ans =
      0      0.5262      9.0626     18.3070
      0      0.3769     16.0777     24.9958
      0      0.6198     17.5072     31.4104
```

```
[H, pValue, Stat, CriticalValue] = ...
    archtest(innovations./sigmas,[10 15 20]',0.05);
[H pValue Stat CriticalValue]
```

```
ans =
      0      0.5625      8.6823     18.3070
      0      0.4408     15.1478     24.9958
      0      0.6943     16.3557     31.4104
```


GARCH Specification Structure

Introduction (p. 3-2)	Introduces the GARCH specification structure and explains how the primary analysis and modeling functions operate on the structure.
Equation Variables and Parameter Names (p. 3-4)	Associates the variables used in the model equations (“Conditional Mean and Variance Models” on page 2-6) with their corresponding parameters in the specification structure.
Examples of Specification Structures (p. 3-5)	Uses examples of specification structures to interpret their contents.
Reading and Writing Specification Structures (p. 3-8)	Describes the creation and modification of a specification structure, as well as the retrieval of values from it.

Introduction

The GARCH Toolbox maintains the parameters that define a model and control the estimation process in a specification structure.

For the default model (see “The Default Model” on page 2-13), `garchfit` can create the specification structure and store the model orders and estimated parameters in it. For more complex models, you must use the function `garchset` to explicitly specify, in a specification structure, the conditional variance model you want, the mean and variance model orders, and possibly the initial coefficient estimates.

The primary analysis and modeling functions, `garchfit`, `garchpred`, and `garchsim`, all operate on the specification structure. This table describes how each function uses the specification structure.

Function	Description	Use of GARCH Specification Structure
<code>garchfit</code>	Estimates the parameters of a conditional mean specification of ARMAX form and a conditional variance specification of GARCH, GJR, or EGARCH form.	<p>Input. Optionally accepts a GARCH specification structure as input. If the structure contains the model orders (R, M, P, Q) but no coefficient vectors (C, AR, MA, Regress, K, ARCH, GARCH, Leverage), <code>garchfit</code> uses maximum likelihood to estimate the coefficients for the specified mean and variance models. If the structure contains coefficient vectors, <code>garchfit</code> uses them as initial estimates for further refinement. If you provide no specification structure, <code>garchfit</code> assumes, and returns, a specification structure for the default model (see “The Default Model” on page 2-13).</p> <p>Output. Returns a specification structure that contains a fully specified ARMAX/GARCH model.</p>

Function	Description	Use of GARCH Specification Structure
garchpred	Provides minimum-mean-square-error (MMSE) forecasts of the conditional mean and standard deviation of a return series, for a specified number of periods into the future.	<p>Input. Requires a GARCH specification structure that contains the coefficient vectors for the model for which garchpred is to forecast the conditional mean and standard deviation.</p> <p>Output. garchpred does not modify or return the specification structure.</p>
garchsim	Uses Monte Carlo methods to simulate sample paths for return series, innovations, and conditional standard deviation processes.	<p>Input. Requires a GARCH specification structure that contains the coefficient vectors for the model for which garchsim is to simulate sample paths.</p> <p>Output. garchsim does not modify or return the specification structure.</p>

Note See the garchset function reference page for descriptions of all the specification structure parameters.

Equation Variables and Parameter Names

For the most part, the names of specification structure parameters that define the ARMAX and GARCH models reflect the variable names of their corresponding components in the conditional mean and variance model equations (see “Conditional Mean and Variance Models” on page 2-6).

Conditional Mean Model

In the conditional mean model,

- R and M represent the order of the ARMA(R,M) conditional mean model.
- C represents the constant C .
- AR represents the R-element autoregressive coefficient vector ϕ_i .
- MA represents the M-element moving average coefficient vector θ_j .
- Regress represents the regression coefficients β_k .

Unlike the other components of the conditional mean equation, X has no representation in the GARCH specification structure. X is an optional matrix of returns that some toolbox functions use as explanatory variables in the regression component of the conditional mean. For example, X could contain return series of a suitable market index collected over the same period as the return series y . Toolbox functions that allow the use of a regression matrix provide a separate argument by which you can specify it.

Conditional Variance Models

In the conditional variance models

- P and Q represent the order of the GARCH(P,Q), GJR(P,Q), or EGARCH(P,Q) conditional variance model.
- K represents the constant K .
- GARCH represents the P-element coefficient vector G_i .
- ARCH represents the Q-element coefficient vector A_j .
- Leverage represents the Q-element leverage coefficient vector, L_j , for asymmetric EGARCH(P,Q) and GJR(P,Q) models.

Examples of Specification Structures

The following example shows the fields of the specification structure, `coeff`, for the estimated default model from “Analysis and Estimation Example Using the Default Model” on page 2-16. The term to the left of the colon (`:`) is the parameter name.

```
coeff
coeff =
    Comment: 'Mean: ARMAX(0,0,0); Variance: GARCH(1,1)'
    Distribution: 'Gaussian'
           C: -6.1919e-005
    VarianceModel: 'GARCH'
           P: 1
           Q: 1
           K: 1.0761e-006
    GARCH: 0.8060
    ARCH: 0.1531
```

When you display a specification structure, only the fields that are applicable to the specified model are displayed. Notice that $R = M = 0$ for this model, and so are not displayed.

By default, the `Comment` field shown above is automatically generated by `garchset` and `garchfit`. It summarizes the ARMAX and GARCH models used for the conditional mean and variance equations. You can use `garchset` to set the value of the `Comment` field, but the value you give it will replace this summary statement.

Following is the display for the MA(1)/GJR(1,1) estimated model from the example “Specifying Presample Data” on page 5-19. Notice that $\text{length}(\text{MA}) = M$, $\text{length}(\text{GARCH}) = P$, and $\text{length}(\text{ARCH}) = Q$.

```
coeff =
    Comment: 'Mean: ARMAX(0,1,0); Variance: GJR(1,1)'
    Distribution: 'Gaussian'
           M: 1
           C: 5.6403e-004
           MA: 0.2501
    VarianceModel: 'GJR'
           P: 1
```

```
Q: 1
K: 1.1907e-005
GARCH: 0.6945
ARCH: 0.0249
Leverage: 0.2454
Display: 'off'
```

If you had created the specification structure for the same MA(1)/GJR(1,1) example, but had not yet estimated the model coefficients, this is what you would see if you displayed the specification structure.

```
spec = garchset('VarianceModel','GJR','M',1,'P',1,'Q',1,...
'Display','off')
```

```
spec =
    Comment: 'Mean: ARMAX(0,1,?); Variance: GJR(1,1)'
    Distribution: 'Gaussian'
    M: 1
    C: []
    MA: []
    VarianceModel: 'GJR'
    P: 1
    Q: 1
    K: []
    GARCH: []
    ARCH: []
    Leverage: []
    Display: 'off'
```

The empty matrix symbols, [], indicate that these fields are required for the specified model, but have not yet been assigned values. For the specification to be *complete*, these fields must be assigned valid values. You can use `garchset` to assign values, e.g., as initial parameter estimates, to these fields. You can also pass such a specification structure to `garchfit`, which uses the parameters it estimates to complete the model specification. You cannot pass such a structure to `garchsim`, `garchinfer`, or `garchpred`. These functions require complete specifications.

Note See the `garchset` function reference page for descriptions of all the specification structure fields.

Reading and Writing Specification Structures

This section discusses

- “Creating and Modifying a Specification Structure” on page 3-8
- “Retrieving Specification Structure Values” on page 3-11

Creating and Modifying a Specification Structure

In general, you must use the function `garchset` to initially create a specification structure that, at a minimum, contains the chosen variance model and the mean and variance model orders. The only exception is the default model, for which `garchfit` can create a specification structure. The model parameters you provide must specify a valid model.

When you create a specification structure, you can specify both the conditional mean and variance models. Alternatively, you can specify either the conditional mean or the conditional variance model. If you do not specify both models, `garchset` assigns default parameters to the one you did not specify. For the conditional mean, the default is a constant ARMA(0,0,?) model. For the conditional variance, the default is a constant GARCH(0,0) model. The question mark (?) indicates that `garchset` doesn't know if you intend to include a regression component (see “Regression Components in Conditional Mean Models” on page 7-1).

The following examples create specification structures and display the results. Note that you only need to type the leading characters that uniquely identify the parameter. As illustrated here, `garchset` ignores case for parameter names.

The Default Model

This is a sampling of statements that all create specification structures for the default model.

```
spec = garchset('R',0,'m',0,'P',1,'Q',1);
```

```
spec = garchset('p',1,'Q',1);
```

```
spec = garchset;
```


The output of each command is the same. The Comment field summarizes the model. Because $R = M = 0$, the fields R, M, AR, and MA are not displayed.

```
spec =
      Comment: 'Mean: ARMAX(0,0,?); Variance: GARCH(1,1)'
      Distribution: 'Gaussian'
          C: []
      VarianceModel: 'GARCH'
          P: 1
          Q: 1
          K: []
      GARCH: []
      ARCH: []
```

ARMA(0,0)/GJR(1,1)

This command accepts the constant default for the mean model.

```
spec = garchset('VarianceModel','GJR','P',1,'Q',1)

spec =
      Comment: 'Mean: ARMAX(0,0,?); Variance: GJR(1,1)'
      Distribution: 'Gaussian'
          C: []
      VarianceModel: 'GJR'
          P: 1
          Q: 1
          K: []
      GARCH: []
      ARCH: []
      Leverage: []
```

AR(2)/GARCH(1,2) with Initial Parameter Estimates

For this command, garchset infers the model orders from the lengths of the coefficient vectors. garchset assumes a GARCH(P,Q) conditional variance process as the default.

```
spec = garchset('C',0,'AR',[0.5 -0.8],'K',0.0002,...
               'GARCH',0.8,'ARCH',[0.1 0.05])
```

```
spec =  
  
    Comment: 'Mean: ARMAX(2,0,?); Variance: GARCH(1,2) '  
    Distribution: 'Gaussian'  
        R: 2  
        C: 0  
        AR: [0.5000 -0.8000]  
    VarianceModel: 'GARCH'  
        P: 1  
        Q: 2  
        K: 2.0000e-004  
    GARCH: 0.8000  
    ARCH: [0.1000 0.0500]
```

Modifying a Specification Structure

This command creates an initial structure, and then updates the existing structure with additional parameter/value pairs. At each step the result must be a valid specification structure.

```
spec = garchset('VarianceModel','EGARCH','M',1,'P',1,'Q',1);  
spec = garchset(spec,'R',1,'Distribution','T')  
  
spec =  
  
    Comment: 'Mean: ARMAX(1,1,?); Variance: EGARCH(1,1) '  
    Distribution: 'T'  
        DoF: []  
        R: 1  
        M: 1  
        C: []  
        AR: []  
        MA: []  
    VarianceModel: 'EGARCH'  
        P: 1  
        Q: 1  
        K: []  
    GARCH: []  
    ARCH: []  
    Leverage: []
```

Retrieving Specification Structure Values

The function `garchget` retrieves the values of the specification structure fields.

This example creates a specification structure, `spec`, by providing the model coefficients, and allowing `garchset` to infer the model orders from the lengths of these vectors. `garchset` assumes the GARCH(P,Q) default variance model. The example then uses `garchget` to retrieve the variance model and the model orders for the conditional mean. Note that you only need to type the leading characters that uniquely identify the parameter. As illustrated here, `garchget` ignores case for parameter names.

```
spec = garchset('C',0,'AR',[0.5 -0.8],'K',0.0002,...
               'GARCH',0.8,'ARCH',[0.1 0.05])
spec =

    Comment: 'Mean: ARMAX(2,0,?); Variance: GARCH(1,2)'
    Distribution: 'Gaussian'
         R: 2
         C: 0
        AR: [0.5000 -0.8000]
VarianceModel: 'GARCH'
         P: 1
         Q: 2
         K: 2.0000e-004
        GARCH: 0.8000
        ARCH: [0.1000 0.0500]

R = garchget(spec,'R')

R =

     2

M = garchget(spec,'m')

M =

     0

var = garchget(spec,'VarianceModel')

var =

    GARCH
```


Simulation

Simulating Sample Paths (p. 4-2)

Shows you how to simulate single and multiple paths for return series, innovations, and conditional standard deviation processes.

Presample Data (p. 4-7)

Explains the use of automatically generated and user-supplied presample data. For automatically generated presample data, this section also discusses response tolerance and the minimization of transient effects.

Simulating Sample Paths

- “Introduction” on page 4-2
- “Simulating a Single Path” on page 4-4
- “Simulating Multiple Paths” on page 4-6

Introduction

Given models for the conditional mean and variance (see “Conditional Mean and Variance Models” on page 2-6), the function `garchsim` can simulate one or more sample paths for return series, innovations, and conditional standard deviation processes.

The section “Analysis and Estimation Example Using the Default Model” on page 2-16 uses the default GARCH(1,1) model to model the Deutschmark/British pound foreign exchange series. These examples use the resulting model

$$y_t = -6.1919e-005 + \varepsilon_t$$

$$\sigma_t^2 = 1.0761e-006 + 0.80598\sigma_{t-1}^2 + 0.15313\varepsilon_{t-1}^2$$

to simulate sample paths for return series, innovations, and conditional standard deviation processes.

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

Use the following commands to restore your workspace if necessary. The text of this example omits the display output of the estimation to save space.

```
load garchdata
dem2gbp = price2ret(DEM2GBP);
[coeff,errors,LLF,innovations,sigmas] = garchfit(dem2gbp);
coeff

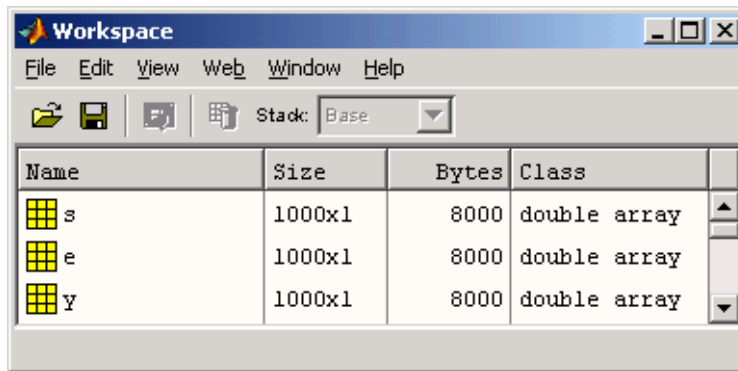
coeff =
    Comment: 'Mean: ARMAX(0,0,0); Variance: GARCH(1,1)'
    Distribution: 'Gaussian'
             C: -6.1919e-005
    VarianceModel: 'GARCH'
                 P: 1
                 Q: 1
                 K: 1.0761e-006
    GARCH: 0.8060
    ARCH: 0.1531
```

Simulating a Single Path

This code generates a single path of 1000 observations starting from the initial MATLAB random number generator state. Assuming there are 250 trading days per year, this is roughly four years' worth of daily data. ("Introduction" on page 4-2 tells you how to generate `coeff` for use in this example.)

```
randn('state',0);  
rand('state',0);  
[e,s,y] = garchsim(coeff,1000);
```

The Workspace Browser shows the result to be a single realization of 1000 observations each for the innovations $\{\varepsilon_t\}$, conditional standard deviations $\{\sigma_t\}$, and returns $\{y_t\}$ processes. These processes are designated by the output variables `e`, `s`, and `y`, respectively.

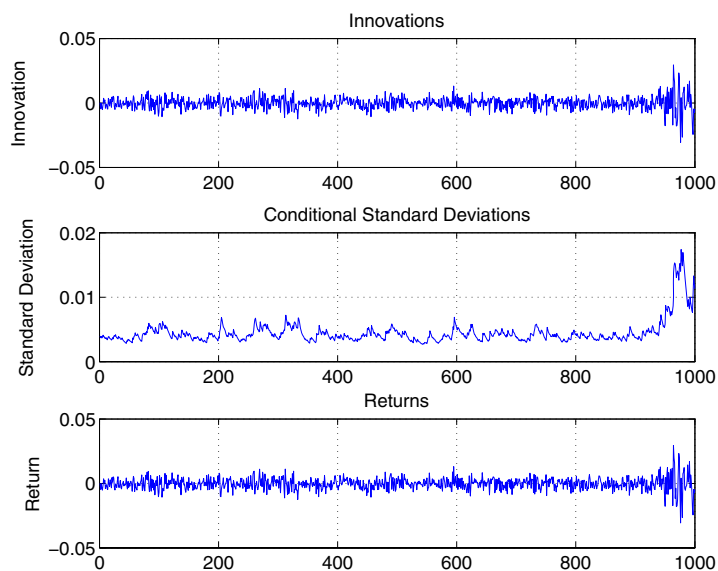


The screenshot shows the MATLAB Workspace Browser window. The title bar reads "Workspace". The menu bar includes "File", "Edit", "View", "Web", "Window", and "Help". Below the menu bar is a toolbar with icons for file operations and a "Stack" dropdown menu set to "Base". The main area contains a table with the following data:

Name	Size	Bytes	Class
s	1000x1	8000	double array
e	1000x1	8000	double array
y	1000x1	8000	double array

Now plot the garchsim output data to see what it looks like.

```
garchplot(e,s,y)
```



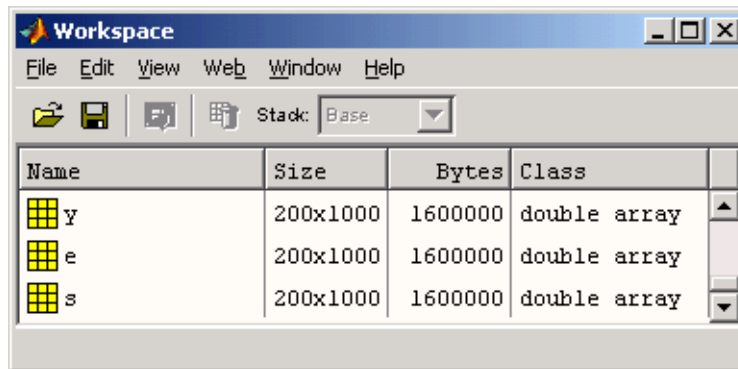
Note If you do not specify the number of observations, the default is 100. For example, `[e,s,y] = garchsim(coeff)` produces a single path of 100 observations.

Simulating Multiple Paths

In some cases, you may need multiple realizations. This example uses the same model to simulate 1000 paths of 200 observations each. (“Introduction” on page 4-2 tells you how to generate `coeff` for use in this example.)

```
[e,s,y] = garchsim(coeff,200,1000);
```

In this example, the $\{\varepsilon_t\}$, $\{\sigma_t\}$, and $\{y_t\}$ processes are 200-by-1000 element matrices. These are relatively large arrays, and demand large chunks of memory. In fact, because of the way the GARCH Toolbox manages transients, simulating this data requires more memory than the 4800000 bytes indicated in the Workspace Browser. (See “Automatically Generated Presample Data” on page 4-7 for more information about transients.)



The screenshot shows the MATLAB Workspace window. The title bar reads "Workspace" and the menu bar includes "File", "Edit", "View", "Web", "Window", and "Help". Below the menu bar is a toolbar with icons for file operations and a "Stack" dropdown menu set to "Base". The main area contains a table with the following data:

Name	Size	Bytes	Class
y	200x1000	1600000	double array
e	200x1000	1600000	double array
s	200x1000	1600000	double array

Presample Data

Because the mean equation and the variance equations can be recursive in nature, they require initial, or presample, data to initiate the simulation. You can use one of these methods.

- “Automatically Generated Presample Data” on page 4-7
- “User-Specified Presample Data” on page 4-13

Automatically Generated Presample Data

When you allow `garchsim` to automatically generate the required initial data,

- `garchsim` performs *independent path simulation*. That is, all simulated realizations are unique in that they evolve independently and share no common presample conditioning data.
- `garchsim` generates the presample data in a way that minimizes transient effects in the output processes.

Automatic Minimization of Transient Effects

`garchsim` generates output processes in (approximately) steady state by attempting to eliminate transients in the data it simulates. `garchsim` first estimates the number of observations needed for the transients to decay to some arbitrarily small value, subject to a 10000 observation maximum. It then generates a number of observations equal to the sum of this estimated value and the number of observations you request. `garchsim` then ignores the earlier estimated number of initial observations needed for the transients to decay sufficiently, and returns only the requested number of later observations.

To do this, `garchsim` interprets a GARCH(P,Q) or GJR(P,Q) conditional variance process as an ARMA(max(P,Q),P) model for the squared innovations, and interprets an EGARCH(P,Q) process as an ARMA(P,Q) model for the log of the conditional variance. (See, for example, Bollerslev [4], p.310.) `garchsim` then interprets the ARMA model as the correlated output of a linear filter and estimates its impulse response by finding the magnitude of the largest eigenvalue of its autoregressive polynomial. Based on this eigenvalue, `garchsim` estimates the number of observations, subject to a 10000 maximum, needed for the magnitude of the impulse response, which begins at 1, to decay below the default response tolerance 0.01 (i.e., 1 percent). If the conditional mean has an ARMA(R,M) component, then `garchsim` also estimates the

number of observations needed for the impulse response to decay below the response tolerance. This number is also subject to a 10000 maximum.

The effect of transients in the simulation process parallels that in the estimation, or inference, process. “Presample Data and Transient Effects” on page 5-23 provides an example of transient effects in the estimation process.

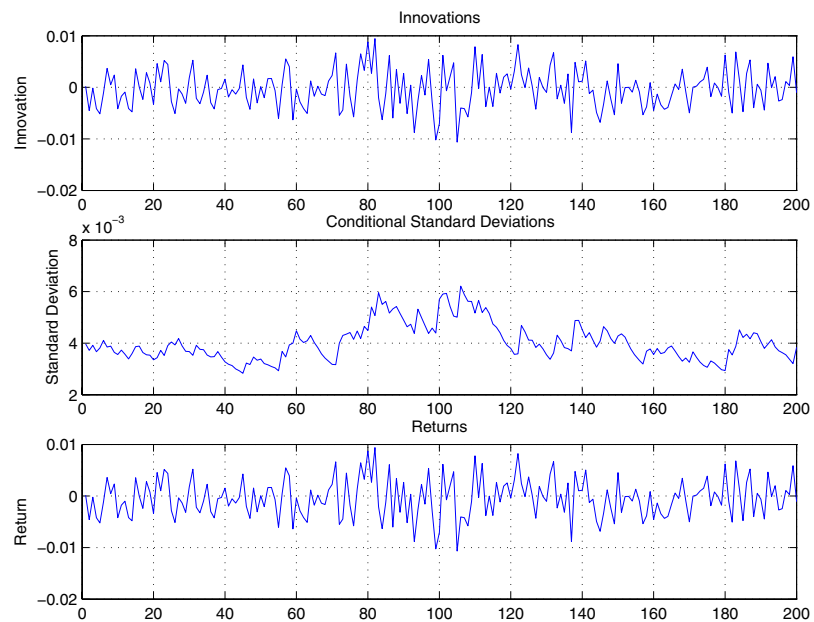
Specifying a Scalar Response Tolerance

If you want to use a response tolerance other than the default 0.01, you can specify it via the `Tolerance` argument. This example compares simulated observations generated using the default response tolerance, 0.01, and a larger tolerance 0.05. It uses the model from “Simulating Sample Paths” on page 4-2.

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

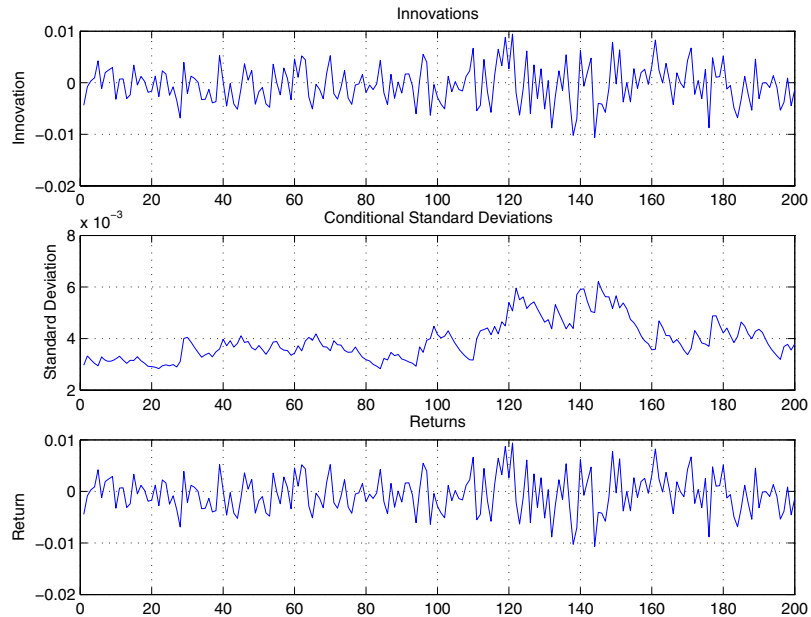
Start by simulating a single path of 200 observations using the default tolerance 0.01, and setting the scalar integer random generator state to its initial state 0.

```
[e1,s1,y1] = garchsim(coeff,200,1,0);  
garchplot(e1,s1,y1)
```



Now repeat the simulation, but specify the scalar `Tolerance` argument as 0.05.

```
[e5,s5,y5] = garchsim(coeff,200,1,0,[],0.05);
garchplot(e5,s5,y5)
```



The observations generated using the 0.05 response tolerance are the same as those generated for the default 0.01 tolerance, but shifted to the right. This is because it took fewer observations for the magnitude of the impulse response to decay below the larger 0.05 tolerance. If you make the `Tolerance` smaller than 0.01, note that `garchsim` might have to generate more observations and could conceivably reach the 10000 observation transient decay period maximum or run out of memory.

Storage Considerations

Depending on the values of the parameters in the simulated conditional mean and variance models, you may need long presample periods for the transients to die out. Although the simulation outputs may be relatively small matrices,

the initial computation of these transients can result in a large memory burden and seriously impact performance. Because of this, `garchsim` imposes a maximum of 10000 observations to the transient decay period of each realization. The example in “Simulating Multiple Paths” on page 4-6, which simulates three 200-by-1000 element arrays, requires intermediate storage for far more than 200 observations.

Other Ways to Minimize Transient Effects

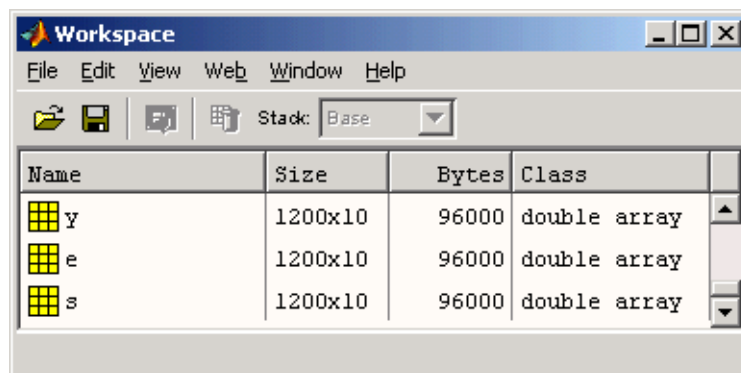
If you suspect that transients persist in the simulated data `garchsim` returns, you can use one of these methods to minimize their effect:

- “Oversampling” on page 4-11
- “Recycling Outputs” on page 4-12

Oversampling. Generate samples that are larger than you need, and delete observations from the beginning of each output series. For example, suppose you would like to simulate 10 independent paths of 1000 observations each for $\{\varepsilon_t\}$, $\{\sigma_t\}$, and $\{y_t\}$ starting from a known scalar random number state (12345).

Start by generating 1200 observations. `garchsim` generates sufficient presample data so that it can ignore initial samples that might be affected by transients. It then returns only the requested 1200 later observations.

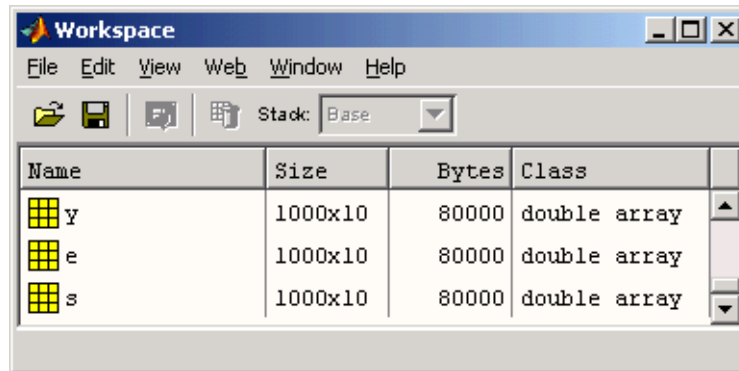
```
[e,s,y] = garchsim(coeff,1200,10,12345);
```



Name	Size	Bytes	Class
y	1200x10	96000	double array
e	1200x10	96000	double array
s	1200x10	96000	double array

Further minimize the effect of transients by retaining only the last 1000 observations of interest.

```
e = e(end-999:end,:);  
s = s(end-999:end,:);  
y = y(end-999:end,:);
```



Note The example above also illustrates how to specify a scalar random number generator state. This use corresponds to the `rand` and `randn` syntaxes, `rand('state',j)` and `randn('state',j)`.

Recycling Outputs. Start by simulating the desired number of observations without explicitly providing presample data i.e., let `garchsim` automatically generate the presample data. Then simulate again, using the simulated observations as the presample data. You can repeat this process until you are satisfied that transient effects have been sufficiently eliminated. See “User-Specified Presample Data” on page 4-13 for information about supplying presample data.

User-Specified Presample Data

Use the time-series input arrays `PreInnovations`, `PreSigmas`, and `PreSeries` to explicitly specify all required presample data. These presample arrays are associated with the `garchsim` outputs `Innovations`, `Sigmas`, and `Series`, respectively. When specified, `garchsim` uses these presample arrays to initiate the filtering process and form the conditioning set upon which the simulated realizations are based.

The `PreInnovations`, `PreSigmas`, and `PreSeries` arrays, as well as their associated outputs, are column-oriented arrays in which each column is associated with a distinct realization, or sample path. The first row of each array stores the oldest data and the last row the most recent.

Note that you can specify `PreInnovations`, `PreSigmas`, and `PreSeries` as matrices (i.e., with multiple columns), or as single-column vectors. In either case, the following table summarizes the minimum number of rows required to successfully initiate the simulation process.

garchsim Input Argument	Minimum Number of Rows	
	GARCH(P,Q), GJR(P,Q)	EGARCH(P,Q)
PreInnovations	$\max(M, Q)$	$\max(M, Q)$
PreSigmas	P	$\max(P, Q)$
PreSeries	R	R

If you specify `PreInnovations`, `PreSigmas`, and `PreSeries` as matrices, `garchsim` uses each column to initiate simulation of the corresponding column of the `Innovations`, `Sigmas`, and `Series` outputs. Each of the presample inputs must have `NUMPATHS` columns.

If you specify `PreInnovations`, `PreSigmas`, and `PreSeries` as column vectors, and `NUMPATHS` is greater than 1, `garchsim` performs *dependent path simulation*. That is, `garchsim` applies the same vector to each column of the corresponding `Innovations`, `Sigmas`, and `Series` outputs. All simulated sample paths share a common conditioning set. Although all realizations evolve independently, they share common presample conditioning data. Dependent path simulation enables the simulated sample paths to evolve from a common

starting point, and allows Monte Carlo simulation of forecasts and forecast error distributions. See “Advanced Example” on page 10-1.

If you specify at least one set, but fewer than three sets, of presample data, `garchsim` does not attempt to derive presample observations for those you omit. If you specify your own presample data, you must specify all that are necessary for the specified conditional mean and variance models. See the example “Specifying Presample Data” on page 5-19.

Note You can also use the `garchsim` input argument `State` to specify your own standardized noise process. See the `garchsim` reference page for details.

Estimation

Maximum Likelihood Estimation (p. 5-2)	Explains how the estimation engine, <code>garchfit</code> , uses maximum likelihood to estimate the parameters needed to fit the specified models to a given univariate return series.
Initial Parameter Estimates (p. 5-4)	Describes the use of both user-supplied and automatically generated initial parameter estimates. It also explains how <code>garchfit</code> uses parameter bounds to provide stability in the optimization process.
Presample Observations (p. 5-11)	Explains how <code>garchfit</code> calculates automatically generated presample data for the conditional mean model and for each of the supported conditional variance models. It also explains how to specify your own presample data.
Termination Criteria and Optimization Results (p. 5-13)	Discusses the optimization parameters that enable you to influence the optimization process.
Examples (p. 5-19)	Illustrates presample data, transient effects, and lower bound constraints. It also offers an alternative technique for estimating ARMA(R,M) parameters.

Maximum Likelihood Estimation

Given models for the conditional mean and variance (see “Conditional Mean and Variance Models” on page 2-6), and an observed univariate return series, the estimation engine `garchfit` infers the innovations (i.e., residuals) from the return series, and estimates, by maximum likelihood, the parameters needed to fit the specified models to the return series.

`garchfit` calls the Optimization Toolbox function `fmincon` to perform constrained optimization of a scalar function of several variables, i.e., the log-likelihood function, given a vector of initial parameter estimates (see “Initial Parameter Estimates” on page 5-4). This is generally referred to as constrained nonlinear optimization or nonlinear programming. In turn, `fmincon` calls the appropriate log-likelihood objective function to estimate the model parameters via maximum likelihood estimation (MLE). The chosen log-likelihood objective function proceeds in three steps:

- 1 Given the vector of current parameter values and the observed data `Series`, the log-likelihood function infers the process innovations (i.e., residuals) by inverse filtering. This inference, or inverse filtering, operation rearranges the conditional mean equation to solve for the current innovation, ε_t :

$$\varepsilon_t = -C + y_t - \sum_{i=1}^R \phi_i y_{t-i} - \sum_{j=1}^M \theta_j \varepsilon_{t-j} - \sum_{k=1}^{Nx} \beta_k X(t, k)$$

This equation is a whitening filter, transforming a (possibly) correlated process y_t into an uncorrelated white noise process ε_t .

- 2 The log-likelihood function then uses the inferred innovations ε_t to infer the corresponding conditional variances σ_t^2 via recursive substitution into the model-dependent conditional variance equations (Eq. (2-4), Eq. (2-5), Eq. (2-6)) above.
- 3 Finally, the function uses the inferred innovations and conditional variances to evaluate the appropriate log-likelihood objective function. If ε_t is Gaussian, the log-likelihood function is

$$\text{LLF} = -\frac{T}{2}\log(2\pi) - \frac{1}{2} \sum_{t=1}^T \log \sigma_t^2 - \frac{1}{2} \sum_{t=1}^T \varepsilon_t^2 / \sigma_t^2 \quad (5-1)$$

If ε_t is Student's t, the log-likelihood function is

$$\begin{aligned} \text{LLF} = T \log \left\{ \frac{\Gamma[(v+1)/2]}{\pi^{1/2} \Gamma(v/2)} (v-2)^{-1/2} \right\} - \frac{1}{2} \sum_{t=1}^T \log \sigma_t^2 \\ - \frac{v+1}{2} \sum_{t=1}^T \log \left[1 + \frac{\varepsilon_t^2}{\sigma_t^2 (v-2)} \right] \end{aligned} \quad (5-2)$$

where T is the sample size, i.e., the number of rows in the series $\{y_t\}$. The degrees of freedom v must be greater than 2.

Notice that the conditional mean equation (Eq. (2-2)) and the conditional variance equations (Eq. (2-4), Eq. (2-5), and Eq. (2-6)) are recursive and, in general, require presample observations to initiate the inverse filtering. For this reason, the objective functions shown above are referred to as conditional log-likelihood functions. That is, evaluation of the log-likelihood function is conditioned, or based, on a set of presample observations. The methods used to specify these presample observations are discussed in "Presample Observations" on page 5-11.

The iterative numerical optimization repeats the three steps described above until suitable termination criteria are reached. See "Termination Criteria and Optimization Results" on page 5-13 for details.

Initial Parameter Estimates

The constrained nonlinear optimizer, `fmincon`, requires a vector of initial parameter estimates. Although `garchfit` computes initial parameter estimates if you provide none, at times it may be helpful to compute and specify your own initial guesses to avoid convergence problems.

This section discusses

- “User-Specified Initial Estimates” on page 5-4
- “Automatically Generated Initial Estimates” on page 5-5
- “Parameter Bounds” on page 5-9

User-Specified Initial Estimates

You can specify complete initial estimates for either or both the conditional mean equation and the conditional variance equation.

For the conditional mean estimates to be complete, you must specify the C , AR, and MA parameters, consistent with the orders you specified for R and M ; i.e., the length of AR must be R , and the length of MA must be M . You must also specify the Regress parameter if you provide a regression matrix X . C , AR, MA, and Regress correspond respectively to C , ϕ_j , θ_i , and β_k in Eq. (2-2).

Note To remove the constant C from the conditional mean model, i.e., to fix $C = 0$ without providing initial parameter estimates for the remaining parameters, set $C = \text{NaN}$ (Not-a-Number). In this case, the value of `FixC` has no effect.

For the conditional variance estimates to be complete, you must specify the K , GARCH, and ARCH specification structure parameters for all conditional variance models, consistent with the orders you specified for P and Q ; i.e., the length of GARCH must be P , and the length of ARCH must be Q . You must also specify the Leverage parameter for GJR and EGARCH conditional variance models. The parameters K , GARCH, ARCH, and Leverage correspond respectively to κ , G_i , A_j , and L_j in Eq. (2-4), Eq. (2-5), and Eq. (2-6).

You can use `garchset` to create the necessary specification structure, `Spec`, or you can modify the `Coeff` structure returned by a previous call to `garchfit`.

If you provide initial parameter estimates for a model equation, you must provide *all* the estimated constants and coefficients consistent with the specified model orders. For example, for an ARMA(2,2) model with no regression matrix, you must specify the parameters C, AR, and MA. If you specify only MA, the specification is *incomplete*, and `garchfit` ignores the MA you specified and automatically generates all the requisite initial estimates.

The following specification structure provides C and AR as initial parameter estimates, but does not provide MA, even though $M = 1$. In this case, `garchfit` ignores the C and AR fields, computes initial parameter estimates, and overwrites any existing parameters in the incomplete conditional mean specification.

```
spec = garchset('R',1,'M',1,'C',0,'AR',0.5,...
               'P',1,'Q',1,'K',0.0005,'GARCH',0.8,'ARCH',0.1)
spec =

      Comment: 'Mean: ARMAX(1,1,?); Variance: GARCH(1,1)'
Distribution: 'Gaussian'
           R: 1
           M: 1
           C: 0
           AR: 0.5000
           MA: []
VarianceModel: 'GARCH'
           P: 1
           Q: 1
           K: 5.0000e-004
           GARCH: 0.8000
           ARCH: 0.1000
```

However, since the structure explicitly sets all fields in the conditional variance model, `garchfit` uses the specified values of K, GARCH, and ARCH as initial estimates subject to further refinement.

Automatically Generated Initial Estimates

If you do not provide initial coefficient estimates for a conditional mean or variance model, or if the estimates you provide are incomplete, `garchfit` automatically generates initial estimates. It first estimates the conditional mean parameters, if necessary, then estimates the conditional variance

parameters, if necessary. Again, note that `garchfit` ignores any incomplete initial estimates. `garchfit` estimates initial conditional mean parameters using standard statistical time-series techniques, dependent upon the parametric form of the conditional mean equation.

- “Conditional Mean Models Without a Regression Component” on page 5-6
- “Conditional Mean Models with a Regression Component” on page 5-6
- “Conditional Variance Models” on page 5-7

Conditional Mean Models Without a Regression Component

ARMA Models. Initial parameter estimates of general ARMA(R,M) conditional mean models are estimated by the three-step method outlined in Box, Jenkins, and Reinsel [8], Appendix A6.2.

- 1 `garchfit` estimates the autoregressive coefficients, ϕ_j , by computing the sample autocovariance matrix and solving the Yule-Walker equations.
- 2 Using these estimated coefficients, `garchfit` filters the observed Series to obtain a pure moving average process.
- 3 `garchfit` computes the autocovariance sequence of the moving average process, and uses it to iteratively estimate the moving average coefficients, θ_i . This last step also provides an estimate of the unconditional variance of the innovations.

Conditional Mean Models with a Regression Component

ARX Models (No Moving Average Terms Allowed). Initial estimates of the autoregressive coefficients, ϕ_j , and the regression coefficients, β_k , of the explanatory data matrix, X , are generated by ordinary least squares regression.

See “Regression Components in Conditional Mean Models” on page 7-1 for more information.

ARMAX Models (Moving Average Terms Included). Initial parameter estimation of the general ARMAX conditional mean models requires two steps:

- 1 `garchfit` estimates an ARX model by ordinary least squares.

- 2 garchfit estimates an MA(M) = ARMA(0,M) model as outlined in “Conditional Mean Models Without a Regression Component” on page 5-6.

Conditional Variance Models

As opposed to conditional mean parameters, initial estimates of conditional variance parameters are based on empirical analysis of financial time series, and are thus ad hoc. The approach is dependent upon the conditional variance model you select.

GARCH(P,Q) Models. For GARCH models, garchfit assumes that the sum of the G_i , ($i = 1, \dots, P$) and the A_j , ($j = 1, \dots, Q$) is close to 1. Specifically, for a general GARCH(P,Q) model (Eq. (2-4)), garchfit assumes that

$$G_1 + \dots + G_P + A_1 + \dots + A_Q = 0.9$$

If $P > 0$ (i.e., lagged conditional variances are included), then garchfit equally allocates 0.85 out of the available 0.90 to the P GARCH coefficients, and allocates the remaining 0.05 equally among the Q ARCH coefficients. $P = 0$ specifies an ARCH(Q) model in which garchfit allocates 0.90 equally to the Q ARCH terms. Some examples will clarify the approach.

The GARCH(1,1) model is by far the most common, and initial estimates are expressed as follows:

$$\sigma_t^2 = \kappa + 0.85\sigma_{t-1}^2 + 0.05\epsilon_{t-1}^2$$

A GARCH(2,1) model would be initially expressed as

$$\sigma_t^2 = \kappa + 0.425\sigma_{t-1}^2 + 0.425\sigma_{t-2}^2 + 0.05\epsilon_{t-1}^2$$

An ARCH(1) model would be initially expressed as

$$\sigma_t^2 = \kappa + 0.9\epsilon_{t-1}^2$$

An ARCH(2) model would be initially expressed as

$$\sigma_t^2 = \kappa + 0.45\epsilon_{t-1}^2 + 0.45\epsilon_{t-2}^2$$

Finally, garchfit estimates the constant κ of the conditional variance model by first estimating the unconditional, or time-independent, variance of $\{\epsilon_t\}$.

$$\sigma^2 = \frac{1}{T} \sum_{t=1}^T \varepsilon_t^2$$

In terms of the parameters this can also be expressed as

$$\sigma^2 = \frac{\kappa}{1 - \sum_{i=1}^P G_i - \sum_{j=1}^Q A_j} = \frac{\kappa}{1 - (0.85 + 0.05)}$$

and so

$$\begin{aligned} \kappa &= \sigma^2 (1 - (0.85 + 0.05)) \\ &= 0.1\sigma^2 \end{aligned}$$

GJR(P,Q) Models. `garchfit` treats a GJR(P,Q) model, described in Eq. (2-5), as a straightforward extension of an equivalent GARCH(P,Q) model with zero leverage terms. Thus, initial parameter estimates of GJR models are identical to those of equivalent order GARCH models (see “GARCH(P,Q) Models” on page 5-7), with the additional assumption that all leverage terms are zero,

$$L_i = 0 \quad 1 \leq i \leq Q$$

EGARCH(P,Q) Models. For EGARCH models `garchfit` assumes that the sum of the G_i , ($i = 1, \dots, P$) is 0.9, and the sum of the A_j , ($j = 1, \dots, Q$) is 0.2. Specifically, for a general EGARCH(P,Q) model (Eq. (2-6)), `garchfit` assumes that

$$G_1 + G_2 + \dots + G_P = 0.9$$

$$A_1 + A_2 + \dots + A_Q = 0.2$$

and

$$L_i = 0 \quad 1 \leq i \leq Q$$

If $P > 0$, i.e., lagged conditional variances are included, then `garchfit` equally allocates the available weight of 0.9 to the P GARCH coefficients, and equally allocates the available weight of 0.2 to the Q ARCH coefficients.

Recall that, in EGARCH models, the standardized innovation, z_t , serves as the forcing variable for both the conditional variance and the error, so that volatility clustering (i.e., persistence) is entirely captured by the G_i terms. In other words, EGARCH models make no allowance for the relationship between the G_i and A_i coefficients regarding initial parameter estimates. Because of this, EGARCH(0,Q) models ignore the persistence effect commonly associated with financial returns and are somewhat unusual. Some examples clarify the approach.

The EGARCH(1,1) model is by far the most common, and initial estimates are expressed as:

$$\log \sigma_t^2 = \kappa + 0.9 \log \sigma_{t-1}^2 + 0.2[|z_{t-1}| - E(|z_{t-1}|)]$$

Initial estimates for an EGARCH(2,2) model are expressed as

$$\begin{aligned} \log \sigma_t^2 = & \kappa + 0.45 \log \sigma_{t-1}^2 + 0.45 \log \sigma_{t-2}^2 + \\ & 0.1[|z_{t-1}| - E(|z_{t-1}|)] + 0.1[|z_{t-2}| - E(|z_{t-2}|)] \end{aligned}$$

An EGARCH(0,1) model would be initially expressed as

$$\log \sigma_t^2 = \kappa + 0.2[|z_{t-1}| - E(|z_{t-1}|)]$$

As you can see, initial parameter estimates for EGARCH models are most effective when $P > 0$.

Finally, you can estimate the constant κ of an EGARCH conditional variance model by noting the approximate relationship between the unconditional variance of the innovations process, σ^2 , and the G_i parameters of an EGARCH(1,1) model:

$$\begin{aligned} \kappa &= (1 - G_1) \log \sigma^2 \\ &= (1 - 0.9) \log \sigma^2 \\ &= 0.1 \log \sigma^2 \end{aligned}$$

Parameter Bounds

`garchfit` bounds some model parameters to provide stability in the optimization process. See the example “Active Lower Bound Constraint” on

page 5-30 for more information on overriding these bounds in the unlikely event they become active.

Conditional Mean Model

For the conditional mean model, Eq. (2-2), `garchfit` bounds the conditional mean constant C and the conditional mean regression coefficients β_k , if any, in the interval $[-10,10]$. However, if the coefficient estimates, whether provided by the user or automatically generated by `garchfit`, are outside this interval, `garchfit` sets the appropriate lower or upper bound equal to the estimated coefficient.

GARCH(P,Q) and GJR(P,Q) Conditional Variance Models

For GARCH(P,Q) and GJR(P,Q) conditional variance models, Eq. (2-3) and Eq. (2-4), `garchfit` uses 5 as an upper bound for the conditional variance constant κ . However, if the initial estimate is greater than 5, `garchfit` uses the estimated value as the upper bound.

EGARCH(P,Q) Conditional Variance Model

For EGARCH(P,Q) conditional variance models, Eq. (2-5), `garchfit` places arbitrary bounds on the conditional variance constant κ , such that $-5 \leq \kappa \leq 5$. However, if the magnitude of the initial estimate is greater than 5, `garchfit` adjusts the bounds accordingly.

Presample Observations

You can explicitly specify all required presample data, or you can allow `garchfit` to automatically generate the necessary presample data. “Maximum Likelihood Estimation” on page 5-2 discusses presample data required to initiate the inverse filtering and evaluate the conditional log-likelihood objective function.

This section discusses

- “User-Specified Presample Observations” on page 5-11
- “Automatically Generated Presample Observations” on page 5-11

User-Specified Presample Observations

Use the time-series column vector inputs `PreInnovations`, `PreSigmas`, and `PreSeries` to explicitly specify all required presample data. The following table summarizes the minimum number of rows required to successfully initiate the optimization process.

garchfit Input Argument	Minimum Number of Rows	
	GARCH(P,Q), GJR(P,Q)	EGARCH(P,Q)
PreInnovations	$\max(M, Q)$	$\max(M, Q)$
PreSigmas	P	$\max(P, Q)$
PreSeries	R	R

If you specify at least one set, but fewer than three sets, of presample data, `garchfit` does not attempt to derive presample observations for those you omit. If you specify your own presample data, you must specify all that are necessary for the specified conditional mean and variance models. See the example “Specifying Presample Data” on page 5-19.

Automatically Generated Presample Observations

If you do not specify any presample data, `garchfit` automatically generates the required presample data.

Conditional Mean Models

For conditional mean models with an autoregressive component, `garchfit` assigns the `R` required presample observations (i.e., `PreSeries`) of y_t the sample mean of `Series`. For models with a moving-average component, it sets the `M` required presample observations (i.e., `PreInnovations`) of ε_t to their expected value of zero. With this presample data, `garchfit` can infer the entire sequence of innovations for any general ARMAX conditional mean model regardless of the conditional variance model you select.

`garchfit` attempts to eliminate the effect of transients in the presample data it generates. This effect parallels that in the simulation process described in “Automatically Generated Presample Data” on page 4-7. The example “Presample Data and Transient Effects” on page 5-23 provides an example of transient effects in the estimation process.

GARCH(P,Q) Models

Once `garchfit` computes the innovations, it assigns the sample mean of the squared innovations

$$\sigma^2 = \frac{1}{T} \sum_{t=1}^T \varepsilon_t^2$$

to the `P` and `Q` required presample observations of σ_t^2 and ε_t^2 , respectively. See Hamilton [19] and Bollerslev [4].

GJR(P,Q) Models

`garchfit` also assigns the average squared innovation to all required presample observations of σ_t^2 and ε_t^2 . In addition, `garchfit` weights the `Q` presample observations of ε_t^2 associated with the leverage terms by 0.5 (i.e., the probability of a negative past residual).

EGARCH(P,Q) Models

`garchfit` also assigns the average squared innovation to all `P` presample observations of σ_t^2 . In addition, it sets all `Q` presample observations of the standardized innovations $z_t = (\varepsilon_t / \sigma_t)$ to zero and $|z_t| = (|\varepsilon_t| / \sigma_t)$ to the mean absolute deviation. This has the effect of setting all `Q` presample ARCH and leverage terms to zero.

Termination Criteria and Optimization Results

There are several fields in the specification structure that allow you to influence the optimization process. In order of importance, these are

<code>TolCon</code>	Termination tolerance on the constraint violation
<code>TolFun</code>	Termination tolerance on the function value
<code>TolX</code>	Termination tolerance on the parameter estimates
<code>MaxFunEvals</code>	Maximum number of function evaluations allowed
<code>MaxIter</code>	Maximum number of iterations allowed

This section discusses

- “MaxIter and MaxFunEvals” on page 5-13
- “TolCon, TolFun, and TolX” on page 5-14
- “Convergence” on page 5-14
- “Optimization Results” on page 5-15
- “Constraint Violation Tolerance” on page 5-16

See the Optimization Toolbox documentation and the `garchset` function for additional information about these parameters.

MaxIter and MaxFunEvals

`MaxIter` is the maximum number of iterations allowed in the estimation process. Each iteration involves an optimization phase in which `garchfit` suitably modifies calculations such as line search, gradient, and step size. The default value of `MaxIter` is 400. Although an estimation rarely exceeds `MaxIter`, you can increase the value if you suspect that the estimation terminated prematurely.

`MaxFunEvals`, a field closely related to `MaxIter`, specifies the maximum number of log-likelihood objective function evaluations. The default value is 100 times the number of parameters estimated in the model. For example, the default model has four parameters, so the default value of `MaxFunEvals` for the default model is 400. When the estimation process terminates prematurely, it is usually because `MaxFunEvals`, rather than `MaxIter`, is exceeded. You can

increase `MaxFunEvals` if you suspect that the estimation terminated prematurely.

The fields `MaxFunEvals` and `MaxIter` are purely mechanical in nature. Although you may encounter situations in which `MaxFunEvals` or `MaxIter` is reached, this is rather uncommon. Increasing `MaxFunEvals` or `MaxIter` may allow successful convergence, but reaching `MaxFunEvals` or `MaxIter` is usually an indication that your model poorly describes the data; in particular, it often indicates that the model is too complicated. Finally, although `MaxFunEvals` and `MaxIter` can cause the function to stop before a solution is found, they do not affect the solution once it is found.

TolCon, TolFun, and TolX

The fields `TolCon`, `TolFun`, and `TolX` are tolerance-related parameters that directly influence how and when convergence is achieved, and can also affect the solution.

`TolCon` is the termination tolerance placed on constraint violations, and represents the maximum value by which parameter estimates can violate a constraint and still allow successful convergence. See “Conditional Mean and Variance Models” on page 2-6 for information about these constraints.

`TolFun` is the termination tolerance placed on the log-likelihood objective function. Successful convergence occurs when the log-likelihood function value changes by less than `TolFun`. See “Optimization Results” on page 5-15 for more information.

`TolX` is the termination tolerance placed on the estimated parameter values. Similar to `TolFun`, successful convergence occurs when the parameter values change by less than `TolX`. See “Optimization Results” on page 5-15 for more information.

Convergence

`TolFun`, and `TolX` have the same default value, $1e-006$. The `TolCon` default is $1e-007$. If you experience extreme difficulty in convergence (e.g., the estimation shows little or no progress, or shows progress but stops early), then increasing one or more of these parameter values, for example, from $1e-006$ to $1e-004$, may allow the estimation to converge. If the estimation appears to converge to a suboptimal solution, then decreasing one or more of these

parameter values (e.g., from 1e-006 to 1e-007) may provide more accurate parameter estimates.

Note You can avoid many convergence difficulties by performing a prefit analysis. “Analysis and Estimation Example Using the Default Model” on page 2-16 describes graphical techniques, e.g., plotting the return series, and examining the ACF and PACF. It also discusses some preliminary tests, including Engle’s ARCH test and the Q-test.

“Model Selection and Analysis” on page 9-1 discusses other tests to help you determine the appropriateness of a specific GARCH model. It also explains how equality constraints can help you assess parameter significance. “GARCH Limitations” on page 1-4 mentions some limitations of GARCH models that could affect convergence.

Optimization Results

In contrast to MaxIter and MaxFunEvals, the tolerance fields TolCon, TolFun, and TolX do affect the optimization results (see “TolCon, TolFun, and TolX” on page 5-14). At successful termination, assuming iterative display is selected, you will typically see a message similar to one of the following:

```
Optimization terminated successfully:  
Magnitude of directional derivative in search direction  
less than 2*options.TolFun and maximum constraint violation  
is less than options.TolCon
```

```
Optimization terminated successfully:  
Search direction less than 2*options.TolX and  
maximum constraint violation is less than options.TolCon
```

```
Optimization terminated successfully:  
First-order optimality measure less than options.TolFun and  
maximum constraint violation is less than options.TolCon
```

Increasing TolFun or TolX from the default of 1e-6 to, for example, 1e-5, relaxes one or both of the first two termination criteria, often resulting in a slightly less accurate solution. Similarly, decreasing TolFun or TolX to, for

example, $1e-7$, restricts one or both of the first two termination criteria, often resulting in a slightly more accurate solution, but may also require more iterations. However, experience has shown that the default value of $1e-6$ for `TolFun` and `TolX` is almost always sufficient, and changing the value is unlikely to significantly affect the estimation results for GARCH modeling. For this reason, it is recommended that you accept the default value for `TolFun` and `TolX`.

The GARCH Toolbox default value of `TolCon` is $1e-7$, and changing the value of `TolCon` can significantly affect the solution in situations in which a constraint is active. For the GARCH Toolbox, from a practical standpoint, `TolCon` is the most important optimization-related field, and an additional discussion of its significance and use is helpful.

Whenever `garchfit` actively imposes parameter constraints (other than user-specified equality constraints) during the estimation process, the statistical results based on the maximum likelihood parameter estimates are invalid. (See Hamilton [19], page 142.) This is because statistical inference relies on the log-likelihood function's being approximately quadratic in the neighborhood of the maximum likelihood parameter estimates. This cannot be the case when the estimates fail to fall in the interior of the parameter space.

Constraint Violation Tolerance

At each step in the optimization process, `garchfit` evaluates the constraints described in “Conditional Mean and Variance Models” on page 2-6 against the current intermediate solution vector. For each user-specified equality constraint, it determines whether or not there is a violation whose absolute value is greater than `TolCon`. For each inequality constraint (including lower and upper bounds), it determines whether or not the inequality is violated by more than `TolCon`. If either `TolFun` or `TolX` exit condition is satisfied, and if the maximum of any violations is less than `TolCon`, then the optimization terminates successfully. (See “`TolCon`, `TolFun`, and `TolX`” on page 5-14.)

Strict Inequality Constraints

For the Optimization Toolbox, the numerical optimizer, `fmincon`, defines inequality constraints as a less than or equal to condition. However, many GARCH Toolbox inequality constraints are strict inequalities that specifically exclude exact equality. For this reason the GARCH Toolbox interprets `TolCon` differently from the Optimization Toolbox.

While TolCon applies to both strict inequalities and those that are not strict, garchfit provides special handling for strict inequalities. Specifically, garchfit associates each strict inequality constraint with its theoretical bound, or limit. However, to avoid the possibility of violating strict inequality constraints, garchfit defines the *actual bound* for each such constraint as the theoretical bound offset by $2 * \text{TolCon}$. Since the optimization can successfully terminate if the actual bound is violated by as much as TolCon, the end result is that any given strict inequality constraint is allowed to approach its theoretical bound to within TolCon.

Single Parameter Strict Inequality Constraints

Although it is possible for an estimate of a strict inequality constraint that involves a single parameter to terminate a distance TolCon from its theoretical bound, experience has shown that this is unlikely. Examples of such constraints are the conditional variance constant $\kappa > 0$ for the GARCH(P,Q) and GJR(P,Q) models, and the degrees of freedom $\nu > 2$ for the Student's t distribution. Typically, when the lower or upper bound of such a single-parameter inequality constraint is active, the estimate remains $2 * \text{TolCon}$ from the bound.

Note that even though the possibility is remote that an estimate of a single parameter constraint will terminate a distance TolCon from its theoretical bound, the garchfit approach for handling strict inequalities still allows for it.

As an illustration, assume TolCon = $1e-7$ (i.e., the GARCH Toolbox default value), and consider the default GARCH(1,1) model:

$$y_t = C + \varepsilon_t$$

$$\sigma_t^2 = \kappa + G_1 \sigma_{t-1}^2 + A_1 \varepsilon_{t-1}^2$$

with constraints

$$\begin{aligned} \kappa &> 0 \\ G_1 + A_1 &< 1 \\ G_1 &\geq 0 \\ A_1 &\geq 0 \end{aligned}$$

When the lower bound constraint $\kappa > 0$ is active, the estimated value of κ is typically $\kappa = 2e-7 = 2 * \text{TolCon}$.

Relaxing Constraint Tolerance Limits

Experience has shown that relaxing TolCon is more apt to remove an active constraint in some cases than in others. For inequality constraints with a single parameter, such as $\kappa > 0$ for the GARCH(P,Q) and GJR(P,Q) models and $\nu > 2$ for the Student's t distribution, decreasing TolCon may relax the constraint such that it is no longer active. The example "Active Lower Bound Constraint" on page 5-30 explains how to identify such a condition by examining the summary output structure.

This is not generally true for linear inequality constraints with multiple parameters. An example is $G_1 + A_1 < 1$. When this constraint is active, the estimated values of G_1 and A_1 are typically such that

$G_1 + A_1 = 0.9999999 = 1.0 - \text{TolCon}$. Decreasing TolCon to, say, $1e-8$ allows $G_1 + A_1$ to approach 1.0 a bit more closely, but the linear inequality constraint is likely to remain active.

Examples

- “Specifying Presample Data” on page 5-19
- “Presample Data and Transient Effects” on page 5-23
- “Alternative Technique for Estimating ARMA(R,M) Parameters” on page 5-29
- “Active Lower Bound Constraint” on page 5-30
- “Determining Convergence Status” on page 5-34

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

Specifying Presample Data

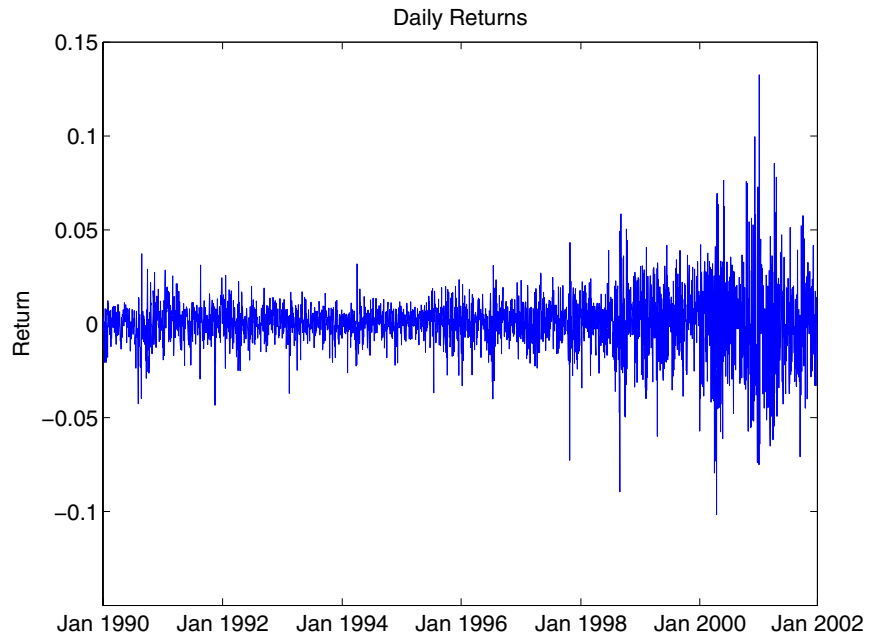
This example shows you how to specify your own presample data to initiate the estimation process. It highlights the formal column-oriented nature of the presample time-series inputs.

- 1 Load the Nasdaq data set and convert prices to returns.

```
load garchdata
nasdaq = price2ret(NASDAQ);
```

Suppose, for example, that you want to segment the Nasdaq data in an effort to compare estimation results obtained from a relatively stable period to those from a period of relatively high volatility. If you examine the Nasdaq returns, there is a rather distinct increase in volatility starting, approximately, in December 1997. Roughly, this is the 2000th observation.

```
plot(nasdaq)
axis([0 length(nasdaq) -0.15 0.15])
set(gca,'XTick',[1 507 1014 1518 2025 2529 3027])
set(gca,'XTickLabel',{'Jan 1990' 'Jan 1992' 'Jan 1994' ...
'Jan 1996' 'Jan 1998' 'Jan 2000' 'Jan 2002'})
ylabel('Return')
title('Daily Returns')
```



- 2** For this example, create a specification structure to model the Nasdaq returns as an MA(1) process with GJR(1,1) residuals,

```
spec = garchset('VarianceModel','GJR','M',1,'P',1,'Q',1,...
'Display','off');
```

- 3** Estimate the parameters, standard errors, and inferred residuals and standard deviations using the first 2000 observations, allowing `garchfit` to automatically generate the necessary presample observations. Then display the estimated coefficients and errors.

```
[coeff,errors,LLF,eFit,sFit] = garchfit(spec,nasdaq(1:2000));
garchdisp(coeff,errors)
```

```
Mean: ARMAX(0,1,0); Variance: GJR(1,1)
```

```
Conditional Probability Distribution: Gaussian
Number of Model Parameters Estimated: 6
```

Parameter	Value	Standard Error	T Statistic
C	0.00056403	0.00023455	2.4048
MA(1)	0.25006	0.024165	10.3480
K	1.1907e-005	1.528e-006	7.7931
GARCH(1)	0.69447	0.033664	20.6295
ARCH(1)	0.024937	0.017695	1.4093
Leverage(1)	0.24541	0.030517	8.0420

Since this particular conditional mean model has no regression component, you can obtain the same estimation results by calling `garchfit` with an empty regression matrix, `X = []`, as a placeholder for the third input,

```
[coeff,errors,LLF,eFit,sFit] = garchfit(spec,...
    nasdaq(1:2000),[]);
```

- 4 However, to specify your own presample data, you need to specify additional inputs. If you provide presample data, you must provide all necessary presample data, and it must be in the form of column vectors of sufficient length. This is because the inputs `PreInnovations`, `PreSigmas`, and `PreSeries` represent time series in a formal sense. (See “Presample Observations” on page 5-11.)

From the table in “Presample Observations”, the length of `PreInnovations` must be at least $\max(M, Q) = 1$, the length of `PreSigmas` must be at least $P = 1$, and `PreSeries` can be empty or unspecified altogether because $R = 0$.

Now estimate the same model from the later high-volatility period, using the inferred residuals and standard deviations from the previous period as the presample data:

```
[coeff,errors] = garchfit(spec,nasdaq(2001:end),[],eFit,sFit);
garchdisp(coeff, errors)
```

```
Mean: ARMAX(0,1,0); Variance: GJR(1,1)
```

```
Conditional Probability Distribution: Gaussian
```

```
Number of Model Parameters Estimated: 6
```

Parameter	Value	Standard Error	T Statistic
C	0.00065398	0.00060488	1.0812
MA(1)	0.012699	0.035131	0.3615
K	1.7845e-005	3.9153e-006	4.5578
GARCH(1)	0.85799	0.026246	32.6906
ARCH(1)	0.016147	0.022595	0.7146
Leverage(1)	0.17433	0.033234	5.2455

Comparing the estimation results from the two periods reveals a marked difference. Notice that the last input, `PreSeries`, is unnecessary and is left unspecified.

Since the example uses only the most recent observations of `PreInnovations`, `PreSigmas`, and `PreSeries`, any of the following calls to `garchfit` produce identical estimation results.

```
[coeff,errors] = garchfit(spec,nasdaq(2001:end),[],...
    eFit(end),sFit(end));
```

```
[coeff,errors] = garchfit(spec,nasdaq(2001:end),[],...
    eFit(end),sFit(end),nasdaq(1:2000));
```

```
[coeff,errors] = garchfit(spec,nasdaq(2001:end),...
    [],eFit,sFit,nasdaq(1999:2000));
```

The first equivalent call passes in the minimum required presample observations of past residuals and standard deviations, which in this case is the last inferred observation of each. The last two equivalent calls specify an unnecessary presample return series, which `garchfit` ignores.

If, for example, the original specification included an AR(2) model (i.e., $R = 2$), then at least the last two Nasdaq returns are needed to initiate estimation. In this case, the last two calls to `garchfit` above would produce identical results for conditional mean models with AR components up to 2nd order.

Presample Data and Transient Effects

This example simulates a return series, `yTrue`, then uses the inference function `garchinfer` to infer $\{\epsilon_t\}$ and $\{\sigma_t\}$ from the simulated return series. First, the example uses automatically generated presample data to infer the approximate residuals and conditional standard deviation processes, and then uses explicitly specified presample data to infer the exact residuals and conditional standard deviation processes. The example finally compares the approximate conditional standard deviation processes with the exact conditional standard deviations processes to reveal the effect of transients in the approximate results. The effect of transients in the estimation, or inference, process parallels that in the simulations process described in “Automatically Generated Presample Data” on page 4-7.

Note This example uses `garchinfer`, rather than `garchfit`, to avoid introducing differences as a result of the optimization. While `garchsim` uses an ARMA model as a linear filter to transform an uncorrelated input innovations process $\{\varepsilon_t\}$ into a correlated output returns process $\{y_t\}$, `garchinfer` reverses this process (as does `garchfit`) by inferring innovations $\{\varepsilon_t\}$ and standard deviation $\{\sigma_t\}$ processes from the observations in $\{y_t\}$.

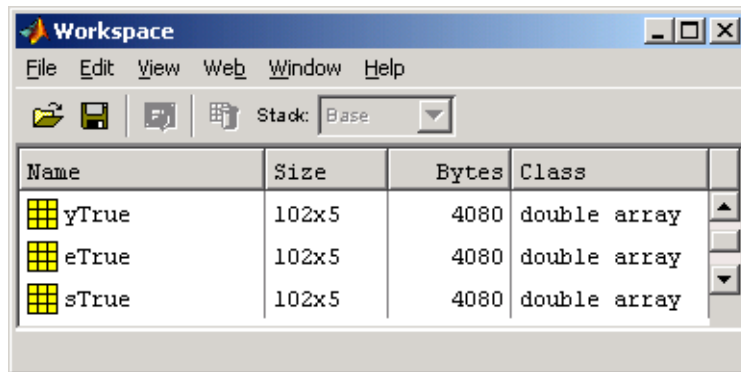
- 1 Specify a time series as an AR(2) conditional mean model and GARCH(1,2) conditional variance model. Note that this is an elaborate specification, typically unwarranted for a real-world financial time series, and is meant for illustration purposes only.

```
spec = garchset('C',0,'AR',[0.5 -0.8],'K',0.0002,...
               'GARCH',0.8,'ARCH',[0.1 0.05])
spec =

      Comment: 'Mean: ARMAX(2,0,?); Variance: GARCH(1,2) '
Distribution: 'Gaussian'
           R: 2
           C: 0
           AR: [0.5000 -0.8000]
VarianceModel: 'GARCH'
           P: 1
           Q: 2
           K: 2.0000e-004
      GARCH: 0.8000
      ARCH: [0.1000 0.0500]
```

- 2 Simulate 102 observations for each of 5 realizations and reserve the first 2 rows of observations for the presample data needed by `garchinfer` in step 4. From the table in “User-Specified Presample Data” on page 4-13, notice that the `PreInnovations` array must have at least $\max(M, Q) = 2$ rows, `PreSigmas` must have at least $P = 1$ row, and `PreSeries` must have at least $R = 2$ rows. Add the initial state = 0 as a trailing input argument.

```
[eTrue,sTrue,yTrue] = garchsim(spec,102,5,0);
```

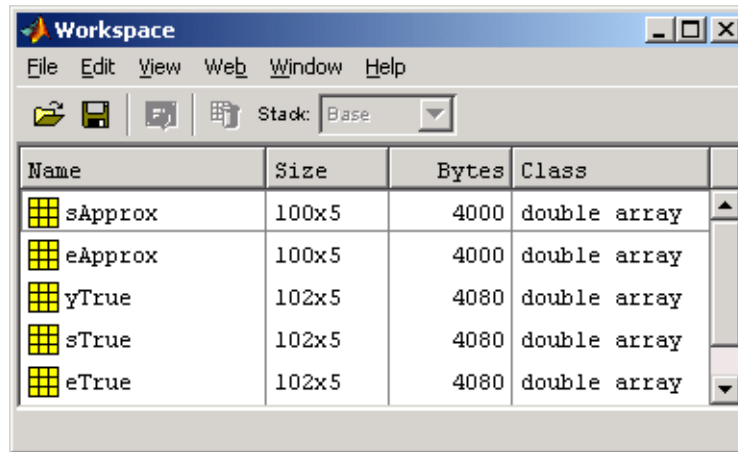


The screenshot shows the MATLAB Workspace window with the following table of variables:

Name	Size	Bytes	Class
yTrue	102x5	4080	double array
eTrue	102x5	4080	double array
sTrue	102x5	4080	double array

- 3 Using observations 3 and beyond as the observed return series input, call `garchinfer` without any explicit presample data to infer the approximate residuals and conditional standard deviations based on the default, or automatic, presample data inference approach (see the `garchfit` and `garchinfer` functions for details).

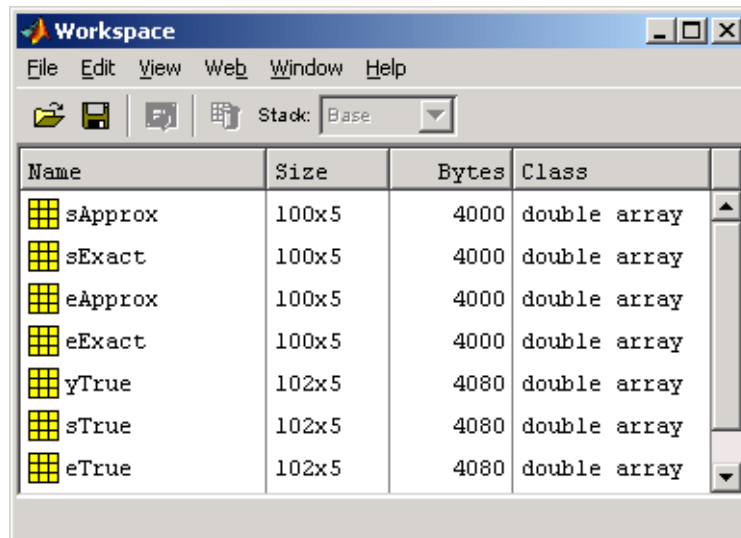
```
[eApprox,sApprox] = garchinfer(spec,yTrue(3:end,:));
```



Name	Size	Bytes	Class
sApprox	100x5	4000	double array
eApprox	100x5	4000	double array
yTrue	102x5	4080	double array
sTrue	102x5	4080	double array
eTrue	102x5	4080	double array

- 4 Call `garchinfer` again, but this time use the first two rows of the true simulated data as presample data. Use of the presample data allows you to infer the exact residuals and conditional standard deviations,

```
[eExact,sExact] = garchinfer(spec,yTrue(3:end,:),[],...  
                          eTrue(1:2,:),sTrue(1:2,:),yTrue(1:2,:));
```

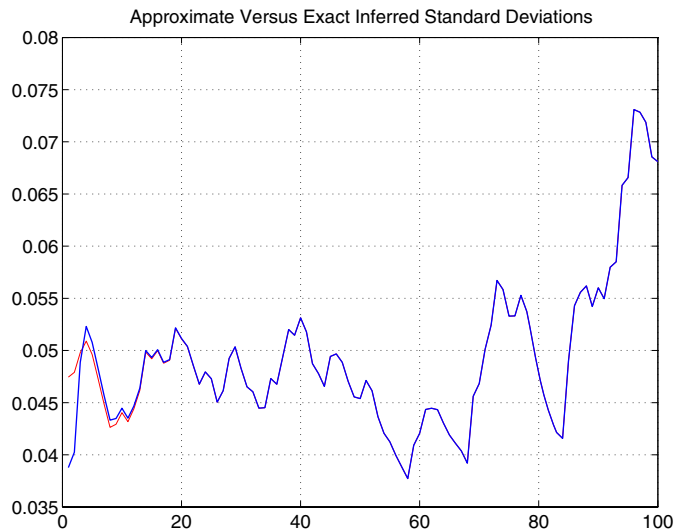


The screenshot shows the MATLAB Workspace window with a menu bar (File, Edit, View, Web, Window, Help) and a toolbar. Below the toolbar is a table listing variables in the workspace. The table has columns for Name, Size, Bytes, and Class. The variables listed are sApprox, sExact, eApprox, eExact, yTrue, sTrue, and eTrue, all of which are double arrays.

Name	Size	Bytes	Class
sApprox	100x5	4000	double array
sExact	100x5	4000	double array
eApprox	100x5	4000	double array
eExact	100x5	4000	double array
yTrue	102x5	4080	double array
sTrue	102x5	4080	double array
eTrue	102x5	4080	double array

- 5 Compare the first realization of the approximate and the exact inferred conditional standard deviations reveals the distinction between automatically generated and user-specified presample data.

```
plot(sApprox(:,1),'red')
grid('on'),hold('on')
plot(sExact(:,1),'blue')
title('Approximate Versus Exact Inferred Standard Deviations')
```



Notice that the approximate and exact standard deviations are asymptotically identical. The only difference between the two curves is attributable to the transients induced by the default initial conditions.

In fact, if you were to plot the first realization of the original simulated conditional standard deviations, `sTrue(3:end,1)`, on the current figure, it would lie completely on top of the blue curve.

Although the figure highlights the first realization of conditional standard deviations, the comparison holds for any realization, and for the inferred residuals as well.

Thus, this example reveals the link between simulation and inference: `garchsim` can be thought of as a correlation filter capable of processing multiple realizations simultaneously, and is the complement of `garchinfer`, which can be thought of as a whitening, or inverse, filter capable of processing multiple realizations simultaneously. Although the estimation engine `garchfit` is capable of processing only a single realization at a time, the transient effects highlighted in this example are exactly the same when applied to the estimation.

Alternative Technique for Estimating ARMA(R,M) Parameters

This example illustrates how to use the presample inputs `PreInnovations` and `PreSeries` to estimate the parameters of ARMA(R,M) models by a popular alternative technique. It assumes a simple constant variance model, and highlights the GARCH Toolbox as a general-purpose univariate time-series processor.

Default Method

As mentioned above, estimation requires presample data to initiate the inverse filtering process. In the absence of any explicit presample data, `garchfit` assigns the R required presample observations of y_t , i.e., `Series`, the sample mean of `Series`. It also assigns the M required presample observations of ε_t , i.e., the innovations, or residuals, their expected value of zero. This method then calculates the log-likelihood objective function value using all the available data in `Series`, and is the default method used by the GARCH Toolbox.

Alternative Technique

Another popular method also sets the M required presample observations of the residuals, ε_t , to zero, but uses the first R actual observations of `Series` as initial values. Thus, $\{y_1, y_2, \dots, y_R\}$ are used to initiate the inverse filter, and the log-likelihood objective function value is based on the remaining observations. See Hamilton [19], page 132, or Box, Jenkins, and Reinsel [8], pages 236-237.

For example, assume you have some hypothetical time series, `xyz`, and you want to estimate an ARMA(R,M) model with constant conditional variances. Using the alternative presample method, you would exclude the first R observations of `xyz` from the input `Series`, and reserve them for the input

PreSeries. Specifically, you would set the input Series = xyz(R+1:end), PreInnovations = zeros(M,1), PreSigmas = [], and PreSeries = xyz(1:R).

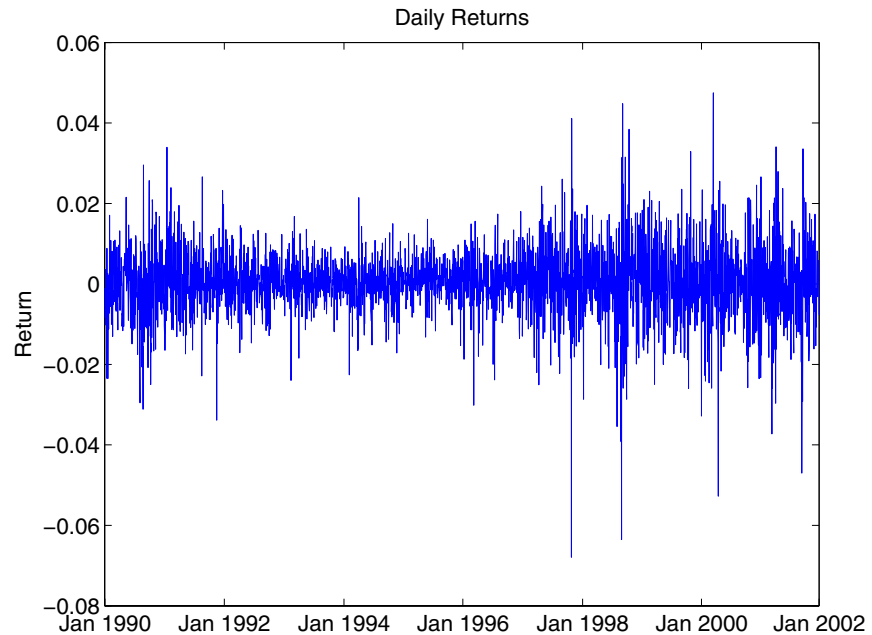
Active Lower Bound Constraint

This example illustrates an active lower bound constraint, $\kappa > 0$, for the conditional variance constant κ . This constraint is required for GARCH and GJR variance models to ensure a positive conditional variance process. It also illustrates how to identify such active constraints, and what to do about this most commonly encountered active constraint. See “Termination Criteria and Optimization Results” on page 5-13.

1 Load the NYSE data set and convert prices to returns.

```
load garchdata
nyse = price2ret(NYSE);

plot(nyse)
axis([0 length(nyse) -0.08 0.06])
set(gca,'XTick',[1 507 1014 1518 2025 2529 3027])
set(gca,'XTickLabel',{'Jan 1990' 'Jan 1992' 'Jan 1994' ...
    'Jan 1996' 'Jan 1998' 'Jan 2000' 'Jan 2002'})
set(gca,'YTick',[-0.08:0.02:0.06])
ylabel('Return')
title('Daily Returns')
```



- 2** Estimate a default GARCH(1,1) model and print the estimation results. For this example, notice that TolCon = 1e-6, and iterative display is disabled for brevity.

```
spec = garchset('Display','off','P',1,'Q',1,'TolCon',1e-6);
[coeff,errors,LLF,eFit,sFit,summary] = garchfit(spec,nyse);
garchdisp(coeff,errors)
```

Mean: ARMAX(0,0,0); Variance: GARCH(1,1)

Conditional Probability Distribution: Gaussian
Number of Model Parameters Estimated: 4

Parameter	Value	Standard Error	T Statistic
C	0.00051941	0.00013701	3.7910
K	2e-006	2.8192e-007	7.0943
GARCH(1)	0.87166	0.0095167	91.5925
ARCH(1)	0.10419	0.0073771	14.1238

- 3** Examination of these results reveals the estimated variance constant $K = 2e-006 = 0 + 2 * TolCon = 2 * TolCon$, i.e., K is equal to the theoretical lower bound plus $2 * TolCon$. You can see this by printing the summary structure and looking at the constraints message field:

```
summary
```

```
summary =
```

```
    warning: 'No Warnings'
    converge: 'Function Converged to a Solution'
  constraints: 'Boundary Constraints Active: Standard
              Errors May Be Inaccurate'
    covMatrix: [4x4 double]
    iterations: 13
functionCalls: 115
      lambda: [1x1 struct]
```

- 4 Print the lower and upper bound LaGrange multipliers and examine them for nonzero values:

```
[summary.lambda.lower summary.lambda.upper]
```

```
ans =
    1.0e+006 *

         0         0
    7.3602         0
         0         0
         0         0
```

Notice that lower and upper bound LaGrange multipliers are ordered exactly as displayed by `garchdisp`. From this result, you can clearly see that the lower bound constraint $K > 0$ is active.

- 5 Repeat the estimation with the default `TolCon = 1e-7` and verify that the constraint is no longer active.

```
spec = garchset('Display','off','P',1,'Q',1);
[coeff,errors,LLF,eFit,sFit,summary] = garchfit(spec,nyse);
garchdisp(coeff,errors)
```

```
Mean: ARMAX(0,0,0); Variance: GARCH(1,1)
```

```
Conditional Probability Distribution: Gaussian
```

```
Number of Model Parameters Estimated: 4
```

Parameter	Value	Standard Error	T Statistic
C	0.00049676	0.00013137	3.7813
K	8.9128e-007	1.5776e-007	5.6495
GARCH(1)	0.91088	0.0069142	131.7410
ARCH(1)	0.079942	0.0058319	13.7077

```
summary
```

```
summary =
    warning: 'No Warnings'
```

```

        converge: 'Function Converged to a Solution'
        constraints: 'No Boundary Constraints'
        covMatrix: [4x4 double]
        iterations: 21
        functionCalls: 208
        lambda: [1x1 struct]

```

```
[summary.lambda.lower    summary.lambda.upper]
```

```

ans =
     0     0
     0     0
     0     0
     0     0

```

Determining Convergence Status

There are two ways to determine whether an estimation achieves convergence. The first, and easiest, is to examine the optimization details of the estimation. By default, `garchfit` displays this information in the MATLAB Command Window. The second way to determine convergence status is to request the `garchfit` optional summary output.

To illustrate these methods, use the DEM2GBP (Deutschmark/British pound foreign exchange rate) data.

```

load garchdata
dem2gbp = price2ret(DEM2GBP);
[coeff,errors,LLF,eFit,sFit,summary] = garchfit(dem2gbp);

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Diagnostic Information
```

```
Number of variables: 4
```

```
Functions
```

```

Objective:          garchllfn
Gradient:           finite-differencing
Hessian:            finite-differencing (or Quasi-Newton)
Nonlinear constraints: armanlc
Gradient of nonlinear constraints: finite-differencing

```

Constraints

Number of nonlinear inequality constraints: 0

Number of nonlinear equality constraints: 0

Number of linear inequality constraints: 1

Number of linear equality constraints: 0

Number of lower bound constraints: 4

Number of upper bound constraints: 4

Algorithm selected

medium-scale

%%%

End diagnostic information

Iter	F-count	f(x)	max constraint	Step-size	Directional derivative	First-order Optimality	Procedure
1	28	-7916.01	-2.01e-006	7.63e-006	857	1.42e+005	
2	36	-7959.65	-1.508e-006	0.25	389	9.8e+007	
3	45	-7963.98	-3.113e-006	0.125	131	5.29e+006	
4	52	-7965.59	-1.586e-006	0.5	55.9	4.45e+007	
5	65	-7966.9	-1.574e-006	0.00781	101	1.46e+007	
6	74	-7969.46	-2.201e-006	0.125	14.9	2.77e+007	
7	83	-7973.56	-2.663e-006	0.125	36.6	1.45e+007	
8	90	-7982.09	-1.332e-006	0.5	-6.39	5.59e+006	
9	103	-7982.13	-1.399e-006	0.00781	6.49	1.32e+006	
10	111	-7982.53	-1.049e-006	0.25	12.5	1.87e+007	
11	120	-7982.56	-1.186e-006	0.125	3.72	3.8e+006	
12	128	-7983.69	-1.11e-006	0.25	0.184	4.91e+006	
13	134	-7983.91	-7.813e-007	1	0.732	1.22e+006	
14	140	-7983.98	-9.265e-007	1	0.186	1.17e+006	
15	146	-7984	-8.723e-007	1	0.0427	9.52e+005	
16	154	-7984	-8.775e-007	0.25	0.0152	6.33e+005	
17	160	-7984	-8.75e-007	1	0.00197	6.98e+005	
18	166	-7984	-8.763e-007	1	0.000931	7.38e+005	
19	173	-7984	-8.759e-007	0.5	0.000469	7.37e+005	
20	179	-7984	-8.761e-007	1	0.00012	7.22e+005	
21	199	-7984	-8.761e-007	-6.1e-005	0.0167	7.37e+005	Hessian modified twice
22	213	-7984	-8.761e-007	0.00391	0.00582	7.26e+005	Hessian modified twice

Optimization terminated successfully:

Search direction less than 2*options.TolX and

maximum constraint violation is less than options.TolCon

No Active Constraints

Notice that the optimization details indicate successful termination. Now, examine the summary output structure.

```
summary

summary =
    warning: 'No Warnings'
    converge: 'Function Converged to a Solution'
    constraints: 'No Boundary Constraints'
    covMatrix: [4x4 double]
    iterations: 22
    functionCalls: 213
    lambda: [1x1 struct]
```

The converge field indicates successful convergence. If the estimation failed to converge, the converge field would contain the message 'Function Did NOT Converge'. If the number of iterations or function evaluations exceeded its specified limits, the converge field would contain the message 'Maximum Function Evaluations or Iterations Reached'. The summary structure also contains fields that indicate the number of iterations (`iterations`) and log-likelihood function evaluations (`functionCalls`).

Forecasting

Minimum Mean Square Error
Forecasting (p. 6-2)

Discusses the outputs of the forecasting engine, garchpred: the conditional standard deviations of future innovations, the conditional mean forecasts of the return series, the MMSE volatility forecasts of returns, and the RMSE associated with conditional mean forecasts.

Presample Observations (p. 6-5)

Explains how garchpred generates the necessary presample data.

Asymptotic Behavior for Long-Range
Forecast Horizons (p. 6-6)

Discusses the asymptotic behavior of the garchpred outputs for long-range forecast horizons.

Examples (p. 6-8)

Computes a forecast of the conditional mean, a volatility forecast, and a forecast with multiple realizations.

Minimum Mean Square Error Forecasting

The forecasting engine, `garchpred`, computes minimum mean square error (MMSE) forecasts of the conditional mean of returns $\{y_t\}$ and conditional standard deviation of the innovations $\{\varepsilon_t\}$ in each period over a user-specified forecast horizon. To do this, it views the conditional mean and variance models from a linear filtering perspective, and applies iterated conditional expectations to the recursive equations, one forecast period at a time.

Each output of `garchpred` is an array with a number of rows equal to the user-specified forecast horizon and with a number of columns the same as the number of columns (i.e., realizations, or paths) in the time-series array of asset returns, `Series`. For a general forecasting example involving multiple realizations, see “Examples” on page 6-8.

This section discusses the four `garchpred` outputs:

- “Conditional Standard Deviations of Future Innovations” on page 6-2
- “Conditional Mean Forecasts of the Return Series” on page 6-3
- “MMSE Volatility Forecasts of Returns” on page 6-3
- “RMSE Associated with Conditional Mean Forecasts” on page 6-4

Conditional Standard Deviations of Future Innovations

The first output of `garchpred`, `SigmaForecast`, is a matrix of conditional standard deviations of future innovations (i.e., residuals) on a per-period basis. This matrix represents the standard deviations derived from the MMSE forecasts associated with the recursive volatility model you defined in the GARCH specification structure.

For GARCH(P,Q) and GJR(P,Q) models, `SigmaForecast` is the square root of the MMSE conditional variance forecasts. For EGARCH(P,Q) models, `SigmaForecast` is the square root of the exponential of the MMSE forecasts of the logarithm of conditional variance.

Since the forecasts are computed iteratively, the first row contains the standard deviation in the first forecast period for each realization of `Series`, the second row contains the standard deviation in the second forecast period, and so on. Thus, if you specify a forecast horizon greater than one, the per-period standard deviations of all intermediate horizons are returned as

well. In this case, the last row contains the standard deviation at the specified forecast horizon for each realization of Series.

Conditional Mean Forecasts of the Return Series

The second output of `garchpred`, `MeanForecast`, is a matrix of MMSE forecasts of the conditional mean of Series on a per-period basis. Again, the first row contains the forecast for each realization of Series in the first forecast period, the second row contains the forecast in the second forecast period, and the last row contains the forecast of Series at the forecast horizon.

MMSE Volatility Forecasts of Returns

The third output of `garchpred`, `SigmaTotal`, is a matrix of volatility forecasts of returns over multiperiod holding intervals. That is, the first row contains the expected standard deviation of returns for assets held for one period for each realization of Series, the second row contains the standard deviation of returns for assets held for two periods, and so on. Thus, the last row contains the forecast of the standard deviation of the cumulative return obtained if an asset was held for the entire forecast horizon.

`garchpred` computes the elements of `SigmaTotal` by taking the square root of

$$\text{var}_t \left[\sum_{i=1}^s y_{t+i} \right] = \sum_{i=1}^s \left[\left(1 + \sum_{j=1}^{s-i} \Psi_j \right)^2 \text{E}_t(\sigma_{t+i}^2) \right] \quad (6-1)$$

where s is the forecast horizon of interest (`NumPeriods`), and Ψ_j is the coefficient of the j th lag of the innovations process in an infinite-order MA representation of the conditional mean model (see the function `garchma`).

In the special case of the default model for the conditional mean, $y_t = C + \varepsilon_t$, this reduces to

$$\text{var}_t \left[\sum_{i=1}^s y_{t+i} \right] = \sum_{i=1}^s \text{E}_t(\sigma_{t+i}^2)$$

The `SigmaTotal` forecasts are correct for continuously compounded returns, and approximate for periodically compounded returns. `SigmaTotal` is the same size as `SigmaForecast` if the conditional mean is modeled as a stationary invertible ARMA process.

For conditional mean models with regression components (i.e., X or XF is specified), `SigmaTotal` is an empty matrix, []. In other words, `garchpred` computes `SigmaTotal` only if the conditional mean is modeled as a stationary invertible ARMA process. See “Regression Components in Conditional Mean Models” on page 7-1.

RMSE Associated with Conditional Mean Forecasts

The fourth output of `garchpred`, `MeanRMSE`, is a matrix of root mean square errors (RMSE) associated with the output forecast array `MeanForecast`. That is, each element of `MeanRMSE` is the conditional standard deviation of the corresponding forecast error (i.e., the standard error of the forecast) in the `MeanForecast` matrix. From Baillie and Bollerslev [1], Equation 19,

$$\text{var}_t(y_{t+s}) = \sum_{i=1}^s \Psi_{s-i}^2 \mathbf{E}_t(\sigma_{t+i}^2)$$

Using this equation and the computed MMSE forecasts of the conditional mean (`MeanForecast`) and the standard errors of the corresponding forecasts (`MeanRMSE`), you can construct approximate confidence intervals for conditional mean forecasts, with the approximation becoming more accurate during periods of relatively stable volatility (see Baillie and Bollerslev [1] and Bollerslev, Engle, and Nelson [6]). As heteroscedasticity in returns disappears (i.e., as the returns approach the homoscedastic, or constant variance, limit), the approximation is exact and you can apply the Box & Jenkins confidence bounds (see Box, Jenkins, and Reinsel [8], pages 133-145).

For conditional mean models with regression components (i.e., X or XF is specified), `MeanRMSE` is an empty matrix, []. In other words, `garchpred` computes `MeanRMSE` only if the conditional mean is modeled as a stationary invertible ARMA process. See “Regression Components in Conditional Mean Models” on page 7-1.

Presample Observations

As mentioned in “Minimum Mean Square Error Forecasting” on page 6-2, `garchpred` computes MMSE forecasts by applying iterated conditional expectations to the conditional mean and variance models one forecast period at a time. Since these models are generally recursive in nature, they often require presample data to initiate the iterative forecasting process. This initial data plays the identical role that the presample time-series inputs `PreInnovations`, `PreSigmas`, and `PreSeries` play in simulation (see `garchsim`) and estimation (see `garchfit` and `garchinfer`).

Since the time-series array of asset returns, `Series`, is a required input, `garchpred` simply takes any initial returns it needs to initiate forecasting of the conditional mean directly from the last (i.e., most recent) rows of `Series`. For example, consider a conditional mean model with an AR(R) autoregressive component. In this case, `garchpred` takes the R observations required to initiate the forecast of each realization of `Series` directly from the last R rows of `Series`.

However, `garchpred` obtains any initial innovations and conditional standard deviations needed to initiate forecasting of the conditional variance model from the input array `Series` via the inverse filtering inference engine `garchinfer`.

For additional details regarding estimation and inverse filtering, see “Maximum Likelihood Estimation” on page 5-2, “Presample Observations” on page 5-11, and the `garchinfer` function.

Asymptotic Behavior for Long-Range Forecast Horizons

If you are working with long-range forecast horizons, the following asymptotic behaviors hold for the outputs of garchpred:

- As mentioned earlier in this section, the conditional standard deviation forecast (i.e., the first garchpred output, sigmaForecast) approaches the unconditional standard deviation of $\{\varepsilon_t\}$. For GARCH(P,Q) models it is given by

$$\sigma = \sqrt{\frac{\kappa}{1 - \sum_{i=1}^P G_i - \sum_{j=1}^Q A_j}}$$

For GJR(P,Q) models, it is given by

$$\sigma = \sqrt{\frac{\kappa}{1 - \sum_{i=1}^P G_i - \sum_{j=1}^Q A_j - \frac{1}{2} \sum_{j=1}^Q L_j}}$$

And for EGARCH(P,Q) models, it is given by

$$\sigma = \sqrt{e^{\frac{\kappa}{1 - \sum_{i=1}^P G_i}}}$$

- GARCH effects do not affect the MMSE forecast of the conditional mean (i.e., the second garchpred output, meanForecast). The forecast approaches the unconditional mean of $\{y_t\}$ as in the constant variance case. That is, the presence of GARCH effects introduces dependence in the variance process, and only affects the uncertainty of the mean forecast, leaving the mean forecast itself unchanged.
- The mean square error of the conditional mean (i.e., the square of the fourth garchpred output, meanRMSE.^2) approaches the unconditional variance of $\{y_t\}$.
- EGARCH(P,Q) models represent the logarithm of the conditional variance as the output of a linear filter, rather than the conditional variance process

itself. Because of this, the MMSE forecasts derived from EGARCH(P,Q) models are optimal for the logarithm of the conditional variance, but are generally downward-biased forecasts of the conditional variance process itself. Since the output arrays `SigmaForecast`, `SigmaTotal`, and `MeanRMSE` are based on the conditional variance forecasts, these outputs generally underestimate their true expected values for conditional variance forecasts derived from EGARCH(P,Q) models. The important exception is the one-period ahead forecast, which is unbiased in all cases. For unbiased multiperiod forecasts of `SigmaForecast`, `SigmaTotal`, and `MeanRMSE`, you can perform Monte Carlo simulation via `garchsim` (see “Advanced Example” on page 10-1).

Examples

- “Computing a Forecast” on page 6-8
- “Volatility Forecasts over Multiple Periods” on page 6-11
- “Computing a Forecast with Multiple Realizations” on page 6-14

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

Computing a Forecast

The section “Analysis and Estimation Example Using the Default Model” on page 2-16 uses the default GARCH(1,1) model to model the Deutschmark/British pound foreign exchange series. This example uses the resulting model

$$y_t = -6.1919e-005 + \varepsilon_t$$

$$\sigma_t^2 = 1.0761e-006 + 0.80598\sigma_{t-1}^2 + 0.15313\varepsilon_{t-1}^2$$

to demonstrate the use of the forecasting function `garchpred`.

- 1 Use the following commands to restore your workspace if necessary. The following text omits the display output of the estimation to save space.

```
load garchdata
dem2gbp = price2ret(DEM2GBP);
[coeff,errors,LLF,innovations,sigmas] = garchfit(dem2gbp);
garchdisp(coeff,errors)
```

```
Mean: ARMAX(0,0,0); Variance: GARCH(1,1)
```

Conditional Probability Distribution: Gaussian
 Number of Model Parameters Estimated: 4

Parameter	Value	Standard Error	T Statistic
C	-6.1919e-005	8.4331e-005	-0.7342
K	1.0761e-006	1.323e-007	8.1341
GARCH(1)	0.80598	0.016561	48.6685
ARCH(1)	0.15313	0.013974	10.9586

- 2** Call `garchpred` to forecast the returns for the Deutschmark/British pound foreign exchange series using the default model parameter estimates. Provide the specification structure `coeff` (the output of `garchfit`) and the FX return series `dem2gbp`, and the number of forecast periods as input.

Note Example results below are displayed in **Short E** numeric format for readability. Select **File -> Preferences -> Command Window -> Text display: short e** before starting the example to duplicate this format.

Use the following command to forecast the conditional mean and standard deviation in each period of a 10-period forecast horizon.

```
[sigmaForecast,meanForecast] = garchpred(coeff,dem2gbp,10);
[sigmaForecast,meanForecast]
```

```
ans =
```

```
3.8340e-003 -6.1919e-005
3.8954e-003 -6.1919e-005
3.9535e-003 -6.1919e-005
4.0084e-003 -6.1919e-005
4.0603e-003 -6.1919e-005
4.1095e-003 -6.1919e-005
4.1562e-003 -6.1919e-005
4.2004e-003 -6.1919e-005
4.2424e-003 -6.1919e-005
```

4.2823e-003 -6.1919e-005

The result consists of the MMSE forecasts of the conditional standard deviations and the conditional mean of the return series dem2gbp for a 10-period default horizon. They show that the default model forecast of the conditional mean is always $C = -6.1919e-05$. This is true for any forecast horizon because the expected value of any innovation, ε_t , is 0.

The conditional standard deviation forecast (`sigmaForecast`) changes from period to period and approaches the unconditional standard deviation of $\{\varepsilon_t\}$, given by

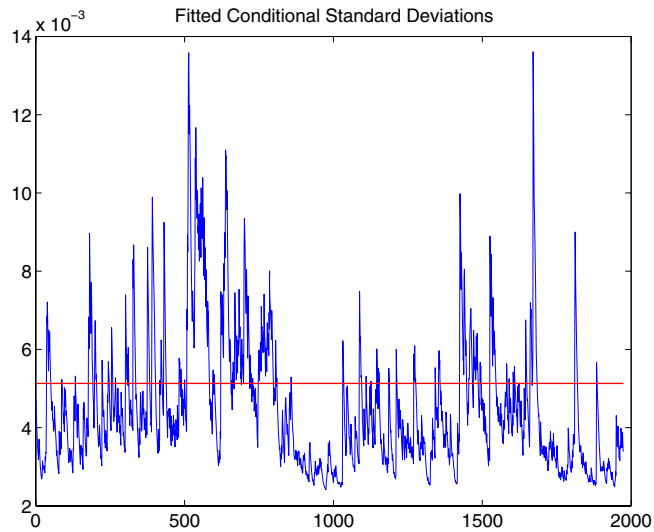
$$\sigma = \sqrt{\frac{K}{1 - \sum_{i=1}^P G_i - \sum_{j=1}^Q A_j}}$$

- 3** Calculate the unconditional standard deviation of $\{\varepsilon_t\}$ as

```
s0 = sqrt(coeff.K/(1 - sum([coeff.GARCH(:);coeff.ARCH(:)])))
s0 =
    5.1300e-003
```

- 4** Plot the unconditional standard deviation, 5.1300e-003, and the conditional standard deviations, `sigmas`, derived from the fitted returns. The plot reveals that the most recent values of σ_t fall below this long-run, asymptotic value.

```
plot(sigmas), hold('on')
plot([0 size(sigmas,1)],[s0 s0],'red')
title('Fitted Conditional Standard Deviations')
hold('off')
```

Volatility Forecasts over Multiple Periods

In addition to computing conditional mean and volatility forecasts on a per-period basis, `garchpred` also computes volatility forecasts of returns for assets held for multiple periods. For example, you could forecast the standard deviation of the return you would obtain if you purchased shares in a mutual fund that mirrors the performance of the New York Stock Exchange Composite Index today, and sold it 10 days from now.

- 1 Use the default GARCH(1,1) model (“The Default Model” on page 2-13) to estimate the model parameters for the NYSE data set. The following text omits the display output of the estimation to save space.

```
load garchdata
nyse = price2ret(NYSE);
[coeff,errors,LLF,innovations,sigmas] = garchfit(nyse);
garchdisp(coeff,errors)
```

```
Mean: ARMAX(0,0,0); Variance: GARCH(1,1)
```

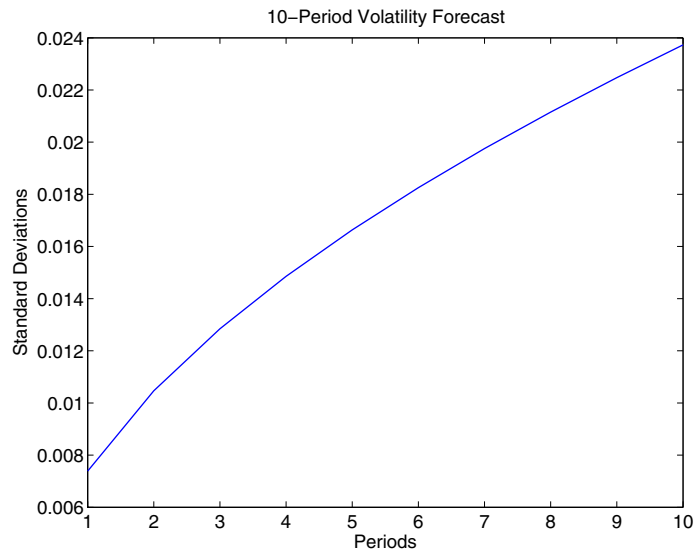
```
Conditional Probability Distribution: Gaussian
Number of Model Parameters Estimated: 4
```

Parameter	Value	Standard Error	T Statistic
C	0.00049676	0.00013137	3.7813
K	8.9128e-007	1.5776e-007	5.6495
GARCH(1)	0.91088	0.0069142	131.7410
ARCH(1)	0.079942	0.0058319	13.7077

- 2** Now, forecast and plot the standard deviation of the return you would obtain if you sold the shares after 10 days.

```
[sigmaForecast,meanForecast,sigmaTotal] = garchpred(coeff,...
                                                    nyse,10);

plot(sigmaTotal)
ylabel('Standard Deviations')
xlabel('Periods')
title('10-Period Volatility Forecast')
hold('off')
```



This plot represents the standard deviation of the returns (`sigmaTotal`) expected if you held the shares for the number of periods shown on the *x*-axis. The value for the tenth period is the volatility forecast of the expected return if you purchased the shares today and held them for 10 periods.

Note that the calculation of `sigmaTotal` is strictly correct for continuously compounded returns only, and is an approximation for periodically compounded returns.

- 3** If you convert the standard deviations `sigmaForecast` and `sigmaTotal` to variances by squaring each element, you can see an interesting relationship between the cumulative sum of `sigmaForecast.^2` and `sigmaTotal.^2`.

```
format short e
[cumsum(sigmaForecast.^2) sigmaTotal.^2]
```

ans =

```
5.4587e-005  5.4587e-005
1.0956e-004  1.0956e-004
1.6493e-004  1.6493e-004
```

```
2.2068e-004 2.2068e-004
2.7680e-004 2.7680e-004
3.3331e-004 3.3331e-004
3.9018e-004 3.9018e-004
4.4743e-004 4.4743e-004
5.0504e-004 5.0504e-004
5.6302e-004 5.6302e-004
```

Although not exactly equivalent, this relationship in the presence of heteroscedasticity is similar to the familiar square-root-of-time rule for converting constant variances of uncorrelated returns expressed on a per-period basis to a variance over multiple periods. This relationship between `sigmaForecast` and `sigmaTotal` holds for the default conditional mean model only (i.e., the relationship is valid for uncorrelated returns).

Computing a Forecast with Multiple Realizations

This example illustrates how to forecast multiple realizations of an MA(1) conditional mean model with an EGARCH(1,1) conditional variance model.

- 1 Load the NYSE data set and convert prices to returns.

```
load garchdata
nyse = price2ret(NYSE);
```

- 2 Create a specification structure template, and estimate and display the estimation results,

```
spec = garchset('VarianceModel','EGARCH','M',1,'P',1,'Q',1,...
               'Display','off');
[coeff,errors] = garchfit(spec,nyse);
garchdisp(coeff,errors)
```

```
Mean: ARMAX(0,1,0); Variance: EGARCH(1,1)
```

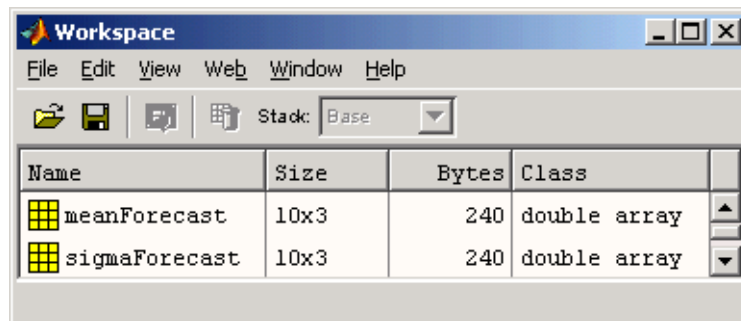
```
Conditional Probability Distribution: Gaussian
Number of Model Parameters Estimated: 6
```

Parameter	Value	Standard Error	T Statistic
C	0.00022434	0.00014038	1.5981
MA(1)	0.10677	0.018795	5.6806
K	-0.25399	0.031452	-8.0755
GARCH(1)	0.97329	0.003231	301.2365
ARCH(1)	0.14514	0.011845	12.2533
Leverage(1)	-0.10359	0.0081483	-12.7128

- 3** Based on the estimation results, simulate 1000 observations for each of three independent realizations and forecast the conditional standard deviations and returns for a 10-period forecast horizon,

```
[innovations,sigmas,series] = garchsim(coeff,1000,3);
[sigmaForecast,meanForecast] = garchpred(coeff,series,10);
```

Examination of the MATLAB workspace reveals that both `sigmaForecast` and `meanForecast` outputs are 10-by-3 arrays. Both arrays have the same number of rows as the specified number of periods. The first row contains the standard deviations and mean forecasts for the first period, and the last row contains these values for the most recent period. Both arrays have the same number of columns as there are realizations, i.e., columns, in the simulated return series, `series`.



Regression Components in Conditional Mean Models

Introduction (p. 7-2)	Introduces the concept of a regression component in the conditional mean model.
Incorporating a Regression Model in an Estimation (p. 7-3)	Shows you how to perform an estimation when the conditional mean model includes a regression component.
Simulation and Inference Using a Regression Component (p. 7-9)	Explains the syntax for including a matrix of explanatory data, i.e., a regression matrix, in calls to <code>garchsim</code> and <code>garchinfer</code> .
Forecasting Using a Regression Component (p. 7-10)	Discusses the need for both explanatory and forecast explanatory data when you incorporate a regression component in a forecast.
Regression in a Monte Carlo Framework (p. 7-14)	Considers Monte Carlo simulation that includes a regression component.

Introduction

The GARCH Toolbox allows conditional mean models with regression components, i.e., of general ARMAX(R,M,Nx) form.

$$y_t = C + \sum_{i=1}^R \phi_i y_{t-i} + \varepsilon_t + \sum_{j=1}^M \theta_j \varepsilon_{t-j} + \sum_{k=1}^{Nx} \beta_k X(t, k)$$

with regression coefficients β_k , and explanatory regression matrix X in which each column is a time series and $X(t, k)$ denotes the t th row and k th column.

Conditional mean models with a regression component introduce additional complexity in the sense that the toolbox functions have no way of knowing what the explanatory data represents or how it was generated. This is in contrast to ARMA models, which have an explicit forecasting mechanism and well-defined stationarity/invertibility requirements.

All the primary functions (i.e., `garchfit`, `garchinfer`, `garchpred`, and `garchsim`) accept an optional regression matrix, X , that represents X in the equation above. You must ensure that the regression matrix you provide is valid and you must

- Collect and format the past history of explanatory data you include in X .
- For forecasting, forecast X into the future to form XF .

Incorporating a Regression Model in an Estimation

This section uses the asymptotic equivalence of autoregressive models and linear regression models to illustrate the use of a regression component. The example has two parts:

- “Fitting a Model to a Simulated Return Series” on page 7-3
- “Fitting a Regression Model to the Same Return Series” on page 7-5

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

Fitting a Model to a Simulated Return Series

This section uses an AR(R)/GARCH(P,Q) model to fit a simulated return series to the defined model.

1 Define an AR(2)/GARCH(1,1) model. Start by creating a specification structure for an AR(2)/GARCH(1,1) composite model. Set the 'Display' parameter 'off' to suppress the optimization details that garchfit normally displays.

```
spec = garchset('AR',[0.5 -0.8],'C',0,'Regress',[0.5 -0.8],...
               'GARCH',0.7,'ARCH',0.1,'K',0.005,...
               'Display','off')
```

```
spec =
    Comment: 'Mean: ARMAX(2,0,?); Variance: GARCH(1,1)'
    Distribution: 'Gaussian'
           R: 2
           C: 0
           AR: [0.5000 -0.8000]
           Regress: [0.5000 -0.8000]
```

```
VarianceModel: 'GARCH'
      P: 1
      Q: 1
      K: 0.0050
      GARCH: 0.7000
      ARCH: 0.1000
      Display: 'off'
```

Notice that in this specification structure, spec

- The model order fields R, M, P, and Q are consistent with the number of coefficients in the AR, MA, GARCH, and ARCH vectors, respectively.
- Although the Regress field indicates two regression coefficients, the Comment field still contains a question mark as a placeholder for the number of explanatory variables.
- There is no model order field for the Regress vector, analogous to the R, M, P, and Q orders of an ARMA(R,M)/GARCH(P,Q) model.

2 Fit the model to a simulated return series. Simulate 2000 observations of the innovations, conditional standard deviations, and returns for the AR(2)/GARCH(1,1) process defined in spec. Use the model defined in spec to estimate the parameters of the simulated return series and then compare the parameter estimates to the original coefficients in spec.

```
[e,s,y] = garchsim(spec,2000,1,0);
[coeff,errors] = garchfit(spec,y);
garchdisp(coeff,errors)
```

```
Mean: ARMAX(2,0,0); Variance: GARCH(1,1)
```

```
Conditional Probability Distribution: Gaussian
Number of Model Parameters Estimated: 6
```

Parameter	Value	Standard Error	T Statistic
C	-0.00044755	0.0034623	-0.1293
AR(1)	0.50257	0.01392	36.1049
AR(2)	-0.8002	0.013981	-57.2344
K	0.0050532	0.001971	2.5637

GARCH(1)	0.70954	0.095319	7.4439
ARCH(1)	0.083296	0.022665	3.6752

The estimated parameters, shown in the Value column, are quite close to the true coefficients in spec.

Because you specified no explanatory regression matrix as input to `garchsim` and `garchfit`, these functions ignore the regression coefficients (Regress). The `garchdisp` output shows a 0 for the order of the regression component.

Fitting a Regression Model to the Same Return Series

To illustrate the use of a regression matrix, fit the return series `y`, an AR(2) process in the mean, to a regression model with two explanatory variables. The regression matrix consists of the first- and second-order lags of the simulated return series `y`. The return series `y` was simulated in the previous topic, “Fitting a Model to a Simulated Return Series” on page 7-3.

1 Remove the AR component. First, remove the AR component from the specification structure.

```
spec = garchset(spec, 'R', 0, 'AR', [])
spec =
    Comment: 'Mean: ARMAX(0,0,?); Variance: GARCH(1,1)'
    Distribution: 'Gaussian'
    C: 0
    Regress: [0.5000 -0.8000]
    VarianceModel: 'GARCH'
    P: 1
    Q: 1
    K: 0.0050
    GARCH: 0.7000
    ARCH: 0.1000
    Display: 'off'
```

2 Create the regression matrix. Create a regression matrix of first- and second-order lags using the simulated returns vector `y` from “Fitting a Model to a Simulated Return Series” on page 7-3 as input. Examine the first 10 rows of `y` and the corresponding rows of the lags.

```
X = lagmatrix(y,[1 2]);
[y(1:10) X(1:10,:)]
ans =
    0.0562         NaN         NaN
    0.0183    0.0562         NaN
   -0.0024    0.0183    0.0562
   -0.1506   -0.0024    0.0183
   -0.3937   -0.1506   -0.0024
   -0.0867   -0.3937   -0.1506
    0.1075   -0.0867   -0.3937
    0.2225    0.1075   -0.0867
    0.1044    0.2225    0.1075
    0.1288    0.1044    0.2225
```

3 Examine the regression matrix. A NaN (an IEEE arithmetic standard for Not-a-Number) in the resulting matrix X indicates the presence of a missing observation. If you use X to fit a regression model to y , `garchfit` produces an error.

```
[coeff,errors] = garchfit(spec,y,X);
??? Error using ==> garchfit
Regression matrix 'X' has insufficient number of observations.
```

The error occurs because there are fewer valid rows (i.e., those rows without a NaN) in the regression matrix X than there are observations in y . The returns vector y has 2000 observations but the most recent number of valid observations in X is only 1998.

4 Repair the regression matrix. You can do one of two things in order to proceed. For a return series of this size it makes little difference which option you choose:

- Strip off the first two observations in y .
- Replace all NaNs in X with some reasonable value.

This example continues by replacing all NaNs with the sample mean of y . Use the MATLAB function `isnan` to identify NaNs and the function `mean` to compute the mean of y .

```
X(isnan(X)) = mean(y);
[y(1:10), X(1:10,:)]
```

```
ans =
    0.0562    0.0004    0.0004
    0.0183    0.0562    0.0004
   -0.0024    0.0183    0.0562
   -0.1506   -0.0024    0.0183
   -0.3937   -0.1506   -0.0024
   -0.0867   -0.3937   -0.1506
    0.1075   -0.0867   -0.3937
    0.2225    0.1075   -0.0867
    0.1044    0.2225    0.1075
    0.1288    0.1044    0.2225
```

Note If the number of valid rows in X exceeds the number of observations in y , then `garchfit` includes in the estimation only the most recent rows of X , equal to the number of observations in y .

5 Fit the regression model. Now that the explanatory regression matrix X is compatible with the return series vector y , use `garchfit` to estimate the model coefficients for the return series using the regression matrix, and display the results.

```
[coeffX,errorsX] = garchfit(spec,y,X);
garchdisp(coeffX,errorsX)
```

Mean: ARMAX(0,0,2); Variance: GARCH(1,1)

Conditional Probability Distribution: Gaussian
 Number of Model Parameters Estimated: 6

Parameter	Value	Standard Error	T Statistic
C	-0.00044754	0.0034628	-0.1292
Regress(1)	0.50257	0.01392	36.1048
Regress(2)	-0.8002	0.013981	-57.2346
K	0.0050526	0.0019708	2.5637
GARCH(1)	0.70957	0.095311	7.4447

ARCH(1)	0.083292	0.022663	3.6752
---------	----------	----------	--------

These estimation results are similar to those shown for the AR model in the section “Fitting a Model to a Simulated Return Series” on page 7-3. This similarity illustrates the asymptotic equivalence of autoregressive models and linear regression models.

By illustrating the extra steps involved in formatting the explanatory matrix, this part of the example also highlights the additional complexity involved in modeling conditional means with regression components.

Simulation and Inference Using a Regression Component

Including a regression component with `garchsim` and `garchinfer` is similar to including one with `garchfit`. (See “Incorporating a Regression Model in an Estimation” on page 7-3.)

For example, the following command simulates a single realization of 2000 observations of the innovations, conditional standard deviations, and returns.

```
[e,s,y] = garchsim(spec,2000,1,[],X);
```

You can also use the same regression matrix X to infer the innovations and conditional standard deviations from the returns.

```
[eInfer,sInfer] = garchinfer(spec,y,X);
```

Forecasting Using a Regression Component

Inclusion of a regression component in forecasting is also similar to including one in an estimation. However, in addition to the explanatory data, you must consider the use of forecasted explanatory data.

This section discusses

- “Forecasted Explanatory Data” on page 7-10
- “Generating Forecasted Explanatory Data” on page 7-11
- “Ordinary Least Squares Regression” on page 7-12

Forecasted Explanatory Data

If you want to forecast the conditional mean of a return series y in each period of a 10-period forecast horizon, the correct calling syntax for `garchpred` is

```
NumPeriods = 10;  
[sigmaForecast,meanForecast] = ...  
    garchpred(spec,y,NumPeriods,X,XF);
```

where X is the same regression matrix shown in “Fitting a Regression Model to the Same Return Series” on page 7-5, and XF is a regression matrix of forecasted explanatory data. In fact, XF represents a projection into the future of the explanatory data in X . Note that the command above produces an error if you execute it in your current workspace because XF is missing.

XF must have the same number of columns as X . In each column of XF , the first row contains the one-period-ahead forecast, the second row the two-period-ahead forecast, and so on. If you specify XF , the number of rows (forecasts) in each column must equal or exceed the forecast horizon, `NumPeriods`. When the number of forecasts in XF exceeds the forecast horizon, `garchpred` uses only the first `NumPeriods` forecasts. If XF is empty (`[]`) or missing, the conditional mean forecast, `meanForecast`, has no regression component.

If you used a regression matrix, X , for simulation and/or estimation, then you should also use a regression matrix when calling `garchpred`. This is because `garchpred` requires a complete conditional mean specification to correctly infer the innovations $\{\varepsilon_t\}$ from the observed return series $\{y_t\}$. Typically, the same regression matrix is used for simulation, estimation, and forecasting.

Forecasting Only the Conditional Standard Deviation

To forecast the conditional standard deviation (i.e., `sigmaForecast`), `XF` is unnecessary, and `garchpred` ignores it if it is present. This is true even if you included the matrix `X` in the simulation and/or estimation process.

For example, you could use the following syntax to forecast only the conditional standard deviation of the innovations $\{\varepsilon_t\}$ over a 10-period forecast horizon.

```
sigmaForecast = garchpred(spec,y,10,X);
```

Forecasting the Conditional Mean

To forecast the conditional mean (i.e., `meanForecast`), if you specify `X`, you must also specify `XF`.

For example, to forecast the conditional mean of the return series `y` over a 10-period forecast horizon,

```
[sigmaForecast,meanForecast] = garchpred(spec,y,10,X,XF);
```

Generating Forecasted Explanatory Data

Typically, the regression matrix `X` contains the observed returns of a suitable market index, collected over the same time interval as the observed data of interest. In this case, `X` is most likely a vector, corresponding to a single explanatory variable, and you must devise some way of generating the forecast of `X` (i.e., `XF`).

One approach, using the GARCH Toolbox, is to first use `garchfit` to fit a suitable `ARMA(R,M)` model to the returns in `X`, then use `garchpred` to forecast the market index returns into the future. Specifically, since you're not interested in fitting the volatility of `X`, you can simplify the estimation process by assuming a constant conditional variance model, e.g., `ARMA(R,M)/GARCH(0,0)`.

Ordinary Least Squares Regression

The following example illustrates an ordinary least squares regression by simulating a return series that scales the daily return values of the New York Stock Exchange Composite Index. It also provides an example of a constant conditional variance model.

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

- 1 Load the NYSE data set and convert the price series to a return series.

```
load garchdata
nyse = price2ret(NYSE);
```

- 2 Create a specification structure. Set the Display flag to 'off' to suppress the optimization details that garchfit normally displays.

```
spec = garchset('P',0,'Q',0,'C',0,'Regress',1.2,'K',0.00015,...
               'Display','off')
```

```
spec =
    Comment: 'Mean: ARMAX(0,0,?); Variance: GARCH(0,0) '
    Distribution: 'Gaussian'
           C: 0
           Regress: 1.2000
    VarianceModel: 'GARCH'
           K: 1.5000e-004
           Display: 'off'
```

- 3 Simulate a single realization of 2000 observations, fit the model, and examine the results.

```
[e,s,y] = garchsim(spec,2000,1,0,nyse);
[coeff,errors] = garchfit(spec,y,nyse);
```

```
garchdisp(coeff,errors)
```

```
Mean: ARMAX(0,0,1); Variance: GARCH(0,0)
```

```
Conditional Probability Distribution: Gaussian
```

```
Number of Model Parameters Estimated: 3
```

Parameter	Value	Standard Error	T Statistic
C	4.9091e-006	0.00027114	0.0181
Regress(1)	1.2251	0.028909	42.3786
K	0.00014662	4.6945e-006	31.2334

These estimation results are just the ordinary least squares (OLS) regression results. In fact, in the absence of GARCH effects and assuming Gaussian innovations, maximum likelihood estimation and least squares regression are the same thing.

Note This example is shown purely for illustration purposes. Although you can use the GARCH Toolbox to perform OLS, it is computationally inefficient and is not recommended.

Regression in a Monte Carlo Framework

In the general case, the functions `garchsim`, `garchinfer`, and `garchpred` process multiple realizations (i.e., sample paths) of univariate time series. That is, the outputs of `garchsim`, as well as the observed return series input to `garchpred` and `garchinfer`, can be time-series matrices in which each column represents an independent realization. `garchfit` is different, in that the input observed return series of interest must be a vector (i.e., a single realization).

When simulating, inferring, and forecasting multiple realizations, the appropriate toolbox function applies a given regression matrix X to each realization of a univariate time series. For example, in the following command, `garchsim` applies a given X matrix to all 10 columns of the output series $\{\varepsilon_t\}$, $\{\sigma_t\}$, and $\{y_t\}$.

```
NumSamples = 100;  
NumPaths = 10;  
[e,s,y] = garchsim(spec,NumSamples,NumPaths,[],X);
```

In a true Monte Carlo simulation of this process, including a regression component, you would call `garchsim` inside a loop 10 times, once for each path. Each iteration would pass in a unique realization of X and produce a single-column output.

Univariate Unit Root Tests

Introduction (p. 8-2)	Introduces augmented Dickey-Fuller and Phillips-Perron univariate unit root tests.
Dickey-Fuller Tests (p. 8-3)	Describes three augmented Dickey-Fuller unit root tests: <code>dfARTest</code> , <code>dfARDTest</code> , and <code>dftSTest</code> .
Phillips-Perron Tests (p. 8-5)	Describes three Phillips-Perron unit root tests: <code>ppARTest</code> , <code>ppARDTest</code> , and <code>ppTSTest</code> .
Interpretation of Results (p. 8-9)	Explains some pitfalls in the interpretation of unit root tests.
Examples (p. 8-10)	Conducts two tests, one with a trend stationary component, and a second with a drift component.

Introduction

The GARCH Toolbox supports several members of the Phillips-Perron and augmented Dickey-Fuller classes of univariate unit root tests. The test statistics for these popular tests are straightforward to evaluate by ordinary least-squares regression, but many of the most common parametric cases follow nonstandard distributions. Therefore, the test statistics need to be compared to critical values derived from Monte Carlo simulations.

Critical Values

Critical values from the simulations are derived for various combinations of sample size and significance level. The significance level sets the probability of a Type I error of incorrectly rejecting the null hypothesis of the underlying process when it is true. Specifically, five million Monte Carlo trials of a given sample size are generated using independent, identically distributed standard Gaussian disturbances. For each sample size, tabulated critical values are the quantiles associated with given cumulative probabilities — significance levels — of the simulated test statistic.

The test suite supports sample sizes as small as 10 and significance levels ranging from 0.001 to 0.999. For small samples, the critical values are exact only for Gaussian residuals. As the sample size becomes larger, critical values are also valid for non-Gaussian residuals. All univariate unit root tests are designed as conventional single-tailed tests.

Serial Dependence

Although augmented Dickey-Fuller and Phillips-Perron tests both attempt to compensate for serial dependence in the residuals process, they do so in different ways. For a given parametric specification of the null hypothesis, Phillips-Perron tests retain the same OLS (ordinary least squares) regression model, but they adjust the test statistics to account for serially dependent residuals. The augmented Dickey-Fuller tests, by contrast, add lagged changes of the observed time series as explanatory variables in the OLS regression model. Hamilton [19] and Greene [16] contain more discussion of these tests.

Dickey-Fuller Tests

The GARCH Toolbox supports three augmented Dickey-Fuller unit root hypothesis tests:

- “dfARTest” on page 8-3
- “dfARDTest” on page 8-4
- “dfTSTest” on page 8-4

In the equations below, define y_t and ε_t as the univariate time series of observed data and model residuals, respectively. Also, define the first difference operator Δ such that $\Delta y_t = y_t - y_{t-1}$.

dfARTest

The first form of the augmented Dickey-Fuller unit root test assumes that a zero drift unit root process underlies the observed time series y_t . Specifically, under the null hypothesis, the true underlying process is a zero drift ARIMA(P,1,0) model

$$y_t = y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$$

which is equivalent to an integrated AR(P+1) model.

As an alternative, the estimated OLS regression model is

$$y_t = \phi y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$$

for some AR(1) coefficient $\phi < 1$.

dfARDTest

The second form of the augmented Dickey-Fuller unit root test also assumes that a zero drift unit root process underlies the observed time series y_t . Specifically, under the null hypothesis, the true underlying process is a zero drift ARIMA(P,1,0) model

$$y_t = y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$$

which is equivalent to an integrated AR(P+1) model.

In this case, the alternative estimated OLS regression model is

$$y_t = C + \phi y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$$

for some constant C and AR(1) coefficient $\phi < 1$.

dfTSTest

The third form of the augmented Dickey-Fuller unit root test assumes that a unit root process with arbitrary drift underlies the observed time series y_t . Specifically, under the null hypothesis, the true process underlying the observed time series y_t is an ARIMA(P,1,0) model with drift

$$y_t = C + y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$$

which is equivalent to an integrated AR(P+1) model.

As an alternative, the estimated OLS regression model includes a time trend,

$$y_t = C + \phi y_{t-1} + \delta t + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$$

for some constant C, AR(1) coefficient $\phi < 1$, and time trend stationary coefficient δ .

Phillips-Perron Tests

The GARCH Toolbox supports three Phillips-Perron unit root hypothesis tests:

- “ppARTest” on page 8-5
- “ppARDTest” on page 8-6
- “ppTSTest” on page 8-6

In the equations below, define y_t and ε_t as the univariate time series of observed data and model residuals, respectively.

ppARTest

The first form of the Phillips-Perron unit root test assumes that a zero drift unit root process underlies the observed time series y_t . Under the null hypothesis, the assumed underlying process is

$$y_t = y_{t-1} + \varepsilon_t$$

As an alternative, the estimated OLS regression model is

$$y_t = \phi y_{t-1} + \varepsilon_t$$

for some AR(1) coefficient $\phi < 1$.

ppARDTest

The second form of the Phillips-Perron unit root test also assumes that a zero drift unit root process underlies the observed time series y_t . Under the null hypothesis, the assumed underlying process is

$$y_t = y_{t-1} + \varepsilon_t$$

As an alternative, the estimated OLS regression model is

$$y_t = C + \phi y_{t-1} + \varepsilon_t$$

for some constant C and AR(1) coefficient $\phi < 1$.

ppTSTest

The third form of the Phillips-Perron unit root test assumes that a unit root process with arbitrary drift underlies the observed time series y_t . Under the null hypothesis, the assumed underlying process is

$$y_t = C + y_{t-1} + \varepsilon_t$$

for an arbitrary constant C. As an alternative, the estimated OLS regression model is

$$y_t = C + \phi y_{t-1} + \delta t + \varepsilon_t$$

for some constant C, AR(1) coefficient $\phi < 1$, and time trend stationary coefficient δ .

How to Test for Unit Roots: Inputs and Outputs

All of the Dickey-Fuller and Phillips-Perron functions share a common interface. In addition to a univariate time series y_t to be tested, all functions accept

- An integer input vector `Lags`
- A vector `Alpha` to set significance levels
- A character string `TestType` to select the form of the test

The output vectors are `H`, `PValue`, `TestStat`, and `CriticalValue`.

See “Examples” on page 8-10 for illustrations of the syntax required for both inputs and outputs. The examples also illustrate how to interpret the results of the tests.

Lags

The input vector `Lags` always serves as a correction for serial correlation of residuals. The precise meaning of the vector, however, differs between the Dickey-Fuller and the Phillips-Perron tests:

- In Dickey-Fuller tests, `Lags` indicates the number of lagged changes or first differences in y_t that are included in the OLS regression model. It is denoted by p in the Dickey-Fuller equations.
- In Phillips-Perron tests, `Lags` indicates the number of lagged autocovariance terms included in the Newey-West estimation of the asymptotic variance of the sample mean of residuals.

In all cases, setting `Lags = 0` applies no correction for serial correlation, and the Dickey-Fuller and Phillips-Perron tests produce identical results.

Significance Level

The significance level `Alpha` always denotes the same probability, or set of probabilities, for all six unit root tests. The significance level is the probability — in the appropriate tail of the distribution — of rejecting the null hypothesis when it is in fact true and should be accepted. See “Critical Values” on page 8-2.

TestType

The input `TestType` specifies the basic form of the test used to construct the test statistic. The three test types are *AR*, *t*, and *F*. All three tests are conventional, single-tailed tests.

AR and *t* Tests

Suppose you conduct a Dickey-Fuller or Phillips-Perron test where you specify no correction for serial dependence (`Lags = 0`). Two possibilities exist:

- Set `TestType = 'AR'` to select a unit root test based on the AR(1) regression coefficient ϕ without the need to calculate the standard error. In this case, the test statistic χ based on T observations of y_t is $\chi = T(\phi - 1)$.
- Set `TestType = 't'` to select a unit root test based on the studentized t test. In this case, the test statistic χ based on the AR(1) regression coefficient ϕ and its standard error σ_ϕ is $\chi = (\phi - 1)/\sigma_\phi$.

When you specify a correction for serial dependence (`Lags > 0`), the test function adjusts the computation of χ . See Hamilton (1994) for details.

Both the *AR* and the *t* test are lower tailed tests, where the null hypothesis is rejected if the test statistic is less than the critical value.

F Tests

Two Dickey-Fuller tests, `dfARDTest` and `dfTSTest`, let you specify joint OLS *F* tests. For `dfARDTest`, the *F* test is of a unit root ($\phi = 1$) with zero drift ($C = 0$). For `dfTSTest`, the *F* test is of a unit root ($\phi = 1$) with a zero trend stationary coefficient ($\delta = 1$). In both cases, the joint *F* test is an upper-tailed test. Reject the null hypothesis if the test statistic is greater than the critical value.

Outputs

The six unit root tests return the same set of output arguments. The first output is a vector of logical indicators, `H`. `H = 0` indicates acceptance of the null hypothesis, and `H = 1` indicates rejection of the null hypothesis. Each element of `H` corresponds to a particular lag of `Lags` and significance level of `Alpha`. Each element of `H` also corresponds to an output vector of *p*-values called `PValue`, a vector of test statistics called `TestStat`, and a vector of critical values called `CriticalValue`.

Interpretation of Results

Analysts often associate rejection of the null unit root hypothesis with the assertion of a stationary AR(1) model. They assume acceptance of the alternative hypothesis implies that the time series y_t is stationary. This assumption is correct in most — but not all — practical applications.

Here is why you should interpret test results with care. An AR(1) model is stationary if and only if the magnitude of the AR coefficient is strictly less than 1 (i.e., $|\phi| < 1$). Assume, for example, that the AR coefficient estimated by OLS is $\phi = -2$. A test statistic based on this coefficient is well under the applicable critical value. You correctly reject the null hypothesis. Yet the time series is nonstationary!

Another pitfall is to confuse unit root tests with random walk tests. For a unit root model to be a random walk, the residuals are generally assumed to be independent and identically distributed Gaussian random variables. Other forms of random walk exist, but all require the residuals to be at least uncorrelated. Since unit root tests are often designed to compensate for serial correlation, unit root processes are more general than random walks. Put another way, a random walk is subsumed by the unit root null hypothesis, but so are many other processes.

Given a stationary process of finite sample size, a unit root process exists that describes it arbitrarily well. In light of that, use a unit root test to formulate a well-performing, simple representation of an observed time series. Do not use the test only to determine whether or not the true underlying process actually contains a unit root.

Examples

The following examples make use of two common, easily-accessible economic time series. Both series were downloaded directly from the U.S. Federal Reserve Economic Data (FRED) web site maintained by the Federal Reserve Bank of St. Louis (<http://research.stlouisfed.org/fred/>):

- “Test GDP by OLS Regression with a Stationary Component” on page 8-11
- “Test T-Bill Rate by OLS Regression with a Drift Component” on page 8-16

The first example is a quarterly time series of seasonally adjusted, annualized, real Gross Domestic Product (GDP) of the United States from January 1, 1947 to April 1, 2005, quoted in billions of year 2000 U.S. dollars, for a total of 234 quarterly observations (Series GDPC96).

The second is a monthly time series of the three-month U.S. Treasury Bill secondary market rate from January 1, 1947 to September 1, 2005, quoted in percent on an annualized discount rate basis, for a total of 705 monthly observations (Series TB3MS).

To prepare for the examples, load `unitRootData`, the file that stores the observed GDP and T-Bill time series and the associated serial dates:

```
load unitRootData
whos
```

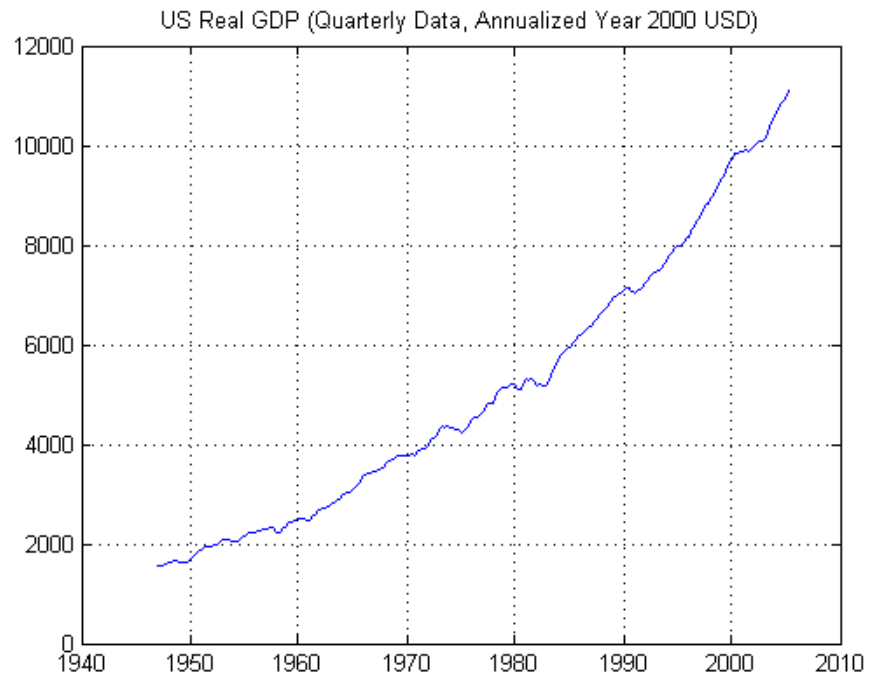
Name	Size	Bytes	Class
GDP	234x1	1872	double array
GDPDates	234x1	1872	double array
TBillDates	705x1	5640	double array
TBillRates	705x1	5640	double array

```
Grand total is 1878 elements using 15024 bytes
```

Test GDP by OLS Regression with a Stationary Component

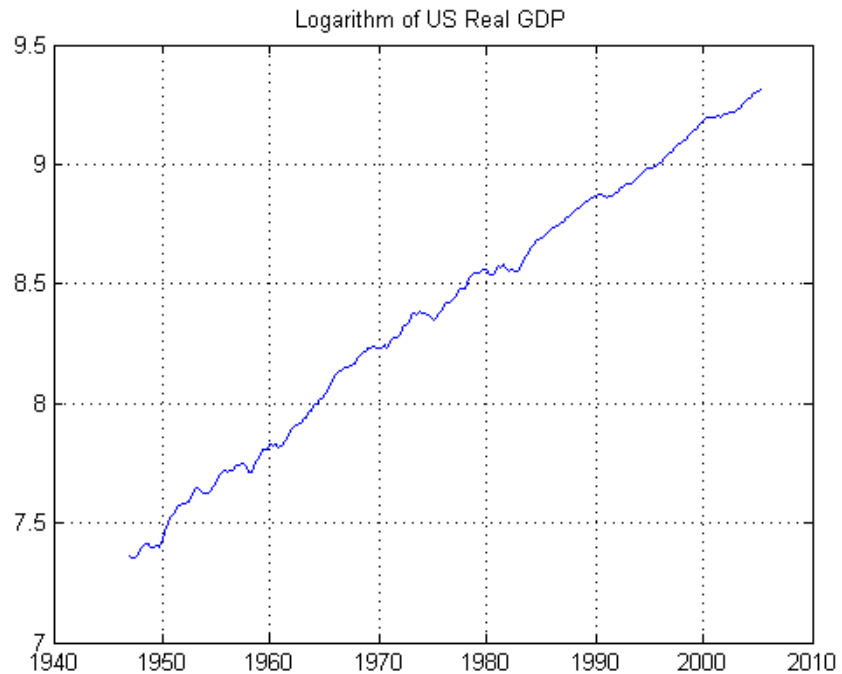
To launch the first example, plot GDP against time:

```
plot(GDPDates, GDP), datetick('x'), grid('on')
title('US Real GDP (Quarterly Data, Annualized Year 2000 USD)')
```



The plot above suggests exponential growth in the time series for real GDP. Therefore, take the logarithm of GDP data to obtain a linear time trend in the plot:

```
y = log(GDP);
plot(GDPDates, y), datetick('x'), grid('on')
title('Logarithm of US Real GDP')
```



Base your unit root test on an OLS regression model that includes a trend stationary component. Compare results of the Dickey-Fuller and Phillips-Perron trend stationary t tests, using `dftSTest` and `ppTSTest` at several common lags and at the 5% significance level:

```
[h, pValue, tStat, cValue] = dftSTest(y, [0:4], [0.05 0.05 0.05
0.05 0.05], 't');
[H, PValue, TStat, CValue] = ppTSTest(y, [0:4], 0.05); % t test
is default
```


Notice the input argument lists in the above example. The inputs to the Dickey-Fuller test explicitly specify a 5% significance level and a studentized t test for all tests. In contrast, the call to the Phillips-Perron test simply specifies a scalar 5% significance level, which is scalar-expanded to match the length of the Lags input. In addition, the syntax for the Phillips-Perron test accepts the default t test when no TestType is specified.

All elements of the logical indicator variables, h and H, are logical zero, indicating that there is no significant statistical evidence to reject the null hypothesis of a unit root ($\phi = 1$):

```
[h ; H]
ans =
    0     0     0     0     0
    0     0     0     0     0
```

Furthermore, compare the p -values, OLS test statistics, and critical values of the Phillips-Perron test in the first line and the Dickey-Fuller test in the second line:

```
[pValue ; PValue]
ans =
    0.6058    0.1841    0.0871    0.1964    0.3372
    0.6058    0.4755    0.3923    0.3555    0.3484

[tStat ; TStat]
ans =
   -1.9708   -2.8428   -3.2012   -2.8079   -2.5181
   -1.9708   -2.2364   -2.4059   -2.4808   -2.4952

[cValue ; CValue]
ans =
   -3.4315   -3.4315   -3.4316   -3.4317   -3.4318
   -3.4315   -3.4315   -3.4315   -3.4315   -3.4315
```

The first element of each Phillips-Perron output vector matches the first element of the Dickey-Fuller output vector, confirmation that the two tests are identical when Lags = 0, that is, when you make no correction for dependence.

Similarly, you can compare *AR* tests at unique combinations of lags and significance levels:

```
[h, pValue, tStat, cValue] = dfTSTest(y, [0 1 2]', [0.01 0.025  
0.05], 'AR');
```

```
[H, PValue, TStat, CValue] = ppTSTest(y, [1 2 3]', [0.01 0.05  
0.075], 'AR');
```

```
[h ; H]
```

```
ans =
```

```
    0    0  
    0    0  
    0    0
```

```
[pValue ; PValue]
```

```
ans =
```

```
    0.6683    0.5199  
    0.1721    0.4156  
    0.0717    0.3671
```

```
[tStat ; TStat]
```

```
ans =
```

```
   -7.2381   -9.4831  
  -15.1154  -11.0620  
  -19.4593  -11.7962
```

```
[cValue ; CValue]
```

```
ans =
```

```
  -28.3742  -28.3742  
  -24.3272  -21.1594  
  -21.1554  -19.2414
```

Now examine the Dickey-Fuller joint F test of a unit root ($\phi = 1$) with zero trend stationary coefficient ($\delta = 0$) under the same conditions:

```
[h, pValue, tStat, cValue] = dfTSTest(y, [0 1 2]', [0.01 0.025  
0.05], 'F');
```

```
[h, pValue, tStat, cValue]
```

```
ans =
```

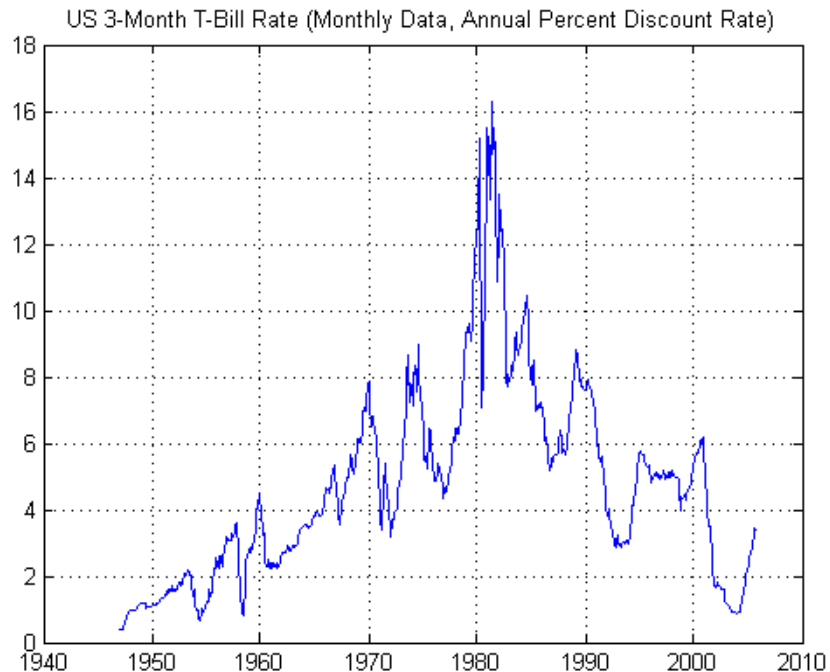
```
0    0.6207    2.5622    8.4814  
0    0.1891    4.4635    7.2834  
0    0.0884    5.5669    6.3548
```

In each of the above comparisons, the row-and-column orientation of the Lags input vector determines the row-and-column orientation of the output vectors.

Test T-Bill Rate by OLS Regression with a Drift Component

To start the second example, plot the three-month T-Bill rate against time:

```
plot(TBillDates, TBillRates), datetick('x'), grid('on')
title('US 3-Month T-Bill Rate (Monthly Data, Annual Percent
Discount Rate)')
```



Although the interest rate for Treasury Bills does not exhibit a time trend, the plot of three-month T-Bill data suggests that the OLS regression model should include an additive constant to account for drift. To incorporate drift in your model, use `ppARDTest` and `dfARDTest`. This example compares the results of these two tests:

```
[h, pValue, tStat, cValue] = dfARDTest(TBillRates, [0:4]', ...
0.05, 't');
[H, PValue, TStat, CValue] = ppARDTest(TBillRates, [0:4]');
% Alpha = 0.05 default
```

The call to the Phillips-Perron test below uses the default significance level for input Alpha, 0.05, and the default TestType, *t*.

Now compare the results:

```
[h, H, pValue, PValue, tStat, TStat, cValue, CValue]
ans =
0      0  0.2133  0.2133 -2.1889 -2.1889 -2.8667 -2.8667
1.0000 0  0.0428  0.1280 -2.9286 -2.4531 -2.8667 -2.8667
0      0  0.1313  0.1166 -2.4409 -2.4995 -2.8667 -2.8667
0      0  0.1191  0.1182 -2.4887 -2.4925 -2.8667 -2.8667
0      0  0.1235  0.1204 -2.4700 -2.4831 -2.8667 -2.8667
```

These results indicate that the null hypothesis of a unit root is actually rejected in the second Dickey-Fuller case, where $H = 1$ at Lags = 1, at the 5% significance level. If, however, you test a first-order correction at a significance level smaller than the reported p value of 0.0428 — say 0.03 — the null hypothesis is now accepted:

```
[h, pValue, tStat, cValue] = dfARDTest(TBillRates, 1, 0.03, 't');
[h, pValue, tStat, cValue]
ans =
      0  0.0428 -2.9286 -3.0633
```

Lastly, examine the Dickey-Fuller joint *F* test of a unit root ($\phi = 1$) with zero drift ($C = 0$):

```
[h, pValue, tStat, cValue] = dfARDTest(TBillRates, [0 2 4]', [0.01
0.02 0.1]);
[h, pValue, tStat, cValue]
ans =
      0  0.2133 -2.1889 -3.4405
      0  0.1313 -2.4409 -3.2078
      0  0.1235 -2.4700 -2.5696
```


Model Selection and Analysis

Likelihood Ratio Tests (p. 9-2)	Uses likelihood ratio tests to determine if evidence exists to support the use of a specific GARCH model.
Akaike and Bayesian Information Criteria (p. 9-5)	Uses Akaike (AIC) and Bayesian (BIC) information criteria to compare alternative models.
Equality Constraints and Parameter Significance (p. 9-7)	Sets and constrains model parameters as a way of assessing the parameters' significance.
Equality Constraints and Initial Parameter Estimates (p. 9-12)	Demonstrates the need for a complete model specification when you specify equality constraints. It also provides tips for using equality constraints.
Simplicity and Parsimony (p. 9-15)	Explains why you should use the smallest, simplest model that adequately describes your data.

See “Analysis and Estimation Example Using the Default Model” on page 2-16 for information about using the autocorrelation and partial autocorrelation functions as qualitative guides in the process of model selection and assessment. This example also introduces the Ljung-Box-Pierce Q-test and Engle's ARCH test functions.

Likelihood Ratio Tests

The section “Analysis and Estimation Example Using the Default Model” on page 2-16 demonstrates that the default GARCH(1,1) model explains most of the variability of the daily returns observations of the Deutschmark/British Pound foreign exchange rate. This example uses the function `lratiotest` to determine whether evidence exists to support the use of a GARCH(2,1) model.

The example first fits the Deutschmark/British Pound foreign exchange rate return series to the default GARCH(1,1) model. It then fits the same series using the following, more elaborate, GARCH(2,1) model.

$$y_t = C + \varepsilon_t$$

$$\sigma_t^2 = \kappa + G_1 \sigma_{t-1}^2 + G_2 \sigma_{t-2}^2 + A_1 \varepsilon_{t-1}^2$$

If the Deutschmark/British Pound foreign exchange rate data is not in your workspace, you can restore it with these commands.

```
load garchdata
dem2gbp = price2ret(DEM2GBP);
```

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

1 Estimate the GARCH(1,1) model. Create a GARCH(1,1) default model with `Display` set to `'off'`. Then estimate the model, including the maximized log-likelihood function value, and display the results.

```
spec11 = garchset('P',1,'Q',1,'Display','off');
[coeff11,errors11,LLF11] = garchfit(spec11,dem2gbp);
garchdisp(coeff11,errors11)
```

```
Mean: ARMAX(0,0,0); Variance: GARCH(1,1)
```


Conditional Probability Distribution: Gaussian
 Number of Model Parameters Estimated: 4

Parameter	Value	Standard Error	T Statistic
C	-6.1919e-005	8.4331e-005	-0.7342
K	1.0761e-006	1.323e-007	8.1341
GARCH(1)	0.80598	0.016561	48.6685
ARCH(1)	0.15313	0.013974	10.9586

- 2 Estimate the GARCH(2,1) model.** Create a GARCH(2,1) specification structure, and again set `Display` to 'off'. Then estimate the GARCH(2,1) model and display the results. Again, calculate the maximized log-likelihood function value.

```
spec21 = garchset('P',2,'Q',1,'Display','off');
[coeff21,errors21,LLF21] = garchfit(spec21,dem2gbp);
garchdisp(coeff21,errors21)
```

Mean: ARMAX(0,0,0); Variance: GARCH(2,1)

Conditional Probability Distribution: Gaussian
 Number of Model Parameters Estimated: 5

Parameter	Value	Standard Error	T Statistic
C	-5.0071e-005	8.4756e-005	-0.5908
K	1.1196e-006	1.5358e-007	7.2904
GARCH(1)	0.49404	0.11249	4.3918
GARCH(2)	0.2938	0.10295	2.8537
ARCH(1)	0.16805	0.016589	10.1305

- 3 Perform the likelihood ratio test.** Of the two models associated with the same return series,

- The default GARCH(1,1) model is a restricted model. That is, you can interpret a GARCH(1,1) model as a GARCH(2,1) model with the restriction that $G_2 = 0$.

- The more elaborate GARCH(2,1) model is an unrestricted model.

Since `garchfit` enforces no boundary constraints during either of the two estimations, you can apply a likelihood ratio test (LRT) (see Hamilton [19], pages 142-144).

In this context, the unrestricted GARCH(2,1) model serves as the alternative hypothesis (i.e., the hypothesis the example gathers evidence to support), while the restricted GARCH(1,1) model serves as the null hypothesis (i.e., the hypothesis the example assumes is true, lacking any evidence to support the alternative).

The LRT statistic is asymptotically chi-square distributed with degrees of freedom equal to the number of restrictions imposed. Since the GARCH(1,1) model imposes one restriction, specify one degrees of freedom in your call to `lratiotest`. Test the models at the 0.05 significance level.

```
[H,pValue,Stat,CriticalValue] = lratiotest(LLF21,LLF11,1,0.05);  
[H,pValue,Stat,CriticalValue]
```

```
ans =  
      1.0000      0.0218      5.2624      3.8415
```

H = 1 indicates that there is sufficient statistical evidence in support of the GARCH(2,1) model.

Alternatively, at the 0.02 significance level,

```
[H,pValue,Stat,CriticalValue] = lratiotest(LLF21,LLF11,1,0.02);  
[H,pValue,Stat,CriticalValue]
```

```
ans =  
      0      0.0218      5.2624      5.4119
```

H = 0 indicates that there is insufficient statistical evidence in support of the GARCH(2,1) model.

Akaike and Bayesian Information Criteria

You can also use Akaike (AIC) and Bayesian (BIC) information criteria to compare alternative models. Since information criteria penalize models with additional parameters, the AIC and BIC model order selection criteria are based on parsimony (see Box, Jenkins, and Reinsel [8], pages 200-201).

The following example uses the default GARCH(1,1) and GARCH(2,1) models developed in the previous section, “Likelihood Ratio Tests” on page 9-2.

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

1 Count the estimated parameters. For both AIC and BIC, you need to provide the number of parameters estimated in the model. For the relatively simple models in the previous example, you can just count the number of parameters. The GARCH(2,1) model estimated five parameters, C , κ , G_1 , G_2 , and A_1 , and the GARCH(1,1) model estimated four parameters, C , κ , G_1 , and A_1 .

Use the function `garchcount` for more elaborate models. `garchcount` accepts the output specification structure created by `garchfit` and returns the number of parameters in the model defined in that structure.

```
n21 = garchcount(coeff21)
```

```
n21 =  
      5
```

```
n11 = garchcount(coeff11)
```

```
n11 =  
      4
```

2 Compute the AIC and BIC criteria. Use the function `aicbic` to compute the AIC and BIC statistics for the GARCH(2,1) model and the GARCH(1,1) model. Note that for the BIC statistic, you must also specify the number of observations in the return series. Set the numeric format to long, to see the results more precisely.

```
format long
[AIC,BIC] = aicbic(LLF21,n21,1974);
[AIC BIC]

ans =
    1.0e+004 *

    -1.59632585502853   -1.59353194641854

[AIC,BIC] = aicbic(LLF11,n11,1974);
[AIC BIC]

ans =
    1.0e+004 *

    -1.59599961321328   -1.59376448632528
```

You can use the relative values of the AIC and BIC statistics as guides in the model selection process. In this example, the AIC criterion favors the GARCH(2,1) model, while the BIC criterion favors the GARCH(1,1) default model with fewer parameters. Notice that since BIC imposes a greater penalty for additional parameters than does AIC, BIC always provides a model with a number of parameters no greater than that chosen by AIC.

Note You can also set the numeric format by selecting **File -> Preferences -> Command Window -> Text display** from the MATLAB desktop.

Equality Constraints and Parameter Significance

The GARCH Toolbox lets you set and constrain model parameters as a way of assessing the parameters' significance.

This section discusses

- “The Specification Structure Fix Fields” on page 9-7
- “The GARCH(2,1) Model as an Example” on page 9-8

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

The Specification Structure Fix Fields

Each of the coefficient fields C, AR, MA, Regress, K, GARCH, ARCH, Leverage, and DoF in the specification structure has a corresponding logical field that lets you hold any individual parameter fixed. These fix fields are `FixC`, `FixAR`, `FixMA`, `FixRegress`, `FixK`, `FixGARCH`, `FixARCH`, `FixLeverage`, and `FixDoF`.

For example, fit the Nasdaq returns series to the default GARCH(1,1) model. If the Nasdaq data is not already in your workspace, you can restore it with these commands.

```
load garchdata
nasdaq = price2ret(NASDAQ);

spec11 = garchset('P',1,'Q',1,'Display','off');
[coeff11,errors11,LLF11] = garchfit(spec11,nasdaq);
garchdisp(coeff11,errors11)
```

```
Mean: ARMAX(0,0,0); Variance: GARCH(1,1)
```

```
Conditional Probability Distribution: Gaussian
```

Number of Model Parameters Estimated: 4

Parameter	Value	Standard Error	T Statistic
C	0.00085852	0.00018353	4.6778
K	2.2595e-006	3.3806e-007	6.6836
GARCH(1)	0.87513	0.0089892	97.3531
ARCH(1)	0.11635	0.0085331	13.6348

Since the estimated model has no equality constraints, all the fixed fields are implicitly empty. For example,

```
garchget(coeff11, 'FixGARCH')
ans =
     []
```

Each fix field, when not empty (`[]`), is the same size as the corresponding coefficient field. A 0 in a particular element of a fix field indicates that the corresponding element of its companion value field is an initial parameter guess that `garchfit` refines during the estimation process. A 1 indicates that `garchfit` holds the corresponding element of its value field fixed during the estimation process (i.e., an equality constraint).

Note To remove the constant C from the conditional mean model, i.e., to fix $C = 0$ without providing initial parameter estimates for the remaining parameters, set $C = \text{NaN}$ (Not-a-Number). In this case, the value of `FixC` has no effect.

The GARCH(2,1) Model as an Example

This example compares the estimation results for the default GARCH(1,1) model with those obtained from fitting a GARCH(2,1) model to the Nasdaq returns. (See “Data Sets” on page 1-11.)

Use these commands to restore your workspace if necessary.

```
load garchdata
nasdaq = price2ret(NASDAQ);
```

1 Estimate the model parameters and display the results.

```
spec21 = garchset('P',2,'Q',1,'Display','off');
[coeff21,errors21,LLF21] = garchfit(spec21,nasdaq);
garchdisp(coeff21,errors21)
```

Mean: ARMAX(0,0,0); Variance: GARCH(2,1)

Conditional Probability Distribution: Gaussian
Number of Model Parameters Estimated: 5

Parameter	Value	Standard Error	T Statistic
C	0.00086237	0.00018378	4.6925
K	2.3016e-006	4.7519e-007	4.8436
GARCH(1)	0.83571	0.18533	4.5092
GARCH(2)	0.036149	0.16562	0.2183
ARCH(1)	0.1195	0.020346	5.8734

The **T Statistic** column is the parameter value divided by the standard error, and is normally distributed for large samples. The T-statistic measures the number of standard deviations the parameter estimate is away from zero, and, as a general rule, a T-statistic greater than 2 in magnitude corresponds to approximately a 95 percent confidence interval. The T-statistics in the table above imply that the GARCH(2) parameter adds little if any explanatory power to the model.

2 Assess significance of the GARCH(2) parameter. Begin by constraining the GARCH(2) parameter at 0.

```
specG2 = garchset(coeff21,'GARCH',[0.8 0],'FixGARCH',[0 1]);
```

Using the specG2 structure, garchfit holds GARCH(2) fixed at 0, and refines GARCH(1) from an initial value of 0.8 during the estimation process. In other words, the specG2 specification structure tests the composite model

$$y_t = C + \varepsilon_t$$

$$\sigma_t^2 = \kappa + G_1 \sigma_{t-1}^2 + 0 \sigma_{t-2}^2 + A_1 \varepsilon_{t-1}^2$$

which is mathematically equivalent to the default GARCH(1,1) model.

Now estimate the model subject to the equality constraint and display the results.

```
[coeffG2,errorsG2,LLFG2] = garchfit(specG2,nasdaq);
garchdisp(coeffG2,errorsG2)
```

Mean: ARMAX(0,0,0); Variance: GARCH(2,1)

Conditional Probability Distribution: Gaussian

Number of Model Parameters Estimated: 4

Parameter	Value	Standard Error	T Statistic
C	0.00085827	0.00018353	4.6766
K	2.2574e-006	3.3785e-007	6.6818
GARCH(1)	0.87518	0.0089856	97.3979
GARCH(2)	0	Fixed	Fixed
ARCH(1)	0.11631	0.0085298	13.6357

Notice that the standard error and T-statistic columns for the second GARCH parameter indicate that garchfit held the GARCH(2) parameter fixed. The number of estimated parameters also decreased from 5 in the original, unrestricted GARCH(2,1) model to 4 in this restricted GARCH(2,1) model. Also notice that the results are virtually identical to those obtained from the GARCH(1,1) model in step 1.

Apply the likelihood ratio test as before.

```
[H,pValue,Stat,CriticalValue] = lratiotest(LLF21,LLFG2,1, 0.05);  
[H pValue Stat CriticalValue]
```

```
ans =  
    0    0.7835    0.0755    3.8415
```

As expected, since the two models are virtually identical, the results support acceptance of the simpler restricted model, which is essentially just the default GARCH(1,1) model.

Equality Constraints and Initial Parameter Estimates

This section highlights some important points regarding equality constraints and initial parameter estimates in the GARCH Toolbox. It discusses

- “Complete Model Specification” on page 9-12
- “Empty Fix Fields” on page 9-13
- “Limiting Use of Equality Constraints” on page 9-14

Note See “The Specification Structure Fix Fields” on page 9-7 for information about using the specification structure fix fields to set equality constraints.

Complete Model Specification

To set equality constraints during estimation, you must provide a complete model specification; i.e., the specification must include initial parameter estimates consistent with the model orders. The only flexibility in this regard is that you can specify the model for either the conditional mean or the conditional variance, without specifying the other.

The following example demonstrates an attempt to set equality constraints for an incomplete conditional mean model and a complete variance model. Create an ARMA(1,1)/GARCH(1,1) specification structure for conditional mean and variance models, respectively.

```
spec = garchset('R',1,'M',1,'C',0,'AR',0.5,'FixAR',1,...
               'P',1,'Q',1,'K',0.0005,'GARCH',0.8,...
               'ARCH',0.1,'FixGARCH',1)

spec =

    Comment: 'Mean: ARMAX(1,1,?); Variance: GARCH(1,1)'
Distribution: 'Gaussian'
           R: 1
           M: 1
           C: 0
          AR: 0.5000
          MA: []
```

```

VarianceModel: 'GARCH'
              P: 1
              Q: 1
              K: 5.0000e-004
              GARCH: 0.8000
              ARCH: 0.1000
              FixAR: 1
              FixGARCH: 1

```

The conditional mean model is incomplete because the MA field is still empty. Since the requested ARMA(1,1) model is an incomplete conditional mean specification, `garchfit` ignores the C, AR, and FixAR fields, computes initial parameter estimates, and overwrites any existing parameters in the incomplete conditional mean specification. It also estimates all conditional mean parameters (i.e., C, AR, and MA) and ignores the request to constrain the AR parameter.

However, since the structure explicitly sets all fields in the conditional variance model, `garchfit` uses the specified values of K and ARCH as initial estimates subject to further refinement, and holds the GARCH parameter at 0.8 throughout the optimization process.

Empty Fix Fields

Any fix field that you leave empty, (`[]`), is equivalent to a vector of zeros of compatible length. That is, when `garchfit` encounters an empty fix field, it automatically estimates the corresponding parameter. For example, the following specification structures produce the same GARCH(1,1) estimation results.

```

spec1 = garchset('K',0.005,'GARCH',0.8,'ARCH',0.1,...
                'FixGARCH',0,'FixARCH',0)

spec2 = garchset('K',0.005,'GARCH',0.8,'ARCH',0.1)

```

Note To remove the constant C from the conditional mean model, i.e., to fix $C = 0$ without providing initial parameter estimates for the remaining parameters, use `garchset` to set $C = \text{NaN}$ (Not-a-Number). In this case, the value of `FixC` is ignored.

Limiting Use of Equality Constraints

Although the ability to set equality constraints is both convenient and useful, equality constraints complicate the estimation process. For this reason, you should avoid setting several equality constraints simultaneously. For example, if you really want to estimate a GARCH(1,1) model, then specify a GARCH(1,1) model instead of a more elaborate model with numerous constraints.

Simplicity and Parsimony

As a general rule, you should specify the smallest, simplest models that adequately describe your data. This is especially relevant for estimation. Simple models are easier to estimate, easier to forecast, and easier to analyze. In fact, certain model selection criteria, such as AIC and BIC discussed in the section “Model Selection and Analysis” on page 9-1, penalize models for their complexity.

It makes sense to use diagnostic tools such as autocorrelation function (ACF) and partial autocorrelation function (PACF) to guide model selection. For example, the section “Analysis and Estimation Example Using the Default Model” on page 2-16 examines the ACF and PACF of the Deutschmark/British Pound foreign exchange rate (see “Data Sets” on page 1-11). The results support the use of a simple constant for the conditional mean model as adequate to describe the data.

The following example illustrates an unnecessarily complicated model specification. It simulates a returns series as a pure GARCH(1,1) innovations process (i.e., the default model), then attempts to overfit an ARMA(1,1)/GARCH(1,1) composite model to the data.

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

- 1 Create a specification structure for the innovations process and simulate the returns.

```
spec = garchset('C',0,'K',0.00005,'GARCH',0.85,'ARCH',0.1,...
              'Display','off');
[e,s,y] = garchsim(spec,5000,1,0);
```

- 2 Fit the default model to the known GARCH(1,1) innovations process and display the estimation results.

```
[coeff,errors] = garchfit(spec,y);
garchdisp(coeff,errors)
```

Mean: ARMAX(0,0,0); Variance: GARCH(1,1)

Conditional Probability Distribution: Gaussian
 Number of Model Parameters Estimated: 4

Parameter	Value	Standard Error	T Statistic
C	-5.8129e-005	0.0004096	-0.1419
K	4.6408e-005	8.3396e-006	5.5648
GARCH(1)	0.85994	0.014612	58.8515
ARCH(1)	0.095354	0.0097535	9.7765

These estimation results indicate that the model that best fits the observed data is approximately

$$y_t = -5.8129e-005 + \varepsilon_t$$

$$\sigma_t^2 = 4.6408e-005 + 0.85994\sigma_{t-1}^2 + 0.95354\varepsilon_{t-1}^2$$

- 3** Continue by fitting the known GARCH(1,1) innovations process to an ARMA(1,1) mean model and display the estimation results.

```
spec11 = garchset(spec,'R',1,'M',1);
[coeff11,errors11] = garchfit(spec11,y);
garchdisp(coeff11,errors11)
```

Mean: ARMAX(1,1,0); Variance: GARCH(1,1)

Conditional Probability Distribution: Gaussian
 Number of Model Parameters Estimated: 6

Parameter	Value	Standard Error	T Statistic
C	-7.1366e-005	0.00052468	-0.1360
AR(1)	-0.24509	0.32706	-0.7494

MA(1)	0.28515	0.32362	0.8811
K	4.6868e-005	8.4098e-006	5.5731
GARCH(1)	0.85917	0.014733	58.3160
ARCH(1)	0.095584	0.0097975	9.7560

- 4** Examine the results. Close examination of the conditional mean equation reveals that the AR(1) and MA(1) parameters are quite similar. In fact, when rewriting the mean equation in backshift (i.e., lag) operator notation, where $By_t = y_{t-1}$,

$$(1 + 0.24509B)y_t = -7.1366e-005 + (1 + 0.28515B)\varepsilon_t$$

the autoregressive and moving-average polynomials come close to canceling each other (see Box, Jenkins, and Reinsel [8], pages 263-267). This is an example of parameter redundancy, or pole-zero cancellation, and supports the use of the simple default model. In fact, the more elaborate ARMA(1,1) model only complicates the analysis by requiring the estimation of two additional parameters.

Advanced Example

Estimating the Model (p. 10-2)

Fits ARMA(1,1) and GJR(1,1) models to the conditional mean and variance processes, respectively, of the Nasdaq return series, assuming conditionally t-distributed residuals.

Forecasting (p. 10-4)

Uses the estimated model from the first part of the example to forecast the conditional standard deviations of residuals, the returns, the standard deviations of multi-period cumulative returns, and the standard errors of the forecast of returns over multiple periods.

Monte Carlo Simulation (p. 10-6)

Uses the estimated model from the first part of the example and vector-format presample data to perform dependent-path Monte Carlo simulation of multiple realizations.

Comparing Forecasts with Simulation Results (p. 10-8)

Illustrates the relationship between forecasting and dependent-path Monte Carlo simulation by comparing and contrasting the forecasts with their counterparts derived from the Monte Carlo simulation.

Estimating the Model

The first part of the example fits the Nasdaq daily returns to an ARMA(1,1)/GJR(1,1) model with conditionally t-distributed residuals. (See “Data Sets” on page 1-11 for more information about the Nasdaq Composite Index data set.)

- 1 Load the Nasdaq data set and convert daily closing prices to daily returns.

```
load garchdata
nasdaq = price2ret(NASDAQ);
```

- 2 Create a specification structure for an ARMA(1,1)/GJR(1,1) model with conditionally t-distributed residuals.

```
spec = garchset('VarianceModel','GJR','R',1,'M',1,'P',1,'Q',1);
spec = garchset(spec,'Display','off','Distribution','T');
```

Note This example is for illustration purposes only. Such an elaborate ARMA(1,1) model is typically unwarranted, and should only be used after you have performed a sound preestimation analysis.

Note The estimation results you obtain when you recreate examples in this book may differ slightly from those shown in the text because of differences in platforms (operating systems), as well as in versions of MATLAB, the Optimization Toolbox, and supporting math libraries. These differences in the optimization results will propagate through any subsequent examples that use the estimation results as input. These differences, however, do not affect the outcome of the examples.

- 3 Estimate the parameters of the mean and conditional variance models via `garchfit`. Make sure that the example returns the estimated residuals and conditional standard deviations inferred from the optimization process so that they can be used as presample data.

```
[coeff,errors,LLF,eFit,sFit] = garchfit(spec,nasdaq);
```

Alternatively, you could replace the above call to `garchfit` with the following successive calls to `garchfit` and `garchinfer`. This is because the estimated residuals and conditional standard deviations are also available from the inference function `garchinfer`,

```
[coeff,errors] = garchfit(spec,nasdaq);
[eFit,sFit] = garchinfer(coeff,nasdaq);
```

Either approach produces the same estimation results.

```
garchdisp(coeff,errors)
```

```
Mean: ARMAX(1,1,0); Variance: GJR(1,1)
```

```
Conditional Probability Distribution: T
Number of Model Parameters Estimated: 8
```

Parameter	Value	Standard Error	T Statistic
C	0.00099709	0.00023381	4.2646
AR(1)	-0.10719	0.11571	-0.9264
MA(1)	0.26272	0.11208	2.3441
K	1.4684e-006	3.8716e-007	3.7927
GARCH(1)	0.89993	0.011223	80.1855
ARCH(1)	0.048844	0.013619	3.5863
Leverage(1)	0.086624	0.016922	5.1189
DoF	7.8274	0.9301	8.4157

Forecasting

The second part of the example uses the estimated model (“Estimating the Model” on page 10-2) to compute forecasts for the Nasdaq return series 30 days into the future.

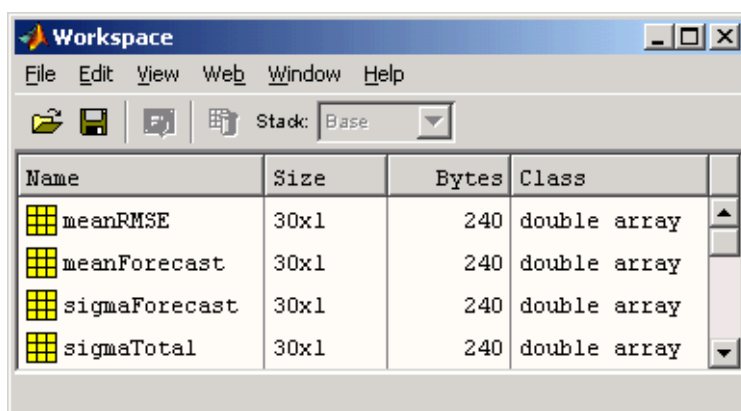
Set the forecast horizon to 30 days (i.e., one month), then call the forecasting engine, `garchpred`, with the estimated model parameters, `coeff`, the Nasdaq returns, and the forecast horizon.

```
horizon = 30; % Define the forecast horizon
[sigmaForecast,meanForecast,sigmaTotal,meanRMSE] = ...
    garchpred(coeff,nasdaq,horizon);
```

This call to `garchpred` returns

- Forecasts of conditional standard deviations of the residuals (`sigmaForecast`)
- Forecasts of the Nasdaq returns (`meanForecast`)
- Forecasts of the standard deviations of the cumulative holding period Nasdaq returns (`sigmaTotal`)
- Standard errors associated with forecasts of Nasdaq returns (`meanRMSE`)

Because the return series `nasdaq` is a vector, all `garchpred` outputs are vectors. Because `garchpred` uses iterated conditional expectations to successively update forecasts, all `garchpred` outputs have 30 rows. The first row stores the 1-period-ahead forecasts, the second row stores the 2-period-ahead forecasts, and so on. Thus, the last row stores the forecasts at the 30-day horizon.



The screenshot shows a 'Workspace' window with a menu bar (File, Edit, View, Web, Window, Help) and a toolbar with icons for file operations and a 'Stack' dropdown menu set to 'Base'. Below the toolbar is a table listing variables in the workspace.

Name	Size	Bytes	Class
meanRMSE	30x1	240	double array
meanForecast	30x1	240	double array
sigmaForecast	30x1	240	double array
sigmaTotal	30x1	240	double array

Monte Carlo Simulation

The third part of the example uses the same estimated model (`coeff`) it used in the second part of the example, “Forecasting” on page 10-4, to simulate 20000 realizations for the same 30-day period.

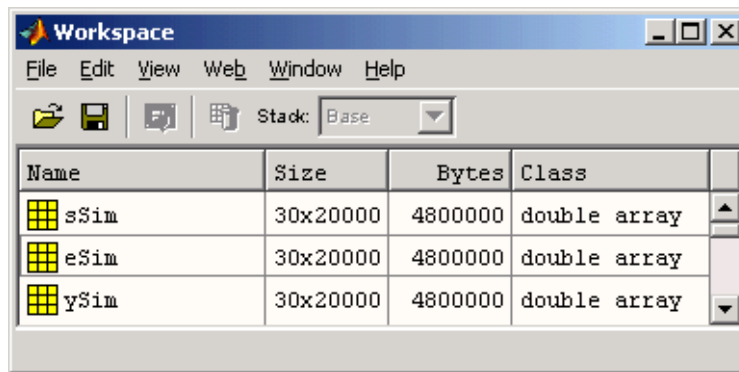
The example explicitly specifies the needed presample data. It uses the inferred residuals (`eFit`) and standard deviations (`sFit`) from the first part of the example, “Estimating the Model” on page 10-2, as the presample inputs `PreInnovations` and `PreSigmas`, respectively. It uses the nasdaq return series as the presample input `PreSeries`. Because all three inputs are vectors, `garchsim` applies the same vector to each column of the corresponding outputs, `Innovations`, `Sigmas`, and `Series`. In this context, referred to as dependent-path simulation, all simulated sample paths share a common conditioning set and evolve from the same set of initial conditions, thus enabling Monte Carlo simulation of forecasts and forecast error distributions.

Note that you can specify `PreInnovations`, `PreSigmas`, and `PreSeries` as matrices, where each column is a realization, or as single-column vectors. In either case, they must have a sufficient number of rows to initiate the simulation (see “User-Specified Presample Data” on page 4-13).

For this application of Monte Carlo simulation, the example generates a relatively large number of realizations, or sample paths, so that it can aggregate across realizations. Because each realization corresponds to a column in the `garchsim` time-series output arrays, the output arrays are large, with many columns.

The following code simulates 20000 paths (i.e., columns). As a result, each time-series output that `garchsim` returns is an array of size horizon-by-nPaths, or 30-by-20000. Although more realizations (e.g., 100000) provide more accurate simulation results, you may want to decrease the number of paths (e.g., to 10000) to avoid memory limitations.

```
nPaths = 20000; % Define the number of realizations.
[eSim,sSim,ySim] = garchsim(coeff,horizon,nPaths,0,[],[],...
                           eFit,sFit,nasdaq);
```



The screenshot shows the MATLAB Workspace window with the following table of variables:

Name	Size	Bytes	Class
sSim	30x20000	4800000	double array
eSim	30x20000	4800000	double array
ySim	30x20000	4800000	double array

Because `garchsim` needs only the last, or most recent, observation of each, the following command produces identical results.

```
[eSim,sSim,ySim] = garchsim(coeff,horizon,nPaths,0,[],[],...  
                           eFit(end),sFit(end),nasdaq(end));
```

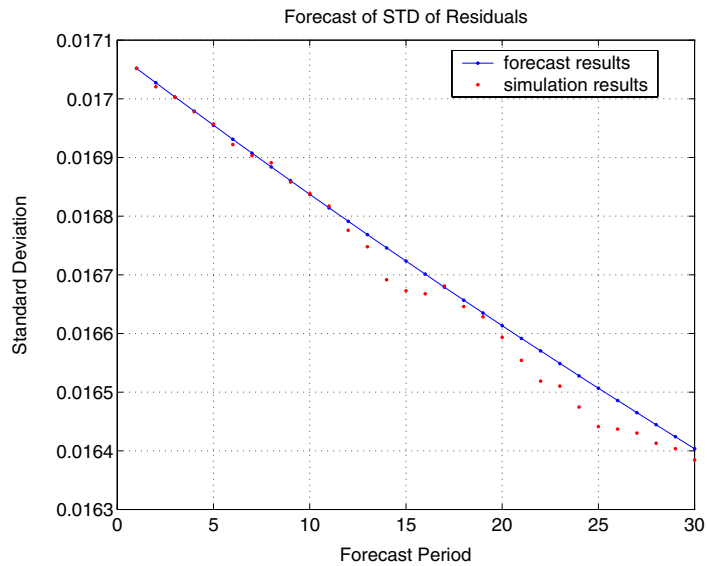
Comparing Forecasts with Simulation Results

The fourth, and last, part of this example graphically compares the forecasts from “Forecasting” on page 10-4 with their counterparts derived from the Monte Carlo experiment described in “Monte Carlo Simulation” on page 10-6. The first four figures directly compare each of the garchpred outputs, in turn, with the corresponding statistical result obtained from simulation. The last two figures illustrate histograms from which approximate probability density functions and empirical confidence bounds can be computed.

Note For an EGARCH model, multi-period MMSE forecasts are generally downward biased and underestimate their true expected values for conditional variance forecasts. This is not true for one-period-ahead forecasts, which are unbiased in all cases. For unbiased multi-period forecasts of `sigmaForecast`, `sigmaTotal`, and `meanRMSE`, you can perform Monte Carlo simulation via `garchsim`. For more information, see “Asymptotic Behavior for Long-Range Forecast Horizons” on page 6-6.

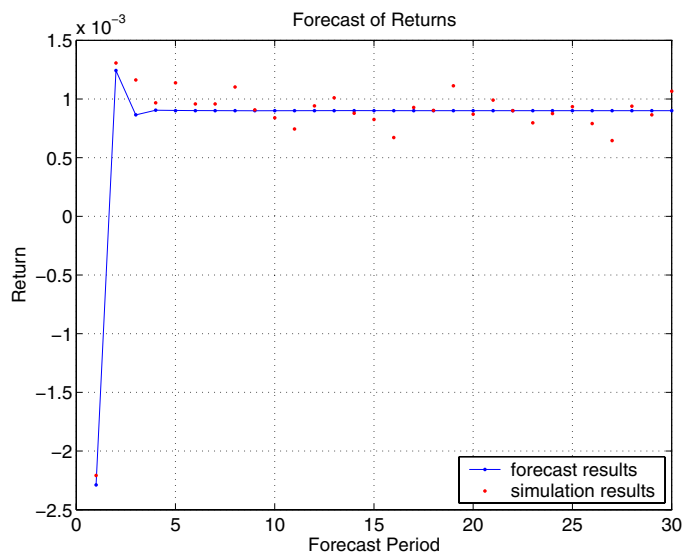
- 1 Compare the first `garchpred` output, `sigmaForecast`, i.e., the conditional standard deviations of future innovations, with its counterpart derived from the Monte Carlo simulation.

```
figure
plot(sigmaForecast, '-b')
hold('on')
grid('on')
plot(sqrt(mean(sSim.^2,2)), '.r')
title('Forecast of STD of Residuals')
legend('forecast results', 'simulation results')
xlabel('Forecast Period')
ylabel('Standard Deviation')
```

- 2 Compare the second garchpred output, meanForecast, i.e., the MMSE forecasts of the conditional mean of the nasdaq return series, with its counterpart derived from the Monte Carlo simulation.

```
figure(2)
plot(meanForecast, '-.b')
hold('on')
grid('on')
plot(mean(ySim,2), '.r')
title('Forecast of Returns')
legend('forecast results','simulation results',4)
xlabel('Forecast Period')
ylabel('Return')
```

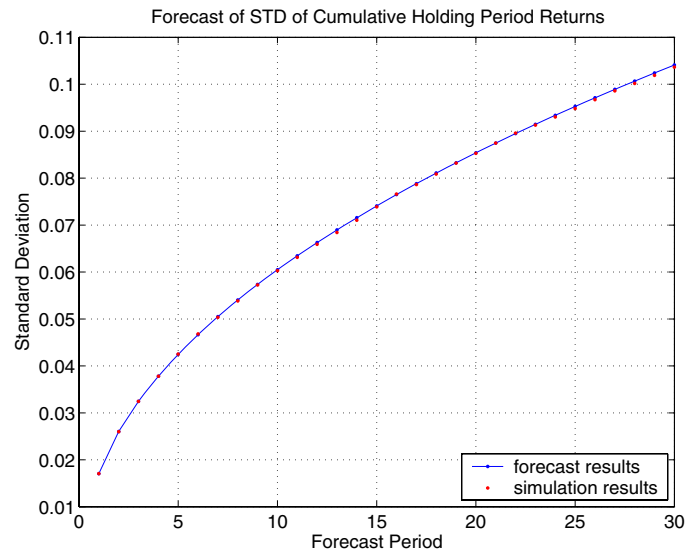


- 3 Compare the third garchpred output, `sigmaTotal`, i.e., cumulative holding period returns, with its counterpart derived from the Monte Carlo simulation.

```

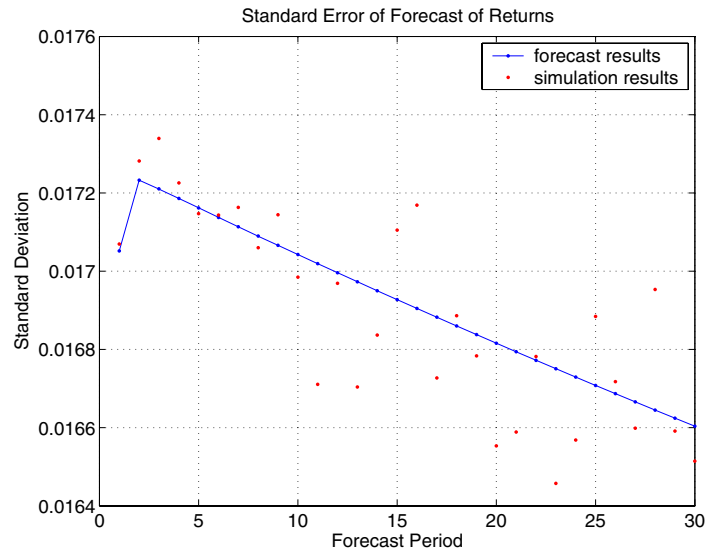
holdingPeriodReturns = log(ret2price(ySim,1));
figure(3)
plot(sigmaTotal, '-.b')
hold('on')
grid('on')
plot(std(holdingPeriodReturns(2:end,:))', '.r')
title('Forecast of STD of Cumulative Holding Period Returns')
legend('forecast results','simulation results',4)
xlabel('Forecast Period')
ylabel('Standard Deviation')

```



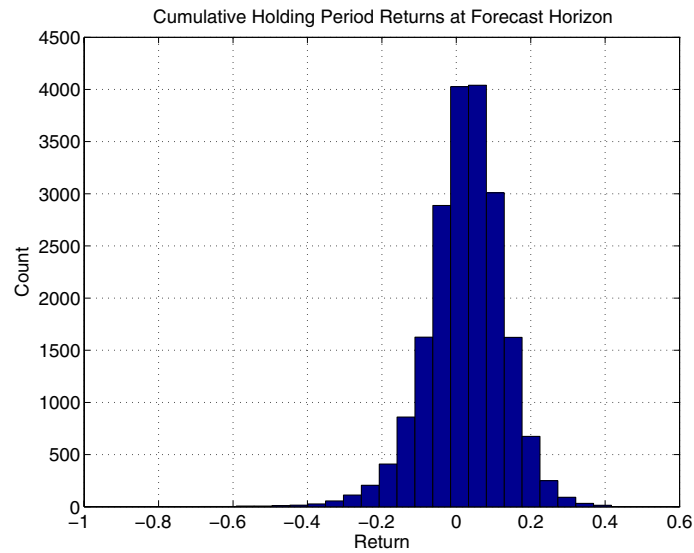
- 4 Compare the fourth garchpred output, meanRMSE, i.e. the root mean square errors (RMSE) of the forecasted returns, with its counterpart derived from the Monte Carlo simulation.

```
figure(4)
plot(meanRMSE, '-.b')
hold('on')
grid('on')
plot(std(ySim), '.r')
title('Standard Error of Forecast of Returns')
legend('forecast results','simulation results')
xlabel('Forecast Period')
ylabel('Standard Deviation')
```



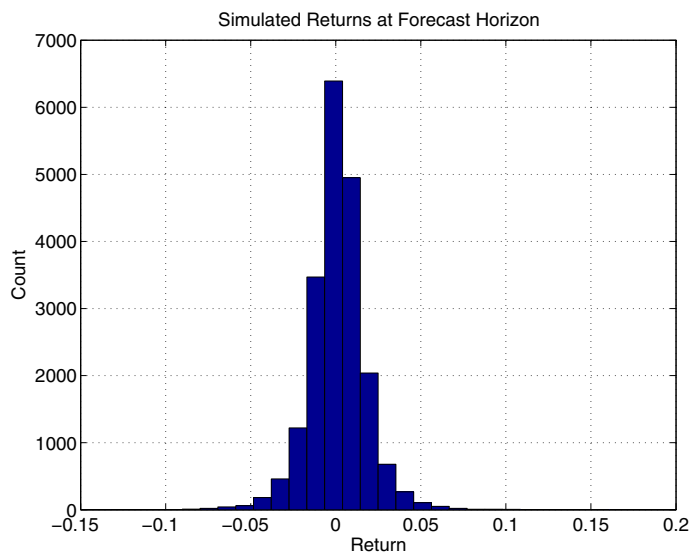
- 5 Use a histogram to illustrate the distribution of the cumulative holding period return obtained if an asset was held for the full 30-day forecast horizon, i.e., the return obtained if you purchased a mutual fund that mirrors the performance of the Nasdaq Composite Index today, and sold after 30 days. Notice that this histogram is directly related to the final red dot in step 3.

```
figure(5)
hist(holdingPeriodReturns(end,:),30)
grid('on')
title('Cumulative Holding Period Returns at Forecast Horizon')
xlabel('Return')
ylabel('Count')
```



- 6 Use a histogram to illustrate the distribution of the single-period return at the forecast horizon, i.e., the return of the same mutual fund the 30th day from now. Notice that this histogram is directly related to the final red dots in steps 2 and 4.

```
figure(6)
hist(ySim(end,:),30)
grid('on')
title('Simulated Returns at Forecast Horizon')
xlabel('Return')
ylabel('Count')
```



Note This example is not meant to imply that such elaborate conditional mean and variance models are required to describe typical financial time series, nor is it meant to imply that most users will need to perform such detailed analyses at all. Furthermore, it is not meant to imply that such a graphical analysis even makes sense for a given model, or that this is the only graphs that could make sense.

This example merely highlights the range of possibilities, and provides a deeper understanding of the interaction between the simulation, forecasting, and estimation engines, `garchsim`, `garchpred`, and `garchfit`, respectively.

Function Reference

- Functions — By Category (p. 11-2) Lists the GARCH Toolbox functions and classes according to their purpose.
- Functions — Alphabetical List (p. 11-5) Lists the GARCH Toolbox functions and classes alphabetically.

Functions – By Category

This section lists the GARCH Toolbox functions according to their purpose.

- “GARCH Modeling” on page 11-2
- “GARCH Innovations Inference” on page 11-2
- “Statistics and Tests” on page 11-2
- “GARCH Specification Structure Interface Functions” on page 11-3
- “Helpers and Utilities” on page 11-3
- “Graphics” on page 11-4

GARCH Modeling

<code>garchfit</code>	Univariate GARCH process parameter estimation.
<code>garchpred</code>	Univariate GARCH process forecasting.
<code>garchsim</code>	Univariate GARCH process simulation.

GARCH Innovations Inference

<code>garchinfer</code>	Inverse filter to infer GARCH innovations and conditional standard deviations from an observed return series.
-------------------------	---

Statistics and Tests

<code>aicbic</code>	Akaike and Bayesian information criteria for model order selection.
<code>archtest</code>	Engle’s hypothesis test for presence of ARCH/GARCH effects.
<code>autocorr</code>	Plot or return computed sample autocorrelation function.
<code>crosscorr</code>	Plot or return computed sample crosscorrelation function.
<code>dfARDTest</code>	Augmented Dickey-Fuller unit root test based on AR model with drift.
<code>dfARTest</code>	Augmented Dickey-Fuller unit root test based on zero drift AR model.

<code>dfTSTest</code>	Augmented Dickey-Fuller unit root test based on trend stationary AR model.
<code>lbqtest</code>	Ljung-Box Q-statistic lack-of-fit hypothesis test.
<code>lratiotest</code>	Likelihood ratio hypothesis test.
<code>parcorr</code>	Plot or return computed sample partial autocorrelation function.
<code>ppARDTest</code>	Phillips-Perron unit root test based on AR(1) model with drift.
<code>ppARTest</code>	Phillips-Perron unit root test based on zero drift AR(1) model.
<code>ppTSTest</code>	Phillips-Perron unit root test based on trend stationary AR(1) model.

GARCH Specification Structure Interface Functions

<code>garchget</code>	GARCH specification structure parameter.
<code>garchset</code>	Create or modify a GARCH specification structure.

Helpers and Utilities

<code>garchar</code>	Convert finite-order ARMA models to infinite-order AR models.
<code>garchcount</code>	Count GARCH estimation coefficients.
<code>garchdisp</code>	GARCH process estimation results.
<code>garchma</code>	Convert finite-order ARMA models to infinite-order MA models.
<code>lagmatrix</code>	Create lagged time-series matrix.
<code>price2ret</code>	Convert price series to return series.
<code>ret2price</code>	Convert return series to price series.

Graphics

`garchplot` Plot matched univariate innovations, volatility, and return series.

Functions — Alphabetical List

This section contains function reference pages listed alphabetically.

aicbic

Purpose Akaike (AIC) and Bayesian (BIC) information criteria for model order selection

Syntax
`AIC = aicbic(LLF, NumParams)`
`[AIC, BIC] = aicbic(LLF, NumParams, NumObs)`

Description `aicbic` computes the Akaike and Bayesian information criteria, using optimized log-likelihood objective function (LLF) values as input. You can obtain the LLF values by fitting models of the conditional mean and variance to a univariate return series.

`AIC = aicbic(LLF, NumParams)` computes only the Akaike (AIC) information criteria.

`[AIC, BIC] = aicbic(LLF, NumParams, NumObs)` computes both the Akaike (AIC) and Bayesian (BIC) information criteria.

Since information criteria penalize models with additional parameters' parsimony is the basis of the AIC and BIC model order selection criteria.

Input Arguments

LLF Vector of optimized log-likelihood objective function (LLF) values associated with parameter estimates of the models to be tested. `aicbic` assumes you obtained the LLF values from the estimation function `garchfit` or the inference function `garchinfer`.

NumParams Number of estimated parameters associated with each LLF value in LLF. `NumParams` can be a scalar applied to all values in LLF, or a vector the same length as LLF. All elements of `NumParams` must be positive integers. Use `garchcount` to compute `NumParams` values.

NumObs Sample size of the observed return series you associate with each value of LLF. `NumObs` can be a scalar applied to all values in LLF, or a vector the same length as LLF. It is required to compute BIC. All elements of `NumObs` must be positive integers.

Output Arguments

AIC Vector of AIC statistics associated with each LLF objective function value. The AIC statistic is defined as

$$AIC = (-2 \times \text{LLF}) + (2 \times \text{NumParams})$$

BIC Vector of BIC statistics associated with each LLF objective function value. The BIC statistic is defined as

$$BIC = (-2 \times \text{LLF}) + (\text{NumParams} \times \log(\text{NumObs}))$$

Examples

See “Akaike and Bayesian Information Criteria” on page 9-5.

See Also

garchdisp, garchfit, garchinfer

References

[1] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.

archtest

Purpose	Engle's hypothesis test for presence of ARCH/GARCH effects						
Syntax	<code>[H, pValue, ARCHstat, CriticalValue] = archtest(Residuals, Lags, Alpha)</code>						
Description	<p><code>[H, pValue, ARCHstat, CriticalValue] = archtest(Residuals, Lags, Alpha)</code> tests the null hypothesis that a time series of sample residuals consists of independent identically distributed (i.i.d.) Gaussian disturbances, i.e., that no ARCH effects exist.</p> <p>Given sample residuals obtained from a curve fit (e.g., a regression model), <code>archtest</code> tests for the presence of Mth order ARCH effects by regressing the squared residuals on a constant and the lagged values of the previous M squared residuals. Under the null hypothesis, the asymptotic test statistic, $T(R^2)$, where T is the number of squared residuals included in the regression and R^2 is the sample multiple correlation coefficient, is asymptotically chi-square distributed with M degrees of freedom. When testing for ARCH effects, a GARCH(P,Q) process is locally equivalent to an ARCH(P+Q) process.</p>						
Input Arguments	<table><tr><td>Residuals</td><td>Time-series column vector of sample residuals obtained from a curve fit, which <code>archtest</code> examines for the presence of ARCH effects. The last row contains the most recent observation.</td></tr><tr><td>Lags</td><td>Vector of positive integers indicating the lags of the squared sample residuals included in the ARCH test statistic. If specified, each lag should be significantly less than the length of Residuals. If Lags = [] or is not specified, the default is 1 lag (i.e., first-order ARCH).</td></tr><tr><td>Alpha</td><td>Significance levels of the hypothesis test. Alpha can be a scalar applied to all lags in Lags, or a vector of significance levels the same length as Lags. If Alpha = [] or is not specified, the default is 0.05. For all elements, α, of Alpha, $0 < \alpha < 1$.</td></tr></table>	Residuals	Time-series column vector of sample residuals obtained from a curve fit, which <code>archtest</code> examines for the presence of ARCH effects. The last row contains the most recent observation.	Lags	Vector of positive integers indicating the lags of the squared sample residuals included in the ARCH test statistic. If specified, each lag should be significantly less than the length of Residuals. If Lags = [] or is not specified, the default is 1 lag (i.e., first-order ARCH).	Alpha	Significance levels of the hypothesis test. Alpha can be a scalar applied to all lags in Lags, or a vector of significance levels the same length as Lags. If Alpha = [] or is not specified, the default is 0.05. For all elements, α , of Alpha, $0 < \alpha < 1$.
Residuals	Time-series column vector of sample residuals obtained from a curve fit, which <code>archtest</code> examines for the presence of ARCH effects. The last row contains the most recent observation.						
Lags	Vector of positive integers indicating the lags of the squared sample residuals included in the ARCH test statistic. If specified, each lag should be significantly less than the length of Residuals. If Lags = [] or is not specified, the default is 1 lag (i.e., first-order ARCH).						
Alpha	Significance levels of the hypothesis test. Alpha can be a scalar applied to all lags in Lags, or a vector of significance levels the same length as Lags. If Alpha = [] or is not specified, the default is 0.05. For all elements, α , of Alpha, $0 < \alpha < 1$.						

Output Arguments

H	Boolean decision vector. 0 indicates acceptance of the null hypothesis that no ARCH effects exist; i.e., there is homoscedasticity at the corresponding element of Lags. 1 indicates rejection of the null hypothesis. The length of H is the same as the length of Lags.
pValue	Vector of P-values (significance levels) at which archtest rejects the null hypothesis of no ARCH effects at each lag in Lags.
ARCHstat	Vector of ARCH test statistics for each lag in Lags.
CriticalValue	Vector of critical values of the chi-square distribution for comparison with the corresponding element of ARCHstat.

Examples

Example 1. Create a time-series column vector of 100 (synthetic) residuals, then test for the first, second, and fourth order ARCH effects at the 10 percent significance level.

```

randn('state', 0)           % Start from a known state.
residuals = randn(100, 1); % 100 Gaussian deviates ~ N(0, 1)
[H, P, Stat, CV] = archtest(residuals, [1 2 4]', 0.10);
[H, P, Stat, CV]

ans =

      0    0.3925    0.7312    2.7055
      0    0.5061    1.3621    4.6052
      0    0.7895    1.7065    7.7794

```

Example 2. See “Analysis and Estimation Example Using the Default Model” on page 2-16 for another example.

See Also

lbqtest

References

[1] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.

[2] Engle, Robert, “Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation,” *Econometrica*, Vol. 50, 1982, pp. 987-1007.

[3] Gouriéroux, C., *ARCH Models and Financial Applications*, Springer-Verlag, 1997.

[4] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, 1994.

Purpose	Plot or return computed sample autocorrelation function
Syntax	<pre>autocorr(Series, nLags, M, nSTDs) [ACF, Lags, Bounds] = autocorr(Series, nLags, M, nSTDs)</pre>
Description	<p><code>autocorr(Series, nLags, M, nSTDs)</code> computes and plots the sample ACF of a univariate, stochastic time series with confidence bounds. To plot the ACF sequence without the confidence bounds, set <code>nSTDs = 0</code>.</p> <p><code>[ACF, Lags, Bounds] = autocorr(Series, nLags, M, nSTDs)</code> computes and returns the ACF sequence.</p>
Input Arguments	<p>Series Column vector of observations of a univariate time series for which <code>autocorr</code> computes or plots the sample autocorrelation function (ACF). The last row of <code>Series</code> contains the most recent observation of the time series.</p> <p>nLags Positive scalar integer indicating the number of lags of the ACF to compute. If <code>nLags = []</code> or is not specified, the default is to compute the ACF at lags 0, 1, 2, ..., T, where $T = \min([20, \text{length}(\text{Series}) - 1])$.</p> <p>M Nonnegative integer scalar indicating the number of lags beyond which the theoretical ACF is effectively 0. <code>autocorr</code> assumes the underlying <code>Series</code> is an MA(M) process, and uses Bartlett's approximation to compute the large-lag standard error for lags greater than M. If <code>M = []</code> or is not specified, the default is 0, and <code>autocorr</code> assumes that <code>Series</code> is Gaussian white noise. If <code>Series</code> is a Gaussian white noise process of length N, the standard error is approximately $1/\sqrt{N}$. M must be less than <code>nLags</code>.</p> <p>nSTDs Positive scalar indicating the number of standard deviations of the sample ACF estimation error to compute. <code>autocorr</code> assumes the theoretical ACF of <code>Series</code> is 0 beyond lag M. When <code>M = 0</code> and <code>Series</code> is a Gaussian white noise process of length N, specifying <code>nSTDs</code> results in confidence bounds at $\pm(n\text{STDs}/\sqrt{N})$. If <code>nSTDs = []</code> or is not specified, the default is 2 (i.e., approximate 95 percent confidence interval).</p>

autocorr

Output Arguments

ACF	Sample autocorrelation function of Series. ACF is a vector of length $nLags+1$ corresponding to lags 0, 1, 2, ..., $nLags$. The first element of ACF is unity, that is, $ACF(1) = 1 = \text{lag } 0 \text{ correlation}$.
Lags	Vector of lags corresponding to $ACF(0, 1, 2, \dots, nLags)$. Since an ACF is symmetric about 0 lag, autocorr ignores negative lags.
Bounds	Two-element vector indicating the approximate upper and lower confidence bounds, assuming that Series is an MA(M) process. Values of ACF beyond lag M that are effectively 0 lie within these bounds. Note that autocorr computes Bounds only for lags greater than M.

Examples

Example 1. Create an MA(2) time series from a column vector of 1000 Gaussian deviates, and assess whether the ACF is effectively zero for lags greater than 2.

```
randn('state', 0)           % Start from a known state.
x = randn(1000, 1);         % 1000 Gaussian deviates ~ N(0, 1).
y = filter([1 -1 1], 1, x); % Create an MA(2) process.
[ACF, Lags, Bounds] = autocorr(y, [], 2); % Compute the ACF with
                                         % 95 percent confidence.

[Lags, ACF]

ans =
     0     1.0000
    1.0000    -0.6487
    2.0000     0.3001
    3.0000     0.0229
    4.0000     0.0196
    5.0000    -0.0489
    6.0000     0.0452
    7.0000     0.0012
    8.0000    -0.0214
    9.0000     0.0235
   10.0000     0.0340
   11.0000    -0.0392
   12.0000     0.0188
   13.0000     0.0504
   14.0000    -0.0600
```

```

15.0000    0.0251
16.0000    0.0441
17.0000   -0.0732
18.0000    0.0755
19.0000   -0.0571
20.0000    0.0485

```

Bounds

```

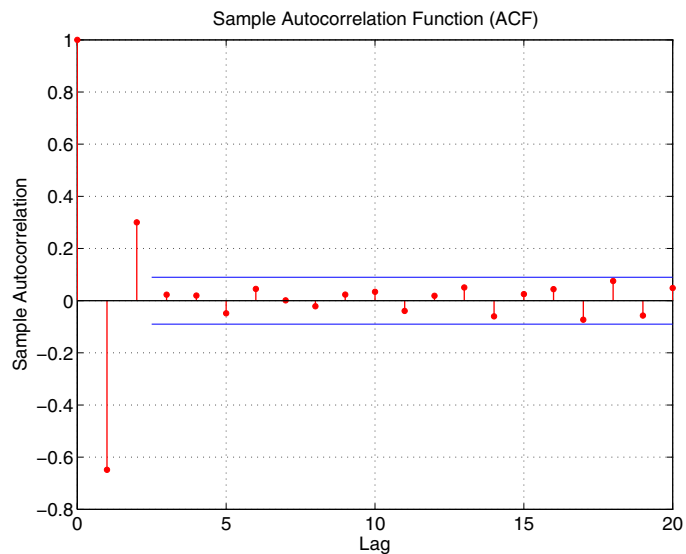
Bounds =
    0.0899
   -0.0899

```

```

autocorr(y, [], 2)    % Use the same example, but plot the ACF
                      % sequence with confidence bounds.

```



Example 2. See “Analysis and Estimation Example Using the Default Model” on page 2-16 for another example.

autocorr

See Also

crosscorr, parcorr
filter (MATLAB)

References

- [1] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.
- [2] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, 1994.

Purpose	Plot or return computed sample crosscorrelation function
Syntax	<pre>crosscorr(Series1, Series2, nLags, nSTDs) [XCF, Lags, Bounds] = crosscorr(Series1, Series2, nLags, nSTDs)</pre>
Description	<p><code>crosscorr(Series1, Series2, nLags, nSTDs)</code> computes and plots the sample crosscorrelation function (XCF) between two univariate, stochastic time series. To plot the XCF sequence without the confidence bounds, set <code>nSTDs = 0</code>.</p> <p><code>[XCF, Lags, Bounds] = crosscorr(Series1, Series2, nLags, nSTDs)</code> computes and returns the XCF sequence.</p>
Input Arguments	<p>Series1 Column vector of observations of the first univariate time series for which <code>crosscorr</code> computes or plots the sample crosscorrelation function (XCF). The last row of <code>Series1</code> contains the most recent observation.</p> <p>Series2 Column vector of observations of the second univariate time series for which <code>crosscorr</code> computes or plots the sample XCF. The last row of <code>Series2</code> contains the most recent observation.</p> <p>nLags Positive scalar integer indicating the number of lags of the XCF to compute. If <code>nLags = []</code> or is not specified, <code>crosscorr</code> computes the XCF at lags $0, \pm 1, \pm 2, \dots, \pm T$, where $T = \min([20, \min([\text{length}(\text{Series1}), \text{length}(\text{Series2})]) - 1])$.</p> <p>nSTDs Positive scalar indicating the number of standard deviations of the sample XCF estimation error to compute, if <code>Series1</code> and <code>Series2</code> are uncorrelated. If <code>nSTDs = []</code> or is not specified, the default is 2 (i.e., approximate 95 percent confidence interval).</p>
Output Arguments	<p>XCF Sample crosscorrelation function between <code>Series1</code> and <code>Series2</code>. XCF is a vector of length $2(\text{nLags}) + 1$, which corresponds to lags $0, \pm 1, \pm 2, \dots, \pm \text{nLAGs}$. The center element of XCF contains the 0th lag cross correlation.</p>

crosscorr

Lags Vector of lags corresponding to XCF(-nLags, ..., +nLags).
Bounds Two-element vector indicating the approximate upper and lower confidence bounds, assuming that Series1 and Series2 are completely uncorrelated.

Examples

Example 1. Create a time-series column vector of 100 Gaussian deviates, and a delayed version lagged by four samples. Compute the XCF, and then plot it to see the XCF peak at the fourth lag.

```
randn('state', 100)                    % Start from a known state.  
x = randn(100, 1);                    % 100 Gaussian deviates, N(0, 1).  
y = lagmatrix(x, 4);                  % Delay it by 4 samples.  
y(isnan(y)) = 0;                      % Replace NaNs with zeros.  
[XCF, Lags, Bounds] = crosscorr(x, y); % Compute the XCF with  
                                      % 95 percent confidence.
```

```
[Lags, XCF]
```

```
ans =
```

```
-20.0000   -0.0210  
-19.0000   -0.0041  
-18.0000    0.0661  
-17.0000    0.0668  
-16.0000    0.0380  
-15.0000   -0.1060  
-14.0000    0.0235  
-13.0000    0.0240  
-12.0000    0.0366  
-11.0000    0.0505  
-10.0000    0.0661  
-9.0000     0.1072  
-8.0000    -0.0893  
-7.0000    -0.0018  
-6.0000     0.0730  
-5.0000     0.0204  
-4.0000     0.0352  
-3.0000     0.0792  
-2.0000     0.0550  
-1.0000     0.0004
```

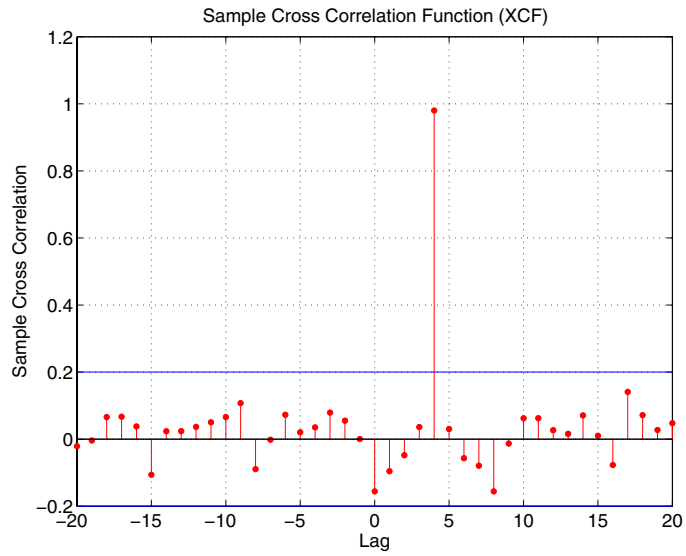
0	-0.1556
1.0000	-0.0959
2.0000	-0.0479
3.0000	0.0361
4.0000	0.9802
5.0000	0.0304
6.0000	-0.0566
7.0000	-0.0793
8.0000	-0.1557
9.0000	-0.0128
10.0000	0.0623
11.0000	0.0625
12.0000	0.0268
13.0000	0.0158
14.0000	0.0709
15.0000	0.0102
16.0000	-0.0769
17.0000	0.1410
18.0000	0.0714
19.0000	0.0272
20.0000	0.0473

Bounds

Bounds =
0.2000
-0.2000

crosscorr(x, y)

% Use the same example, but plot the XCF
% sequence. Note the peak at the 4th lag.



Example 2. See “Analysis and Estimation Example Using the Default Model” on page 2-16 for another example.

See Also

autocorr, parcorr
filter (MATLAB)

Purpose	Augmented Dickey-Fuller unit root test based on AR model with drift
Syntax	<pre>[H, PValue, TestStat, CriticalValue] = dfARDTest(Y) [H, PValue, TestStat, CriticalValue] = dfARDTest(Y, Lags, Alpha, TestType)</pre>
Description	<p>[H, PValue, TestStat, CriticalValue] = dfARDTest(Y) performs an augmented Dickey-Fuller univariate unit root test under the assumption that the true underlying process is a zero drift unit root process. As an alternative, a (P+1)th order autoregressive (AR(P+1)) model plus additive constant is estimated by OLS regression.</p> <p>Specifically, if y_t and ε_t are the time series of observed data and model residuals, respectively, and $\Delta y_t = y_t - y_{t-1}$ is the first difference operator, then under the null hypothesis the true underlying process is a zero drift ARIMA(P,1,0) model</p> $y_t = y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$ <p>which is equivalent to an integrated AR(P+1) model.</p> <p>As an alternative, the estimated OLS regression model is</p> $y_t = C + \phi y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$ <p>for some constant C and AR(1) coefficient $\phi < 1$.</p>
Input Arguments	<p>Y Time-series vector of observed data tested for a unit root. The last element contains the most recent observation. Missing values are indicated by NaNs and are removed, thereby reducing the sample size.</p> <p>Lags (Optional) Scalar or vector of nonnegative integers indicating the number of lagged changes (i.e., first differences) of Y included in the OLS regression model (see P above). Lags serves as a correction for serial correlation of residuals. If empty or missing, the default is 0 (no correction for serial correlation).</p>

dfARDTest

Alpha	(Optional) Scalar or vector of significance levels of the test. All elements of the input argument must be $0.0001 \leq \text{Alpha} \leq 0.999$.
TestType	(Optional) Character string indicating the type of unit root test. Possible choices are t, AR, and F, indicating an OLS t test of the AR(1) coefficient, a test of the unstudentized AR(1) coefficient, and a joint OLS F test of a unit root ($\phi = 1$) with zero drift ($C = 0$), respectively. If empty or missing, the default is a t test. A case-insensitive check is made of TestType.
H	Logical decision vector. Elements of $H = 0$ indicate acceptance of the null hypothesis; elements of $H = 1$ indicate rejection of the null hypothesis. Each element of H is associated with a particular lag of Lags and significance level of Alpha.
PValue	Vector of P values (significance levels) associated with the test decision vector H. Each element of PValue represents the probability of observing a test statistic at least as extreme as that calculated from the OLS regression model when the null hypothesis is true. P values are obtained by interpolation into the appropriate table of critical values, and a NaN is returned if the test statistic lies beyond the range of the table.
TestStat	Vector of test statistics associated with the decision vector H.
CriticalValue	Vector of critical values associated with the decision vector H.

Output Arguments

Notes

Both Lags and Alpha may be vectors. If both are specified as vectors, however, they must be the same length (i.e., have the same number of elements). If one is specified as a scalar and the other as a vector, a scalar expansion is performed to enforce vectors of identical length. If Lags is a scalar or an empty matrix, by default, all outputs are column vectors.

All vector outputs are the same length as vector inputs Alpha and/or Lags. By default, all vector outputs are column vectors. If Lags is a row vector, however, all vector outputs are row vectors.

This univariate unit root test is a conventional *single-tailed* test, and the acceptance or rejection of the test is based on a comparison of the test statistic with the critical value:

- The AR and t tests are *lower-tailed* tests. Reject the null hypothesis if the test statistic is *less than* the critical value.
- The joint F test is an *upper-tailed* test. Reject the null hypothesis if the test statistic is *greater than* the critical value.

See Also

dfARTest, dfTSTest, ppARDTest, ppARTest, ppTSTest

References

- [1] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, Princeton, NJ, 1994.
- [2] Greene, W.H., *Econometric Analysis*, Prentice Hall, Fifth edition, Upper Saddle River, NJ, 2003.
- [3] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, New York, 1995.
- [4] Campbell, J.Y., A.W. Lo, and A.C. MacKinlay, *The Econometrics of Financial Markets*, Princeton University Press, Princeton, NJ, 1997.

dfARTest

Purpose	Augmented Dickey-Fuller unit root test based on zero drift AR model
Syntax	<pre>[H, PValue, TestStat, CriticalValue] = dfARTest(Y) [H, PValue, TestStat, CriticalValue] = dfARTest(Y, Lags, Alpha, TestType)</pre>
Description	<p>[H, PValue, TestStat, CriticalValue] = dfARTest(Y) performs an augmented Dickey-Fuller univariate unit root test under the assumption that the true underlying process is a zero drift unit root process. As an alternative, a zero drift (P+1)th order autoregressive (AR(P+1)) model is estimated by OLS regression.</p> <p>Specifically, if y_t and ε_t are the time series of observed data and model residuals, respectively, and $\Delta y_t = y_t - y_{t-1}$ is the first difference operator, then under the null hypothesis the true underlying process is a zero drift ARIMA(P,1,0) model</p> $y_t = y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$ <p>which is equivalent to an integrated AR(P+1) model.</p> <p>As an alternative, the estimated OLS regression model is</p> $y_t = \phi y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$ <p>for some AR(1) coefficient $\phi < 1$.</p>
Input Arguments	<p>Y Time-series vector of observed data tested for a unit root. The last element contains the most recent observation. Missing values are indicated by NaNs and are removed, thereby reducing the sample size.</p> <p>Lags (Optional) Scalar or vector of nonnegative integers indicating the number of lagged changes (i.e., first differences) of Y included in the OLS regression model (see P above). Lags serves as a correction for serial correlation of residuals. If empty or missing, the default is 0 (no correction for serial correlation).</p>

Output Arguments

Alpha	(Optional) Scalar or vector of significance levels of the test. All elements of the input argument must be $0.0001 \leq \text{Alpha} \leq 0.999$.
TestType	(Optional) Character string indicating the type of unit root test. Possible choices are t and AR, indicating an OLS t test of the AR(1) coefficient and a test of the unstudentized AR(1) coefficient, respectively. If empty or missing, the default is a t test. A case-insensitive check is made of TestType.
H	Logical decision vector. Elements of H = 0 indicate acceptance of the null hypothesis; elements of H = 1 indicate rejection of the null hypothesis. Each element of H is associated with a particular lag of Lags and significance level of Alpha.
PValue	Vector of P values (significance levels) associated with the test decision vector H. Each element of PValue represents the probability of observing a test statistic at least as extreme as that calculated from the OLS regression model when the null hypothesis is true. P values are obtained by interpolation into the appropriate table of critical values, and a NaN is returned if the test statistic lies beyond the range of the table.
TestStat	Vector of test statistics associated with the decision vector H.
CriticalValue	Vector of critical values associated with the decision vector H.

Notes

Both Lags and Alpha may be vectors. If both are specified as vectors, however, they must be the same length (i.e., have the same number of elements). If one is specified as a scalar and the other as a vector, a scalar expansion is performed to enforce vectors of identical length. If Lags is a scalar or an empty matrix, by default, all outputs are column vectors.

All vector outputs are the same length as vector inputs Alpha and/or Lags. By default all vector outputs are column vectors. If Lags is a row vector, however, all vector outputs are row vectors.

This univariate unit root test is a conventional *lower tailed* test, and the acceptance or rejection of the test is based on a comparison of the test statistic with the critical value. If the test statistic is *less than* the critical value, reject the null hypothesis.

See Also

dfARDTest, dfTSTest, ppARDTest, ppARTest, ppTSTest

References

- [1] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, Princeton, NJ, 1994.
- [2] Greene, W.H., *Econometric Analysis*, Prentice Hall, Fifth edition, Upper Saddle River, NJ, 2003.
- [3] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, New York, 1995.
- [4] Campbell, J.Y., A.W. Lo, and A.C. MacKinlay, *The Econometrics of Financial Markets*, Princeton University Press, Princeton, NJ, 1997.

Purpose	Augmented Dickey-Fuller unit root test based on trend stationary AR model
Syntax	<pre>[H, PValue, TestStat, CriticalValue] = dfTSTest(Y) [H, PValue, TestStat, CriticalValue] = dfTSTest(Y, Lags, Alpha, TestType)</pre>
Description	<p>[H, PValue, TestStat, CriticalValue] = dfTSTest(Y) performs an augmented Dickey-Fuller univariate unit root test under the assumption that the true underlying process is a unit root process with drift. As an alternative, a trend stationary (P+1)th order autoregressive (AR(P+1)) model plus additive constant is estimated by OLS regression.</p> <p>Specifically, if y_t and ε_t are the time series of observed data and model residuals, respectively, and $\Delta y_t = y_t - y_{t-1}$ is the first difference operator, then under the null hypothesis the true underlying process is an ARIMA(P,1,0) model with drift</p> $y_t = C + y_{t-1} + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$ <p>which is equivalent to an integrated AR(P+1) model.</p> <p>As an alternative, the estimated OLS regression model is</p> $y_t = C + \phi y_{t-1} + \delta t + \zeta_1 \Delta y_{t-1} + \zeta_2 \Delta y_{t-2} + \dots + \zeta_p \Delta y_{t-p} + \varepsilon_t$ <p>for some constant C, AR(1) coefficient $\phi < 1$, and trend stationary coefficient δ.</p>
Input Arguments	<p>Y Time-series vector of observed data tested for a unit root. The last element contains the most recent observation. Missing values are indicated by NaNs and are removed, thereby reducing the sample size.</p> <p>Lags (Optional) Scalar or vector of nonnegative integers indicating the number of lagged changes (i.e., first differences) of Y included in the OLS regression model (see P above). Lags serves as a correction for serial correlation of residuals. If empty or missing, the default is 0 (no correction for serial correlation).</p>

dfTSTest

Alpha	(Optional) Scalar or vector of significance levels of the test. All elements of the input argument must be $0.0001 \leq \text{Alpha} \leq 0.999$.
TestType	(Optional) Character string indicating the type of unit root test. Possible choices are t, AR, and F, indicating an OLS t test of the AR(1) coefficient, a test of the unstudentized AR(1) coefficient, and a joint OLS F test of a unit root ($\phi = 1$) with zero trend stationary coefficient ($\delta = 1$), respectively. If empty or missing, the default is a t test. A case-insensitive check is made of TestType.
H	Logical decision vector. Elements of $H = 0$ indicate acceptance of the null hypothesis; elements of $H = 1$ indicate rejection of the null hypothesis. Each element of H is associated with a particular lag of Lags and significance level of Alpha.
PValue	Vector of P values (significance levels) associated with the test decision vector H. Each element of PValue represents the probability of observing a test statistic at least as extreme as that calculated from the OLS regression model when the null hypothesis is true. P values are obtained by interpolation into the appropriate table of critical values, and a NaN is returned if the test statistic lies beyond the range of the table.
TestStat	Vector of test statistics associated with the decision vector H.
CriticalValue	Vector of critical values associated with the decision vector H.

Output Arguments

Notes

Both Lags and Alpha may be vectors. If both are specified as vectors, however, they must be the same length (i.e., have the same number of elements). If one is specified as a scalar and the other as a vector, a scalar expansion is performed to enforce vectors of identical length. If Lags is a scalar or an empty matrix, by default, all outputs are column vectors.

All vector outputs are the same length as vector inputs Alpha and/or Lags. By default all vector outputs are column vectors. If Lags is a row vector, however, all vector outputs are row vectors.

This univariate unit root test is a conventional *single-tailed* test, and the acceptance or rejection of the test is based on a comparison of the test statistic with the critical value:

- The AR and t tests are *lower-tailed* tests. Reject the null hypothesis if the test statistic is *less than* the critical value.
- The joint F test is an *upper-tailed* test. Reject the null hypothesis if the test statistic is *greater than* the critical value.

See Also

dfARDTest, dfARTest, ppARDTest, ppARTest, ppTSTest

References

- [1] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, Princeton, NJ, 1994.
- [2] Greene, W.H., *Econometric Analysis*, Prentice Hall, Fifth edition, Upper Saddle River, NJ, 2003.
- [3] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, New York, 1995.
- [4] Campbell, J.Y., A.W. Lo, and A.C. MacKinlay, *The Econometrics of Financial Markets*, Princeton University Press, Princeton, NJ, 1997.

garchar

Purpose	Convert finite-order ARMA models to infinite-order autoregressive (AR) models						
Syntax	<code>InfiniteAR = garchar(AR, MA, NumLags)</code>						
Description	<code>InfiniteAR = garchar(AR, MA, NumLags)</code> computes the coefficients of an infinite-order AR model, using the coefficients of the equivalent univariate, stationary, invertible, finite-order ARMA(R,M) model as input. <code>garchar</code> truncates the infinite-order AR coefficients to accommodate a user-specified number of lagged AR coefficients.						
Input Arguments	<table><tr><td>AR</td><td><i>R</i>-element vector of autoregressive coefficients associated with the lagged observations of a univariate return series modeled as a finite-order, stationary, invertible ARMA(R,M) model.</td></tr><tr><td>MA</td><td><i>M</i>-element vector of moving-average coefficients associated with the lagged innovations of a finite-order, stationary, invertible univariate ARMA(R,M) model.</td></tr><tr><td>NumLags</td><td>(optional) Number of lagged AR coefficients that <code>garchar</code> includes in the approximation of the infinite-order AR representation. <code>NumLags</code> is an integer scalar and determines the length of the infinite-order AR output vector. If <code>NumLags = []</code> or is not specified, the default is 10.</td></tr></table>	AR	<i>R</i> -element vector of autoregressive coefficients associated with the lagged observations of a univariate return series modeled as a finite-order, stationary, invertible ARMA(R,M) model.	MA	<i>M</i> -element vector of moving-average coefficients associated with the lagged innovations of a finite-order, stationary, invertible univariate ARMA(R,M) model.	NumLags	(optional) Number of lagged AR coefficients that <code>garchar</code> includes in the approximation of the infinite-order AR representation. <code>NumLags</code> is an integer scalar and determines the length of the infinite-order AR output vector. If <code>NumLags = []</code> or is not specified, the default is 10.
AR	<i>R</i> -element vector of autoregressive coefficients associated with the lagged observations of a univariate return series modeled as a finite-order, stationary, invertible ARMA(R,M) model.						
MA	<i>M</i> -element vector of moving-average coefficients associated with the lagged innovations of a finite-order, stationary, invertible univariate ARMA(R,M) model.						
NumLags	(optional) Number of lagged AR coefficients that <code>garchar</code> includes in the approximation of the infinite-order AR representation. <code>NumLags</code> is an integer scalar and determines the length of the infinite-order AR output vector. If <code>NumLags = []</code> or is not specified, the default is 10.						
Output Arguments	<table><tr><td><code>InfiniteAR</code></td><td>Vector of coefficients of the infinite-order AR representation associated with the finite-order ARMA model specified by the AR and MA input vectors. <code>InfiniteAR</code> is a vector of length <code>NumLags</code>. The <i>j</i>th element of <code>InfiniteAR</code> is the coefficient of the <i>j</i>th lag of the input series in an infinite-order AR representation. Note that Box, Jenkins, and Reinsel refer to the infinite-order AR coefficients as “π weights.”</td></tr></table>	<code>InfiniteAR</code>	Vector of coefficients of the infinite-order AR representation associated with the finite-order ARMA model specified by the AR and MA input vectors. <code>InfiniteAR</code> is a vector of length <code>NumLags</code> . The <i>j</i> th element of <code>InfiniteAR</code> is the coefficient of the <i>j</i> th lag of the input series in an infinite-order AR representation. Note that Box, Jenkins, and Reinsel refer to the infinite-order AR coefficients as “ π weights.”				
<code>InfiniteAR</code>	Vector of coefficients of the infinite-order AR representation associated with the finite-order ARMA model specified by the AR and MA input vectors. <code>InfiniteAR</code> is a vector of length <code>NumLags</code> . The <i>j</i> th element of <code>InfiniteAR</code> is the coefficient of the <i>j</i> th lag of the input series in an infinite-order AR representation. Note that Box, Jenkins, and Reinsel refer to the infinite-order AR coefficients as “ π weights.”						

In the following ARMA(R,M) model, $\{y_t\}$ is the return series of interest and $\{\varepsilon_t\}$ the innovations noise process.

$$y_t = \sum_{i=1}^R \phi_i y_{t-i} + \varepsilon_t + \sum_{j=1}^M \theta_j \varepsilon_{t-j}$$

If you write this model equation as

$$y_t = \phi_1 y_{t-1} + \dots + \phi_R y_{t-R} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_M \varepsilon_{t-M}$$

you can specify the garchar input coefficient vectors, AR and MA, exactly as you read them from the model. In general, the j th elements of AR and MA are the coefficients of the j th lag of the return series and innovations processes y_{t-j} and ε_{t-j} , respectively. garchar assumes that the current-time-index coefficients of y_t and ε_t are 1 and are *not* part of AR and MA.

In theory, you can use the π weights returned in InfiniteAR to approximate y_t as a pure AR process.

$$y_t = \sum_{i=1}^{\infty} \pi_i y_{t-i} + \varepsilon_t$$

Consistently, the j th element of the truncated infinite-order autoregressive output vector, π_j or InfiniteAR(j), is the coefficient of the j th lag of the observed return series, y_{t-j} , in this equation. See Box, Jenkins, and Reinsel [8], Section 4.2.3, pages 106-109.

Examples

For the following ARMA(2,2) model, use garchar to obtain the first 20 weights of the infinite-order AR approximation.

$$y_t = 0.5y_{t-1} - 0.8y_{t-2} + \varepsilon_t - 0.6\varepsilon_{t-1} + 0.08\varepsilon_{t-2}$$

From this model,

$$\begin{aligned} \text{AR} &= [0.5 \ -0.8] \\ \text{MA} &= [-0.6 \ 0.08] \end{aligned}$$

Since the current-time-index coefficients of y_t and ε_t are defined to be 1, the example omits them from AR and MA. This saves time and effort when you specify parameters using the garchset and garchget interfaces.

$$\begin{aligned} \text{PI} &= \text{garchar}([0.5 \ -0.8], [-0.6 \ 0.08], 20); \\ \text{PI}' & \end{aligned}$$

```
ans =  
-0.1000  
-0.7800  
-0.4600  
-0.2136  
-0.0914  
-0.0377  
-0.0153  
-0.0062  
-0.0025  
-0.0010  
-0.0004  
-0.0002  
-0.0001  
-0.0000  
-0.0000  
-0.0000  
-0.0000  
-0.0000  
-0.0000  
-0.0000  
-0.0000
```

See Also

garchfit, garchma, garchpred

References

[1] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.

Purpose	Count number of GARCH estimation coefficients		
Syntax	<code>NumParams = garchcount(Coeff)</code>		
Description	<code>NumParams = garchcount(Coeff)</code> counts and returns the number of estimated coefficients from a specification structure, as returned by <code>garchfit</code> , containing coefficient estimates and equality constraint information. <code>garchcount</code> is a helper utility designed to support the model selection function <code>aicbic</code> .		
Input Arguments	<table><tr><td><code>Coeff</code></td><td>Specification structure containing coefficient estimates and equality constraints. <code>Coeff</code> is an output of the estimation function <code>garchfit</code>.</td></tr></table>	<code>Coeff</code>	Specification structure containing coefficient estimates and equality constraints. <code>Coeff</code> is an output of the estimation function <code>garchfit</code> .
<code>Coeff</code>	Specification structure containing coefficient estimates and equality constraints. <code>Coeff</code> is an output of the estimation function <code>garchfit</code> .		
Output Arguments	<table><tr><td><code>NumParams</code></td><td>Number of estimated parameters, i.e., coefficients, included in the conditional mean and variance specifications, less any parameters held constant, as equality constraints, during the estimation. The <code>aicbic</code> function needs <code>NumParams</code> to calculate the Akaike (AIC) and Bayesian (BIC) statistics.</td></tr></table>	<code>NumParams</code>	Number of estimated parameters, i.e., coefficients, included in the conditional mean and variance specifications, less any parameters held constant, as equality constraints, during the estimation. The <code>aicbic</code> function needs <code>NumParams</code> to calculate the Akaike (AIC) and Bayesian (BIC) statistics.
<code>NumParams</code>	Number of estimated parameters, i.e., coefficients, included in the conditional mean and variance specifications, less any parameters held constant, as equality constraints, during the estimation. The <code>aicbic</code> function needs <code>NumParams</code> to calculate the Akaike (AIC) and Bayesian (BIC) statistics.		
Example	See “Akaike and Bayesian Information Criteria” on page 9-5.		
See Also	<code>aicbic</code> , <code>garchdisp</code> , <code>garchfit</code>		

garchdisp

Purpose GARCH process estimation results

Syntax `garchdisp(Coeff, Errors)`

Description `garchdisp(Coeff, Errors)` displays coefficient estimates, standard errors, and T-statistics from a GARCH specification structure that was output by the estimation function `garchfit`.

This function displays estimation results, and returns no output arguments. The tabular display includes parameter estimates, standard errors, and T-statistics for each parameter in the conditional mean and variance models. Parameters held fixed during the estimation process have the word 'Fixed' displayed in the standard error and T-statistic columns, indicating that the parameter was set as an equality constraint.

Input Arguments

<code>Coeff</code>	GARCH specification structure containing estimated coefficients and equality constraint information. <code>Coeff</code> is an output of the estimation function <code>garchfit</code> .
<code>Errors</code>	Structure containing the estimation errors (i.e., the standard errors) of the coefficients in <code>Coeff</code> . <code>Errors</code> is also an output of the estimation function <code>garchfit</code> .

Examples The following code uses `garchfit` to generate the GARCH specification structure `Coeff` and the standard errors structure `Errors` for a return series of 1000 simulated observations based on a GARCH(1,1) model. It then calls `garchdisp` to display the estimation results. Setting 'Display' to 'off' suppresses display of the iterative optimization information produced by `garchfit`.

```
spec = garchset('C', 0, 'K', 0.0001, 'GARCH', 0.9, 'ARCH', ...  
0.05, 'Display', 'off');  
[e, s, y] = garchsim(spec, 1000);  
[Coeff, Errors] = garchfit(spec, y);  
garchdisp(Coeff, Errors)
```

```
Mean: ARMAX(0, 0, 0); Variance: GARCH(1, 1)
```

```
Conditional Probability Distribution: Gaussian  
Number of Model Parameters Estimated: 4
```

Parameter	Value	Standard Error	T Statistic
C	-0.0024759	0.0012919	-1.9165
K	4.6877e-005	5.3555e-005	0.8753
GARCH(1)	0.93904	0.041604	22.5707
ARCH(1)	0.035503	0.015123	2.3477

See Also

garchcount, garchfit

garchfit

Purpose Univariate GARCH process parameter estimation

Syntax

```
[Coeff, Errors, LLF, Innovations, Sigmas, Summary] =  
    garchfit(Series)  
[...] = garchfit(Spec, Series)  
[...] = garchfit(Spec, Series, X)  
[...] = garchfit(Spec, Series, X, PreInnovations, PreSigmas,  
    PreSeries)  
garchfit(...)
```

Description Given an observed univariate return series, `garchfit` estimates the parameters of a conditional mean specification of ARMAX form, and conditional variance specification of GARCH, EGARCH, or GJR form. The estimation process infers the innovations (i.e., residuals) from the return series, and fits the model specification to the return series by maximum likelihood.

`[Coeff, Errors, LLF, Innovations, Sigmas, Summary] = garchfit(Series)` models an observed univariate return series as a constant, C , plus GARCH(1,1) conditionally Gaussian innovations. For models beyond this simple (yet common) model, you must provide model parameters in the GARCH specification structure `Spec`.

`[...] = garchfit(Spec, Series)` infers the innovations from the return series and fits the model specification, contained in `Spec`, to the return series by maximum likelihood.

`[...] = garchfit(Spec, Series, X)` provides a regression component X for the conditional mean.

`[...] = garchfit(Spec, Series, X, PreInnovations, PreSigmas, PreSeries)` uses presample observations, contained in the time-series column vectors `PreInnovations`, `PreSigmas`, and `PreSeries`, to infer the outputs `Innovations` and `Sigmas`. These vectors form the conditioning set used to initiate the inverse filtering, or inference, process. If you provide no explicit presample data, the necessary presample observations are derived by conventional time-series techniques (see “Automatic Minimization of Transient Effects” on page 4-7).

If you specify at least one set, but fewer than three sets, of presample data, `garchsim` does not attempt to derive presample observations for those you omit. If you specify your own presample data, you must specify all that are necessary for the specified conditional mean and variance models. See “User-Specified Presample Observations” on page 5-11.

`garchfit(...)` with input arguments as shown above but with no output arguments, displays the final parameter estimates and standard errors. It also produces a tiered plot of the original return series, the inferred innovations, and the corresponding conditional standard deviations.

Input Arguments

Spec	GARCH specification structure containing the conditional mean and variance specifications. It also contains the optimization parameters needed for the estimation. Create this structure by calling <code>garchset</code> , or use the <code>Coeff</code> output structure returned by <code>garchfit</code> .
Series	Time-series column vector of observations of the underlying univariate return series of interest. <code>Series</code> is the response variable representing the time series to be fitted to conditional mean and variance specifications. The last element of <code>Series</code> holds the most recent observation.
X	Time-series regression matrix of observed explanatory data. Typically, <code>X</code> is a matrix of asset returns (e.g., the return series of an equity index), and represents the past history of the explanatory data. Each column of <code>X</code> is an individual time series used as an explanatory variable in the regression component of the conditional mean. In each column, the first row contains the oldest observation and the last row the most recent. The number of valid (non-NaN) most recent observations in each column of <code>X</code> must equal or exceed the number of valid most recent observations in <code>Series</code> . If the number of valid observations in a column of <code>X</code> exceeds that of <code>Series</code> , <code>garchfit</code> uses only the most recent observations of <code>X</code> . If <code>X = []</code> or is not specified, the conditional mean has no regression component.

- PreInnovations** Time-series column vector of presample innovations that `garchfit` uses to condition the recursive mean and variance models. This column vector can have any number of rows, provided it contains sufficient observations to initialize the mean and variance equations. I.e., if M and Q are the number of lagged innovations required by the conditional mean and variance equations, respectively, then `PreInnovations` must have at least $\max(M, Q)$ rows. If the number of rows exceeds $\max(M, Q)$, then `garchfit` uses only the last (i.e., most recent) $\max(M, Q)$ rows.
- PreSigmas** Time-series column vector of positive presample conditional standard deviations that `garchfit` uses to condition the recursive variance model. This vector can have any number of rows, provided it contains sufficient observations to initialize the conditional variance equation. I.e., if P and Q are the number of lagged conditional standard deviations and lagged innovations required by the conditional variance equation, respectively, then `PreSigmas` must have at least P rows for GARCH and GJR models, and at least $\max(P, Q)$ rows for EGARCH models. If the number of rows exceeds the requirement, then `garchfit` uses only the last (i.e., most recent) rows.
- PreSeries** Time-series column vector of presample observations of the return series of interest that `garchfit` uses to condition the recursive mean model. This vector can have any number of rows, provided it contains sufficient observations to initialize the conditional mean equation. Thus, if R is the number of lagged observations of the return series required by the conditional mean equation, then `PreSeries` must have at least R rows. If the number of rows exceeds R , then `garchfit` uses only the last (i.e., most recent) R rows.

Output Arguments

Coeff	GARCH specification structure containing the estimated coefficients. <code>Coeff</code> is of the same form as the <code>Spec</code> input structure. Toolbox functions such as <code>garchset</code> , <code>garchget</code> , <code>garchsim</code> , <code>garchinfer</code> , and <code>garchpred</code> can accept either <code>Spec</code> or <code>Coeff</code> as input arguments.				
Errors	Structure containing the estimation errors (i.e., the standard errors) of the coefficients. <code>Errors</code> is of the same form as the <code>Spec</code> and <code>Coeff</code> structures. In the event an error occurs in the calculation of the standard errors, all fields associated with estimated coefficients are set to NaN.				
LLF	Optimized log-likelihood objective function value associated with the parameter estimates found in <code>Coeff</code> . <code>garchfit</code> performs the optimization using the <code>fmincon</code> function of the Optimization Toolbox.				
Innovations	Innovations (i.e., residuals) time-series column vector inferred from <code>Series</code> . The size of <code>Innovations</code> is the same as the size of <code>Series</code> . In the event of an error, <code>Innovations</code> is a vector of NaNs.				
Sigmas	Conditional standard deviation vector corresponding to <code>Innovations</code> . The size of <code>Sigmas</code> is the same as the size of <code>Series</code> . In the event of an error, <code>Sigmas</code> is a vector of NaNs.				
Summary	Structure of summary information about the optimization process. The fields and their possible values are <table> <tr> <td><code>exitFlag</code></td> <td>Describes the exit condition: >0 Log-likelihood objective function converged to a solution. 0 Maximum number of function evaluations or iterations was exceeded. <0 Function did not converge to a solution.</td> </tr> <tr> <td><code>warning</code></td> <td>One of the following strings: 'No Warnings' 'ARMA Model Is Not Stationary/Invertible'</td> </tr> </table>	<code>exitFlag</code>	Describes the exit condition: >0 Log-likelihood objective function converged to a solution. 0 Maximum number of function evaluations or iterations was exceeded. <0 Function did not converge to a solution.	<code>warning</code>	One of the following strings: 'No Warnings' 'ARMA Model Is Not Stationary/Invertible'
<code>exitFlag</code>	Describes the exit condition: >0 Log-likelihood objective function converged to a solution. 0 Maximum number of function evaluations or iterations was exceeded. <0 Function did not converge to a solution.				
<code>warning</code>	One of the following strings: 'No Warnings' 'ARMA Model Is Not Stationary/Invertible'				

converge	One of the following strings: 'Function Converged to a Solution' 'Function Did NOT Converge' 'Maximum Function Evaluations or Iterations Reached'
constraints	One of the following strings: 'No Boundary Constraints' 'Boundary Constraints Active; Errors May Be Inaccurate'
covMatrix	Covariance matrix of the parameter estimates
iterations	Number of iterations
functionCalls	Number of function evaluations
lambda	Structure, output by fmincon, containing the Lagrange multipliers at the solution x

Note garchfit calculates the error covariance matrix of the parameter estimates Summary.covMatrix, and the corresponding standard errors found in the Errors output structure using finite difference approximation. In particular, it calculates the standard errors using the outer-product method (see Hamilton [8], Section 5.8, bottom of page 143).

Example 1

The following code uses garchfit to estimate the parameters for a return series of 1000 simulated observations based on a GARCH(1,1) model. Because the 'Display' parameter defaults to 'on', garchfit displays diagnostic and iterative information.

```
spec = garchset('C', 0, 'K', 0.0001, 'GARCH', 0.9, 'ARCH', 0.05);  
[e, s, y] = garchsim(spec, 1000);  
[Coeff, Errors] = garchfit(spec, y);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
Diagnostic Information
```

```
Number of variables: 4
```

```

Functions
Objective:          garchllfn
Gradient:          finite-differencing
Hessian:           finite-differencing (or Quasi-Newton)
Nonlinear constraints:  armanlc
Gradient of nonlinear constraints: finite-differencing
    
```

```

Constraints
Number of nonlinear inequality constraints: 0
Number of nonlinear equality constraints:  0

Number of linear inequality constraints:   1
Number of linear equality constraints:     0
Number of lower bound constraints:        4
Number of upper bound constraints:        4
    
```

```

Algorithm selected
  medium-scale
    
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
End diagnostic information
    
```

Iter	F-count	f(x)	max constraint	Step-size	Directional derivative	First-order optimality	Procedure
1	22	-1762.62	-9.975e-005	0.000488	1.32e+004	1.47e+004	
2	35	-1763.04	-9.897e-005	0.00781	126	2.13e+005	
3	43	-1764.69	-7.423e-005	0.25	4.19	1.28e+005	
4	57	-1764.72	-7.477e-005	0.00391	6.92	1.12e+005	
5	64	-1765.27	-4.128e-005	0.5	0.228	3.59e+003	
6	78	-1765.28	-4.751e-005	0.00391	3.34	2.98e+004	
7	89	-1765.28	-4.617e-005	0.0313	0.0725	2.91e+004	
8	101	-1765.29	-4.927e-005	0.0156	0.39	84	
9	107	-1765.29	-4.73e-005	1	-0.000969	6.06	
10	114	-1765.29	-4.668e-005	0.5	-0.000135	213	
11	134	-1765.29	-4.668e-005	-6.1e-005	-2.4e-005	213	Hessian modified
12	140	-1765.29	-4.668e-005	1	1.39e-007	19.5	Hessian modified twice

```

Optimization terminated successfully:
Magnitude of directional derivative in search direction
less than 2*options.TolFun and maximum constraint violation
is less than options.TolCon
No Active Constraints
    
```

Example 2 Using the same data as above, the example sets 'Display' to 'off' and calls garchfit with no output arguments. In this case, garchfit displays the final parameter estimates and standard errors, then produces a tiered plot.

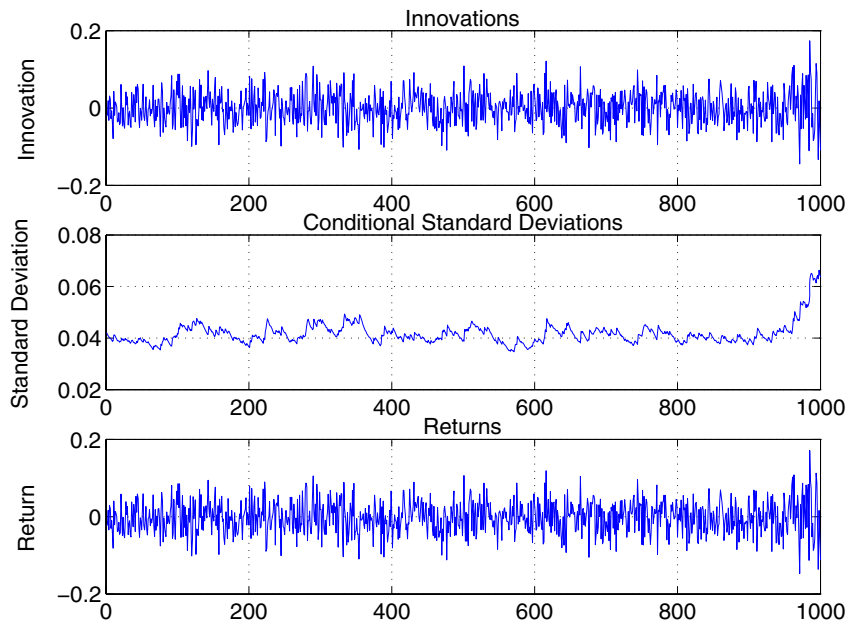
```
spec = garchset(spec, 'Display', 'off');  
garchfit(spec, y)
```

Mean: ARMAX(0, 0, 0); Variance: GARCH(1, 1)

Conditional Probability Distribution: Gaussian
Number of Model Parameters Estimated: 4

Parameter	Value	Standard Error	T Statistic
C	-0.0024759	0.0012919	-1.9165
K	4.6877e-005	5.3555e-005	0.8753
GARCH(1)	0.93904	0.041604	22.5707
ARCH(1)	0.035503	0.015123	2.3477

Log Likelihood Value: 1765.29



See Also

garchpred, garchset, garchsim
fmincon (in the Optimization Toolbox)

References

[1] Bollerslev, T., "A Conditionally Heteroskedastic Time Series Model for Speculative Prices and Rates of Return," *Review of Economics and Statistics*, Vol. 69, 1987, pp 542-547.

[2] Bollerslev, T., "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, Vol. 31, 1986, pp 307-327.

[3] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.

[4] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, 1995.

[5] Engle, Robert, "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica*, Vol. 50, 1982, pp 987-1007.

[6] Engle, R.F., D.M. Liliien, and R.P. Robins, "Estimating Time Varying Risk Premia in the Term Structure: The ARCH-M Model," *Econometrica*, Vol. 59, 1987, pp 391-407.

[7] Glosten, L.R., R. Jagannathan, and D.E. Runkle, "On the Relation Between Expected Value and the Volatility of the Nominal Excess Return on Stocks," *The Journal of Finance*, Vol.48, 1993, pp 1779-1801.

[8] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, 1994.

[9] Nelson, D.B., "Conditional Heteroskedasticity in Asset Returns: A New Approach," *Econometrica*, Vol. 59, 1991, pp 347-370.

garchget

Purpose	GARCH specification structure parameter				
Syntax	<code>ParameterValue = garchget(Spec, 'ParameterName')</code>				
Description	<code>ParameterValue = garchget(Spec, 'ParameterName')</code> returns the value of the specified parameter from the GARCH specification structure <code>Spec</code> .				
Input Arguments	<table><tr><td><code>Spec</code></td><td>GARCH specification structure returned by <code>garchset</code>, or the output (<code>Coeff</code>) of the estimation function <code>garchfit</code>.</td></tr><tr><td><code>ParameterName</code></td><td>String indicating the name of the parameter whose value <code>garchget</code> extracts from <code>Spec</code>. It is sufficient to type only the leading characters that uniquely identify a parameter name. See <code>garchset</code> for a list of valid parameter names. <code>ParameterName</code> is case insensitive.</td></tr></table>	<code>Spec</code>	GARCH specification structure returned by <code>garchset</code> , or the output (<code>Coeff</code>) of the estimation function <code>garchfit</code> .	<code>ParameterName</code>	String indicating the name of the parameter whose value <code>garchget</code> extracts from <code>Spec</code> . It is sufficient to type only the leading characters that uniquely identify a parameter name. See <code>garchset</code> for a list of valid parameter names. <code>ParameterName</code> is case insensitive.
<code>Spec</code>	GARCH specification structure returned by <code>garchset</code> , or the output (<code>Coeff</code>) of the estimation function <code>garchfit</code> .				
<code>ParameterName</code>	String indicating the name of the parameter whose value <code>garchget</code> extracts from <code>Spec</code> . It is sufficient to type only the leading characters that uniquely identify a parameter name. See <code>garchset</code> for a list of valid parameter names. <code>ParameterName</code> is case insensitive.				
Output Arguments	<table><tr><td><code>ParameterValue</code></td><td>Value of the named parameter extracted from the structure <code>Spec</code>. <code>garchget</code> returns the appropriate model default value if the specified parameter is not defined in the specification structure.</td></tr></table>	<code>ParameterValue</code>	Value of the named parameter extracted from the structure <code>Spec</code> . <code>garchget</code> returns the appropriate model default value if the specified parameter is not defined in the specification structure.		
<code>ParameterValue</code>	Value of the named parameter extracted from the structure <code>Spec</code> . <code>garchget</code> returns the appropriate model default value if the specified parameter is not defined in the specification structure.				
Examples	<pre>Spec = garchset('P', 1, 'Q', 1) % Create a GARCH(P=1, Q=1) model. Spec = Comment: 'Mean: ARMAX(0, 0, ?); Variance: GARCH(1, 1)' Distribution: 'Gaussian' C: [] VarianceModel: 'GARCH' P: 1 Q: 1 K: [] GARCH: [] ARCH: [] P = garchget(Spec, 'P') % Retrieve the order P. P = 1</pre>				
See Also	<code>garchfit</code> , <code>garchpred</code> , <code>garchset</code> , <code>garchsim</code>				

Purpose	Infer GARCH innovation processes from return series
Syntax	<pre>[Innovations, Sigmas, LLF] = garchinfer(Spec, Series) [...] = garchinfer(Spec, Series, X) [...] = garchinfer(Spec, Series, X, PreInnovations, PreSigmas, PreSeries)</pre>
Description	<p>[Innovations, Sigmas, LLF] = garchinfer(Spec, Series) given a conditional mean specification of ARMAX form and conditional variance specification of GARCH, EGARCH, or GJR form, infers the innovations and conditional standard deviations from an observed univariate return series. Since garchinfer is an interface to the appropriate log-likelihood objective function, the log-likelihood value is also computed for convenience.</p> <p>[...] = garchinfer(Spec, Series, X) also accepts a time-series regression matrix X of observed explanatory data. garchinfer treats each column of X as an individual time series, and uses it as an explanatory variable in the regression component of the conditional mean.</p> <p>[...] = garchinfer(Spec, Series, X, PreInnovations, PreSigmas, PreSeries) uses presample observations, contained in the time-series matrices or column vectors PreInnovations, PreSigmas, and PreSeries, to infer the outputs Innovations and Sigmas. These vectors form the conditioning set used to initiate the inverse filtering, or inference, process.</p> <p>If you specify the presample data as matrices, the number of columns (realizations) of each <i>must</i> be the same as the number of columns (realizations) of the Series input. In this case, the presample information of a given column is used to infer the residuals and standard deviations of the corresponding column of Series. If you specify the presample data as column vectors, the vectors are applied to each column of Series.</p> <p>If you provide no explicit presample data, the necessary presample observations are derived by conventional time-series techniques (see “Automatic Minimization of Transient Effects” on page 4-7).</p> <p>If you specify at least one set, but fewer than three sets, of presample data, garchsim does not attempt to derive presample observations for those you omit. If you specify your own presample data, you must specify all that are necessary</p>

for the specified conditional mean and variance models. See “User-Specified Presample Observations” on page 5-11.

Input Arguments

Spec	GARCH specification structure containing the conditional mean and variance specifications. It also contains the optimization parameters needed for the estimation. Create this structure by calling <code>garchset</code> , or use the <code>Coeff</code> output structure returned by <code>garchfit</code> .
Series	Time-series matrix or column vector of observations of the underlying univariate return series of interest. <code>Series</code> is the response variable representing the time series fitted to conditional mean and variance specifications. Each column of <code>Series</code> is an independent realization (i.e., path). The last row of <code>Series</code> holds the most recent observation of each realization.
X	Time-series regression matrix of explanatory variables. Typically, <code>X</code> is a regression matrix of asset returns (e.g., the return series of an equity index). Each column of <code>X</code> is an individual time series used as an explanatory variable in the regression component of the conditional mean. In each column, the first row contains the oldest observation and the last row the most recent. The number of valid (non- <code>NaN</code>) observations below the last <code>NaN</code> in each column of <code>X</code> must equal or exceed the number of valid observations below the last <code>NaN</code> in <code>Series</code> . If the number of valid observations in a column of <code>X</code> exceeds that of <code>Series</code> , <code>garchinfer</code> uses only the most recent. If <code>X = []</code> or is not specified, the conditional mean has no regression component.

PreInnovations Time-series matrix or column vector of presample innovations on which the recursive mean and variance models are conditioned. This array can have any number of rows, provided it contains sufficient observations to initialize the mean and variance equations. I.e., if M and Q are the number of lagged innovations required by the conditional mean and variance equations, respectively, then **PreInnovations** must have at least $\max(M, Q)$ rows. If the number of rows exceeds $\max(M, Q)$, then only the last (i.e., most recent) $\max(M, Q)$ rows are used. If **PreInnovations** is a matrix, then the number of columns must be the same as the number of columns in **Series**. If **PreInnovations** is a column vector, then the vector is applied to each column (i.e., realization) of **Series**.

PreSigmas	<p>Time-series matrix or column vector of positive presample conditional standard deviations on which the recursive variance model is conditioned. This array can have any number of rows, provided it contains sufficient observations to initialize the conditional variance equation. I.e., if P and Q are the number of lagged conditional standard deviations and lagged innovations required by the conditional variance equation, respectively, then <code>PreSigmas</code> must have at least P rows for GARCH and GJR models, and at least $\max(P, Q)$ rows for EGARCH models.</p> <p>If the number of rows exceeds the requirement, then only the last (i.e., most recent) rows are used. If <code>PreSigmas</code> is a matrix, then the number of columns must be the same as the number of columns in <code>Series</code>. If <code>PreSigmas</code> is a column vector, then the vector is applied to each column (i.e., realization) of <code>Series</code>.</p>
PreSeries	<p>Time-series matrix or column vector of presample observations of the return series of interest on which the recursive mean model is conditioned. This array can have any number of rows, provided it contains sufficient observations to initialize the conditional mean equation. Thus, if R is the number of lagged observations of the return series required by the conditional mean equation, then <code>PreSeries</code> must have at least R rows. If the number of rows exceeds R, then only the last (i.e., most recent) R rows are used. If <code>PreSeries</code> is a matrix, then the number of columns must be the same as the number of columns in <code>Series</code>. If <code>PreSeries</code> is a column vector, then the vector is applied to each column (i.e., realization) of <code>Series</code>.</p>

Output Arguments

Innovations	Innovations time-series matrix inferred from Series. The size of Innovations is the same as the size of Series.
Sigmas	Conditional standard deviation time-series matrix corresponding to Innovations. The size of Sigmas is the same as the size of Series.
LLF	Row vector of log-likelihood objective function values for each realization of Series. The length of LLF is the same as the number of columns in Series.

Remarks

garchinfer performs essentially the same operation as garchfit, but without the optimization. While garchfit calls the appropriate log-likelihood objective function indirectly via the iterative numerical optimizer, garchinfer allows you direct access to the same suite of log-likelihood objective functions.

Note that, for garchinfer, inputs Series, PreInnovations, PreSigmas, and PreSeries, and outputs Innovations and Sigmas, are column-oriented time-series arrays in which each column is associated with a unique realization, or random path. For garchfit, these same inputs and outputs cannot have multiple columns; i.e., they must all represent single realizations of a univariate time series.

For additional details about estimation and inverse filtering, see “Maximum Likelihood Estimation” on page 5-2 and “Presample Observations” on page 5-11.

Examples

See “Presample Data and Transient Effects” on page 5-23, “Presample Observations” on page 6-5, and “Estimating the Model” on page 10-2.

See Also

garchfit, garchpred, garchset, garchsim

References

[1] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.

[2] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, 1994.

garchma

Purpose Convert finite-order ARMA models to infinite-order moving average (MA) models

Syntax `InfiniteMA = garchma(AR, MA, NumLags)`

Description `InfiniteMA = garchma(AR, MA, NumLags)` computes the coefficients of an infinite-order MA model, using the coefficients of the equivalent univariate, stationary, invertible, finite-order ARMA(R,M) model as input. `garchma` truncates the infinite-order MA coefficients to accommodate the number of lagged MA coefficients you specify in `NumLags`.

This function is particularly useful for calculating the standard errors of minimum mean square error forecasts of univariate ARMA models.

Arguments

AR	<i>R</i> -element vector of autoregressive coefficients associated with the lagged observations of a univariate return series modeled as a finite-order, stationary, invertible ARMA(R,M) model.
MA	<i>M</i> -element vector of moving-average coefficients associated with the lagged innovations of a finite-order, stationary, invertible, univariate ARMA(R,M) model.
NumLags	(optional) Number of lagged MA coefficients that <code>garchma</code> includes in the approximation of the infinite-order MA representation. <code>NumLags</code> is an integer scalar and determines the length of the infinite-order MA output vector. If <code>NumLags = []</code> or is not specified, the default is 10.

Output Arguments

<code>InfiniteMA</code>	Vector of coefficients of the infinite-order MA representation associated with the finite-order ARMA model specified by <code>AR</code> and <code>MA</code> . <code>InfiniteMA</code> is a vector of length <code>NumLags</code> . The <i>j</i> th element of <code>InfiniteMA</code> is the coefficient of the <i>j</i> th lag of the innovations noise sequence in an infinite-order MA representation. Note that Box, Jenkins, and Reinsel refer to the infinite-order MA coefficients as the “ ψ weights.”
-------------------------	---

In the following ARMA(R,M) model, $\{y_t\}$ is the return series of interest and $\{\varepsilon_t\}$ the innovations noise process.

$$y_t = \sum_{i=1}^R \phi_i y_{t-i} + \varepsilon_t + \sum_{j=1}^M \theta_j \varepsilon_{t-j}$$

If you write this model equation as

$$y_t = \phi_1 y_{t-1} + \dots + \phi_R y_{t-R} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_M \varepsilon_{t-M}$$

you can specify the garchma input coefficient vectors, AR and MA, exactly as you read them from the model. In general, the j th elements of AR and MA are the coefficients of the j th lag of the return series and innovations processes y_{t-j} and ε_{t-j} , respectively. garchma assumes that the current-time-index coefficients of y_t and ε_t are 1 and are not part of AR and MA.

In theory, you can use the ψ weights returned in `InfiniteMA` to approximate y_t as a pure MA process.

$$y_t = \varepsilon_t + \sum_{i=1}^{\infty} \psi_i \varepsilon_{t-i}$$

Consistently, the j th element of the truncated infinite-order moving-average output vector, ψ_j or `InfiniteMA(j)`, is the coefficient of the j th lag of the innovations process, ε_{t-j} , in this equation. See Box, Jenkins, and Reinsel [8], Section 5.2.2, pages 139-141.

Examples

Suppose you want a forecast horizon of 10 periods for the following ARMA(2,2) model.

$$y_t = 0.5y_{t-1} - 0.8y_{t-2} + \varepsilon_t - 0.6\varepsilon_{t-1} + 0.08\varepsilon_{t-2}$$

To obtain probability limits for these forecasts, use garchma to compute the first 9 (i.e., $10 - 1$) weights of the infinite order MA approximation.

From the model, AR = [0.5 -0.8] and MA = [-0.6 0.08].

Since the current-time-index coefficients of y_t and ε_t are 1, the example omits them from AR and MA. This saves time and effort when you specify parameters via the garchset and garchget user interfaces.

garchma

```
PSI = garchma([0.5 -0.8], [-0.6 0.08], 9);  
PSI'
```

```
ans =
```

```
-0.1000  
-0.7700  
-0.3050  
0.4635  
0.4758  
-0.1329  
-0.4471  
-0.1172  
0.2991
```

See Also

garchar, garchpred

References

[1] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.

Purpose	Plot matched univariate innovations, volatility, and return series
Syntax	<code>garchplot(Innovations, Sigmas, Series)</code>
Description	<p><code>garchplot</code> lets you visually compare matched innovations, conditional standard deviations, and returns. It provides a convenient way to compare innovations series, simulated using <code>garchsim</code> or estimated using <code>garchfit</code>, with companion conditional standard deviations, or returns series. You can also use <code>garchplot</code> to plot forecasts, computed using <code>garchpred</code>, of conditional standard deviations and returns.</p> <p>In general, <code>garchplot</code> produces a tiered plot of matched time series. <code>garchplot</code> does not display an empty or missing input array; i.e., <code>garchplot</code> allocates no space in the tiered figure window to the array. <code>garchplot</code> displays valid (nonempty) <code>Innovations</code>, <code>Sigmas</code>, and <code>Series</code> arrays in the top, center, and bottom plots, respectively. Since <code>garchplot</code> assigns a title and label to each plot according to its position in the argument list, you can ensure correct plot annotation by using empty matrices (<code>[]</code>) as placeholders.</p> <p>You can plot several realizations of each array simultaneously because <code>garchplot</code> color codes corresponding realizations of each input array. However, the plots can become cluttered if you try to display more than a few realizations of each input at one time.</p>
Input Arguments	<p><code>Innovations</code> Time-series column vector or matrix of innovations. As a column vector, <code>Innovations</code> represents a single realization of a univariate time series in which the first element contains the oldest observation and the last element the most recent. As a matrix, each column of <code>Innovations</code> represents a single realization of a univariate time series in which the first row contains the oldest observation of each realization and the last row the most recent. If <code>Innovations = []</code>, then <code>Innovations</code> is not displayed.</p>

Sigmas	Time-series column vector or matrix of conditional standard deviations. In general, Innovations and Sigmas are the same size, and form a matching pair of arrays. If Sigmas = [], then Sigmas is not displayed.
Series	Time-series column vector or matrix of asset returns. In general, Series is the same size as Innovations and Sigmas, and is organized in exactly the same manner. If Series = [] or is not specified, then Series is not displayed.

Examples

Example 1. Assume that Innovations, Sigmas, and Series are not empty.

```
garchplot(Innovations)           % Plot Innovations only.

garchplot(Innovations, [], Series)% Plot Innovations and
                                % Series only.

garchplot([], Sigmas, Series) % Plot Sigmas and Series only.

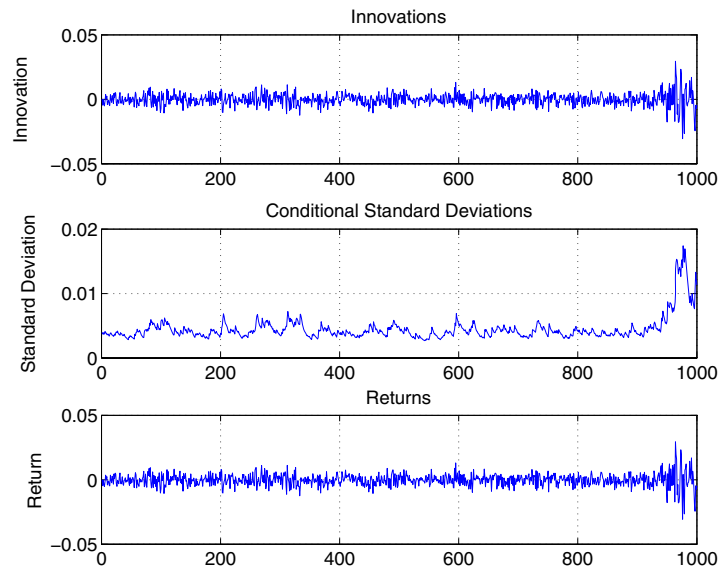
garchplot(Innovations, Sigmas, Series) % Plot all three vectors.

garchplot(Innovations, Sigmas, [])    % Plot Innovations and
                                % Sigmas only.

garchplot(Innovations, Sigmas)       % Plot Innovations and
                                % Sigmas only.
```

Example 2. The following code uses the default GARCH(1,1) model to model the Deutschmark/British pound foreign exchange series (see “Data Sets” on page 1-11). It then uses the estimated model to generate a single path of 1000 observations for return series, innovations, and conditional standard deviation processes.

```
load garchdata
dem2gbp = price2ret(DEM2GBP);
[coeff, errors, LLF, innovations, sigmas] = garchfit(dem2gbp);
[e, s, y] = garchsim(coeff, 1000);
garchplot(e, s, y)
```



See Also

`garchfit`, `garchpred`, `garchsim`

garchpred

Purpose Univariate GARCH process forecasting

Syntax

```
[SigmaForecast, MeanForecast] = garchpred(Spec, Series, NumPeriods)
[SigmaForecast, MeanForecast] = garchpred(Spec, Series, NumPeriods,
    X, XF)
[SigmaForecast, MeanForecast, SigmaTotal, MeanRMSE] =
    garchpred(Spec, Series, NumPeriods)
```

Description garchpred forecasts the conditional mean of the univariate return series and the standard deviation of the innovations NumPeriods into the future, using specifications for the conditional mean and variance of an observed univariate return series as input. garchpred also computes volatility forecasts of asset returns over multiperiod holding intervals, and the standard errors of conditional mean forecasts. The conditional mean is of general ARMAX form and the conditional variance can be of GARCH, EGARCH, or GJR form. (See “Conditional Mean and Variance Models” on page 2-6.)

[SigmaForecast, MeanForecast] = garchpred(Spec, Series, NumPeriods) uses the conditional mean and variance specifications defined in Spec to forecast the conditional mean, MeanForecast, of the univariate return series and the standard deviation, SigmaForecast, of the innovations NumPeriods into the future. The NumPeriods default is 1.

[SigmaForecast, MeanForecast] = garchpred(Spec, Series, NumPeriods, X, XF) includes the time-series regression matrix of observed explanatory data X and the time-series regression matrix of forecasted explanatory data XF in the calculation of MeanForecast. For MeanForecast, if you specify X, you must also specify XF. Typically, X is the same regression matrix of observed returns, if any, that you used for simulation (garchsim) or estimation (garchfit).

[SigmaForecast, MeanForecast, SigmaTotal, MeanRMSE] = garchpred(Spec, Series, Numperiods) also computes the volatility forecasts, SigmaTotal, of the cumulative returns for assets held for multiple periods, and the standard errors MeanRMSE associated with MeanForecast.

Input Arguments

Spec	Specification structure for the conditional mean and variance models. You can create Spec using the function <code>garchset</code> or the estimation function <code>garchfit</code> .
Series	Matrix of observations of the underlying univariate return series of interest for which <code>garchpred</code> generates forecasts. Each column of Series is an independent realization (i.e., path). The last row of Series holds the most recent observation of each realization. <code>garchpred</code> treats those observations as valid that are below the most recent NaN in any column. <code>garchpred</code> assumes that Series is a stationary stochastic process. It also assumes that the ARMA component of the conditional mean model (if any) is stationary and invertible.
NumPeriods	Positive scalar integer representing the forecast horizon of interest. It is expressed in periods, and should be compatible with the sampling frequency of Series. If NumPeriods = [] or is not specified, the default is 1.

- X** Time-series regression matrix of observed explanatory data that represents the past history of the explanatory data. Typically, X is a regression matrix of asset returns, e.g., the return series of an equity index. Each column of X is an individual time series used as an explanatory variable in the regression component of the conditional mean. In each column, the first row contains the oldest observation and the last row the most recent. The most recent number of valid (non-NaN) observations in each column of X must equal or exceed the most recent number of valid observations in `Series`. If the number of valid observations in a column of X exceeds that of `Series`, `garchpred` uses only the most recent observations of X . If $X = []$ or is not specified, the conditional mean has no regression component.
- XF** Time-series matrix of forecasted explanatory data. XF represents the evolution into the future of the same explanatory data found in X . Because of this, XF and X must have the same number of columns. In each column of XF , the first row contains the one-period-ahead forecast, the second row contains the two-period-ahead forecast, and so on. The number of rows (forecasts) in each column (time series) of XF must equal or exceed the forecast horizon `NumPeriods`. When the number of forecasts in XF exceeds `NumPeriods`, `garchpred` uses only the first `NumPeriods` forecasts. If $XF = []$ or is not specified, the conditional mean forecast (`MeanForecast`) has no regression component.

Output Arguments

- SigmaForecast** Matrix of conditional standard deviations of future innovations (i.e., model residuals) on a per period basis. This matrix represents the standard deviations derived from the minimum mean square error (MMSE) forecasts associated with the recursive volatility model, e.g., 'GARCH', 'GJR', or 'EGARCH', specified for the 'VarianceModel' parameter in Spec. For GARCH(P,Q) and GJR(P,Q) models, SigmaForecast is the square root of the MMSE conditional variance forecasts. For EGARCH(P,Q) models, SigmaForecast is the square root of the exponential of the MMSE forecasts of the logarithm of conditional variance. SigmaForecast has NumPeriods rows and the same number of columns as Series. The first row contains the standard deviation in the first period for each realization of Series, the second row contains the standard deviation in the second period, and so on. If you specify a forecast horizon greater than 1, i.e., NumPeriods > 1, garchpred returns the per-period standard deviations of all intermediate horizons as well; in this case, the last row contains the standard deviation at the specified forecast horizon.
- MeanForecast** Matrix of MMSE forecasts of the conditional mean of Series on a per-period basis. MeanForecast is the same size as SigmaForecast. The first row contains the forecast in the first period for each realization of Series, the second row contains the forecast in the second period, and so on. Both X and XF must be nonempty for MeanForecast to have a regression component. If X and XF are empty ([]) or not specified, MeanForecast is based on an ARMA model. If you specify X and XF, MeanForecast is based on the full ARMAX model.

<code>SigmaTotal</code>	<p>Matrix of MMSE volatility forecasts of <code>Series</code> over multiperiod holding intervals. <code>SigmaTotal</code> is the same size as <code>SigmaForecast</code>. The first row contains the standard deviation of returns expected for assets held for one period for each realization of <code>Series</code>, the second row contains the standard deviation of returns expected for assets held for two periods, and so on. The last row contains the standard deviations of the cumulative returns obtained if an asset was held for the entire <code>NumPeriods</code> forecast horizon. If you specify <code>X</code> or <code>XF</code>, <code>SigmaTotal</code> = [].</p>
<code>MeanRMSE</code>	<p>Matrix of root mean square errors (RMSE) associated with <code>MeanForecast</code>. That is, <code>MeanRMSE</code> is the conditional standard deviation of the forecast errors (i.e., the standard error of the forecast) of the corresponding <code>MeanForecast</code> matrix. <code>MeanRMSE</code> is the same size as <code>MeanForecast</code> and <code>garchpred</code> organizes it in exactly the same manner, provided the conditional mean is modeled as a stationary/invertible ARMA process. If you specify <code>X</code> or <code>XF</code>, <code>MeanRMSE</code> = [].</p>

Note garchpred calls the function garchinfer to access the past history of innovations and conditional standard deviations inferred from Series. If you need the innovations and conditional standard deviations, call garchinfer directly.

Notes

EGARCH(P,Q) models represent the logarithm of the conditional variance as the output of a linear filter. As such, the minimum mean square error forecasts derived from EGARCH(P,Q) models are optimal for the logarithm of the conditional variance, but are generally downward-biased forecasts of the conditional variance process itself. Since the output arrays SigmaForecast, SigmaTotal, and MeanRMSE are based upon the conditional variance forecasts, these outputs generally underestimate their true expected values for conditional variances derived from EGARCH(P,Q) models. The important exception is the one-period-ahead forecast, which is unbiased in all cases.

Examples

See “Examples” on page 6-8 and “Forecasting” on page 10-4.

See Also

garchfit, garchinfer, garchma, garchset, garchsim

References

- [1] Baillie, R.T., and T. Bollerslev, “Prediction in Dynamic Models with Time-Dependent Conditional Variances,” *Journal of Econometrics*, Vol. 52, 1992, pp 91-113.
- [2] Bollerslev, T., “Generalized Autoregressive Conditional Heteroskedasticity,” *Journal of Econometrics*, Vol. 31, 1986, pp 307-327.
- [3] Bollerslev, T., “A Conditionally Heteroskedastic Time Series Model for Speculative Prices and Rates of Return,” *The Review Economics and Statistics*, Vol. 69, 1987, pp 542-547.
- [4] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.
- [5] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, 1995.
- [6] Engle, Robert, “Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation,” *Econometrica*, Vol. 50, 1982, pp 987-1007.

[7] Engle, R.F., D.M. Lilen, and R.P. Robins, "Estimating Time Varying Risk Premia in the Term Structure: The ARCH-M Model," *Econometrica*, Vol. 59, 1987, pp 391-407.

[8] Glosten, L.R., R. Jagannathan, and D.E. Runkle, "On the Relation Between Expected Value and the Volatility of the Nominal Excess Return on Stocks," *Journal of Finance*, Vol.48, 1993, pp 1779-1801.

[9] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, 1994.

[10] Nelson, D.B., "Conditional Heteroskedasticity in Asset Returns: A New Approach," *Econometrica*, Vol. 59, 1991, pp 347-370.

Purpose	Create or modify GARCH specification structure
Syntax	<pre>Spec = garchset('Parameter1', Value1, 'Parameter2', Value2, ...) Spec = garchset(OldSpec, 'Parameter1', Value1, ...) Spec = garchset garchset</pre>
Description	<p><code>Spec = garchset('Parameter1', Value1, 'Parameter2', Value2, ...)</code> creates a GARCH model specification structure <code>Spec</code> using the parameter-value pairs specified in the input argument list. Use <code>garchget</code> to retrieve the values of specification structure parameters.</p> <p><code>Spec = garchset(OldSpec, 'Parameter1', Value1, ...)</code> modifies an existing GARCH specification structure <code>OldSpec</code> by changing the named parameters to the specified values. <code>garchset</code> returns an error if the new parameter values would create an invalid model.</p> <p><code>Spec = garchset</code> creates a GARCH specification structure <code>Spec</code> for the GARCH Toolbox default model. For this model, the conditional mean equation is a simple constant plus additive noise, while the conditional variance equation of the additive noise is a GARCH(1,1) model. You can use this <code>Spec</code> as input to <code>garchfit</code>, but it is invalid as input to <code>garchinfer</code>, <code>garchpred</code>, and <code>garchsim</code>.</p> <p><code>garchset</code> (with no input arguments and no output arguments) displays all parameter names and the default values where appropriate.</p>
Input Arguments	<p><code>Parameter1</code>, <code>Parameter2</code>, ... String representing a valid parameter field of the output structure <code>Spec</code>. “Parameters” on page 11-62 lists the valid parameters and describes their allowed values. A parameter name needs to include only sufficient leading characters to uniquely identify the parameter. Parameter names are case insensitive.</p> <p><code>Value1</code>, <code>Value2</code>, ... Value assigned to the corresponding Parameter.</p> <p><code>OldSpec</code> Existing GARCH specification structure as generated by <code>garchset</code> or <code>garchfit</code>.</p>

Output Arguments

Spec GARCH specification structure containing the style, orders, and coefficients (if specified) of the conditional mean and variance specifications of a GARCH model. It also contains the parameters associated with the function `fmincon` in the Optimization Toolbox.

Parameters

A GARCH specification structure includes these parameters. Except as noted, `garchset` sets all parameters you do not specify to their respective defaults.

- “General Parameters” on page 11-62
- “Conditional Mean Parameters” on page 11-62
- “Conditional Variance Parameters” on page 11-63
- “Equality Constraint Parameters” on page 11-64
- “Optimization Parameters” on page 11-65

General Parameters

Parameter	Value	Description
Comment	String. Default is a model summary.	User-defined summary comment. An example of the default is 'Mean: ARMAX(0, 0, ?); Variance: GARCH(1, 1)'.
Distribution	'T' or 'Gaussian'. Default is 'Gaussian'.	Conditional distribution of innovations.
DoF	Scalar. Default = [].	Degrees of freedom parameter for t distributions (must be > 2).

Conditional Mean Parameters

If you specify coefficient vectors `AR` and `MA`, but not their corresponding model orders `R` and `M`, `garchset` infers the values of the model orders from the lengths of the coefficient vectors.

Parameter	Value	Description
R	Nonnegative integer scalar. Default is 0.	Autoregressive model order of an ARMA(R,M) model.

Parameter	Value	Description
M	Nonnegative integer scalar. Default is 0.	Moving-average model order of an ARMA(R,M) model.
C	Scalar coefficient. Default is [].	Conditional mean constant. If $C = \text{NaN}$, garchfit ignores C, effectively fixing $C = 0$, without requiring initial estimates for the remaining parameters.
AR	R-element vector. Default is [].	Conditional mean autoregressive coefficients that imply a stationary polynomial.
MA	M-element vector. Default is [].	Conditional mean moving-average coefficients that imply an invertible polynomial.
Regress	Vector of coefficients. Default is [].	Conditional mean regression coefficients.

Conditional Variance Parameters

If you specify coefficient vectors GARCH and ARCH, but not their corresponding model orders P and Q, garchset infers the values of the model orders from the lengths of the coefficient vectors.

Parameter	Value	Description
VarianceModel	'GARCH', 'EGARCH', 'GJR', or 'Constant'. Default is 'GARCH'.	Conditional variance model.
P	Nonnegative integer scalar. P must be 0 if Q is 0. Default is 0.	Model order of GARCH(P,Q), EGARCH(P,Q), and GJR(P,Q) models.
Q	Nonnegative integer scalar. Default is 0.	Model order of GARCH(P,Q), EGARCH(P,Q), and GJR(P,Q) models.

Parameter	Value	Description
K	Scalar coefficient. Default is [].	Conditional variance constant.
GARCH	P-element vector. Default is [].	Coefficients related to lagged conditional variances.
ARCH	Q-element vector. Default is [].	Coefficients related to lagged innovations (i.e., residuals).
Leverage	Q-element vector. Default is [].	Leverage coefficients for asymmetric EGARCH(P,Q) and GJR(P,Q) models.

Equality Constraint Parameters

These parameters are used only by `garchfit` during estimation. Use these parameters cautiously. The problem can experience difficulty converging if the fixed value is not well suited to the data at hand.

Parameter	Value	Description
FixDoF	Logical scalar. Default is [].	Equality constraint indicator for DoF parameter.
FixC	Logical scalar. Default is [].	Equality constraint indicator for C constant.
FixAR	R-element logical vector. Default is [].	Equality constraint indicator for AR coefficients.
FixMA	M-element logical vector. Default is [].	Equality constraint indicator for MA coefficients.
FixRegress	Logical vector. Default is [].	Equality constraint indicator for the REGRESS coefficients.
FixK	Logical scalar. Default is [].	Equality constraint indicator for the K constant.

Parameter	Value	Description
FixGARCH	P-element logical vector. Default is [].	Equality constraint indicator for the GARCH coefficients.
FixARCH	Q-element logical vector. Default is [].	Equality constraint indicator for the ARCH coefficients.
FixLeverage	Q-element logical vector. Default is [].	Equality constraint indicator for Leverage coefficients.

Optimization Parameters

garchfit uses these parameters in calling the Optimization Toolbox function `fmincon` during estimation.

Parameter	Value	Description
Display	'on' or 'off'. Default is 'on'.	Display iterative optimization information.
MaxFunEvals	Positive integer. Default = (100*number of estimated parameters).	Maximum number of objective function evaluations allowed.
MaxIter	Positive integer. Default is 400.	Maximum number of iterations allowed.
TolCon	Positive scalar. Default is 1e-007.	Termination tolerance on the constraint violation.
TolFun	Positive scalar. Default is 1e-006.	Termination tolerance on the objective function value.
TolX	Positive scalar. Default is 1e-006.	Termination tolerance on parameter estimates.

Examples

This example creates a GARCH(1,1) model, then changes it to a GARCH(1,2) model. In each case, it displays the relevant fields in the specification structure. Use `garchget` to retrieve the values of individual fields.

garchset

```
spec = garchset('P', 1, 'Q', 1) % Create a GARCH(P=1, Q=1) model.

spec =

    Comment: 'Mean: ARMAX(0, 0, ?); Variance: GARCH(1, 1)'
    Distribution: 'Gaussian'
    C: []
    VarianceModel: 'GARCH'
    P: 1
    Q: 1
    K: []
    GARCH: []
    ARCH: []

spec = garchset(spec, 'Q', 2) % Change it to a GARCH(P=1, Q=2)
% model.

spec =

    Comment: 'Mean: ARMAX(0, 0, ?); Variance: GARCH(1, 2)'
    Distribution: 'Gaussian'
    C: []
    VarianceModel: 'GARCH'
    P: 1
    Q: 2
    K: []
    GARCH: []
    ARCH: []
```

See Also

`garchfit`, `garchget`, `garchpred`, `garchsim`
`fmincon` (in the Optimization Toolbox)

Purpose Univariate GARCH process simulation

Syntax

```
[Innovations, Sigmas, Series] = garchsim(Spec)
[...] = garchsim(Spec, NumSamples, NumPaths)
[...] = garchsim(Spec, NumSamples, NumPaths, State)
[...] = garchsim(Spec, NumSamples, NumPaths, State, X)
[...] = garchsim(Spec, NumSamples, NumPaths, State, X, Tolerance)
[...] = garchsim(Spec, NumSamples, NumPaths, State, X, Tolerance,
    ... PreInnovations, PreSigmas, PreSeries)
```

Description

[Innovations, Sigmas, Series] = garchsim(Spec), given specifications for the conditional mean and variance of a univariate time series, simulates a sample path with 100 observations for the return series, innovations, and conditional standard deviation processes. The conditional mean can be of general ARMA form and the conditional variance of general GARCH, EGARCH, or GJR form. [...] = garchsim(Spec, NumSamples, NumPaths) simulates NumPaths sample paths. Each path is sampled at NumSamples observations.

[...] = garchsim(Spec, NumSamples, NumPaths, State) specifies the state of the standardized (zero mean, unit variance), independent, identically distributed random noise process.

[...] = garchsim(Spec, NumSamples, NumPaths, State, X) accepts a time-series regression matrix X of observed explanatory data. garchsim treats each column of X as an individual time series, and uses it as an explanatory variable in the regression component of the conditional mean.

[...] = garchsim(Spec, NumSamples, NumPaths, State, X, Tolerance) accepts a scalar transient response tolerance, such that $Tolerance > 0$ and ≤ 1 . garchsim estimates the number of observations needed for the magnitude of the impulse response, which begins at 1, to decay below the Tolerance value. The number of observations associated with the transient decay period is subject to a maximum of 10,000 to prevent out-of-memory conditions. Tolerance is ignored when you specify presample observations (PreInnovations, PreSigmas, and PreSeries).

Use Tolerance to manage the conflict between transient minimization and memory usage. Smaller Tolerance values generate output processes that more closely approximate true steady-state behavior, but require more memory for

the additional filtering required. Conversely, larger Tolerance values require less memory, but produce outputs in which transients tend to persist.

If you do not explicitly specify presample data (see below), the impulse response estimates are based on the magnitude of the largest eigenvalue of the autoregressive polynomial.

[...] = `garchsim`(Spec, NumSamples, NumPaths, State, X, Tolerance, ... PreInnovations, PreSigmas, PreSeries) uses presample observations, contained in the time-series matrices or column vectors PreInnovations, PreSigmas, and PreSeries, to simulate the outputs Innovations, Sigmas, and Series, respectively. When specified, these presample arrays are used to initiate the filtering process, and thus form the conditioning set upon which the simulated realizations are based.

If you specify the presample data as matrices, they *must* have NumPaths columns. `garchsim` uses the presample information from a given column to initiate the simulation of the corresponding column of the Innovations, Sigmas, and Series outputs. If you specify the presample data as column vectors, the vectors are applied to each column of the corresponding Innovations, Sigmas, and Series outputs.

If you provide no explicit presample data, the necessary presample observations are derived automatically (see “Automatic Minimization of Transient Effects” on page 4-7).

PreInnovations and PreSigmas are usually companion inputs. Although both are optional, when specified, they are typically entered together. A notable exception would be a GARCH(0,Q) (i.e., an ARCH(Q)) model in which the conditional variance equation does not require lagged conditional variance forecasts. Similarly, PreSeries is only necessary when you want to simulate the output return Series, and when the conditional mean equation has an autoregressive component.

If the conditional mean or the conditional variance equation (“Conditional Mean and Variance Models” on page 2-6) is not recursive in any way, then certain presample information is unnecessary to jump-start the models. However, specifying redundant presample information is *not* an error, and `garchsim` ignores any presample observations you specify for models that require no such information.

Input Arguments

Spec	GARCH specification structure for the conditional mean and variance models. You create Spec by calling the function garchset or the estimation function garchfit. The conditional mean can be of general ARMAX form and the conditional variance of general GARCH form.
NumSamples	(optional) Positive integer indicating the number of observations garchsim generates for each path of the Innovations, Sigmas, and Series outputs. If NumSamples = [] or is not specified, the default is 100.
NumPaths	(optional) Positive integer indicating the number of sample paths (realizations) garchsim generates for the Innovations, Sigmas, and Series outputs. If NumPaths = [] or is not specified, the default is 1; i.e. Innovations, Sigmas and Series are column vectors.
PreInnovations	Time-series matrix or column vector of presample innovations on which the recursive mean and variance models are conditioned. This array can have any number of rows, provided it contains sufficient observations to initialize the mean and variance equations. That is, if M and Q are the number of lagged innovations required by the conditional mean and variance equations, respectively, then PreInnovations must have at least $\max(M, Q)$ rows. If the number of rows exceeds $\max(M, Q)$, then only the last (i.e., most recent) $\max(M, Q)$ rows are used. If PreInnovations is a matrix, then it must have NumPaths columns.

PreSigmas	<p>Time-series matrix or column vector of positive presample conditional standard deviations on which the recursive variance model is conditioned. This array can have any number of rows, provided it contains sufficient observations to initialize the conditional variance equation. That is, if P and Q are the number of lagged conditional standard deviations and lagged innovations required by the conditional variance equation, respectively, then PreSigmas must have at least P rows for GARCH and GJR models, and at least $\max(P, Q)$ rows for EGARCH models.</p> <p>If the number of rows exceeds the requirement, then only the last (i.e., most recent) rows are used. If PreSigmas is a matrix, then it must have NumPaths columns.</p>
PreSeries	<p>Time-series matrix or column vector of presample observations of the return series of interest on which the recursive mean model is conditioned. This array can have any number of rows, provided it contains sufficient observations to initialize the conditional mean equation. Thus, if R is the number of lagged observations of the return series required by the conditional mean equation, then PreSeries must have at least R rows. If the number of rows exceeds R, then only the last (i.e., most recent) R rows are used. If PreSeries is a matrix, then it must have NumPaths columns.</p>

State

State of the standardized (mean zero, unit variance), independent, identically distributed (i.i.d.) noise process that drives the output Innovations process (see below). State can be a scalar or a matrix.

When State is a scalar J , it is passed directly to the primary random number generators `rand` and `randn`, and resets each generator to its J th state.

When State is a matrix, it represents a user-specified time-series matrix of standardized, i.i.d. disturbances that drive the output Innovations time-series process. As a matrix, State must have exactly `NumPaths` columns and at least `NumSamples` rows in which the first row contains the oldest observation and the last row the most recent. Additional presample observations required to minimize transients, if any, are generated automatically based on the distribution found in the input specification structure `Spec` and prepended to the input State time-series matrix. If State has more observations (rows) than necessary, then only the most recent observations are used.

If State is empty or missing, `garchsim` uses the current states of the random number generators. You can set and query these states directly by calling `rand` and `randn`. See their reference pages for details.

Tolerance	Scalar transient response tolerance, such that $0 < \text{Tolerance} \leq 1$. This tolerance parameter is ignored if presample conditioning information is specified (see <code>PreInnovations</code> , <code>PreSigmas</code> , and <code>PreSeries</code>). If empty or missing, the default is 0.01 (i.e., 1%).
X	Time-series regression matrix of observed explanatory data. Typically, X is a matrix of asset returns (e.g., the return series of an equity index), and represents the past history of the explanatory data. Each column of X is an individual time series used as an explanatory variable in the regression component of the conditional mean. In each column, the first row contains the oldest observation and the last row the most recent. If <code>X = []</code> or is not specified, the conditional mean has no regression component. If specified, then at least the most recent <code>NumSamples</code> observations of each return series must be valid (i.e., non-NaN). When the number of valid observations in each series exceeds <code>NumSamples</code> , <code>garchsim</code> uses only the most recent <code>NumSamples</code> observations of X.

Output Arguments

Innovations	<code>NumSamples</code> by <code>NumPaths</code> matrix of innovations, representing a mean zero, discrete-time stochastic process. The <code>Innovations</code> time series follows the conditional variance specification defined in <code>Spec</code> . Rows are sequential observations, columns are realizations.
Sigmas	<code>NumSamples</code> by <code>NumPaths</code> matrix of conditional standard deviations of the corresponding <code>Innovations</code> matrix. <code>Innovations</code> and <code>Sigmas</code> are the same size. Rows are sequential observations. Columns are realizations.
Series	<code>NumSamples</code> by <code>NumPaths</code> matrix of the return series of interest. <code>Series</code> is the dependent stochastic process and follows the conditional mean specification of general ARMAX form defined in <code>Spec</code> . Rows are sequential observations. Columns are realizations.

Examples

Example 1. State as an Integer Scalar

The input `State` can be specified as an integer scalar, or as a time-series matrix. As an integer scalar, it represents the random number generator state `J` and corresponds exactly to the syntax `rand('state', J)` and `randn('state', J)`.

This example creates a simple GARCH specification structure for simulation, and specifies a scalar random number generator state, `J = 12345`.

```
spec = garchset('C', 0.0001, 'K', 0.00005, 'GARCH', 0.8, 'ARCH',
... 0.15);
J = 12345;
```

In this situation, the following two calls to `garchsim` produce the same simulated output processes (i.e., $e_1 = e_2$, $s_1 = s_2$, and $y_1 = y_2$).

```
rand('state', J); randn('state', J);

[e1, s1, y1] = garchsim(spec, 100, 1);
[e2, s2, y2] = garchsim(spec, 100, 1, J);
```

Example 2. State as a Standardized Noise Matrix

When `State` is a matrix, it represents a user-specified time-series matrix of standardized (mean zero, unit variance), i.i.d. disturbances $\{z(t)\}$ that drive the output time-series processes $\{e(t)\}$, $\{s(t)\}$, and $\{y(t)\}$. For example, if you run `garchsim` once, then standardize the simulated residuals and pass them into `garchsim` as the i.i.d. noise input for a second run, the standardized residuals from both runs will be identical. This verifies that the specified input noise matrix is indeed the "in-sample" i.i.d. noise process $\{z(t)\}$ for both.

```
spec = garchset('C', 0.0001, 'K', 0.00005, 'GARCH', 0.8, 'ARCH',
... 0.1);

[e1, s1, y1] = garchsim(spec, 100, 1);
z1 = e1./s1; % Standardize residuals
[e2, s2, y2] = garchsim(spec, 100, 1, z1);
z2 = e2./s2; % Standardize residuals
```

In this case, $z_1 = z_2$.

However, although the "in-sample" standardized noise processes are identical, in the absence of presample data the simulated output processes $\{e(t)\}$, $\{s(t)\}$, and $\{y(t)\}$ will differ. This is because, in the absence of presample data, any additional standardized noise observations necessary to minimize transients must be simulated from the distribution, 'Gaussian' or 'T', found in the specification structure.

Now specify all required presample data and repeat the experiment.

```
[e3, s3, y3] = garchsim(spec, 100, 1, [], [], [], 0.02, 0.06);
z3 = e3./s3; % Standardize residuals
[e4, s4, y4] = garchsim(spec, 100, 1, z3, [], [], 0.02, 0.06);
z4 = e4./s4; % Standardize residuals
```

In this case, $e3 = e4$, $s3 = s4$, $y3 = y4$ as well as $z3 = z4$.

More Examples

For more examples of simulation, see “Simulating Sample Paths” on page 4-2, “Fitting a Model to a Simulated Return Series” on page 7-3, and “Monte Carlo Simulation” on page 10-6.

For more comprehensive examples that make use of this functionality, see the GARCH Toolbox demos “Market Risk Using GARCH, Bootstrapping and Filtered Historical Simulation,” and “Market Risk Using GARCH, Extreme Value Theory, and Copulas.” These demos are available only within MATLAB.

See Also

garchfit, garchget, garchpred, garchset
rand, randn (MATLAB)

References

- [1] Bollerslev, T., “A Conditionally Heteroskedastic Time Series Model for Speculative Prices and Rates of Return,” *Review of Economics and Statistics*, Vol. 69, 1987, pp 542-547.
- [2] Bollerslev, T., “Generalized Autoregressive Conditional Heteroskedasticity,” *Journal of Econometrics*, Vol. 31, 1986, pp 307-327.
- [3] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.
- [4] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, 1995.

- [5] Engle, Robert, "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica*, Vol. 50, 1982, pp 987-1007.
- [6] Engle, R.F., D.M. Lilien, and R.P. Robins, "Estimating Time Varying Risk Premia in the Term Structure: The ARCH-M Model," *Econometrica*, Vol. 59, 1987, pp 391-407.
- [7] Glosten, L.R., R. Jagannathan, and D.E. Runkle, "On the Relation Between Expected Value and the Volatility of the Nominal Excess Return on Stocks," *The Journal of Finance*, Vol.48, 1993, pp 1779-1801.
- [8] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, 1994.
- [9] Nelson, D.B., "Conditional Heteroskedasticity in Asset Returns: A New Approach," *Econometrica*, Vol. 59, 1991, pp 347-370.

lagmatrix

Purpose Create lagged time-series matrix

Syntax `XLAG = lagmatrix(X, Lags)`

Description `XLAG = lagmatrix(X, Lags)` creates a lagged (i.e., shifted) version of a time-series matrix. The `lagmatrix` function is useful for creating a regression matrix of explanatory variables for fitting the conditional mean of a return series.

Input Arguments

X Time-series of explanatory data. `X` can be a column vector or a matrix. As a column vector, `X` represents a univariate time series whose first element contains the oldest observation and whose last element contains the most recent observation. As a matrix, `X` represents a multivariate time series whose rows correspond to time indices in which the first row contains the oldest observations and the last row contains the most recent observations. `lagmatrix` assumes that observations across any given row occur at the same time. Each column is an individual time series.

Lags Vector of integer lags. `lagmatrix` applies the first lag to every series in `X`, then applies the second lag to every series in `X`, and so forth. To include a time series as is, include a 0 lag. Positive lags correspond to delays, and shift a series back in time. Negative lags correspond to leads, and shift a series forward in time.

Output Arguments

XLAG Lagged transform of the time series `X`. To create `XLAG`, `lagmatrix` shifts each time series in `X` by the first lag, then shifts each time series in `X` by the second lag, and so forth. Since `XLAG` represents an explanatory regression matrix, each column is an individual time series. `XLAG` has the same number of rows as there are observations in `X`, but its column dimension is equal to the product of the number of columns in `X` and the length of `Lags`. `lagmatrix` uses a NaN (Not-a-Number) to indicate an undefined observation.

Examples

Example 1. The following example creates a bivariate time-series matrix `X` with five observations each, then creates a lagged matrix `XLAG` composed of `X` and the first two lags of `X`. The result, `XLAG`, is a 5-by-6 matrix.

```
X = [1 -1; 2 -2 ;3 -3 ;4 -4 ;5 -5] % Create a simple bivariate  
% series.
```

```
X =  
    1    -1  
    2    -2  
    3    -3  
    4    -4  
    5    -5
```

```
XLAG = lagmatrix(X,[0 1 2]) % Create the lagged matrix.
```

```
XLAG =  
    1    -1   NaN   NaN   NaN   NaN  
    2    -2     1    -1   NaN   NaN  
    3    -3     2    -2     1    -1  
    4    -4     3    -3     2    -2  
    5    -5     4    -4     3    -3
```

Example 2. See “Fitting a Regression Model to the Same Return Series” on page 7-5 for another example.

See Also

filter, isnan, and nan (all in MATLAB)

lbqtest

Purpose Ljung-Box Q-statistic lack-of-fit hypothesis test

Syntax [H, pValue, Qstat, CriticalValue] = lbqtest(Series, Lags, Alpha, DoF)

Description [H, pValue, Qstat, CriticalValue] = lbqtest(Series, Lags, Alpha, DoF) performs the Ljung-Box lack-of-fit hypothesis test for model misspecification, which is based on the Q-statistic

$$Q = N(N+2) \sum_{k=1}^L \frac{r_k^2}{(N-k)}$$

where N = sample size, L = number of autocorrelation lags included in the statistic, and r_k^2 is the squared sample autocorrelation at lag k . Once you fit a univariate model to an observed time series, you can use the Q-statistic as a lack-of-fit test for a departure from randomness. Under the null hypothesis that the model fit is adequate, the test statistic is asymptotically chi-square distributed.

Input Arguments

Series Vector of observations of a univariate time series for which lbqtest computes the sample Q-statistic. The last row of Series contains the most recent observation of the stochastic sequence. Typically, Series is either the sample residuals derived from fitting a model to an observed time series, or the standardized residuals obtained by dividing the sample residuals by the conditional standard deviations.

Lags Vector of positive integers indicating the lags of the sample autocorrelation function included in the Q-statistic. If specified, each lag must be less than the length of Series. If Lags = [] or is not specified, the default is Lags = min([20, length(Series)-1]).

Alpha Significance levels. Alpha can be a scalar applied to all lags, or a vector the same length as Lags. If Alpha = [] or is not specified, the default is 0.05. For all elements, α , of Alpha, $0 < \alpha < 1$.

DoF Degrees of freedom. DoF can be a scalar applied to all lags, or a vector the same length as Lags. If specified, all elements of DoF must be positive integers less than the corresponding element of Lags. If DoF = [] or is not specified, the elements of Lags serve as the default degrees of freedom for the chi-square distribution.

Output Arguments

H Boolean decision vector. 0 indicates acceptance of the null hypothesis that the model fit is adequate (no serial correlation at the corresponding element of Lags). 1 indicates rejection of the null hypothesis. H is the same size as Lags.

pValue Vector of P-values (significance levels) at which lbqtest rejects the null hypothesis of no serial correlation at each lag in Lags.

Qstat Vector of Q-statistics for each lag in Lags.

CriticalValue Vector of critical values of the chi-square distribution for comparison with the corresponding element of Qstat.

Examples

Example 1. Create a vector of 100 Gaussian random numbers, then compute the Q-statistic for autocorrelation lags 20 and 25 at the 10 percent significance level.

```
randn('state', 100)           % Start from a known state.
Series = randn(100, 1);      % 100 Gaussian deviates ~ N(0, 1)
[H, P, Qstat, CV] = lbqtest(Series, [20 25]', 0.10);
[H, P, Qstat, CV]
```

ans =

0	0.9615	10.3416	28.4120
0	0.9857	12.1015	34.3816

Example 2. See “Prestimation Analysis” on page 2-16 for another example.

See Also

archtest, autocorr

References

- [1] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.
- [2] Gouriéroux, C., *ARCH Models and Financial Applications*, Springer-Verlag, 1997.

Purpose	Likelihood ratio hypothesis test				
Syntax	<code>[H, pValue, Ratio, CriticalValue] = lratiotest(BaseLLF, NullLLF, ... DoF, Alpha)</code>				
Description	<p><code>[H, pValue, Ratio, CriticalValue] = lratiotest(BaseLLF, NullLLF, DoF, Alpha)</code> performs the likelihood ratio hypothesis test. <code>lratiotest</code> uses as input the optimized log-likelihood objective function (LLF) value associated with an unrestricted maximum likelihood parameter estimate, and the LLF values associated with restricted parameter estimates.</p> <p>The unrestricted LLF is the baseline case used to fit conditional mean and variance specifications to an observed univariate return series. The restricted models determine the null hypotheses of each test, and the number of restrictions they impose determines the degrees of freedom of the resulting chi-square distribution.</p> <p><code>BaseLLF</code> is usually the LLF of a larger estimated model and serves as the alternative hypothesis. Elements of <code>NullLLF</code> are then the LLFs associated with smaller, restricted specifications. <code>BaseLLF</code> should exceed the values in <code>NullLLF</code>, and the asymptotic distribution of the test statistic is chi-square distributed with degrees of freedom equal to the number of restrictions.</p>				
Input Arguments	<table border="0"> <tr> <td style="vertical-align: top;"><code>BaseLLF</code></td> <td>Scalar value of the optimized log-likelihood objective function of the baseline, unrestricted estimate. <code>lratiotest</code> assumes <code>BaseLLF</code> is the output of the estimation function <code>garchfit</code> or the inference function <code>garchinfer</code>.</td> </tr> <tr> <td style="vertical-align: top;"><code>NullLLF</code></td> <td>Vector of optimized log-likelihood objective function values of the restricted estimates. <code>lratiotest</code> assumes you obtained the <code>NullLLF</code> values using <code>garchfit</code> or <code>garchinfer</code>.</td> </tr> </table>	<code>BaseLLF</code>	Scalar value of the optimized log-likelihood objective function of the baseline, unrestricted estimate. <code>lratiotest</code> assumes <code>BaseLLF</code> is the output of the estimation function <code>garchfit</code> or the inference function <code>garchinfer</code> .	<code>NullLLF</code>	Vector of optimized log-likelihood objective function values of the restricted estimates. <code>lratiotest</code> assumes you obtained the <code>NullLLF</code> values using <code>garchfit</code> or <code>garchinfer</code> .
<code>BaseLLF</code>	Scalar value of the optimized log-likelihood objective function of the baseline, unrestricted estimate. <code>lratiotest</code> assumes <code>BaseLLF</code> is the output of the estimation function <code>garchfit</code> or the inference function <code>garchinfer</code> .				
<code>NullLLF</code>	Vector of optimized log-likelihood objective function values of the restricted estimates. <code>lratiotest</code> assumes you obtained the <code>NullLLF</code> values using <code>garchfit</code> or <code>garchinfer</code> .				

lratiotest

DoF	Degrees of freedom (i.e., the number of parameter restrictions) associated with each value in NullLLF. DoF can be a scalar applied to all values in NullLLF, or a vector the same length as NullLLF. All elements of DoF must be positive integers.
Alpha	Significance levels of the hypothesis test. Alpha can be a scalar applied to all values in NullLLF, or a vector the same length as NullLLF. If Alpha = [] or is not specified, the default is 0.05. For all elements, α , of Alpha, $0 < \alpha < 1$.
H	Vector of Boolean decisions the same size as NullLLF. A 0 indicates acceptance of the restricted model under the null hypothesis. 1 indicates rejection of the restricted, null hypothesis model relative to the unrestricted alternative associated with BaseLLF.
pValue	Vector of P-values (significance levels) at which lratiotest rejects the null hypothesis of each restricted model. pValue is the same size as NullLLF.
Ratio	Vector of likelihood ratio test statistics the same size as NullLLF. The test statistic is $\text{Ratio} = 2(\text{BaseLLF} - \text{NullLLF})$
CriticalValue	Vector of critical values of the chi-square distribution. CriticalValue is the same size as NullLLF.

Output Arguments

Examples

See “Likelihood Ratio Tests” on page 9-2 and “Equality Constraints and Parameter Significance” on page 9-7.

See Also

garchfit, garchinfer

References

[1] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, 1994.

Purpose Plot or return computed sample partial autocorrelation function

Syntax [PartialACF, Lags, Bounds] = `parcorr`(Series, nLags, R, nSTDs)

Description `parcorr`(Series, nLags, R, nSTDs) computes and plots the sample partial autocorrelation function (partial ACF) of a univariate, stochastic time series. `parcorr` computes the partial ACF by fitting successive autoregressive models of orders 1, 2, ... by ordinary least squares, retaining the last coefficient of each regression. To plot the partial ACF sequence without the confidence bounds, set `nSTDs = 0`.

[PartialACF, Lags, Bounds] = `parcorr`(Series, nLags, R, nSTDs) computes and returns the partial ACF sequence.

Input Arguments

Series Vector of observations of a univariate time series for which `parcorr` returns or plots the sample partial autocorrelation function (partial ACF). The last element of `Series` contains the most recent observation of the stochastic sequence.

nLags Positive scalar integer indicating the number of lags of the partial ACF to compute. If `nLags = []` or is not specified, `parcorr` computes the partial ACF sequence at lags 0, 1, 2, ..., T , where $T = \min([20, \text{length}(\text{Series}) - 1])$.

- R** Nonnegative integer scalar indicating the number of lags beyond which `parcorr` assumes the theoretical partial ACF is zero. Assuming that `Series` is an AR(R) process, the estimated partial ACF coefficients at lags greater than `R` are approximately zero-mean, independently distributed Gaussian variates. In this case, the standard error of the estimated partial ACF coefficients of a fitted `Series` with N observations is approximately $1/\sqrt{N}$ for lags greater than `R`. If `R = []` or is not specified, the default is 0. The value of `R` must be less than `nLags`.
- nSTDs** Positive scalar indicating the number of standard deviations of the sample partial ACF estimation error to display, assuming that `Series` is an AR(R) process. If the `R`th regression coefficient (i.e., the last ordinary least squares (OLS) regression coefficient of `Series` regressed on a constant and `R` of its lags) includes N observations, specifying `nSTDs` results in confidence bounds at $\pm(nSTDs/\sqrt{N})$. If `nSTDs = []` or is not specified, the default is 2 (i.e., approximate 95 percent confidence interval).

Output Arguments

- PartialACF** Sample partial ACF of `Series`. `PartialACF` is a vector of length `nLags + 1` corresponding to lags 0, 1, 2, ..., `nLags`. The first element of `PartialACF` is unity, i.e., `PartialACF(1) = 1 = OLS regression coefficient of Series regressed upon itself`. `parcorr` includes this element as a reference.
- Lags** Vector of lags, of length `nLags + 1`. The elements correspond to the elements of `PartialACF`.
- Bounds** Two-element vector indicating the approximate upper and lower confidence bounds, assuming that `Series` is an AR(R) process. Note that `Bounds` is approximate for lags greater than `R` only.

Examples

Example 1. Create a stationary AR(2) process from a sequence of 1000 Gaussian deviates, and then visually assess whether the partial ACF is zero for lags greater than 2.

```
randn('state', 0) % Start from a known state.
```

```
x = randn(1000, 1);           % 1000 Gaussian deviates ~ N(0, 1).
y = filter(1, [1 -0.6 0.08], x); % Create a stationary AR(2)
                                % process.
[PartialACF, Lags, Bounds] = parcorr(y, [], 2); % Compute the
                                                % partial ACF with 95 percent
                                                % confidence.
```

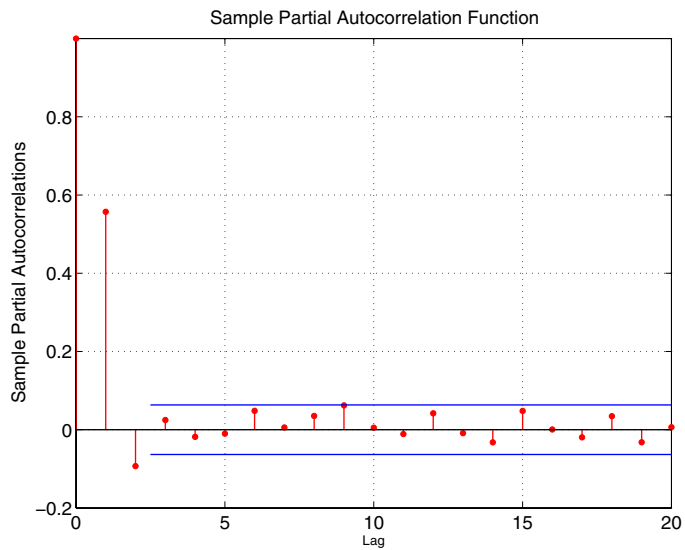
```
[Lags, PartialACF]
```

```
ans =
      0      1.0000
  1.0000      0.5570
  2.0000     -0.0931
  3.0000      0.0249
  4.0000     -0.0180
  5.0000     -0.0099
  6.0000      0.0483
  7.0000      0.0058
  8.0000      0.0354
  9.0000      0.0623
 10.0000      0.0052
 11.0000     -0.0109
 12.0000      0.0421
 13.0000     -0.0086
 14.0000     -0.0324
 15.0000      0.0482
 16.0000      0.0008
 17.0000     -0.0192
 18.0000      0.0348
 19.0000     -0.0320
 20.0000      0.0062
```

```
Bounds
```

```
Bounds =
      0.0633
     -0.0633
```

```
parcorr(y, [], 2)           % Use the same example, but plot
                             % the partial ACF sequence with
                             % confidence bounds.
```



Example 2. See “Preestimation Analysis” on page 2-16 for another example.

See Also

autocorr, crosscorr
filter (MATLAB)

References

- [1] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, 1994.
- [2] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, 1994.

Purpose	Phillips-Perron unit root test based on AR(1) model with drift
Syntax	<pre>[H, PValue, TestStat, CriticalValue] = ppARDTest(Y) [H, PValue, TestStat, CriticalValue] = ppARDTest(Y, Lags, Alpha, TestType)</pre>
Description	<p>[H, PValue, TestStat, CriticalValue] = ppARDTest(Y) performs a Phillips-Perron univariate unit root test under the assumption that the true underlying process is a zero drift unit root process. As an alternative, a first-order autoregressive (AR(1)) model plus additive constant is estimated by OLS regression.</p> <p>Specifically, if y_t and ε_t are the time series of observed data and model residuals, respectively, then under the null hypothesis the true underlying process is assumed to be</p> $y_t = y_{t-1} + \varepsilon_t$ <p>As an alternative, the estimated OLS regression model is</p> $y_t = C + \phi y_{t-1} + \varepsilon_t$ <p>for some constant C and AR(1) coefficient $\phi < 1$.</p>
Input Arguments	<p>Y Time-series vector of observed data tested for a unit root. The last element contains the most recent observation. Missing values are indicated by NaNs and are removed, thereby reducing the sample size.</p> <p>Lags (Optional) Scalar or vector of nonnegative integers indicating the number of autocovariance lags included in the Newey-West estimation of the asymptotic variance of the sample mean of the residuals. Lags serves as a correction for serial correlation of residuals. If empty or missing, the default is 0 (no correction for serial correlation).</p>

ppARDTest

Alpha	(Optional) Scalar or vector of significance levels of the test. All elements of the input argument must be $0.0001 \leq \text{Alpha} \leq 0.999$.
TestType	(Optional) Character string indicating the type of unit root test. Possible choices are t and AR, indicating an OLS t test of the AR coefficient and a test of the unstudentized AR coefficient, respectively. If empty or missing, the default is a t test. A case-insensitive check is made of TestType.
H	Logical decision vector. Elements of H = 0 indicate acceptance of the null hypothesis; elements of H = 1 indicate rejection of the null hypothesis. Each element of H is associated with a particular lag of Lags and significance level of Alpha.
PValue	Vector of P values (significance levels) associated with the test decision vector H. Each element of PValue represents the probability of observing a test statistic at least as extreme as that calculated from the OLS regression model when the null hypothesis is true. P values are obtained by interpolation into the appropriate table of critical values, and a NaN is returned if the test statistic lies beyond the range of the table.
TestStat	Vector of test statistics associated with the decision vector H.
CriticalValue	Vector of critical values associated with the decision vector H.

Output Arguments

Notes

Both Lags and Alpha may be vectors. If both are specified as vectors, however, they must be the same length (i.e., have the same number of elements). If one is specified as a scalar and the other as a vector, a scalar expansion is performed to enforce vectors of identical length. If Lags is a scalar or an empty matrix, by default, all outputs are column vectors.

All vector outputs are the same length as vector inputs Alpha and/or Lags. By default all vector outputs are column vectors. If Lags is a row vector, however, all vector outputs are row vectors.

This univariate unit root test is a conventional *lower-tailed* test, and the acceptance or rejection of the test is based on a comparison of the test statistic with the critical value. If the test statistic is *less than* the critical value, then the null hypothesis is rejected.

See Also

dfARDTest, dfARTest, dfTSTest, ppARTest, ppTSTest

References

- [1] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, Princeton, NJ, 1994.
- [2] Greene, W.H., *Econometric Analysis*, Prentice Hall, Fifth edition, Upper Saddle River, NJ, 2003.
- [3] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, New York, 1995.
- [4] Campbell, J.Y., A.W. Lo, and A.C. MacKinlay, *The Econometrics of Financial Markets*, Princeton University Press, Princeton, NJ, 1997.

ppARTest

Purpose

Phillips-Perron unit root test based on zero drift AR(1) model

Syntax

```
[H, PValue, TestStat, CriticalValue] = ppARTest(Y)
[H, PValue, TestStat, CriticalValue] = ppARTest(Y, Lags, Alpha,
    TestType)
```

Description

[H, PValue, TestStat, CriticalValue] = ppARTest(Y) performs a Phillips-Perron univariate unit root test under the assumption that the true underlying process is a zero drift unit root process. As an alternative, a zero drift first-order autoregressive (AR(1)) model is estimated by OLS regression.

Specifically, if y_t and ε_t are the time series of observed data and model residuals, respectively, then under the null hypothesis the true underlying process is assumed to be

$$y_t = y_{t-1} + \varepsilon_t$$

As an alternative, the estimated OLS regression model is

$$y_t = \phi y_{t-1} + \varepsilon_t$$

for some AR(1) coefficient $\phi < 1$.

Input Arguments

- | | |
|------|---|
| Y | Time-series vector of observed data tested for a unit root. The last element contains the most recent observation. Missing values are indicated by NaNs and are removed, thereby reducing the sample size. |
| Lags | (Optional) Scalar or vector of nonnegative integers indicating the number of autocovariance lags included in the Newey-West estimation of the asymptotic variance of the sample mean of the residuals. Lags serves as a correction for serial correlation of residuals. If empty or missing, the default is 0 (no correction for serial correlation). |

Output Arguments

Alpha	(Optional) Scalar or vector of significance levels of the test. All elements of the input argument must be $0.0001 \leq \text{Alpha} \leq 0.999$.
TestType	(Optional) Character string indicating the type of unit root test. Possible choices are t and AR, indicating an OLS t test of the AR coefficient and a test of the unstudentized AR coefficient, respectively. If empty or missing, the default is a t test. A case-insensitive check is made of TestType.
H	Logical decision vector. Elements of $H = 0$ indicate acceptance of the null hypothesis; elements of $H = 1$ indicate rejection of the null hypothesis. Each element of H is associated with a particular lag of Lags and significance level of Alpha.
PValue	Vector of P values (significance levels) associated with the test decision vector H. Each element of PValue represents the probability of observing a test statistic at least as extreme as that calculated from the OLS regression model when the null hypothesis is true. P values are obtained by interpolation into the appropriate table of critical values, and a NaN is returned if the test statistic lies beyond the range of the table.
TestStat	Vector of test statistics associated with the decision vector H.
CriticalValue	Vector of critical values associated with the decision vector H.

Notes

Both Lags and Alpha may be vectors. If both are specified as vectors, however, they must be the same length (i.e., have the same number of elements). If one is specified as a scalar and the other as a vector, a scalar expansion is performed to enforce vectors of identical length. If Lags is a scalar or an empty matrix, by default, all outputs are column vectors.

All vector outputs are the same length as vector inputs Alpha and/or Lags. By default, all vector outputs are column vectors. If Lags is a row vector, however, all vector outputs are row vectors.

This univariate unit root test is a conventional *lower-tailed* test, and the acceptance or rejection of the test is based on a comparison of the test statistic with the critical value. If the test statistic is *less than* the critical value, reject the null hypothesis.

See Also

dfARDTest, dfARTest, dfTSTest, ppARDTest, ppTSTest

References

- [1] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, Princeton, NJ, 1994.
- [2] Greene, W.H., *Econometric Analysis*, Prentice Hall, Fifth edition, Upper Saddle River, NJ, 2003.
- [3] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, New York, 1995.
- [4] Campbell, J.Y., A.W. Lo, and A.C. MacKinlay, *The Econometrics of Financial Markets*, Princeton University Press, Princeton, NJ, 1997.

Purpose	Phillips-Perron unit root test based on trend stationary AR(1) model
Syntax	<pre>[H, PValue, TestStat, CriticalValue] = ppTSTest(Y) [H, PValue, TestStat, CriticalValue] = ppTSTest(Y, Lags, Alpha, TestType)</pre>
Description	<p>[H, PValue, TestStat, CriticalValue] = ppTSTest(Y) performs a Phillips-Perron univariate unit root test under the assumption that the true underlying process is a unit root process with drift. As an alternative, a trend stationary first-order autoregressive (AR(1)) model plus additive constant is estimated by OLS regression.</p> <p>Specifically, if y_t and ε_t are the time series of observed data and model residuals, respectively, then under the null hypothesis the true underlying process is assumed to be</p> $y_t = C + y_{t-1} + \varepsilon_t$ <p>for an arbitrary constant C. As an alternative, the estimated OLS regression model is</p> $y_t = C + \phi y_{t-1} + \delta t + \varepsilon_t$ <p>for some constant C, AR(1) coefficient $\phi < 1$, and trend stationary coefficient δ.</p>
Input Arguments	<p>Y Time-series vector of observed data tested for a unit root. The last element contains the most recent observation. Missing values are indicated by NaNs and are removed, thereby reducing the sample size.</p> <p>Lags (Optional) Scalar or vector of nonnegative integers indicating the number of autocovariance lags included in the Newey-West estimation of the asymptotic variance of the sample mean of the residuals. Lags serves as a correction for serial correlation of residuals. If empty or missing, the default is 0 (no correction for serial correlation).</p>

ppTSTest

Alpha	(Optional) Scalar or vector of significance levels of the test. All elements of the input argument must be $0.0001 \leq \text{Alpha} \leq 0.999$.
TestType	(Optional) Character string indicating the type of unit root test. Possible choices are t and AR, indicating an OLS t test of the AR coefficient and a test of the unstudentized AR coefficient, respectively. If empty or missing, the default is a t test. A case-insensitive check is made of TestType.
H	Logical decision vector. Elements of H = 0 indicate acceptance of the null hypothesis; elements of H = 1 indicate rejection of the null hypothesis. Each element of H is associated with a particular lag of Lags and significance level of Alpha.
PValue	Vector of P values (significance levels) associated with the test decision vector H. Each element of PValue represents the probability of observing a test statistic at least as extreme as that calculated from the OLS regression model when the null hypothesis is true. P values are obtained by interpolation into the appropriate table of critical values, and a NaN is returned if the test statistic lies beyond the range of the table.
TestStat	Vector of test statistics associated with the decision vector H.
CriticalValue	Vector of critical values associated with the decision vector H.

Output Arguments

Notes

Both Lags and Alpha may be vectors. If both are specified as vectors, however, they must be the same length (i.e., have the same number of elements). If one is specified as a scalar and the other as a vector, a scalar expansion is performed to enforce identical length vectors. If Lags is a scalar or an empty matrix, by default, all outputs are column vectors.

All vector outputs are the same length as vector inputs Alpha and/or Lags. By default all vector outputs are column vectors. If Lags is a row vector, however, all vector outputs are row vectors.

This univariate unit root test is a conventional *lower-tailed* test, and the acceptance or rejection of the test is based on a comparison of the test statistic with the critical value. If the test statistic is *less than* the critical value, reject the null hypothesis.

See Also

dfARDTest, dfARTest, dfTSTest, ppARDTest, ppARTest

References

- [1] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, Princeton, NJ, 1994.
- [2] Greene, W.H., *Econometric Analysis*, Prentice Hall, Fifth edition, Upper Saddle River, NJ, 2003.
- [3] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, New York, 1995.
- [4] Campbell, J.Y., A.W. Lo, and A.C. MacKinlay, *The Econometrics of Financial Markets*, Princeton University Press, Princeton, NJ, 1997.

price2ret

Purpose	Convert price series to return series						
Syntax	<code>[RetSeries, RetIntervals] = price2ret(TickSeries, TickTimes, Method)</code>						
Description	<code>[RetSeries, RetIntervals] = price2ret(TickSeries, TickTimes, Method)</code> computes asset returns for NUMOBS price observations of NUMASSETS assets.						
Input Arguments	<table><tr><td>TickSeries</td><td>Time series of price data. TickSeries can be a column vector or a matrix:<ul style="list-style-type: none">• As a vector, TickSeries represents a univariate price series. The length of the vector is the number of observations (NUMOBS). The first element contains the oldest observation, and the last element the most recent.• As a matrix, TickSeries represents a NUMOBS-by-number of assets (NUMASSETS) matrix of asset prices. Rows correspond to time indices. The first row contains the oldest observations and the last row the most recent. price2ret assumes that the observations across a given row occur at the same time for all columns, and each column is a price series of an individual asset.</td></tr><tr><td>TickTimes</td><td>A NUMOBS element vector of monotonically increasing observation times. Times are numeric and taken either as serial date numbers (day units), or as decimal numbers in arbitrary units (e.g., yearly). If TickTimes = [] or is not specified, then price2ret assumes sequential observation times from 1, 2, ..., NUMOBS.</td></tr><tr><td>Method</td><td>Character string indicating the compounding method to compute asset returns. If Method = 'Continuous', = [], or is not specified, then price2ret computes continuously compounded returns. If Method = 'Periodic', then price2ret assumes simple periodic returns. Method is case insensitive.</td></tr></table>	TickSeries	Time series of price data. TickSeries can be a column vector or a matrix: <ul style="list-style-type: none">• As a vector, TickSeries represents a univariate price series. The length of the vector is the number of observations (NUMOBS). The first element contains the oldest observation, and the last element the most recent.• As a matrix, TickSeries represents a NUMOBS-by-number of assets (NUMASSETS) matrix of asset prices. Rows correspond to time indices. The first row contains the oldest observations and the last row the most recent. price2ret assumes that the observations across a given row occur at the same time for all columns, and each column is a price series of an individual asset.	TickTimes	A NUMOBS element vector of monotonically increasing observation times. Times are numeric and taken either as serial date numbers (day units), or as decimal numbers in arbitrary units (e.g., yearly). If TickTimes = [] or is not specified, then price2ret assumes sequential observation times from 1, 2, ..., NUMOBS.	Method	Character string indicating the compounding method to compute asset returns. If Method = 'Continuous', = [], or is not specified, then price2ret computes continuously compounded returns. If Method = 'Periodic', then price2ret assumes simple periodic returns. Method is case insensitive.
TickSeries	Time series of price data. TickSeries can be a column vector or a matrix: <ul style="list-style-type: none">• As a vector, TickSeries represents a univariate price series. The length of the vector is the number of observations (NUMOBS). The first element contains the oldest observation, and the last element the most recent.• As a matrix, TickSeries represents a NUMOBS-by-number of assets (NUMASSETS) matrix of asset prices. Rows correspond to time indices. The first row contains the oldest observations and the last row the most recent. price2ret assumes that the observations across a given row occur at the same time for all columns, and each column is a price series of an individual asset.						
TickTimes	A NUMOBS element vector of monotonically increasing observation times. Times are numeric and taken either as serial date numbers (day units), or as decimal numbers in arbitrary units (e.g., yearly). If TickTimes = [] or is not specified, then price2ret assumes sequential observation times from 1, 2, ..., NUMOBS.						
Method	Character string indicating the compounding method to compute asset returns. If Method = 'Continuous', = [], or is not specified, then price2ret computes continuously compounded returns. If Method = 'Periodic', then price2ret assumes simple periodic returns. Method is case insensitive.						

Output Arguments

RetSeries

Array of asset returns:

- When TickSeries is a NUMOBS element column vector, RetSeries is a NUMOBS-1 column vector.
- When TickSeries is a NUMOBS-by-NUMASSETS matrix, RetSeries is a (NUMOBS-1)-by-NUMASSETS matrix. price2ret quotes the i th return of an asset for the period TickTimes(i) to TickTimes($i+1$) and normalizes it by the time interval between successive price observations.

Assuming that

$$\text{RetIntervals}(i) = \text{TickTimes}(i+1) - \text{TickTimes}(i)$$

then if Method = 'Continuous', = [], or is not specified, price2ret computes the continuously compounded i th return of an asset as

$$\text{RetSeries}(i) = \frac{\log\left[\frac{\text{TickSeries}(i+1)}{\text{TickSeries}(i)}\right]}{\text{RetIntervals}(i)}$$

If Method = 'Periodic', then price2ret computes the i th simple return as

$$\text{RetSeries}(i) = \frac{\left[\frac{\text{TickSeries}(i+1)}{\text{TickSeries}(i)}\right] - 1}{\text{RetIntervals}(i)}$$

RetIntervals

NUMOBS-1 element vector of interval times between observations. If TickTimes = [] or is not specified, price2ret assumes that all intervals are 1.

Examples

Create a stock price process continuously compounded at 10 percent, then convert the price series to a 10 percent return series.

```
S = 100*exp(0.10 * [0:19]'); % Create the stock price series
R = price2ret(S); % Convert the price series to a 10 percent
% return series
[S [R;NaN]] % Pad the return series so vectors are of same
% length. price2ret computes the ith return from
% the ith and i+1th prices.
```

price2ret

```
ans =  
100.0000    0.1000  
110.5171    0.1000  
122.1403    0.1000  
134.9859    0.1000  
149.1825    0.1000  
164.8721    0.1000  
182.2119    0.1000  
201.3753    0.1000  
222.5541    0.1000  
245.9603    0.1000  
271.8282    0.1000  
300.4166    0.1000  
332.0117    0.1000  
366.9297    0.1000  
405.5200    0.1000  
448.1689    0.1000  
495.3032    0.1000  
547.3947    0.1000  
604.9647    0.1000  
668.5894      NaN
```

See Also

ret2price

Purpose	Convert return series to price series						
Syntax	<pre>[TickSeries, TickTimes] = ret2price(RetSeries, StartPrice, ... RetIntervals, StartTime, Method)</pre>						
Description	<pre>[TickSeries, TickTimes] = ret2price(RetSeries, StartPrice, RetIntervals, StartTime, Method)</pre> generates price series for the specified assets, given the asset starting prices and the return observations for each asset.						
Input Arguments	<table><tr><td>RetSeries</td><td>Time-series array of returns. RetSeries can be a column vector or a matrix:<ul style="list-style-type: none">• As a vector, RetSeries represents a univariate series of returns of a single asset. The length of the vector is the number of observations (NUMOBS). The first element contains the oldest observation, and the last element the most recent.• As a matrix, RetSeries represents a NUMOBS-by-number of assets (NUMASSETS) matrix of asset returns. Rows correspond to time indices. The first row contains the oldest observations and the last row the most recent. ret2price assumes that the observations across a given row occur at the same time for all columns, and each column is a return series of an individual asset.</td></tr><tr><td>StartPrice</td><td>A NUMASSETS element vector of initial prices for each asset, or a single scalar initial price applied to all assets. If StartPrice = [] or is not specified, all asset prices start at 1.</td></tr><tr><td>RetIntervals</td><td>A NUMOBS element vector of time intervals between return observations, or a single scalar interval applied to all observations. If RetIntervals = [] or is not specified, ret2price assumes all intervals have length 1.</td></tr></table>	RetSeries	Time-series array of returns. RetSeries can be a column vector or a matrix: <ul style="list-style-type: none">• As a vector, RetSeries represents a univariate series of returns of a single asset. The length of the vector is the number of observations (NUMOBS). The first element contains the oldest observation, and the last element the most recent.• As a matrix, RetSeries represents a NUMOBS-by-number of assets (NUMASSETS) matrix of asset returns. Rows correspond to time indices. The first row contains the oldest observations and the last row the most recent. ret2price assumes that the observations across a given row occur at the same time for all columns, and each column is a return series of an individual asset.	StartPrice	A NUMASSETS element vector of initial prices for each asset, or a single scalar initial price applied to all assets. If StartPrice = [] or is not specified, all asset prices start at 1.	RetIntervals	A NUMOBS element vector of time intervals between return observations, or a single scalar interval applied to all observations. If RetIntervals = [] or is not specified, ret2price assumes all intervals have length 1.
RetSeries	Time-series array of returns. RetSeries can be a column vector or a matrix: <ul style="list-style-type: none">• As a vector, RetSeries represents a univariate series of returns of a single asset. The length of the vector is the number of observations (NUMOBS). The first element contains the oldest observation, and the last element the most recent.• As a matrix, RetSeries represents a NUMOBS-by-number of assets (NUMASSETS) matrix of asset returns. Rows correspond to time indices. The first row contains the oldest observations and the last row the most recent. ret2price assumes that the observations across a given row occur at the same time for all columns, and each column is a return series of an individual asset.						
StartPrice	A NUMASSETS element vector of initial prices for each asset, or a single scalar initial price applied to all assets. If StartPrice = [] or is not specified, all asset prices start at 1.						
RetIntervals	A NUMOBS element vector of time intervals between return observations, or a single scalar interval applied to all observations. If RetIntervals = [] or is not specified, ret2price assumes all intervals have length 1.						

ret2price

StartTime (optional) Scalar starting time for the first observation, applied to the price series of all assets. The default is 0.

Method Character string indicating the compounding method used to compute asset returns. If Method = 'Continuous', = [], or is not specified, then ret2price computes continuously compounded returns. If Method = 'Periodic' then ret2price computes simple periodic returns. Method is case insensitive.

Output Arguments

TickSeries Array of asset prices:

- When RetSeries is a NUMOBS element column vector, TickSeries is a NUMOBS+1 column vector. The first element contains the starting price of the asset, and the last element the most recent price.
- When RetSeries is a NUMOBS-by-NUMASSETS matrix, then RetSeries is a (NUMOBS+1)-by-NUMASSETS matrix. The first row contains the starting price of the assets, and the last row contains the most recent prices.

TickTimes A NUMOBS+1 element vector of price observation times. The initial time is zero unless specified in StartTime.

Examples

Example 1. Create a stock price process continuously compounded at 10 percent. Compute 10 percent returns for reference, then convert the resulting return series to the original price series and compare results.

```
S = 100*exp(0.10 * [0:19]'); % Create the stock price series
R = price2ret(S);           % Convert the price series to a
                             % 10 percent return series
P = ret2price(R, 100);      % Convert to the original price
                             % series
[S P]                       % Compare the original and
                             % computed price series

ans =

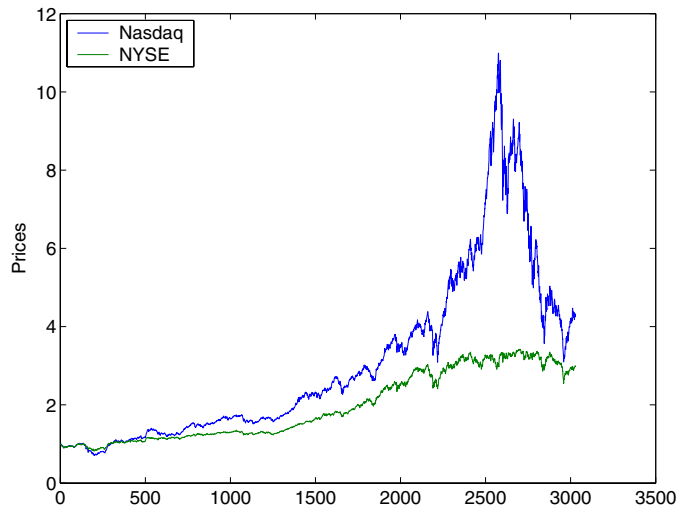
    100.0000    100.0000
```

110.5171	110.5171
122.1403	122.1403
134.9859	134.9859
149.1825	149.1825
164.8721	164.8721
182.2119	182.2119
201.3753	201.3753
222.5541	222.5541
245.9603	245.9603
271.8282	271.8282
300.4166	300.4166
332.0117	332.0117
366.9297	366.9297
405.5200	405.5200
448.1689	448.1689
495.3032	495.3032
547.3947	547.3947
604.9647	604.9647
668.5894	668.5894

Example 2. This example compares the relative price performance of the Nasdaq and the NYSE indexes (see “Data Sets” on page 1-11). Before plotting the series, the example converts the prices to returns, then converts them back to prices specifying the same starting price, 100, for each series. In the plot, the blue (upper) plot shows the NASDAQ price series, the green (lower) plot shows the NYSE price series.

```
load garchdata
nasdaq = price2ret(NASDAQ);
nyse = price2ret(NYSE);
plot(ret2price(price2ret([NASDAQ NYSE]), 100))
ylabel('Prices')
legend('Nasdaq', 'NYSE', 2)
```

ret2price



See Also

[price2ret](#)

Bibliography

- [1] Baillie, R.T., and T. Bollerslev, "Prediction in Dynamic Models with Time-Dependent Conditional Variances," *Journal of Econometrics*, Vol. 52, 1992, pp 91–113.
- [2] Bera, A.K., and H.L. Higgins, "A Survey of ARCH Models: Properties, Estimation and Testing," *Journal of Economic Surveys*, Vol. 7, No. 4, 1993.
- [3] Bollerslev, T., "A Conditionally Heteroskedastic Time Series Model for Speculative Prices and Rates of Return," *Review of Economics and Statistics*, Vol. 69, 1987, pp 542–547.
- [4] Bollerslev, T., "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, Vol. 31, 1986, pp 307–327.
- [5] Bollerslev, T., R.Y. Chou, and K.F. Kroner, "ARCH Modeling in Finance: A Review of the Theory and Empirical Evidence," *Journal of Econometrics*, Vol. 52, 1992, pp 5–59.
- [6] Bollerslev, T., R.F. Engle, and D.B. Nelson, "ARCH Models," *Handbook of Econometrics*, Volume IV, Chapter 49, pp 2959–3038, Elsevier Science B.V., Amsterdam, The Netherlands, 1994.
- [7] Bollerslev, T., and E. Ghysels, "Periodic Autoregressive Conditional Heteroscedasticity," *Journal of Business and Economic Statistics*, Vol. 14, 1996, pp 139–151.
- [8] Box, G.E.P., G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis: Forecasting and Control*, Third edition, Prentice Hall, Upper Saddle River, NJ, 1994.
- [9] Brooks, C., S.P. Burke, and G. Persaud, "Benchmarks and the Accuracy of GARCH Model Estimation," *International Journal of Forecasting*, Vol. 17, 2001, pp 45–56.
- [10] Campbell, J.Y., A.W. Lo, and A.C. MacKinlay, "The Econometrics of Financial Markets," *Nonlinearities in Financial Data*, Chapter 12, Princeton University Press, Princeton, NJ, 1997.
- [11] Enders, W., *Applied Econometric Time Series*, John Wiley & Sons, New York, 1995.
- [12] Engle, Robert F., "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica*, Vol. 50, 1982, pp 987–1007.

-
- [13] Engle, Robert F., D.M. Lilien, and R.P. Robins, "Estimating Time Varying Risk Premia in the Term Structure: The ARCH-M Model," *Econometrica*, Vol. 59, 1987, pp 391–407.
- [14] Glosten, L.R., R. Jagannathan, and D.E. Runkle, "On the Relation between Expected Value and the Volatility of the Nominal Excess Return on Stocks," *The Journal of Finance*, Vol. 48, 1993, pp 1779–1801.
- [15] Gouriéroux, C., *ARCH Models and Financial Applications*, Springer-Verlag, 1997.
- [16] Greene, W.H., *Econometric Analysis*, Fifth edition, Prentice Hall, Upper Saddle River, NJ, 2003.
- [17] Hagerud, G.E., "Modeling Nordic Stock Returns with Asymmetric GARCH," *Working Paper Series in Economics and Finance*, No. 164, Department of Finance, Stockholm School of Economics, 1997.
- [18] Hagerud, G.E., "Specification Tests for Asymmetric GARCH," *Working Paper Series in Economics and Finance*, No. 163, Department of Finance, Stockholm School of Economics, 1997.
- [19] Hamilton, J.D., *Time Series Analysis*, Princeton University Press, Princeton, NJ, 1994.
- [20] Hentschel, L., "All in the Family: Nesting Symmetric and Asymmetric GARCH Models," *Journal of Financial Economics*, Vol. 39, 1995, pp 71–104.
- [21] Johnson, N.L., S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions*, Vol. 2, Second edition, John Wiley & Sons, New York, 1995.
- [22] McCullough, B.D., and C.G. Renfro, "Benchmarks and Software Standards: A Case Study of GARCH Procedures," *Journal of Economic and Social Measurement*, Vol. 25, 1998, pp 59–71.
- [23] Nelson, D.B., "Conditional Heteroskedasticity in Asset Returns: A New Approach," *Econometrica*, Vol. 59, 1991, pp 347–370.
- [24] Peters, J.P., "Estimating and Forecasting Volatility of Stock Indices Using Asymmetric GARCH Models and Skewed Student-t Densities," Working Paper, École d'Administration des Affaires, University of Liège, Belgium, March 20, 2001.

Akaike information criteria (AIC)	A model-order selection criterion based on parsimony. More complicated models are penalized for the inclusion of additional parameters. <i>See also</i> Bayesian information criteria .
AR	Autoregressive. AR models include past observations of the dependent variable in the forecast of future observations.
ARCH	Autoregressive Conditional Heteroscedasticity. A time-series technique in which past observations of the variance are used to forecast future variances. <i>See also</i> GARCH .
ARMA	Autoregressive Moving Average. A time-series model that includes both AR and MA components. <i>See also</i> AR and MA .
autocorrelation function (ACF)	Correlation sequence of a random time series with itself. <i>See also</i> crosscorrelation function .
autoregressive	<i>See</i> AR .
Bayesian information criteria (BIC)	A model-order selection criterion based on parsimony. More complicated models are penalized for the inclusion of additional parameters. Since BIC imposes a greater penalty for additional parameters than AIC, BIC always provides a model with a number of parameters no greater than that chosen by AIC. <i>See also</i> Akaike information criteria .
conditional	Time-series technique with explicit dependence on the past sequence of observations.
conditional mean	Time-series model for forecasting the expected value of the return series itself.
conditional variance	Time-series model for forecasting the expected value of the variance of the return series.
crosscorrelation function (XCF)	Correlation sequence between two random time series. <i>See also</i> autocorrelation function .
equality constraint	A constraint, imposed during parameter estimation, by which a parameter is held fixed at a user-specified value.
excess kurtosis	A characteristic, relative to a standard normal probability distribution, whereby an area under the probability density function is reallocated from the center of the distribution to the tails (fat tails). Samples obtained from distributions with excess kurtosis have a higher probability of containing outliers than samples drawn from a normal (Gaussian) density. Time series that exhibit a fat tail distribution are often referred to as leptokurtic.

explanatory variables	Time series used to explain the behavior of another observed series of interest. Explanatory variables are typically incorporated into a regression framework.
fat tails	<i>See</i> excess kurtosis .
GARCH	Generalized Autoregressive Conditional Heteroscedasticity. A time-series technique in which past observations of the variance and variance forecast are used to forecast future variances. <i>See also</i> ARCH .
heteroscedasticity	Time-varying, or time-dependent, variance.
homoscedasticity	Time-independent variance. The GARCH Toolbox also refers to homoscedasticity as constant conditional variance.
i.i.d.	Independent, identically distributed.
innovations	A sequence of unanticipated shocks, or disturbances. The GARCH Toolbox uses innovations and residuals interchangeably.
leptokurtic	<i>See</i> excess kurtosis .
MA	Moving average. MA models include past observations of the innovations noise process in the forecast of future observations of the dependent variable of interest.
MMSE	Minimum mean square error. A technique designed to minimize the variance of the estimation or forecast error. <i>See also</i> RMSE .
moving average	<i>See</i> MA .
objective function	The function to be numerically optimized. In the GARCH Toolbox, the objective function is the log-likelihood function of a random process.
partial autocorrelation function (PACF)	Correlation sequence estimated by fitting successive order autoregressive models to a random time series by least squares. The PACF is useful for identifying the order of an autoregressive model.
path	A random trial of a time-series process.
P-value	The lowest level of significance at which a test statistic is significant.
realization	<i>See</i> path .
residuals	<i>See</i> innovations .
RMSE	Root mean square error. The square root of the mean square error. <i>See also</i> MMSE .

standardized innovations	The innovations divided by the corresponding conditional standard deviation.
time series	Discrete-time sequence of observations of a random process. The type of time series of interest in the GARCH Toolbox is typically a series of returns, or relative changes of some underlying price series.
transient	A response, or behavior, of a time series that is heavily dependent on the initial conditions chosen to begin a recursive calculation. The transient response is typically undesirable, and initially masks the true steady-state behavior of the process of interest.
unconditional	Time-series technique in which explicit dependence on the past sequence of observations is ignored. Equivalently, the time stamp associated with any observation is ignored.
volatility	The risk, or uncertainty, measure associated with a financial time series. The GARCH Toolbox associates volatility with standard deviation.

A

- ACF 11-11
- AIC
 - model selection 9-5
- aicbic 11-6
- Akaike information criteria
 - model selection 9-5
- analysis example
 - using default model 2-16
- AR model
 - converting from ARMA model 11-28
- ARCH/GARCH effects
 - hypothesis test 11-8
- archtest function 11-8
- ARMA model
 - converting to AR model 11-28
 - converting to MA model 11-48
- array size 1-7
- arrays
 - time series 1-7
- asymptotic behavior
 - for long-range forecast horizons 6-6
 - long-range forecasts 6-6
- autocorr function 11-11
- autocorrelation function 11-11
- autoregressive model
 - converting from ARMA model 11-28

B

- Bayesian information criteria
 - model selection 9-5
- BIC
 - model selection 9-5

C

- compounding
 - continuous and periodic 1-8
- conditional mean models 2-6
 - regression components 7-2, 8-2
- conditional standard deviations
 - inferred from return series 11-43
 - of forecast errors 11-54
 - simulating 11-67
- conditional variance models 2-7
- conditional variances
 - constant 7-12
- constraints
 - active lower bound example 5-30
 - equality 9-7
 - fixing model parameters 9-7
 - tolerance limits 5-16
- conventions
 - technical 1-7
- convergence
 - avoiding problems 2-16
 - determining status 5-34
 - showing little progress 2-16
 - suboptimal solution 2-16
 - tolerance options 5-14
- crosscorr function 11-15
- crosscorrelation function 11-15

D

- data sets 1-11
 - Deutschmark/British Pound FX price series
 - 1-11
 - Nasdaq Composite Index 1-12
 - New York Stock Exchange Composite Index
 - 1-12

- default
 - GARCH model 2-13
- default model
 - estimation and analysis example 2-16
 - estimation example 2-16
- Deutschmark/British Pound FX price series
 - 1-11
- dfARDTest 11-19
- dfARTest 11-22
- dfTSTest 11-25
- distributions
 - supported 2-5

E

- EGARCH(P,Q) conditional variance model 2-9
- Engle's hypothesis test 11-8
- equality constraints
 - initial parameter estimates 9-12
 - parameter significance 9-7
- estimation 5-1
 - advanced example 10-2
 - control of optimization process 5-13
 - convergence 5-14
 - convergence to suboptimal solution 2-16
 - count of coefficients 9-5, 11-31
 - incorporating a regression model 7-3
 - initial parameter estimates 5-4
 - maximum likelihood 5-2
 - number of function evaluations 5-13
 - number of iterations 5-13
 - of GARCH process 11-34
 - optimization results 5-15
 - parameter bounds 5-9
 - plotting results 11-51
 - premature termination 2-16
 - presample observations 5-11

- summary information 11-37
- termination criteria 5-13
- tolerance options 5-14

- estimation example
 - estimating model parameters 2-25
 - postestimation analysis 2-28
 - preestimation analysis 2-16
 - using default model 2-16

F

- fat tails 2-2
- financial time series
 - characteristics 2-2
 - modeling 2-2
- fixing model constraints 9-7
- forecast errors
 - conditional standard deviations 11-54
- forecast results
 - compare with simulation results 10-8
- forecasted explanatory data 7-10
- forecasting 6-1
 - advanced example 10-4
 - asymptotic behavior for long-range 6-6
 - basic example 6-8
 - conditional mean of returns 6-3
 - conditional standard deviations of innovations
 - 6-2
 - minimum mean square error 6-2
 - multiperiod volatility example 6-11
 - multiple realizations example 6-14
 - plotting results 11-51
 - presample data 6-5
 - RMSE of mean forecast 6-4
 - using a regression model 7-10
 - volatility of returns 6-3
- function evaluation count

- maximum 5-13
- functions
 - example showing relationships 10-1
 - primary engines 2-14
- G**
- GARCH
 - brief description 1-3
 - limitations 1-4
 - uses for 1-3
- GARCH process
 - forecasting 11-54
 - inferring innovations 11-43
 - parameter estimation 11-34
 - count of coefficients 11-31
 - displaying results 11-32
 - simulation 11-67
- GARCH specification structure
 - contents 3-5
 - creating and modifying parameters 3-8
 - definition of fields 11-62
 - retrieving parameters 11-42
- GARCH Toolbox
 - conventions and clarifications
 - compounding 1-8
 - primary functions 2-14
- GARCH(P,Q) conditional variance model 2-7
- garchar function 11-28
- garchcount function 11-31
- garchdisp function 11-32
- garchfit function 11-34
- garchget function 11-42
- garchinfer function 11-43
- garchma function 11-48
- garchplot function 11-51
- garchpred function 11-54

- garchset function 11-61
- garchsim function 11-67
- GJR(P,Q) conditional variance model 2-8

H

- hypothesis tests
 - likelihood ratio 11-81
 - Ljung-Box lack-of-fit 11-78

I

- inference
 - conditional standard deviations 11-43
 - GARCH innovations 11-43
 - transient effects example 5-23
 - using a regression model 7-9
- initial parameter estimates 5-4
 - conditional mean models with regression 5-6
 - conditional mean models without regression 5-6
 - conditional variance models 5-7
 - equality constraints 9-12
- innovations
 - distribution 2-5
 - forecasting conditional standard deviations 6-2
 - inferred from return series 11-43
 - serial dependence 2-5
 - simulating 11-67
- iteration count
 - maximum 5-13

L

- lack-of-fit hypothesis test 11-78
- lagged time-series matrix 11-76

- lagmatrix function 11-76
- lbqtest function 11-78
- length
 - vector 1-7
- leverage effects 2-2
- likelihood ratio hypothesis test 11-81
- likelihood ratio tests
 - model selection 9-2
- Ljung-Box lack-of-fit hypothesis test 11-78
- log-likelihood functions 5-2
 - optimized value parameters 11-34
- long-range forecasting
 - asymptotic behavior 6-6
- lratiotest function 11-81

M

- MA model
 - converting from ARMA model 11-48
- maximum likelihood
 - estimation 5-2
- minimum mean square error
 - forecasting 6-2
- MMSE
 - forecasting 6-2
- model parameters
 - complete specification 9-12
 - empty fix fields 9-13
 - equality constraints 9-7
 - estimating 2-25
 - fixing 9-7
 - parsimony 9-15
- model selection and analysis 9-1
 - AIC and BIC 9-5
 - correlation in return series 2-20
 - correlation in squared returns 2-22
 - Engle's ARCH test 2-24

- likelihood ratio tests 9-2
 - Ljung-Box-Pierce Q-test 2-23
- modeling
 - financial time series 2-2
- models
 - complete specification 9-12
 - conditional mean and variance 2-6
 - GARCH default 2-13
- Monte Carlo simulation 7-14
 - advanced example 10-6
 - compare with forecast results 10-8
- moving average model
 - converting from ARMA model 11-48

N

- Nasdaq Composite Index 1-12
- New York Stock Exchange Composite Index 1-12
- nonstationary time series 1-9
- NYSE Composite Index 1-12

O

- ordinary least squares regression 7-12

P

- PACF 11-83
- parameter estimates
 - bounds 5-9
 - displaying results 11-32
 - equality constraints 9-12
 - initial 5-4
 - automatically generated 5-5
 - user-specified 5-4
- parameter estimation
 - plotting results 11-51

- univariate GARCH process 11-34
- parcorr function 11-83
- parsimonious parameterization 9-15
- partial autocorrelation function 11-83
- plotting
 - autocorrelation function 11-11
 - crosscorrelation function 11-15
 - forecasted results 11-51
 - parameter estimation results 11-51
 - partial autocorrelation function 11-83
 - simulation results 11-51
- ppARDTest 11-87
- ppARTest 11-90
- ppTSTest 11-93
- precision 1-8
- prerequisites 1-5
- presample data
 - estimation
 - automatically generated 5-11
 - deriving from actual data 5-29
 - example 5-19
 - user-specified 5-11
 - forecasting 6-5
 - simulation
 - automatically generated 4-7
 - user-specified 4-13
- price series
 - converting from return series 11-99
 - converting to return series 11-96
- price2ret function 11-96

R

- regression
 - in Monte Carlo framework 7-14
- regression components
 - conditional mean models 7-2, 8-2

- estimation 7-3
- forecasting 7-10
- inference 7-9
- simulation 7-9
- response tolerance
 - for simulated data 4-8
- ret2price function 11-99
- return series
 - converting from price series 11-96
 - converting to price series 11-99
 - forecasting conditional mean 6-3
 - forecasting RMSE of mean forecast 6-4
 - forecasting volatility 6-3
 - simulating 11-67

S

- shifted time-series matrix 11-76
- simulation 4-1
 - compare with forecast results 10-8
 - plotting results 11-51
 - presample data 4-7
 - response tolerance 4-8
 - sample paths 4-2
 - storage considerations 4-10
 - univariate GARCH processes 11-67
 - using a regression model 7-9
 - using ordinary least squares regression 7-12
- size
 - array and vector 1-7
- specification structure
 - contents 3-5
 - creating and modifying parameters 3-8
 - definition of fields 11-62
 - fixing model parameters 9-7
 - retrieving parameters 11-42
- stationary time series 1-9

T

- termination criteria
 - estimation 5-13
- time series
 - characteristics of financial 2-2
 - correlation of observations 2-4
 - modeling financial 2-2
 - stationary and nonstationary 1-9
 - stationary, nonstationary 1-9
- time-series matrix 1-7
 - lagged or shifted 11-76
- tolerance options 5-14
 - constraint violation 5-16
 - effect on convergence 5-14
 - effect on optimization results 5-15
- transients
 - automatic minimization 4-7
 - in presample simulation data 4-7
 - inference example 5-23
 - minimization techniques 4-11
 - simulation process 4-7

V

- vector length 1-7
- vector size 1-7
- volatility
 - forecasting 6-3
 - forecasting example 6-11
- volatility clustering 2-2

X

- XCF 11-15