FANPAC

Documentation Version: December 1, 2000

Contents

Chapter 1

Installation

1.1 UNIX

If you are unfamiliar with UNIX, see your system administrator or system documentation for information on the system commands referred to below. The device names given are probably correct for your system.

1. Use `cd` to make the directory containing **GAUSS** the current working directory.
2. Use `tar` to extract the files.

   `tar xvf` *device_name*

   If this software came on diskettes, repeat the `tar` command for each diskette.

The following device names are suggestions. See your system administrator. If you are using Solaris 2.x, see Section 1.1.1.

Operating System	3.5-inch diskette	1/4-inch tape	DAT tape
Solaris 1.x SPARC	`/dev/rfd0`	`/dev/rst8`	
Solaris 2.x SPARC	`/dev/rfd0a` (vol. mgt. off)	`/dev/rst12`	`/dev/rmt/1l`
Solaris 2.x SPARC	`/vol/dev/aliases/floppy0`	`/dev/rst12`	`/dev/rmt/1l`
Solaris 2.x x86	`/dev/rfd0c` (vol. mgt. off)		`/dev/rmt/1l`
Solaris 2.x x86	`/vol/dev/aliases/floppy0`		`/dev/rmt/1l`
HP-UX	`/dev/rfloppy/c20Ad1s0`		`/dev/rmt/0m`
IBM AIX	`/dev/rfd0`	`/dev/rmt.0`	
SGI IRIX	`/dev/rdsk/fds0d2.3.5hi`		

1.1.1 Solaris 2.x Volume Management

If Solaris 2.x volume management is running, insert the floppy disk and type

```
volcheck
```

to signal the system to mount the floppy.

The floppy device names for Solaris 2.x change when the volume manager is turned off and on. To turn off volume management, become the superuser and type

```
/etc/init.d/volmgt off
```

To turn on volume management, become the superuser and type

```
/etc/init.d/volmgt on
```

1.2 DOS

1. Place the diskette in a floppy drive.
2. Log onto the root directory of the diskette drive. For example:

   ```
   A:<enter>
   cd\<enter>
   ```

3. Type: **ginstall** *source_drive* *target_path*

source_drive	Drive containing files to install with colon included For example: **A:**
target_path	Main drive and subdirectory to install to without a final \ For example: **C:\GAUSS**

   A directory structure will be created if it does not already exist and the files will be copied over.

*target_path***src**	source code files
*target_path***lib**	library files
*target_path***examples**	example files

4. The screen output option used may require that the DOS screen driver ANSI.SYS be installed on your system. If ANSI.SYS is not already installed on your system, you can put the command like this one in your CONFIG.SYS file:

   DEVICE=C:\DOS\ANSI.SYS

   (This particular statement assumes that the file ANSI.SYS is on the subdirectory DOS; modify as necessary to indicate the location of your copy of ANSI.SYS.)

1.3 Differences Between the UNIX and DOS Versions

- In the DOS version, when the global **__output** = 2, information may be written to the screen using commands requiring the ANSI.SYS screen driver. These are not available in the current UNIX version, and therefore setting **__output** = 2 may have the same effect as setting **__output** = 1.
- If the functions can be controlled during execution by entering keystrokes from the keyboard, it may be necessary to press *Enter* after the keystroke in the UNIX version.
- On the Intel math coprocessors used by the DOS machines, intermediate calculations have 80-bit precision, while on the current UNIX machines, all calculations are in 64-bit precision. For this reason, **GAUSS** programs executed under UNIX may produce slightly different results, due to differences in roundoff, from those executed under DOS.

Chapter 2

Financial Analysis Package

written by

Ronald Schoenberg

This package provides procedures for the econometric analysis of financial data.

2.1 Getting Started

GAUSS 3.2.17+	DOS
GAUSS 3.2.28+	OS/2
GAUSS 3.2.32+	Windows NT/95
GAUSS 3.2.34+	UNIX

is required to use these routines.

2.1.1 README Files

The file **README.fan** contains any last minute information on this module. Please read it before using the procedures in this module.

2.1.2 Setup

The **FANPAC** library must be active in order to use the procedures in the *Financial Analysis Package*. Please make certain to include `fanpac` in the **LIBRARY** statement at the top of your program or command file. This will enable **GAUSS** to find the *Financial Analysis Procedures*.

```
library fanpac,pgraph;
```

If you plan to make any right hand references to the global variables (described in the *REFERENCE* sections), you also need the statement:

```
#include fanpac.ext;
```

Finally, to reset global variables in succeeding executions of the command file, the following instruction can be used:

```
clearSession;
```

This could be included with the above statements without harm and would ensure the proper definition of the global variables for all executions of the command file.

The version number of each module is stored in a global variable:

_fan_ver 3×1 matrix: the first element contains the major version number of the *Financial Analysis Package*, the second element the minor version number, and the third element the revision number.

If you call for technical support, please have the version number of your copy of this module on hand.

2.2 Modelling with FANPAC

FANPAC is a set of keyword commands and procedures for the estimation of parameters of time series models via the maximum likelihood method. The package is divided into two parts: (1) easy-to-program keyword commands which simplify the modelling process; and (2) **GAUSS** procedures, which can be called directly to perform the computations.

The **FANPAC** keyword commands considerably simplify the work for the analysis of time series. For example, the following command file (which may also be entered interactively)

```
library fanpac,pgraph;
session test 'Analysis of 1996 Intel Stock Prices';
setDataset stocks;
setSeries intel;
estimate run1 garch(1,1);
estimate run2 arima(1,2,1);
showResults;
plotSeries;
plotCV;
```

replaces about a hundred lines of **GAUSS** code using procedures. See Chapter 4 for a description of the keyword commands.

Summary of Keyword Commands

clearSession	clears session from memory, resets global variables
constrainPDCovPar	sets **NLP** global for constraining covariance matrix of parameters to be positive definite
computeLogReturns	computes log returns from price data
computePercentReturns	computes percent returns from price data
estimate	estimates parameters of a time series model
forecast	generates a time series and conditional variance forecast
getCV	puts conditional variances or variance-covariance matrices into global vector **_fan_CV**
getCOR	puts conditional correlations into global variable **_fan_COR**
getEstimates	puts model estimates into global variable **_fan_Estimates**
getResiduals	puts unstandardized residuals into global vector
getSeriesACF	puts autocorrelations into global variable **_fan_ACF**
getSeriesPACF	puts partial autocorrelations into global variable **_fan_PACF**
getSession	retrieves a data analysis session
getSR	puts standardized residuals into global vector
plotCOR	plots conditional correlations
plotCSD	plots conditional standard deviations
plotCV	plots conditional variances
plotQQ	generates quantile-quantile plot
plotSeries	plots time series
plotSeriesACF	plots autocorrelations

plotSeriesPACF	plots partial autocorrelations
plotSR	plots standardized residuals
session	initializes a data analysis session
setAlpha	sets inference alpha level
setConstraintType	sets type of constraints on parameters
setCovParType	sets type of covariance matrix of parameters
setCVIndEqs	declares list of independent variables to be included in conditional variance equations
setDataset	sets dataset name
setIndEqs	declares list of independent variables to be included in mean equations
setInferenceType	sets type of inference
setIndVars	declares names of independent variables
setLagTruncation	sets lags included for FIGARCH model
setLagInitialization	sets lags excluded for FIGARCH model
setLjungBoxOrder	sets order for Ljung-Box statistic
setOutputFile	sets output file name
setRange	sets range of data
setSeries	declares names of time series
setVarNames	sets variable names for data stored in ASCII file
showEstimates	displays estimates in simple format
showResults	displays results of estimations
showRuns	displays runs
simulate	generates simulation
testSR	generates skew, kurtosis, Ljung-Box statistics

If the computations performed by the **FANPAC** keyword commands do not precisely fit your needs, you may design your own command files using the **FANPAC** procedures. For example, you may want to impose alternative sets of constraints on the parameters of a FIGARCH model. To do this you would design your own FIGARCH estimation using the **FANPAC** procedures discussed in Section 2.7 in this chapter, and described in Chapter 4.

You might also want to write your own procedures for models not included in **FANPAC**. To do this you will need to write a procedure for computing the log-likelihood and call **NLP** procedures for the estimation. These procedures are discussed in Section 2.8 in this chapter, and are described in Chapter 5.

When the **FANPAC** keyword commands are used, analysis results are stored in a file on disk. This information can be retrieved or modified as necessary. Results are not stored if there is an error, and thus the original results are not lost when this happens. These keyword commands can be invoked either in command files or interactively from the **GAUSS** command line. They may also be mixed with other **GAUSS** commands either in a command file or interactively.

The following models are available in **FANPAC**:

ols	normal linear regression model
$tols$	t distribution linear regression model
$arima(p, d, q)$	normal ARIMA model
$tarima(p, d, q)$	t distribution ARIMA model
$arch(q)$	normal ARCH model
$tarch(q)$	t distribution ARCH model
$archm(q)$	normal ARCH-in-mean model
$tarchm(q)$	t distribution ARCH-in-mean model
$archv(q)$	normal ARCH-in-cv model
$tarchv(q)$	t distribution ARCH-in-cv model
$garch(p, q)$	normal GARCH model
$tgarch(p, q)$	t distribution GARCH model
$garchm(p, q)$	normal GARCH-in-mean model
$tgarchm(p, q)$	t distribution GARCH-in-mean model
$garchv(p, q)$	normal GARCH-in-cv model
$tgarchv(p, q)$	t distribution GARCH-in-cv model
$igarch(p, q)$	normal integrated GARCH model
$itgarch(p, q)$	t distribution integrated GARCH model
$egarch(p, q)$	exponential GARCH model
$figarch(p, q)$	normal fractionally integrated GARCH model
$fitgarch(p, q)$	t distribution fractionally integrated GARCH model
$figarch(p, q)$	normal fractionally integrated GARCH model
$fitgarch(p, q)$	t distribution fractionally integrated GARCH model
$dvarch(p, q)$	normal DVEC multivariate ARCH model
$cdvarch(p, q)$	constant correlation normal DVEC multivariate ARCH model
$bkarch(p, q)$	normal BEKK multivariate ARCH model
$dvtarch(p, q)$	t distribution DVEC multivariate ARCH model
$cdvtarch(p, q)$	constant correlation t distribution DVEC multivariate ARCH model
$bktarch(p, q)$	t distribution BEKK multivariate ARCH model
$dvarchm(p, q)$	normal DVEC multivariate ARCH-in-mean model
$cdvarchm(p, q)$	constant correlation normal DVEC multivariate ARCH-in-mean model

dvtarchm(p, q)	t distribution DVEC multivariate ARCH-in-mean model
cdvtarchm(p, q)	constant correlation t distribution DVEC multivariate ARCH-in-mean model
bktarchm(p, q)	t distribution BEKK multivariate ARCH-in-mean model
dvarchv(p, q)	normal DVEC multivariate ARCH-in-cv model
cdvarchv(p, q)	constant correlation normal DVEC multivariate ARCH-in-cv model
dvtarchv(p, q)	t distribution DVEC multivariate ARCH-in-cv model
cdvtarchv(p, q)	constant correlation t distribution DVEC multivariate ARCH-in-cv model
dvgarch(p, q)	normal DVEC multivariate GARCH model
cdvgarch(p, q)	constant correlation normal DVEC multivariate GARCH model
dvtgarch(p, q)	t distribution DVEC multivariate GARCH model
cdvtgarch(p, q)	constant correlation t distribution DVEC multivariate GARCH model
bkgarch(p, q)	normal BEKK multivariate GARCH model
bktgarch(p, q)	t distribution BEKK multivariate GARCH model
dvgarchm(p, q)	normal DVEC multivariate GARCH-in-mean model
cdvgarchm(p, q)	constant correlation normal DVEC multivariate GARCH-in-mean model
dvtgarchm(p, q)	t distribution DVEC multivariate GARCH-in-mean model
cdvtgarchm(p, q)	constant correlation t distribution DVEC multivariate GARCH-in-mean model
dvgarchv(p, q)	normal DVEC multivariate GARCH-in-cv model
cdvgarchv(p, q)	constant correlation normal DVEC multivariate GARCH-in-cv model
dvtgarchv(p, q)	t distribution DVEC multivariate GARCH-in-cv model
cdvtgarchv(p, q)	constant correlation t distribution DVEC multivariate GARCH-in-cv model

If the models are declared without numbers in parentheses, then p, q, and d are assumed to be one.

2.3 Univariate Time Series Models

2.3.1 ARCH

For the autoregressive conditional heteroskedastic (ARCH) model, define the series

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed time series, x_t an observed time series of exogenous variables including a column of ones, and β a vector of coefficients. Furthermore,

$$\epsilon_t \equiv \eta_t\sigma_t$$

where $E(\eta_t) = 0$, $Var(\eta_t) = 1$, and

$$\sigma_t^2 = \omega + \alpha_1\epsilon_{t-1}^2 + \cdots + \alpha_q\epsilon_{t-q}^2$$

For maximum likelihood estimation of this model we first provide a distribution for η_t. Two distributions are available for ARCH in **FANPAC**, the Normal and Student's t. The log-likelihood also requires q initial variances. The observed unconditional variance is used to initialize the process.

ARCH-in-cv

For the ARCH-in-cv model, independent variables may be added to the equation for the conditional variance

$$\sigma_t^2 = \omega + \alpha_1\epsilon_{t-1}^2 + \cdots + \alpha_q\epsilon_{t-q}^2 + Z_t\Gamma$$

where Z_t is the t-th vector of observed independent variables and Γ a matrix of coefficients.

ARCH-in-mean

For the ARCH-in-mean (or ARCHM) model, the time series mean equation is modified to include the conditional variance

$$\epsilon_t = y_t - x_t\beta - \delta\sigma_t$$

log-likelihood

The conditional log-likelihood, given the above requirements, of the ARCH model with $\eta_t \sim N(0,1)$ is

$$logL = -\frac{T-q}{2} log(2\pi) - \sum_{t=q}^{T} log(\sigma_t) - \frac{1}{2} \sum_{t=q}^{T} \frac{\epsilon_t^2}{\sigma_t^2}$$

where

$$\sigma_{t+q-1}^2 = \sigma_{t+q-2}^2 = \cdots = \sigma_1^2 = \frac{1}{T} \sum_{T}^{t=1} \epsilon_t^2$$

The unit t distribution with ν degrees of freedom and variance σ^2 is

$$f(u) = \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)(\nu-2)^{1/2}\pi^{1/2}\sigma} \left(1 + \frac{u^2}{(\nu-2)\sigma^2}\right)^{-(\nu+1)/2)}$$

The conditional log-likelihood for $\eta_t \sim t(0,1,\nu)$ is then

$$\begin{aligned} logL &= -\tfrac{T-q}{2} log\left(\tfrac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)(\nu-2)^{1/2}\pi^{1/2}\sigma}\right) - \textstyle\sum_{t=q}^{T} log(\sigma_t) \\ &\quad -\tfrac{\nu+1}{2} \textstyle\sum_{t=q}^{T} log\left(1 + \tfrac{\epsilon_t^2}{(\nu-2)\sigma_t^2}\right) \end{aligned}$$

constraints

Constraints on the parameters are necessary to enforce the stationarity of the ARCH model as well as the nonnegativity of the conditional variances. The nonnegativity of the conditional variances is assured by the following constraints on the parameters (Nelson and Cao, 1992)

$$\omega \geq 0$$

$$\alpha_i > 0, \; i = 1, \cdots, q$$

and strict stationarity by (Gouriéroux, 1997)

$$\sum_i \alpha_i < 1$$

Stationarity in the ARCH-in-cv model is conditional on the exogenous variables included in the conditional variance equation. There is no assurance of unconditional stationarity without further constraints or assumptions with respect to the exogenous variables.

2.3.2 GARCH

The generalized autoregressive conditional heteroskedastic (GARCH) model is an important variation on the ARCH model. Define the time series

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed time series, x_t an observed time series of exogenous variables including a column of ones, and β a vector of coefficients.

Also define

$$\epsilon_t \equiv \eta_t\sigma_t$$

where $E(\eta_t) = 0$, $Var(\eta_t) = 1$, and

$$\begin{aligned} \sigma_t^2 &= \omega + \alpha_1\epsilon_{t-1}^2 + \cdots + \alpha_q\epsilon_{t-q}^2 + \\ &\quad \beta_1\sigma_{t-1}^2 + \cdots + \beta_p\sigma_{t-p}^2 \end{aligned}$$

For maximum likelihood estimation of the GARCH model, we provide two distributions for η_t, the Normal and Student's t.

GARCH-in-cv

For the GARCH-in-cv model, independent variables may be added to the conditional variance equation

$$\begin{aligned} \sigma_t^2 &= \omega + \alpha_1\epsilon_{t-1}^2 + \cdots + \alpha_q\epsilon_{t-q}^2 + \\ &\quad \beta_1\sigma_{t-1}^2 + \cdots + \beta_p\sigma_{t-p}^2 + Z_t\Gamma \end{aligned}$$

where Z_t is the t-th vector of observed independent variables and Γ a matrix of coefficients.

GARCH-in-mean

For the GARCH-in-mean (or GARCHM) model, the time series mean equation is modified to include the conditional variance

$$\epsilon_t = y_t - x_t\beta - \delta\sigma_t$$

FANPAC

log-likelihood

The log-likelihood conditional on $\mu = max(p,q)$ initial estimates of the conditional variances is, for $\eta_t \sim N(0,1)$

$$logL = -\frac{T-\mu}{2}log(2\pi) - \sum_{t+\mu+1}^{T} log(\sigma_t) - \frac{1}{2}\sum_{t+\mu+1}^{T}\frac{\epsilon_t^2}{\sigma_t^2}$$

where

$$\sigma_1^2 = \sigma_2^2 = \cdots = \sigma_\mu^2 = \frac{1}{T}\sum_{t=1}^{T}\epsilon_t^2$$

The unit t distribution with ν degrees of freedom and variance σ^2 is

$$f(u) = \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)(\nu-2)^{1/2}\pi^{1/2}\sigma}\left(1+\frac{u^2}{(\nu-2)\sigma^2}\right)^{-(\nu+1)/2}$$

The conditional log-likelihood for $\eta_t \sim t(0,1,\nu)$ is then

$$\begin{aligned} logL \quad = \quad & -\tfrac{T-\mu}{2}log\left(\tfrac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)(\nu-2)^{1/2}\pi^{1/2}\sigma}\right) - \textstyle\sum_{t+\mu}^{T} log(\sigma_t) \\ & -\tfrac{\nu+1}{2}\textstyle\sum_{t+\mu}^{T} log\left(1+\tfrac{\epsilon_t^2}{(\nu-2)\sigma_t^2}\right) \end{aligned}$$

Nonnegativity of Conditional Variances

Constraints may be placed on the parameters to enforce the stationarity of the GARCH model as well as the nonnegativity of the conditional variances.

Nelson and Cao (1992) established necessary and sufficient conditions for nonnegativity of the conditional variances for the GARCH(1,q) and GARCH(2,q) models.

GARCH(1,q).

$$\omega \geq 0$$

$$\beta_1 \geq 0$$

$$\sum_{j=0}^{k}\alpha_{j+1}\beta^{k-j} \geq 0,\ k = 0,\cdots,q-1$$

GARCH(2,q). Define Δ_1 and Δ_2 as the roots of

$$1 - \beta_1 Z^{-1} - \beta_2 Z^{-2}$$

Then

$$\omega/(1-\Delta_1-\Delta_2+\Delta_1\Delta_2)\geq 0$$
$$\beta 1^2+4\beta_2\geq 0$$
$$\Delta_1>0$$
$$\sum_{j=0}^{q-1}\alpha_{j+1}\Delta^{-j}>0$$

and,

$$\phi_k\geq 0, k=0,\cdots,q$$

where

$$\begin{aligned}
\phi_0 &= \alpha_1\\
\phi_1 &= \beta_1\phi_0+\alpha_2\\
\phi_2 &= \beta_1\phi_1+\beta_2\phi_0+\alpha_3\\
&\vdots\\
\phi_q &= \beta_1\phi_{q-1}+\beta_2\phi_{q-2}
\end{aligned}$$

GARCH(p,q). General constraints for $p>2$ haven't been worked out. For such models, **FANPAC** directly constrains the conditional variances to be greater than zero. It also constrains the roots of the polynomial

$$1-\beta_1 Z-\beta_2 Z^2-\cdots-\beta_p Z^p$$

to be outside the unit circle. This only guarantees that the conditional variances will be nonnegative in the sample, and does not guarantee that the conditional variances will be nonnegative for all realizations of the data.

Stationarity

To ensure that the GARCH process is covariance stationary, the roots of

$$1-(\alpha_1+\beta_1)Z-(\alpha_2+\beta_2)Z^2-\cdots$$

may be constrained to be outside the unit circle (Gouriéroux, 1997, page 37).

Most GARCH models reported in the economics literature are estimated using software that cannot impose nonlinear constraints on parameters and thus either impose a more highly restrictive set of linear constraints than the ones described here, or impose no constraints at all. The procedures provided in **FANPAC** ensure that you have the best fitting solution that satisfies the conditions of stationarity and nonnegative of conditional variances.

Stationarity in the GARCH-in-cv model is conditional on the exogenous variables included in the conditional variance equation. There is no assurance of unconditional stationarity without further constraints or assumptions with respect to the exogenous variables.

Initialization

The calculation of the log-likelihood is recursive and requires initial values for the conditional variance. Following standard practice, the first q values of the conditional variances are fixed to the sample unconditional variance of the series.

2.3.3 IGARCH

The IGARCH(p,q) model is a GARCH(p,q) model with a unit root. This is accomplished in **FANPAC** by adding the equality constraint

$$\sum_i \alpha_1 + \sum_i \beta_i = 1$$

2.3.4 FIGARCH

Define the time series

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed time series, x_t an observed time series of exogenous variables including a column of ones, and β a vector of coefficients.

Further define

$$\epsilon_t \equiv \eta_t\sigma_t$$

where $E(\eta_t) = 0$, $Var(\eta_t) = 1$.

Let

$$A(L) = \alpha_1 L + \alpha_2 L^2 + \cdots + \alpha_q L^q$$

and

$$A(L) = \beta_1 L + \beta_2 L^2 + \cdots + \beta_p L^p$$

where L is the lag operator. In this notation, the GARCH(p,q) model can be specified

$$\sigma_t^2 = \omega + A(L)\epsilon_t^2 + B(L)\sigma_t^2$$

The GARCH(p,q) model can be re-specified as an ARMA(max(p,q),p) model (Bailie, et al., 1996)

$$[1 - A(L) - B(L)]\epsilon_t^2 = \omega + [1 - B(L)]\nu_t$$

where $\nu_t \equiv \epsilon_t^2 - \sigma_t^2$ is the "innovation" at time t for the conditional variance process.

Using this notation, the IGARCH(p,q) model is

$$\theta(L)(1 - L)\epsilon_t^2 = \omega + [1 - B(L)]\nu_t$$

where $\theta(L) = [1 - A(L) - B(L)](1 - L)^{-1}$. The *fractionally integrated* GARCH or FIGARCH(p,q) model is

$$\theta(L)(1 - L)^d\epsilon_t^2 = \omega + [1 - B(L)]\nu_t$$

where $0 < d < 1$.

FIGARCH-in-cv

For the FIGARCH-in-cv model, independent variables may be added to the conditional variance equation

$$\sigma_t^2 = \omega + A(L)\epsilon_t^2 + B(L)\sigma_t^2 + Z_t\Gamma$$

where Z_t is the t-th vector of observed independent variables and Γ a matrix of coefficients.

GARCH-in-mean

For the GARCH-in-mean (or GARCHM) model, the time series mean equation is modified to include the conditional variance

$$\epsilon_t = y_t - x_t\beta - \delta\sigma_t$$

log-likelihood

The conditional variance in the FIGARCH(p,q) model is the sum of an infinite series of prior conditional variances:

$$\begin{aligned}\sigma_t^2 &= \omega + [1 - B(L) - \theta(L)(1-L)^d]\epsilon_t^2 + B(L)\sigma_t^2 \\ &= \omega + [1 - B(L) - [1 - A(L) - B(L)](1-L)^{d-1}]\epsilon_t^2 + B(L)\sigma_t^2 \\ &= \omega + (\phi_1 L - \phi_2 L^2 - \cdots)\epsilon_t^2 + B(L)\sigma_t^2\end{aligned}$$

$$\phi_k = \alpha_k - \pi_k + \sum_{i=1}^{k-1}\pi_i(\alpha_{k-i} + \beta_{k-i})$$

where $\alpha_j = 0, j > q$ and $\beta_j = 0, j > p$, and

$$\pi_k = \frac{1}{k!}\prod_{i=1}^{k}(i-d)$$

In practice, the log-likelihood will be computed from available data and this means that the calculation of the conditional variance will be truncated. To minimize this error, the log-probabilities for initial observations can be excluded from the log-likelihood. The default is one half of the observations.

This can be modified by calling the keyword command **setLagTruncation** with an argument specifying the number of observations to be *included* in the log-likelihood, or by directly setting the **FANPAC** global, **_fan_init** to the number of initial observations to be *excluded* from the log-likelihood.

The log-likelihood for $\eta_t \sim N(0,1)$ is

$$logL = -\frac{T-\rho}{2}log(2\pi) - \sum_{t+\mu}^{T} log(\sigma_t) - \frac{1}{2}\sum_{t+\mu}^{T}\frac{\epsilon_t^2}{\sigma_t^2}$$

and for $\eta_t \sim t(0,1,\nu)$ is

$$\begin{aligned} logL \quad = \quad & -\tfrac{T-q}{2}log\left(\tfrac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)(\nu-2)^{1/2}\pi^{1/2}\sigma}\right) - \textstyle\sum_{t=q}^{T} log(\sigma_t) \\ & -\tfrac{\nu+1}{2}\textstyle\sum_{t=q}^{T} log\left(1+\tfrac{\epsilon_t^2}{(\nu-2)\sigma_t^2}\right) \end{aligned}$$

where $\mu =$ **_fan_init**, the number of lags used to initialize the process.

Stationarity

The unconditional variance of FIGARCH models is infinite, and thus is not covariance stationary. However, Baillie, et al. (1996) point out that FIGARCH models are ergodic and strictly stationary for $0 \leq d \leq 1$ using a direct extension of proofs for the IGARCH case (Nelson, 1990).

In addition to the constraint on d, it is also necessary to constrain the roots of

$$1 - \beta_1 Z - \beta_2 Z^2 - \cdots - \beta_p Z^p$$

to be outside the unit circle.

Nonnegative conditional variances

General methods to ensure the nonnegativity of the conditional variances haven't been established. However, in **FANPAC** the conditional variances are directly constrained to be nonnegative. This guarantees nonnegative conditional variances in the sample, but does not do so for all realizations of the time series.

Stationarity in the FIGARCH-in-cv model is conditional on the exogenous variables included in the conditional variance equation. There is no assurance of unconditional stationarity without further constraints or assumptions with respect to the exogenous variables.

2.3.5 EGARCH

Define the time series

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed time series, x_t an observed time series of exogenous variables including a column of ones, and β a vector of coefficients.

Also define

$$\epsilon_t \equiv \eta_t\sigma_t$$

where $E(\eta_t) = 0$, $Var(\eta_t) = 1$. For the EGARCH (or Exponential GARCH) model, the conditional variance is modelled

$$\begin{aligned} log\,\sigma_t^2 \quad = \quad & \omega + \beta_1 log\,\sigma_{t-1}^2 + \cdots + \beta_p log\,\sigma_{t-p}^2 \\ & \alpha_1(|\epsilon_{t-1}| + E|\epsilon_{t-1}| + \delta\epsilon_{t-1}) \\ & \alpha_2(|\epsilon_{t-2}| + E|\epsilon_{t-2}| + \delta\epsilon_{t-2}) + \cdots + \\ & \alpha_q(|\epsilon_{t-q}| + E|\epsilon_{t-q}| + \delta\epsilon_{t-q}) \end{aligned}$$

In this model, ϵ_t has the generalized error distribution

$$f(\epsilon_t) = \frac{\rho\Gamma(3/\rho)^{\frac{1}{2}}}{2\sigma_t^2\Gamma(1/\rho)^{\frac{3}{2}}} e^{-\frac{1}{2}|\frac{\epsilon_t}{\lambda\sigma_t}|^\rho}$$

where $\rho > 0$ is a parameter measuring the thickness of the tails, δ is a *leverage* parameter,

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

and

$$E|\epsilon_t| = \Gamma(2/\rho)^{\frac{1}{2}}\Gamma(1/\rho)^{-\frac{1}{2}}\Gamma(3/\rho)^{-\frac{1}{2}}$$

log-likelihood

The log-likelihood for the EGARCH model is

$$logL = log(\frac{\rho}{2}) + \frac{1}{2}log\Gamma(3/\rho) - \frac{3}{2}log\Gamma(1/\rho) - \frac{1}{2}\sum_{t+\mu}^{T}\left|\frac{\epsilon^2}{\lambda\sigma_t}\right|^\rho - \sum_{t+\mu}^{T}\sigma_t^2$$

where $\mu = max(p, q) + 1$.

FANPAC

2.3.6 ARIMA

Let

$$\Phi(L) = \phi_1 L + \phi_2 L^2 + \cdots + \phi_p L^p$$

and

$$\Theta(L) = \theta_1 L + \theta_2 L^2 + \cdots + \theta_p L^q$$

then the ARIMA model can be described in lag operator form

$$\Phi(L)[(1-L)^d y_t - x_t \beta] = \Theta(L)\epsilon_t$$

where $t = 1, 2, ...T$, and y_t an observed time series, x_t an observed time series of exogenous variables including a column of ones, and β a vector of coefficients.

2.3.7 OLS

Define the series

$$\epsilon_t = y_t - x_t \beta$$

where $t = 1, 2, ...T$, and y_t an observed time series, x_t an observed time series of exogenous variables including a column of ones, and β a vector of coefficients.

For $\epsilon_t \sim N(0, 1)$, the ordinary least squares estimator

$$\hat{\beta} = (X'X)^{-1} X'Y$$

where x'_t and y_t are the i-th rows of X and Y, respectively, is maximum likelihood.

For ϵ_t with a t distribution with ν degrees of freedom and variance σ^2, the log-likelihood is

$$logL = -\frac{T}{2} log\left(\frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)(\nu-2)^{1/2}\pi^{1/2}\sigma}\right) - \sum_{t=q}^{T} log(\sigma) - \frac{\nu+1}{2}\sum_{t=q}^{T} log\left(1 + \frac{\epsilon_t^2}{(\nu-2)\sigma^2}\right)$$

It is also necessary to constrain ν to be greater than 2.

2.4 Multivariate Time Series Models

2.4.1 DVEC ARCH

Define a vector of ℓ residuals

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed multiple time series, x_t an observed time series of exogenous variables including a column of ones, and β a matrix of coefficients.

Let Σ_t be the conditional variance-covariance matrix of ϵ_t. Each nonredundant element of Σ_t is a separate GARCH model:

$$\Sigma_{t,ij} = \Omega_{ij} + A_{1,ij}\epsilon_{i,t-1}\epsilon_{j,t-1} + \cdots + A_{q,ij}\epsilon_{i,t-q}\epsilon_{j,t-q}$$

where $\Omega_{ij} = \Omega_{ji}$ and $A_{k,ij} = A_{k,ji}$.

DVEC ARCH-in-cv

For the DVEC ARCH-in-cv (or DVARCHV) model, independent variables are added to the equation for the conditional variance

$$\Sigma_{t,ij} = \Omega_{ij} + A_{1,ij}\epsilon_{i,t-1}\epsilon_{j,t-1} + \cdots + A_{q,ij}\epsilon_{i,t-q}\epsilon_{j,t-q} + Z_t\Gamma_{ij}$$

where Z_t is the t-th vector of observed independent variables and Γ_{ij} a matrix of coefficients.

DVEC ARCH-in-mean

For the DVEC ARCH-in-mean (or DVARCHM model), the time series equation is modified to include the conditional variance

$$\epsilon_{i,t} = y_{i,t} - x_t\beta_i' - \delta_i\Sigma_{t,ii})$$

where β_i is the i-th row of B, an $\ell \times k$ coefficient matrix.

FANPAC

log-likelihood

The log-likelihood conditional on $\mu = max(p,q)$ initial estimates of the conditional variance-covariances matrices is

$$logL = -\frac{k(T-\mu)}{2}log(2\pi) - \frac{1}{2}\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid - \frac{1}{2}\sum_{t+\mu+1}^{T}(\epsilon_t' \Sigma_t^{-1} \epsilon_t)$$

where

$$\Sigma_1 = \Sigma_2 = \cdots = \Sigma_\mu = \frac{1}{T}\sum_{t=1}^{T}\epsilon_t' \epsilon_t$$

The conditional log-likelihood for a t-distributed ϵ is

$$\begin{aligned} logL \quad = \quad & -\tfrac{T-\mu}{2}(log\Gamma((\nu+k)/2) - log\Gamma(\nu/2) - \tfrac{1}{2}log((\nu-2)\pi)) \\ & -\tfrac{1}{2}\textstyle\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid - \tfrac{\nu+k}{2}\textstyle\sum_{t+\mu+1}^{T} log(1+\epsilon_t'\Sigma_t^{-1}\epsilon_t/(\nu-2)) \end{aligned}$$

Positive Definiteness of Conditional Variances

Constraints on the parameters are necessary to enforce the positive definiteness of the conditional variance-covariances matrices. This requirement is assured by directly constraining the eigenvalues of the conditional variance-covariance matrices to be greater than zero.

Stationarity

Stationarity is assured if the roots of the determinantal equation

$$\mid I - A_1 z - A_2 z^2 - ... \mid$$

lie outside the unit circle (Gourieroux, 1997).

Stationarity in the DVEC ARCH-in-cv model is conditional on the exogenous variables included in the conditional variance equation. There is no assurance of unconditional stationarity without further constraints or assumptions with respect to the exogenous variables.

2.4.2 Constant Correlation DVEC ARCH Model

Define a vector of ℓ residuals

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed multiple time series, x_t an observed time series of exogenous variables including a column of ones, and β a matrix of coefficients.

Let Σ_t be the conditional variance-covariance matrix of ϵ_t with constant correlation matrix R. Then each diagonal element of Σ_t is modelled as a separate GARCH model

$$\Sigma_{t,ii} = \Omega_i + A_{i,1}\epsilon^2_{i,t-1} + \cdots + A_{i,q}\epsilon^2_{i,t-q}$$

The elements of the conditional variance-covariance matrix, then, are

$$\Sigma_{t,ij} = R_{ij}\sqrt{\Sigma_{t,ii}\Sigma_{t,jj}}$$

constant correlation DVEC ARCH-in-cv

For the constant correlation DVEC ARCH-in-cv (or CDVARCHV) model, independent variables are added to the equation for the conditional variance

$$\Sigma_{t,ii} = \Omega_i + A_{i,1}\epsilon^2_{i,t-1} + \cdots + A_{i,q}\epsilon^2_{i,t-q} + Z_t\Gamma_i$$

where Z_t is the t-th vector of observed independent variables and Γ_i a matrix of coefficients.

constant correlation DVEC ARCH-in-mean

For the constant correlation DVEC ARCH-in-mean (or CDVARCHM) model, the time series equation is modified to include the conditional variance

$$\epsilon_{i,t} = y_{i,t} - x_{i,t}\beta_i' - \delta_i\Sigma_{t,ii}$$

where β_i is the i-th row of B, an $\ell \times k$ coefficient matrix.

log-likelihood

The log-likelihood conditional on $\mu = max(p, q)$ initial estimates of the conditional variance-covariances matrices is

$$logL = -\frac{k(T-\mu)}{2}log(2\pi) - \frac{1}{2}\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid -\frac{1}{2}\sum_{t+\mu+1}^{T}(\epsilon_t'\Sigma_t^{-1}\epsilon_t)$$

FANPAC

where

$$\Sigma_1 = \Sigma_2 = \cdots = \Sigma_\mu = \frac{1}{T}\sum_{t=1}^{T} \epsilon_t' \epsilon_t$$

and where

$$\sigma_{t,ii})^{-1} \sigma_{t,ij} \Sigma_{t,jj})^{-1} = r_{ij}$$

where the r_{ij} are constants.

The conditional log-likelihood for a t-distributed ϵ is

$$\begin{aligned} logL &= -\tfrac{T-\mu}{2}(log\Gamma((\nu+k)/2) - log\Gamma(\nu/2) - \tfrac{1}{2}log((\nu-2)\pi)) \\ &\quad -\tfrac{1}{2}\textstyle\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid - \tfrac{\nu+k}{2}\textstyle\sum_{t+\mu+1}^{T} log(1 + \epsilon_t' \Sigma_t^{-1} \epsilon_t / (\nu - 2)) \end{aligned}$$

Positive Definiteness of Conditional Variances

Constraints on the parameters are necessary to enforce the positive definiteness of the conditional variance-covariances matrices. This requirement is assured by directly constraining the eigenvalues of the conditional variance-covariance matrices to be greater than zero.

Stationarity

Stationarity is assured if the roots of the determinantal equation (Gourieroux, 1997)

$$\mid I - A_1 z - A_2 z^2 - \cdots \mid$$

lie outside the unit circle. Since the A_i and B_i are diagonal matrices, this amounts to determining the roots of k polynomials

$$1 - A_{111} z - A_{211} z^2 - \cdots \tag{2.1}$$
$$1 - A_{122} z - A_{222} z^2 - \cdots \tag{2.2}$$
$$\vdots \tag{2.3}$$
$$1 - A_{1kk} z - A_{2kk} z^2 - \cdots \tag{2.4}$$

Stationarity in the constant correlation DVEC ARCH-in-cv model is conditional on the exogenous variables included in the conditional variance equation. There is no assurance of unconditional stationarity without further constraints or assumptions with respect to the exogenous variables.

2.4.3 BEKK ARCH

Define a vector of ℓ residuals

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed multiple time series, x_t an observed time series of exogenous variables including a column of ones, and β a matrix of coefficients.

Further define the conditional variance Σ_t of ϵ_t

$$\Sigma_t = \Omega + A_1\epsilon'_{t-1}\epsilon_{t-1}A'_1 + \cdots + A_q\epsilon'_{t-q}\epsilon_{t-q}A'_q$$

where Ω is a symmetric matrix and the A_i are square matrices.

log-likelihood

The log-likelihood conditional on $\mu = max(p, q)$ initial estimates of the conditional variance-covariances matrices is

$$logL = -\frac{k(T-\mu)}{2}log(2\pi) - \frac{1}{2}\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid -\frac{1}{2}\sum_{t+\mu+1}^{T}(\epsilon'_t\Sigma_t^{-1}\epsilon_t)$$

where

$$\Sigma_1 = \Sigma_2 = \cdots = \Sigma_\mu = \frac{1}{T}\sum_{t=1}^{T}\epsilon'_t\epsilon_t$$

The conditional log-likelihood for a t-distributed ϵ is

$$\begin{aligned} logL \;\; = \;\; & -\tfrac{T-\mu}{2}(log\Gamma((\nu+k)/2) - log\Gamma(\nu/2) - \tfrac{1}{2}log((\nu-2)\pi)) \\ & -\tfrac{1}{2}\textstyle\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid -\tfrac{\nu+k}{2}\textstyle\sum_{t+\mu+1}^{T} log(1+\epsilon'_t\Sigma_t^{-1}\epsilon_t/(\nu-2)) \end{aligned}$$

Positive Definiteness of Conditional Variances

Constraints on the parameters are necessary to enforce the positive definiteness of the conditional variance-covariances matrices. This requirement is assured by directly constraining the eigenvalues of the conditional variance-covariance matrices to be greater than zero.

2.4.4 DVEC GARCH

Define a vector of ℓ residuals

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed multiple time series, x_t an observed time series of exogenous variables including a column of ones, and β a matrix of coefficients.

Let Σ_t be the conditional variance-covariance matrix of ϵ_t. Each nonredundant element of Σ_t is a separate GARCH model

$$\begin{aligned}\Sigma_{t,ij} &= \Omega_{ij} + A_{1,ij}\epsilon_{i,t-1}\epsilon_{j,t-1} + \cdots + A_{q,ij}\epsilon_{i,t-q}\epsilon_{j,t-q} \\ & B_{1,ij}\Sigma_{t,ij-1} + \cdots + B_{p,ij}\Sigma_{t,ij-p}\end{aligned}$$

where $\Omega_{ij} = \Omega_{ji}$ and $A_{k,ij} = A_{k,ji}$.

DVEC GARCH-in-cv

For the DVEC GARCH-in-cv (or DVGARCHV) model, independent variables are added to the equation for the conditional variance

$$\begin{aligned}\Sigma_{t,ij} &= \Omega_{ij} + A_{1,ij}\epsilon_{i,t-1}\epsilon_{j,t-1} + \cdots + A_{q,ij}\epsilon_{i,t-q}\epsilon_{j,t-q} \\ & +B_{1,ij}\Sigma_{t,ij-1} + \cdots + B_{p,ij}\Sigma_{t,ij-p} + Z_t\Gamma_{ij}\end{aligned}$$

where Z_t is the t-th vector of observed independent variables and Γ_{ij} a matrix of coefficients.

DVEC GARCH-in-mean

For the DVEC GARCH-in-mean (or DVGARCHM) model, the time series equation is modified to include the conditional variance

$$\epsilon_{i,t} = y_{i,t} - x_t\beta_i' - \delta_i\Sigma_{t,ii}$$

where β_i is the i-th row of B, an $\ell \times k$ coefficient matrix.

log-likelihood

The log-likelihood conditional on $\mu = max(p, q)$ initial estimates of the conditional variance-covariances matrices is

$$logL = -\frac{k(T-\mu)}{2}log(2\pi) - \frac{1}{2}\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid -\frac{1}{2}\sum_{t+\mu+1}^{T}(\epsilon_t'\Sigma_t^{-1}\epsilon_t)$$

where

$$\Sigma_1 = \Sigma_2 = \cdots = \Sigma_\mu = \frac{1}{T}\sum_{t=1}^{T}\epsilon_t'\epsilon_t$$

The conditional log-likelihood for a t-distributed ϵ is

$$\begin{aligned}logL &= -\tfrac{T-\mu}{2}(log\Gamma((\nu+k)/2) - log\Gamma(\nu/2) - \tfrac{1}{2}log((\nu-2)\pi)) \\ & -\tfrac{1}{2}\textstyle\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid -\tfrac{\nu+k}{2}\sum_{t+\mu+1}^{T} log(1 + \epsilon_t'\Sigma_t^{-1}\epsilon_t/(\nu-2))\end{aligned}$$

Positive Definiteness of Conditional Variances

Constraints on the parameters are necessary to enforce the positive definiteness of the conditional variance-covariances matrices. This requirement is assured by directly constraining the eigenvalues of the conditional variance-covariance matrices to be greater than zero.

Stationarity

Stationarity is assured if the roots of the determinantal equation

$$| I - (A_1 + B_1)z - (A_2 + B_2)z^2 - ... |$$

lie outside the unit circle (Gourieroux, 1997).

Stationarity in the DVEC GARCH-in-cv model is conditional on the exogenous variables included in the conditional variance equation. There is no assurance of unconditional stationarity without further constraints or assumptions with respect to the exogenous variables.

2.4.5 Constant Correlation DVEC GARCH Model

Define a vector of ℓ residuals

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed multiple time series, x_t an observed time series of exogenous variables including a column of ones, and β a matrix of coefficients.

Let Σ_t be the conditional variance-covariance matrix of ϵ_t with constant correlation matrix R. Then each diagonal element of Σ_t is modelled as a separate GARCH model

$$\Sigma_{t,ii} = \Omega_i + A_{i,1}\epsilon^2_{i,t-1} + \cdots + A_{i,q}\epsilon^2_{i,t-q} + B_{i,1}\Sigma_{i,t-1} + \cdots + B_{i,p}\Sigma_{i,t-p}$$

The elements of the conditional variance-covariance matrix, then, are

$$\Sigma_{t,ij} = R_{ij}\sqrt{\Sigma_{t,ii}\Sigma_{t,jj}}$$

constant correlation DVEC GARCH-in-cv

For the constant correlation DVEC GARCH-in-cv (or CDVGARCHV) model, independent variables are added to the equation for the conditional variance

$$\Sigma_{t,ii} = \Omega_i + A_{i,1}\epsilon^2_{i,t-1} + \cdots + A_{i,q}\epsilon^2_{i,t-q} + B_{i,1}\Sigma_{i,t-1} + \cdots + B_{i,p}\Sigma_{i,t-p} + Z_t\Gamma_{ij}$$

where Z_t is the t-th vector of observed independent variables and Γ_{ij} a matrix of coefficients.

constant correlation DVEC GARCH-in-mean

For the constant correlation DVEC GARCH-in-mean (or DVGARCHM) model, the time series equation is modified to include the conditional variance

$$\epsilon_{i,t} = y_{i,t} - x_t \beta_i' - \delta_i \Sigma_{t,ii}$$

where β_i is the i-th row of B, an $\ell \times k$ coefficient matrix.

log-likelihood

The log-likelihood conditional on $\mu = max(p,q)$ initial estimates of the conditional variance-covariances matrices is

$$logL = -\frac{k(T-\mu)}{2} log(2\pi) - \frac{1}{2} \sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid - \frac{1}{2} \sum_{t+\mu+1}^{T} (\epsilon_t' \Sigma_t^{-1} \epsilon_t)$$

where

$$\Sigma_1 = \Sigma_2 = \cdots = \Sigma_\mu = \frac{1}{T} \sum_{t=1}^{T} \epsilon_t' \epsilon_t$$

and where

$$\sigma_{t,ij} = r_{ij} \sqrt{\sigma_{t,ii} \Sigma_{t,jj}}$$

where r_{ij} is a constant parameter to be estimated.

The conditional log-likelihood for a t-distributed ϵ is

$$\begin{aligned} logL \;=\; & -\tfrac{T-\mu}{2}(log\Gamma((\nu+k)/2) - log\Gamma(\nu/2) - \tfrac{1}{2} log((\nu-2)\pi)) \\ & -\tfrac{1}{2} \textstyle\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid - \tfrac{\nu+k}{2} \textstyle\sum_{t+\mu+1}^{T} log(1 + \epsilon_t' \Sigma_t^{-1} \epsilon_t / (\nu - 2)) \end{aligned}$$

Positive Definiteness of Conditional Variances

Constraints on the parameters are necessary to enforce the positive definiteness of the conditional variance-covariances matrices. This requirement is assured by directly constraining the eigenvalues of the conditional variance-covariance matrices to be greater than zero.

Stationarity

Stationarity is assured if the roots of the determinantal equation (Gourieroux, 1997)

$$| I - (A_1 + B_1)z - (A_2 + B_2)z^2 - \cdots |$$

lie outside the unit circle. Since the A_i and B_i are diagonal matrices, this amounts to determining the roots of k polynomials

$$1 - (A_{111} + B_{111})z - (A_{211} + B_{211})z^2 - \cdots \quad (2.5)$$
$$1 - (A_{122} + B_{122})z - (A_{222} + B_{222})z^2 - \cdots \quad (2.6)$$
$$\vdots \quad (2.7)$$
$$1 - (A_{1kk} + B_{1kk})z - (A_{2kk} + B_{2kk})z^2 - \cdots \quad (2.8)$$

Stationarity in the constant correlation DVEC GARCH-in-cv model is conditional on the exogenous variables included in the conditional variance equation. There is no assurance of unconditional stationarity without further constraints or assumptions with respect to the exogenous variables.

2.4.6 BEKK GARCH

Define a vector of ℓ residuals

$$\epsilon_t = y_t - x_t\beta$$

where $t = 1, 2, ...T$, and y_t an observed multiple time series, x_t an observed time series of exogenous variables including a column of ones, and β a matrix of coefficients.

Further define the conditional variance Σ_t of ϵ_t

$$\begin{aligned}\Sigma_t \quad = \quad & \Omega + A_1\epsilon'_{t-1}\epsilon_{t-1}A'_1 + \cdots + A_q\epsilon'_{t-q}\epsilon_{t-q}A'_q + \\ & B_1\Sigma_{t-1}B'_1 + \cdots + B_p\Sigma_{t-p}B'_p\end{aligned}$$

where Ω is a symmetric matrix, and A_i and B_i are square matrices.

log-likelihood

The log-likelihood conditional on $\mu = max(p, q)$ initial estimates of the conditional variance-covariances matrices is

$$logL = -\frac{k(T-\mu)}{2}log(2\pi) - \frac{1}{2}\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid - \frac{1}{2}\sum_{t+\mu+1}^{T} (\epsilon'_t\Sigma_t^{-1}\epsilon_t)$$

where

$$\Sigma_1 = \Sigma_2 = \cdots = \Sigma_\mu = \frac{1}{T}\sum_{t=1}^{T} \epsilon_t' \epsilon_t$$

The conditional log-likelihood for a t-distributed ϵ is

$$\begin{aligned} logL \quad = \quad & -\tfrac{T-\mu}{2}(log\Gamma((\nu+k)/2) - log\Gamma(\nu/2) - \tfrac{1}{2}log((\nu-2)\pi)) \\ & -\tfrac{1}{2}\textstyle\sum_{t+\mu+1}^{T} log \mid \Sigma_t \mid - \tfrac{\nu+k}{2}\textstyle\sum_{t+\mu+1}^{T} log(1+\epsilon_t'\Sigma_t^{-1}\epsilon_t/(\nu-2)) \end{aligned}$$

Positive Definiteness of Conditional Variances

Constraints on the parameters are necessary to enforce the positive definiteness of the conditional variance-covariances matrices. This requirement is assured by directly constraining the eigenvalues of the conditional variance-covariance matrices to be greater than zero.

2.5 Inference

The parameters of time series models in general are highly constrained. This presents severe difficulties for statistical inference. The usual method for statistical inference, comprising the calculation of the covariance matrix of the parameters and constructing t-statistics from the standard errors of the parameters, fails in the context of inequality constrained parameters because confidence regions will not generally be symmetric about the estimates. For this reason **FANPAC** does not compute t-statistics, but rather computes and reports confidence limits.

The most common type of inference is based on the Wald statistic. A $(1-\alpha)$ joint Wald-type confidence region for θ is the hyper-ellipsoid

$$JF(J, N-K;\alpha) = (\theta-\hat{\theta})'V^{-1}(\theta-\hat{\theta}), \tag{2.9}$$

where V is the covariance matrix of the parameters. The confidence limits are the maximum and minimum solution of

$$\min\left\{\eta_k'\theta \mid (\theta-\hat{\theta})'V^{-1}(\theta-\hat{\theta}) \geq JF(J, N-K;\alpha))\right\}, \tag{2.10}$$

where η can be an arbitrary vector of constants and $J = \sum \eta_k \neq 0$.

When there are no constraints, the solution to this problem for a given parameter is the well known

$$\hat{\theta} \pm t_{(1-\alpha)/2, T-k}\sigma_{\hat{\theta}}$$

where $\sigma_{\hat{\theta}}$ is the square root of the diagonal element of V associated with $\hat{\theta}$.

When there are constraints in the model, two things happen that render the classical method invalid. First, the solution to (2.10) is no longer (2.5) and second, (2.9) is not valid whenever the hyper-ellipsoid is on or near a constraint boundary.

(2.9) is based on an approximation to the likelihood ratio statistic. This approximation fails in the region of constraint boundaries because the likelihood ratio statistic itself is known to be distributed there as a *mixture* of chi-squares (Gouriéroux, et al.; 1982, Wolak, 1991). In finite samples these effects occur in the *region* of the constraint boundary, specifically when the true value is within $\epsilon = \sqrt{(\sigma_e^2/N)\chi^2_{(1-\alpha,k)}}$ of the constraint boundary.

Here, and in **FANPAC**, we consider only the solution for a given parameter, a "parameter of interest;" all other parameters are "nuisance parameters." There are three cases to consider:

(1) parameter constrained, no nuisance parameters constrained;

(2) parameter unconstrained, one or more nuisance parameters constrained;

(3) parameter constrained, one or more nuisance parameters constrained.

Case 1: When the true value is on the boundary, the statistics are distributed as a simple mixture of two chi-squares. Monte Carlo evidence presented Schoenberg (1997) shows that this holds as well in finite samples for true values within ϵ of the constraint boundary.

Case 2: The statistics are distributed as weighted mixtures of chi-squares when the correlation of the constrained nuisance parameter with the unconstrained parameter of interest is greater than about .8. A correction for these effects is feasible. However, for finite samples, the effects on the statistics due to a true value of a constrained nuisance parameter being within ϵ of the boundary are greater and more complicated than the effects of actually being on the constraint boundary. There is no systematic strategy available for correcting for these effects.

Case 3: The references disagree. Gouriéroux, et al., (1982) and Wolak (1991) state that the statistics are distributed as a mixture of chi-squares. However, Self and Liang (1987) argue that when the distributions of the parameter of interest and the nuisance parameter are correlated, the distributions of the statistics are not chi-square mixtures.

There is no known solution for these problems with the type of confidence limits discussed here. Bayesian limits produce correct limits (Geweke, 1995), but they are considerably more computationally intensive. With the correction described in Schoenberg (1997), however, confidence limits computed via the inversion of the Wald statistic will be correct provided that no nuisance parameter within ϵ of a constraint boundary is correlated with the parameter of interest by more than about .6.

2.5.1 Confidence Limits

FANPAC computes, by default, confidence limits computed in the standard way from t-statistics. These limits suffer from the deficiencies reported in the previous section – they are symmetric about the estimate, which is not usually the case for constrained parameters, and they can include undefined regions of the parameter space.

By request, **FANPAC** computes confidence limits by inversion of the Wald statistic. This includes a correction for the chi-squared statistic when the limit falls within $\epsilon = \sqrt{(\sigma_e^2/N)\chi^2_{(1-\alpha,k)}}$ of a constraint boundary. These will be correct confidence limits provided there is no nuisance parameter within ϵ of a constraint boundary correlated with the parameter of interest by more than about .6.

To get confidence limits by inversion of the Wald statistic, call the keyword command

```
setInferenceType WALD
```

2.5.2 Covariance Matrix of Parameters

FANPAC computes a covariance matrix of the parameters that is an approximate estimate when there are constrained parameters in the model (Gallant, 1987, Wolfgang and Hartwig, 1995). When the model includes inequality constraints, the covariance matrix computed directly from the Hessian, the usual method for computing this covariance matrix, is incorrect because they do not account for boundaries placed on the distributions of the parameters by the inequality constraints.

An argument based on a Taylor-series approximation to the likelihood function (e.g., Amemiya, 1985, page 111) shows that

$$\hat{\theta} \to N(\theta, A^{-1}BA^{-1}),$$

where

$$\begin{aligned} A &= E\left[\frac{\partial^2 L}{\partial\theta\partial\theta'}\right], \\ B &= E\left[\left(\frac{\partial L}{\partial\theta}\right)'\left(\frac{\partial L}{\partial\theta}\right)\right]. \end{aligned}$$

Estimates of A and B are

$$\begin{aligned} \hat{A} &= \frac{1}{N}\sum_i^N \frac{\partial^2 L_i}{\partial\theta\partial\theta'}, \\ \hat{B} &= \frac{1}{N}\sum_i^N \left(\frac{\partial L_i}{\partial\theta}\right)'\left(\frac{\partial L_i}{\partial\theta}\right). \end{aligned}$$

Assuming the correct specification of the model plim(A) = plim(B),

$$\hat{\theta} \to N(\theta, \hat{A}^{-1}).$$

Without loss of generality we may consider two types of constraints: the nonlinear equality, and the nonlinear inequality constraints (the linear constraints are included in nonlinear, and the bounds are regarded as a type of linear inequality). Furthermore, the inequality constraints may be treated as equality constraints with the introduction of "slack" parameters into the model:

$$H(\theta) \geq 0$$

is changed to

$$H(\theta) = \zeta^2,$$

where ζ is a conformable vector of slack parameters.

Further, we distinguish *active* from *inactive* inequality constraints. Active inequality constraints have nonzero Lagrangeans, γ_j, and zero slack parameters, ζ_j, while the reverse is true for inactive inequality constraints. Keeping this in mind, define the diagonal matrix, Z, containing the slack parameters, ζ_j, for the inactive constraints, and another diagonal matrix, Γ, containing the Lagrangean coefficients. Also, define $H_\oplus(\theta)$ representing the active constraints, and $H_\ominus(\theta)$ the inactive.

The likelihood function augmented by constraints is then

$$\begin{aligned} L_A &= L + \lambda_1 g(\theta)_1 + \cdots + \lambda_I g(\theta)^I + \gamma_1 h_{\oplus 1}(\theta) + \cdots + \gamma_J h_{\oplus J}(\theta) \\ &\quad + h_{\ominus 1}(\theta)_i - \zeta_1^2 + \cdots + h_{\ominus K}(\theta) - \zeta_K^2, \end{aligned}$$

and the Hessian of the augmented likelihood is

$$E\left(\frac{\partial^2 L_A}{\partial\theta\partial\theta'}\right) = \begin{bmatrix} \Sigma & 0 & 0 & \dot{G}' & \dot{H}'_\oplus & \dot{H}'_\ominus \\ 0 & 2\Gamma & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2Z \\ \dot{G} & 0 & 0 & 0 & 0 & 0 \\ \dot{H}_\oplus & 0 & 0 & 0 & 0 & 0 \\ \dot{H}_\ominus & 0 & 2Z & 0 & 0 & 0 \end{bmatrix},$$

where the dot represents the Jacobian with respect to θ, $L = \sum_{i=1}^N \log P(Y_i;\theta)$, and $\Sigma = \partial^2 L/\partial\theta\partial\theta'$. The covariance matrix of the parameters, Lagrangeans, and slack parameters is the Moore-Penrose inverse of this matrix.

Construct the partitioned array

$$\tilde{B} == \begin{bmatrix} \dot{G} \\ \dot{H}_\oplus \\ \dot{H}_\ominus \end{bmatrix}.$$

Let Ξ be the orthonormal basis for the null space of $\tilde{B}$, then the covariance matrix of the parameters is

$$\Xi(\Xi'\Sigma\Xi)^{-1}\Xi'.$$

Rows of this matrix associated with active inequality constraints may not be available, i.e., the rows and columns of Ω associated with those parameters may be all zeros.

FANPAC

2.5.3 Quasi-Maximum Likelihood Covariance Matrix of Parameters

FANPAC computes a QML covariance matrix of the parameters when requested. Define $B = (\partial L_A/\partial\theta)'(\partial L_A/\partial\theta)$ evaluated at the estimates. Then the covariance matrix of the parameters is $\Omega B \Omega$.

To request the QML covariance matrix, call the keyword command

```
setCovParType QML
```

The default ML covariance matrix can be set by

```
setCovParType ML
```

2.5.4 Ill-Conditioning and Singularity

Occasionally FANPAC fails to produce the covariance matrix of the parameters because the Hessian, that is the matrix of second derivatives of the log-likelihood with respect to the parameters, fails to invert. The failure to invert indicates that the sampling distribution of the parameters is collinear, generating a linear dependency, or a near linear dependency, in the Hessian. The consequence of this is a failure of the Hessian to invert.

There are two types of sampling distribution problems producing linear dependencies. In the first type, the dependency exists in the population being sampled; that is, it's part of the data generating process. For example, two or more exogenous variables may be measured as proportions, and they necessarily add up to a constant in every sample. In the second type, the sampling distribution is sufficiently close to a linear dependency that a certain percentage of the samples will contain linear dependencies. For example, an exogenous variable may be nearly constant in the population so that in some samples it will be sufficiently constant, making it indistinguishable from the intercept in the equation.

In either type, the linear dependency can be described by an equality constraint in the covariance matrix of the parameters. The difference between the types is that the constraint is non-stochastic in the first type, whereas it is stochastic in the second type.

The estimation of the linear dependency is conducted by **NLP** during the iterations. The **FANPAC** keyword command

```
constrainPDCovPar ON;
```

sets an **NLP** global (**_nlp_constrainHess**) which causes **NLP** to generate at each iteration a pivoted QR factorization of the Hessian or estimated Hessian (default = OFF). This factorization "pivots" small values on the diagonal to the end of the matrix. If the trailing values on the diagonal are sufficiently small, the R matrix is partitioned into that part with the diagonal values that are sufficiently large and that part where they are small. Suppose that there are k elements that are sufficiently large, then

```
b = inv(R[1:k,1:k])*R[1:k,k+1:rows(R)]
```

describes the linear dependency between the first k columns and the last rows(R)-k columns of R. The pivot vector of the QR factorization stipulates the relationship of the columns of R to the columns of the Hessian.

From the `b` matrix and the pivoting vector, **NLP** constructs an equality constraint matrix and adds it to the other constraints on the model, if any. This constraint enhances the progress of the iterations that would otherwise have some difficulty because of the poor condition of the Hessian. The constraint is in the form $Ax = 0$, where x is the vector of parameters, zero is a conformable vector of zeros, and A is a coefficient matrix constructed from the `b` matrix.

NLP imposes the equality constraint only on the iterations where it is necessary, and removes it when it is not needed. If it is needed at convergence, the equality constraint is applied to the calculation of the covariance matrix of parameters, i.e., to the inversion of the Hessian. With the feature turned on, the covariance matrix of parameters, including the standard errors, will almost always be generated. If the equality constraints found by **NLP** describe a "structural" condition of the data generating process; i.e., it holds for all samples, the covariance matrix of the parameters computed in this manner is consistent (Gallant,1987). If the equality constraint is stochastic, i.e., it is the second type, the statistical properties of this estimator aren't established.

2.6 FANPAC Keyword Commands

Summary of Keyword Commands

clearSession	clears session from memory, resets global variables
constrainPDCovPar	sets **NLP** global for constraining covariance matrix of parameters to be positive definite
computeLogReturns	computes log returns from price data
computePercentReturns	computes percent returns from price data
estimate	estimates parameters of a time series model
forecast	generates a time series and conditional variance forecast
getCV	puts conditional variances or variance-covariance matrices into global vector **_fan_CV**
getCOR	puts conditional correlations into global variable **_fan_COR**
getEstimates	puts model estimates into global variable **_fan_Estimates**
getResiduals	puts unstandardized residuals into global vector
getSeriesACF	puts autocorrelations into global variable **_fan_ACF**
getSeriesPACF	puts partial autocorrelations into global variable **_fan_PACF**
getSession	retrieves a data analysis session
getSR	puts standardized residuals into global vector
plotCOR	plots conditional correlations
plotCSD	plots conditional standard deviations
plotCV	plots conditional variances
plotQQ	generates quantile-quantile plot
plotSeries	plots time series
plotSeriesACF	plots autocorrelations
plotSeriesPACF	plots partial autocorrelations
plotSR	plots standardized residuals
session	initializes a data analysis session
setAlpha	sets inference alpha level
setConstraintType	sets type of constraints on parameters
setCovParType	sets type of covariance matrix of parameters
setCVIndEqs	declares list of independent variables to be included in conditional variance equations
setDataset	sets dataset name
setIndEqs	declares list of independent variables to be included in mean equations
setInferenceType	sets type of inference
setIndVars	declares names of independent variables

setLagTruncation	sets lags included for FIGARCH model
setLagInitialization	sets lags excluded for FIGARCH model
setLjungBoxOrder	sets order for Ljung-Box statistic
setOutputFile	sets output file name
setRange	sets range of data
setSeries	declares names of time series
setVarNames	sets variable names for data stored in ASCII file
showEstimates	displays estimates in simple format
showResults	displays results of estimations
showRuns	displays runs
simulate	generates simulation
testSR	generates skew, kurtosis, Ljung-Box statistics

2.6.1 Initializing the Session

First, an analysis session must be established.

```
session ses1 'time series analysis';
```

will start a new session, and

```
getSession ses1;
```

will retrieve a previous analysis session.

In either command, *ses1* is the name of the session and is required. It must be no more than eight characters, and the analysis results will be stored in a **GAUSS** matrix file of the same name with a *.fmt* extension. Thus the results of either of the above sessions will be stored in a file with the name *ses1.fmt*.

2.6.2 Entering Data

Before any analysis can be done, the time series must be brought into memory. If the time series resides in a **GAUSS** dataset, enter

```
setDataset stocks;
setSeries intel;
```

FANPAC looks for a **GAUSS** dataset called stocks.dat, and then looks into that dataset for a variable with name intel. If it exists, the time series is inserted into the **FANPAC** global **_fan_Series**.

If the time series is stored in a "flat" ASCII file, it is first necessary to declare the column names. This can be done using the **FANPAC** keyword command, **setVarNames**:

```
setVarNames  date intel intelvol;
setDataset intel.asc;
setSeries intel;
```

The **setVarNames** command puts the variable labels into the **FANPAC** global, **_fan_VarNames**.

2.6.3 The Date Variable

FANPAC assumes that the first observation is the oldest and the last observation is the newest. It also assumes that the date variable, if available, is stored in the yyyymmdd format. One or the other, or both, of the conditions may not be met in an ASCII data file.

Many ASCII files containing stock data will have the date stored as mm/dd/yy or mm/dd/yyyy. **FANPAC** will convert the dates to the standard format and the observations will be sorted. For example:

```
library fanpac,pgraph;
session nissan 'Analysis of Nissan daily log-returns';
setVarNames  date nsany;
setDataset nsany.asc;
setSeries nsany;
estimate run1 garch(1,3);
showResults;
```

`nsany.asc` is an ASCII file, and the command **setDataset** causes **FANPAC** to create a **GAUSS** dataset of the data with the same name as the name of the file in the keyword command argument preceding the extension. Thus a **GAUSS** dataset with file name *nsany.dat* is created with two variables in it with variable names *date* and *nsany*. If you wish the **GAUSS** data file to have a different name, include an argument in the keyword command with the desired name of the **GAUSS** dataset. For example:

```
setDataset nsany.asc newnsany;
```

It is important that the new **GAUSS** dataset file name come after the name of the ASCII data file.

2.6.4 Scaling Data

A keyword command is available for computing log returns from price data. Thus if the time series in the dataset is price data, the log returns can be computed by entering

```
computeLogReturns 251;
```

The argument is a scale factor. This function computes

$$LR_t = \kappa \; log(\frac{P_t}{P_{t-1}})$$

where P_i is price at time i, and κ is the scale factor. For best numerical results, data should be scaled to the year time scale. Thus for monthly data, $\kappa = 12$, and for daily data, $\kappa = 251$.

An additional keyword command is available for computing log *percent* returns from price data by calling **computePercentReturns**. This function computes

$$PCTR_i = \kappa \, \frac{P_t - P_{t-1}}{P_{t-1}}$$

where P_i is price at time i, and κ is the scale factor. For interpretation as a percent, the scale factor should be set to 100.

```
computePercentReturns 100;
```

FANPAC

2.6.5 Independent Variables

To add independent variables to the session, enter their names using

```
setIndVars intelvol;
```

This command assumes that the independent variables are stored in the same location as the time series. The independent variables are stored in a **FANPAC** global, **_fan_IndVars**.

The effect of the sequence of commands ending in setSeries is to store the time series in a global variable, **_fan_Series**; the independent variables, if any, in **_fan_IndVars**; and to store the names of the session, dataset time series and independent variables in a packed matrix on the disk.

2.6.6 Selecting Observations

A subset of the time series can be analyzed by specifying row numbers or, if a date variable exists in the dataset, by date. The date variable must be in the format, yyyymmdd. For the Intel dataset described above, the following are equivalent subsets:

```
setVarNames   date intel intelvol;
setDataset   intel.asc;
setSeries   intel 19960530 19961231;
```

or

```
setVarNames   date intel intelvol;
setDataset   intel.asc;
setSeries   intel 54 203;
```

The beginning and end of the time series may be specified by `start` and `end`:

```
setVarNames   date intel intelvol;
setDataset   intel.asc;
setSeries   intel start 19961231;
```

or

```
setVarNames   date intel intelvol;
setDataset   intel.asc;
setSeries   intel 19960530 end;
```

2.6.7 Simulation

A keyword command is available for simulating data from the various models in **FANPAC**. First, a string array is constructed containing the information required for the simulation, and the name of this array is passed to the keyword command. For example:

```
library fanpac;

string ss = {

  "Model garch(1,2)",
  "NumObs 300",
  "DataSetName example",
  "TimeSeriesName Y",
  "Omega .2",
  "GarchParameter .5",
  "ArchParameter .4 -.1",
  "Constant .5",
  "Seed 7351143"
};

simulate ss;
```

This produces a simulation of a GARCH(1,2) model with 300 observations and puts it into a **GAUSS** dataset named `example`.

The following simulation parameters may be included in the string array:

Model model name (required)

NumObs number of observations (required)

DataSetName name of **GAUSS** dataset into which simulated data will be put (required)

TimeSeriesName variable label of time series

Omega GARCH process constant, required for GARCH models

GarchCoefficients GARCH coefficients, required for GARCH models

ArchCoefficients ARCH coefficients, required for GARCH models

ARCoefficients AR coefficients, required for ARIMA models

MACoefficients MA coefficients, required for ARIMA models

RegCoefficients Regression coefficients, required for OLS models

DFCoefficient degrees of freedom parameter for t-density. If set, t-density will be used; otherwise Normal density

Constant constant (required)

Seed seed for random number generator (optional)

Note: Only the first two characters of the field identifier are actually looked at.

FANPAC

2.6.8 Setting Type of Constraints

By Default constraints described in Nelson and Cao (1992) are imposed on GARCH(1,q) and GARCH(2,q) models to ensure stationarity and nonnegativity of conditional variances (as described in Section2.3.2). These are the least restrictive constraints for these models.

Most GARCH estimation reported in the economics literature employ more restrictive constraints for ensuring stationarity. They are invoked primarily because the optimization software does not provide for nonlinear constraints on parameters. In this case, the GARCH parameters are simultaneously constrained to be positive and to sum to less than 1. For several reasons, including comparisons with published results, you may want to impose either no constraints or the commonly employed more highly restrictive constraints. A keyword function is provided in **FANPAC** for selecting these types of constraints:

```
setConstraintType standard
```

selects the Nelson and Cao (1992) constraints (described in Section/ref:consts). These are the least restrictive constraints that ensure stationarity and nonnegativity of the conditional variances, and are imposed by default.

```
setConstraintType unconstrained
```

will produce GARCH estimates without constraints to ensure stationarity. Nonnegativity of conditional variances is maintained by bounds constraints placed directly on the conditional variances themselves.

```
setConstraintType bounds
```

imposes the more highly restrictive linear constraints on the parameters. They constrain the coefficients in the conditional variance equation simultaneously to be greater than zero and to sum to less than one.

2.6.9 The Analysis

The **estimate** command is used for all analysis. Once the time series itself has been stored in the global, **_fan_Series**, it can be analyzed. The following performs a GARCH estimation:

```
estimate run1 garch;
estimate run2 garch(2,2);
estimate run3 egarch;
estimate run4 arima(2,1,1);
```

The first argument, the run name, is necessary. All results of this estimation will be stored in the session matrix under that name.

With the exception of OLS, these estimations are iterative using **NLP** (Section 2.8). In some cases, therefore, the iterations may be time consuming. **NLP** permits you to monitor the iterations, as well as modify descent methods, line search methods, etc., "on-the-fly" using keystrokes. To cause **NLP** to print iteration information to the screen, press "o". To get a list of options that can be modifed, press "h".

The following models may be estimated:

ols	normal linear regression model
$tols$	t distribution linear regression model
$arima(p, d, q)$	normal arima model
$tarima(p, d, q)$	t distribution arima model
$arch(q)$	normal arch model
$tarch(q)$	t distribution arch model
$archm(q)$	normal arch-in-mean model
$tarchm(q)$	t distribution arch-in-mean model
$archv(q)$	normal arch-in-cv model
$tarchv(q)$	t distribution arch-in-cv model
$garch(p, q)$	normal garch model
$tgarch(p, q)$	t distribution garch model
$garchm(p, q)$	normal garch-in-mean model
$tgarchm(p, q)$	t distribution garch-in-mean model
$garchv(p, q)$	normal garch-in-cv model
$tgarchv(p, q)$	t distribution garch-in-cv model
$igarch(p, q)$	normal integrated garch model
$itgarch(p, q)$	t distribution integrated garch model
$egarch(p, q)$	exponential garch model
$figarch(p, q)$	normal fractionally integrated garch model
$fitgarch(p, q)$	t distribution fractionally integrated garch model
$figarch(p, q)$	normal fractionally integrated garch model
$fitgarch(p, q)$	t distribution fractionally integrated garch model
$dvarch(p, q)$	normal DVEC multivariate ARCH model
$cdvarch(p, q)$	constant correlation normal DVEC multivariate ARCH model
$bkarch(p, q)$	normal BEKK multivariate ARCH model
$dvtarch(p, q)$	t distribution DVEC multivariate ARCH model
$cdvtarch(p, q)$	constant correlation t distribution DVEC multivariate ARCH model
$bktarch(p, q)$	t distribution BEKK multivariate ARCH model
$dvarchm(p, q)$	normal DVEC multivariate ARCH-in-mean model

$cdvarchm(p,q)$	constant correlation normal DVEC multivariate ARCH-in-mean model
$dvtarchm(p,q)$	t distribution DVEC multivariate ARCH-in-mean model
$cdvtarchm(p,q)$	constant correlation t distribution DVEC multivariate ARCH-in-mean model
$dvarchv(p,q)$	normal DVEC multivariate ARCH-in-cv model
$cdvarchv(p,q)$	constant correlation normal DVEC multivariate ARCH-in-cv model
$dvtarchv(p,q)$	t distribution DVEC multivariate ARCH-in-cv model
$cdvtarchv(p,q)$	constant correlation t distribution DVEC multivariate ARCH-in-cv model
$dvgarch(p,q)$	normal DVEC multivariate GARCH model
$cdvgarch(p,q)$	constant correlation normal DVEC multivariate GARCH model
$dvtgarch(p,q)$	t distribution DVEC multivariate GARCH model
$cdvtgarch(p,q)$	constant correlation t distribution DVEC multivariate GARCH model
$bkgarch(p,q)$	normal BEKK multivariate GARCH model
$bktgarch(p,q)$	t distribution BEKK multivariate GARCH model
$dvgarchm(p,q)$	normal DVEC multivariate GARCH-in-mean model
$cdvgarchm(p,q)$	constant correlation normal DVEC multivariate GARCH-in-mean model
$dvtgarchm(p,q)$	t distribution DVEC multivariate GARCH-in-mean model
$cdvtgarchm(p,q)$	constant correlation t distribution DVEC multivariate GARCH-in-mean model
$dvgarchv(p,q)$	normal DVEC multivariate GARCH-in-cv model
$cdvgarchv(p,q)$	constant correlation normal DVEC multivariate GARCH-in-cv model
$dvtgarchv(p,q)$	t distribution DVEC multivariate GARCH-in-cv model
$cdvtgarchv(p,q)$	constant correlation t distribution DVEC multivariate GARCH-in-cv model

If the models are declared without numbers in parentheses, then p, q, and d are assumed to be one.

2.6.10 Results

After the estimations have finished, results are printed using the command

```
showResults;
```

Results for individual runs can be printed by listing them in the command

```
showResults run1 run3;
```

2.6.11 Standardized and Unstandardized Residuals

It may be useful to generate standardized residuals and analyze their moments or plot their cumulative distribution against their predicted cumulative distributions. Thus

```
plotSR;
plotQQ;
```

produces a plot of the standardized results (for all model estimations by default, or specified ones if listed in the command), and plots the observed against the theoretical cumulative distributions. Both of these commands put the requested standardized residuals into the global **_fan_SR**. If you wish only to store the standardized residuals in the global, use

```
getSR;
```

or to get a particular standardized residual

```
getSR run2;
```

Unstandardized residuals are stored in **_fan_Residuals** with the following command

```
getResiduals run2;
```

A request can also be made to test the standardized residuals. The keyword command

```
testSR;
```

will generate an analysis of the time series and residuals. Skew and kurtosis statistics are computed and a heteroskedastic-consistent Ljung-Box statistic (Gouriéroux, 1997) is computed that tests the time series and residuals for autocorrelation. For example

```
======================================================================
                         Session: example1
----------------------------------------------------------------------
                         wilshire example
----------------------------------------------------------------------

                   Time Series

==================================================
                  Series: cwret
```

FANPAC

```
--------------------------------------------------
     skew      -266.1720   pr =      0.000
 kurtosis      8558.5534   pr =      0.000

 heteroskedastic-consistent
 Ljung/Box       39.0881   pr =      0.124
--------------------------------------------------


                    Residuals

==================================================
               run1: GARCH(2,1)
--------------------------------------------------
     skew        -3.9581   pr =      0.047
 kurtosis         8.3773   pr =      0.004

 heteroskedastic-consistent
 Ljung/Box       17.2809   pr =      0.969

==================================================
               run2: TGARCH(2,1)
--------------------------------------------------
     skew        -4.3003   pr =      0.038
 kurtosis        10.4523   pr =      0.001

 heteroskedastic-consistent
 Ljung/Box       18.9296   pr =      0.941
--------------------------------------------------
```

2.6.12 Conditional Variances and Standard Deviations

For the GARCH models, the conditional variances are of particular interest. To plot these, enter

```
plotCV;
```

This also stores them in **_fan_CV**. To store them in a global without plotting, use

```
getCV;
```

In some contexts the conditional standard deviations, that is, the square roots of the conditional variances, are more useful. To generate a plot, enter

```
plotCSD;
```

If percentage scaling has been used for the time series, you may want to annualize the data by scaling. This can be done by adding a scale factor in the call to **plotCSD**. For example, if the data are monthly, enter a value of 12 for the scale factor:

```
plotCSD 12;
```

2.6.13 Example

The following example analyzes daily data on Intel common stock stored in an ASCII file. The raw price data is plotted, then it is transformed to log returns. Next, several models are fitted to the transformed data, the results are printed, and the conditional variances are plotted.

```
library fanpac,pgraph;
session example1 'wilshire example';
setDataset wilshire;
setSeries cwret; /* capitalization weighted returns */
setInferenceType InvWald;
estimate run1 arch(2);
estimate run2 garch(1,2);
showResults;
testSR;
plotCV;
```

```
=====================================================================
                          Session: example1
---------------------------------------------------------------------
                          wilshire example
---------------------------------------------------------------------
 FANPAC Version 1.0.0       Data Set:  wilshire      3/10/98 11:04 am
=====================================================================


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                            Run: run1
---------------------------------------------------------------------
---------------------------------------------------------------------

return code =    0
```

FANPAC

```
normal convergence


Model:  ARCH(2)

Number of Observations     : 320
Observations in likelihood : 318
Degrees of Freedom         : 314

AIC         1859.43
BIC         1874.48
LRS         1851.43

          roots
     ---------------

      -4.5302737
       3.7149351

       Abs(roots)
     ---------------

       4.5302737
       3.7149351


  unconditional variance
----------------------------
       18.963071


Maximum likelihood covariance matrix of parameters
0.95 confidence limits computed from inversion of Wald statistic
Series: cwret

Parameters  Estimates   Standard     Lower       Upper
                         Errors      Limits      Limits
------------------------------------------------------------
omega         17.836       2.003     13.896     21.618
Arch1          0.048       0.068      0.000      0.171
Arch2          0.059       0.113      0.000      0.271
Const          1.163       0.275      0.622      1.700


Correlation Matrix of Parameters
```

```
omega        1.000    0.056   -0.528    0.130
Arch1        0.056    1.000   -0.628    0.405
Arch2       -0.528   -0.628    1.000   -0.389
Const        0.130    0.405   -0.389    1.000


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                             Run: run2
---------------------------------------------------------------------
---------------------------------------------------------------------

return code =    0
normal convergence


Model:  GARCH(1,2)

Number of Observations     : 320
Observations in likelihood : 318
Degrees of Freedom         : 313

AIC         1852.20
BIC         1871.01
LRS         1842.20

          roots
     ---------------

      -8.3913093
       1.0304374
       1.1890779

       Abs(roots)
     ---------------

       8.3913093
       1.0304374
       1.1890779


  unconditional variance
----------------------------
       1.0524876
```

FANPAC

```
Maximum likelihood covariance matrix of parameters
0.95 confidence limits computed from inversion of Wald statistic
Series: cwret

Parameters  Estimates   Standard     Lower       Upper
                         Errors      Limits      Limits
------------------------------------------------------------
omega           0.931      0.692      0.009      2.292
Garch1          0.841      0.048      0.747      0.862
Arch1           0.010      0.030      0.000      0.043
Arch2           0.116      0.063      0.012     -0.009
Const           0.993      0.230      0.541      1.445


Correlation Matrix of Parameters

omega         1.000   -0.587    0.131   -0.214    0.079
Garch1       -0.587    1.000    0.005   -0.534    0.142
Arch1         0.131    0.005    1.000   -0.586    0.180
Arch2        -0.214   -0.534   -0.586    1.000   -0.274
Const         0.079    0.142    0.180   -0.274    1.000



==================================================================
                        Session: example1
------------------------------------------------------------------
                        wilshire example
------------------------------------------------------------------

                   Time Series

==================================================
                  Series: cwret
--------------------------------------------------
       skew      -266.1720   pr =      0.000
   kurtosis      8558.5534   pr =      0.000

   heteroskedastic-consistent
   Ljung/Box    39.0881   pr =      0.124
--------------------------------------------------

 o
                    Residuals
```

```
==================================================
                    run1: ARCH
--------------------------------------------------
       skew        -3.4061   pr =      0.065
   kurtosis        11.0414   pr =      0.001

   heteroskedastic-consistent
   Ljung/Box    20.9290   pr =      0.890

==================================================
               run2: GARCH(1,2)
--------------------------------------------------
       skew        -4.2150   pr =      0.040
   kurtosis         8.8373   pr =      0.003

   heteroskedastic-consistent
   Ljung/Box    17.5968   pr =      0.965
--------------------------------------------------
```


2.6.14 Altering NLP global variables

When an estimation is invoked (i.e., when **estimate** is called) **FANPAC** calls **nlpset** which sets the **NLP** globals to their default values. This is required so that **FANPAC** will know what the **NLP** globals are set to. However, this prevents the user from setting certain **NLP** globals like **_nlp_MaxIters** to non-default values. To allow re-setting **NLP** globals, first, add a proc to the command file that has no input or output arguments, and include in the proc statements re-defining the **NLP** globals. For example,

```
proc(0)=globs;
    _nlp_MaxIters = 100;
    _nlp_IterInfo = 10;
endp;
```

Then in the command file prior to the invokation of **estimate** assign a pointer to that function to the **FANPAC** global, **_fan_NLPglobals**. For example,

```
_fan_NLPglobals = &globs;
```

The procedure will be called by **FANPAC** before it calls **NLP** for the optimization of the log-likelihood objective function, and this will re-set these globals to new values for the optimization.

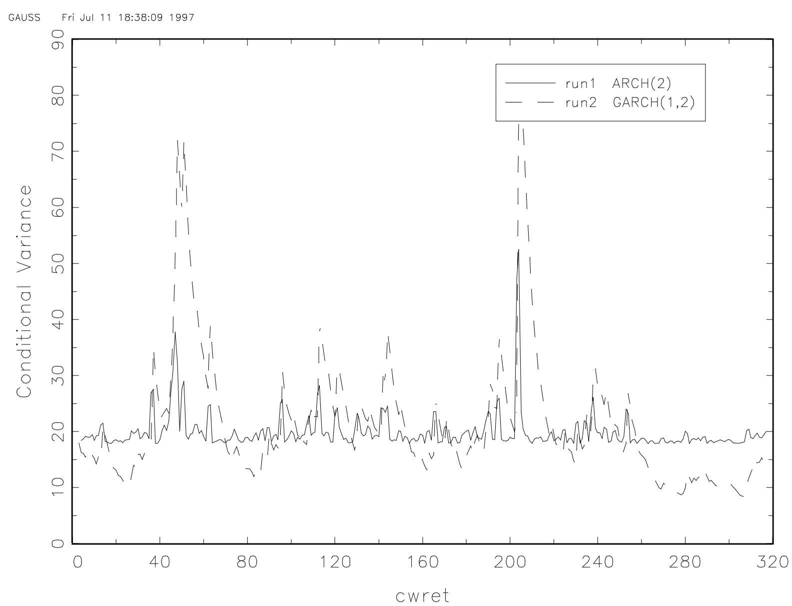

Figure 2.1: Plot of conditional variances for ARCH and GARCH models on a 27-year monthly series of the capitalization-weighted Wilshire 5000 index

2.6.15 Multivariate Models

Most keyword commands behave in the same way for multivariate models as for univariate. The specification of the time series being analyzed, for example, merely requires adding another name to the keyword command

```
setSeries AMZN YHOO;
```

The specification of the independent variables is slightly different. **FANPAC** allows specifying different sets of independent variables for each equation. A simple list of independent variables, as is done for the unvariate models, causes all independent variables to be included in all equations:

```
setIndVars AMZNvol YHOOvol
```

To specify a different list of independent variables for each equation, add the name of the dependent variable to the list, and call **setIndVars** for each dependent variable as needed. Any equation for which **setIndVars** is not called will contain all the independent variables.

```
setIndVars AMZN AMZNvol
setIndVars YHOO YHOOvol;
```

2.6.16 Example

```
library fanpac,pgraph;

session mult 'May 15, 1997 to November 9, 1998';
setDataSet stocks;
setSeries AMZN YHOO;
computeLogReturns 251;
setIndVars AMZNvol YHOOvol;
setCVIndEqs AMZN AMZNvol;
setCVIndEqs YHOO YHOOvol;
setIndEqs none;
constrainPDCovPar on;
estimate run1 dvtgarchv(2,1);
showResults;
forecast 5;
plotCV;
plotCOR;
```

```
=====================================================================
                        Session: mult
---------------------------------------------------------------------
                May 15, 1997 to November 9, 1998
---------------------------------------------------------------------
 FANPAC Version 1.0.0   Data Set:  stocks        11/13/1998 16:05:54
=====================================================================


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                          Run: run1
---------------------------------------------------------------------
---------------------------------------------------------------------

return code =    0
normal convergence


Model:  DVTGARCHV(2,1)

Number of Observations     : 375
Observations in likelihood : 373
Degrees of Freedom         : 354

AIC         3967.68
BIC         4042.19
LRS         3929.68

Maximum likelihood covariance matrix of parameters
0.95 confidence limits computed from standard errors

Series 1: AMZN
Series 2: YHOO

Parameters  Estimates   Standard     Lower       Upper
                         Errors      Limits      Limits
------------------------------------------------------------
OM11           2.508      0.095       2.321       2.695
OM12           0.143      0.092      -0.038       0.325
OM22           1.877      0.093       1.694       2.060
G111           1.029      0.078       0.876       1.183
G112           0.141      0.063       0.018       0.265
G122           0.907      0.044       0.820       0.993
G211          -0.196      0.061      -0.317      -0.075
```

```
G212          0.712     0.062     0.590     0.834
G222          0.015     0.035    -0.055     0.084
A111          0.109     0.025     0.059     0.159
A112          0.065     0.023     0.020     0.110
A122          0.051     0.023     0.005     0.097
B01           0.369     0.084     0.204     0.534
B02           0.506     0.082     0.344     0.668
AMZNcv11     -0.019     0.104    -0.224     0.186
AMZNcv21      0.123     0.080    -0.034     0.280
YHOOcv21     -0.003     0.057    -0.114     0.108
YHOOcv22     -0.091     0.039    -0.167    -0.014
nu            5.962     0.093     5.778     6.145


=======================================================
                       FORECAST
-------------------------------------------------------

-------------------------------------------------------
                run1: DVTGARCHV(2,1)
-------------------------------------------------------

      time series
       forecast

          0.36907         0.50618
          0.36907         0.50618
          0.36907         0.50618
          0.36907         0.50618
          0.36907         0.50618
-------------------------------------------------------


------------------------------

       forecast of
   conditional variance

         26.47835        32.28485
         33.24599        36.93126
         40.16543        41.47222
         46.71268        45.88915
         52.80663        50.18575

------------------------------
```

FANPAC

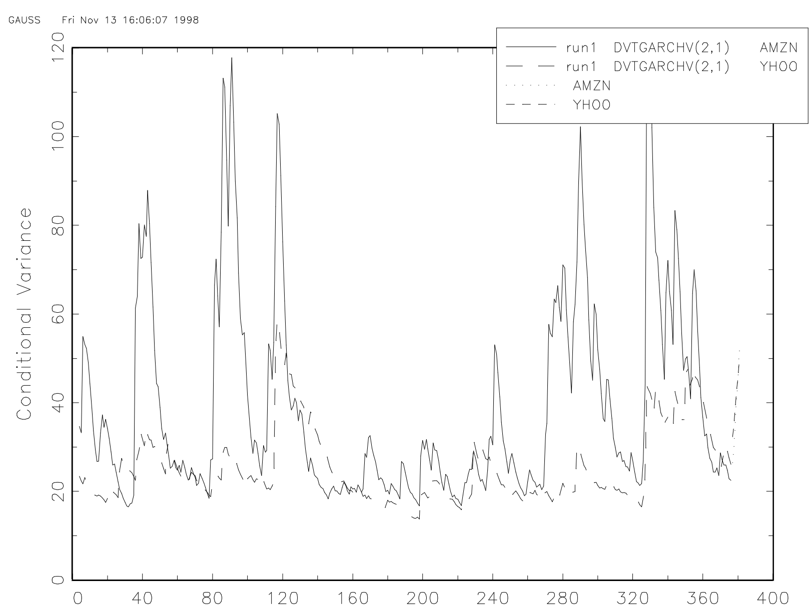

Figure 2.2: Plot of conditional variances for AMZN and YHOO using a Diagonal Vec multivariate GARCH model with t-distribution

FANPAC

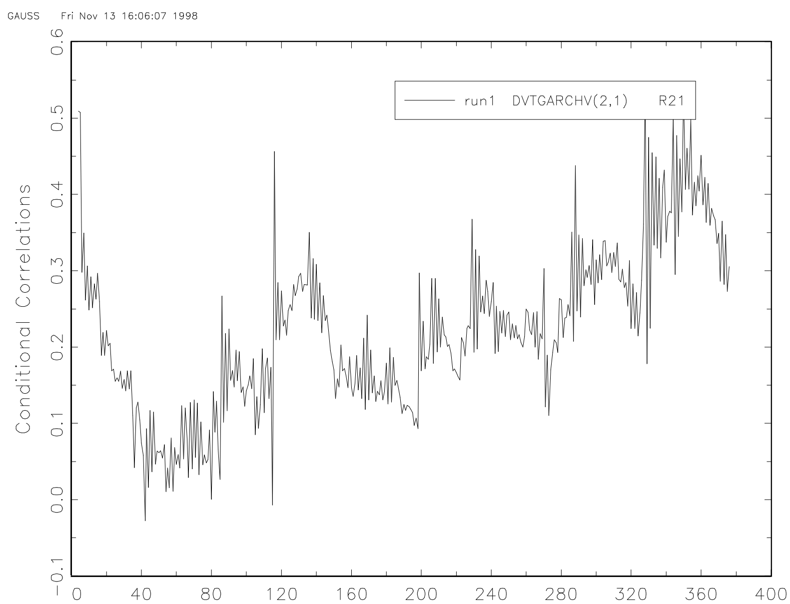

Figure 2.3: Plot of conditional correlations of AMZN and YHOO

2.7 FANPAC Procedures

The **FANPAC** procedures used by the keyword commands can be called directly. The maximum likelihood procedures for each of the **FANPAC** models can be put into command files and estimates generated using the **NLP** optimization procedures.

For example, the following is a command file for estimating a GARCH model. It estimates the model in two ways: first, using the Nelson and Cao constraints; and second, using standard constraints. The results follow the command file.

```
library fanpac;
fanset;    /* resets globals to default values */
           /* when the command file is re-run  */


_fan_p = 1;   /*  garch(1,3) model */
_fan_q = 2;

_fan_series = loadd("example");

nobs = rows(_fan_Series);


start = { .2,    /* omega */
          .3,    /* garch_1 */
          .2,    /* arch_1 */
          .1,    /* arch_2 */
           1 };  /* constant */

 _nlp_GradProc = &garch_n_grd;  /* gradient proc */


/*
**  Nelson and Cao constraints
*/

_nlp_IneqProc = &garch_ineq;  /* inequality constraints */

_nlp_Bounds = { .001   1e250,  /* omega > 0    */
                 0     1,      /* garch_1 >= 0 */
                -1e250 1e250,
                -1e250 1e250,
                -1e250 1e250 };

{ coefs,fct,grad,rcode } = nlp(&garch_n,start);
```

```
print;
print;
print "       Nelson & Cao constraints";
print;

if rcode <= 2;
    /* covariance matrix of parameters */
    h = NLPCovPar(coefs,&garch_n,&garch_n_grd,nobs,1);
else;
    h = error(0);
endif;

format /rd 12,4;
if not scalmiss(h);
    alpha = .05;  /* 95% cl's */

    dv = cdftci(0.5*alpha,nobs-rows(coefs))*sqrt(diag(h));
    print "confidence limits from standard errors";
    print;
    print "  Coefficients    lower cl    upper cl";
    print coefs~(coefs-dv)~(coefs+dv);
    print;

    /* confidence limits by inversion of Wald statistic   */
    sel = 0;      /* selection vector, if zero, all cl's computed */
    cl = NLPclimits(coefs,h,nobs,alpha,sel);
endif;

format /rd 12,4;
if not scalmiss(cl) and not scalmiss(h);

    print "confidence limits from inversion of Wald statistic";
    print;
    print "  Coefficients    lower cl    upper cl";
    print coefs~cl;

else;

    print "  Coefficients";
    print coefs;

endif;


/*
** standard constraints
```

FANPAC

```
*/


_nlp_IneqProc = {.};

_nlp_C = { 0 -1 -1 -1 0 };   /* garch + arch < 1 */
_nlp_D = { -1 };

_nlp_Bounds = { .001   1, /* omega > 0    */
                 0     1,     /* garch_1 >= 0 */
                 0     1,     /* arch_1 >= 0  */
                 0     1,     /* arch_2 >= 0  */
                -1e250 1e250 };



{ coefs,fct,grad,rcode } = nlp(&garch_n,start);

print;
print;
print "       standard constraints";

if rcode <= 2;
    /* covariance matrix of parameters */
    h = NLPCovPar(coefs,&garch_n,&garch_n_grd,nobs,1);
else;
    h = error(0);
endif;


format /rd 12,4;
if not scalmiss(h);
    alpha = .05;  /* 95% cl's */

    dv = cdftci(0.5*alpha,nobs-rows(coefs))*sqrt(diag(h));
    print;
    print "confidence limits from standard errors";
    print;
    print "  Coefficients    lower cl    upper cl";
    print coefs~(coefs-dv)~(coefs+dv);
    print;

    /* confidence limits by inversion of Wald statistic   */
    sel = 0;      /* selection vector, if zero, all cl's computed */
    cl = NLPClimits(coefs,h,nobs,alpha,sel);
endif;
```

```
format /rd 12,4;
if not scalmiss(cl) and not scalmiss(h);
    print;
    print "confidence limits from inversion of Wald statistic";
    print "  Coefficients    lower cl    upper cl";
    print coefs~cl;

else;

    print "  Coefficients";
    print coefs;

endif;
```



```
        Nelson & Cao constraints

 confidence limits from standard errors

   Coefficients    lower cl    upper cl

       0.2437       0.0329       0.4546
       0.3224      -0.1957       0.8405
       0.5338       0.2904       0.7771
      -0.0714      -0.4648       0.3220
       0.4806       0.3949       0.5664

 confidence limits from inversion of Wald statistic

   Coefficients    lower cl    upper cl

       0.2437       0.0670       0.4291
       0.3224       0.1337       0.3017
       0.5338       0.3297       0.7378
      -0.0714      -0.1721       0.0867
       0.4806       0.3949       0.5664


        standard constraints

 confidence limits from standard errors
```

```
  Coefficients    lower cl    upper cl

      0.2730       0.1358      0.4101
      0.2390       0.0224      0.4557
      0.5214       0.2866      0.7561
      0.0000            .           .
      0.4809       0.3953      0.5666


confidence limits from inversion of Wald statistic
  Coefficients    lower cl    upper cl

      0.2730       0.1580      0.4101
      0.2390       0.0574      0.4207
      0.5214       0.2866      0.7182
      0.0000            .           .
      0.4809       0.3953      0.5666
```

2.7.1 Bibliography

Amemiya, Takeshi, 1985. *Advanced Econometrics.* Cambridge, MA: Harvard University Press.

Baillie, Richard T., Bollerslev, Tim, and Mikkelsen, Hans Ole Æ., 1996. "Fractionally integrated generalized autoregressive conditional heteroskedasticity", *Journal of Econometrics*, 74:3-28.

Gallant, A. R., 1987. *Nonlinear Statistical Models.* New York: Wiley.

Geweke, John, 1995. "Posterior simulators in econometrics," Working Paper 555, Research Department, Federal Reserve Bank of Minneapolis.

Gouriéroux, Christian, 1997. *ARCH Models and Financial Applications.* New York: Springer-Verlag.

Gouriéroux, Christian, Holly, Alberto, and Monfort, Alain, 1982. "Likelihood ratio test, Wald Test, and Kuhn-Tucker test in linear models with inequality constraints on the regression parameters," *Econometrica*, 50:63-80.

Hartmann, Wolfgang M. and Hartwig, Robert E., 1995. "Computing the Penrose-Moore inverse for the covariance matrix in constrained nonlinear estimation," SAS Institute, Inc., Cary, NC.

Nelson, Daniel B. and Cao, Charles Q., 1992. "Inequality constraints in the univariate GARCH model," *Journal of Business and Economic Statistics*, 10:229-235.

O'Leary, Dianne P. and Rust, Bert W., 1986. "Confidence intervals for inequality-constrained least squares problems, with applications to ill-posed problems," *American Journal for Scientific and Statistical Computing*, 7(2):473-489.

Schoenberg, Ronald J., 1997. "Constrained maximum likelihood," *Computational Economics*, 10:251-266.

Self, Steven G. and Liang, Kung-Yee, 1987. "Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under nonstandard conditions," *Journal of the American Statistical Association*, 82:605-610.

White, H., 1981. "Consequences and detection of misspecified nonlinear regression models." *Journal of the American Statistical Association*, 76:419-433.

White, H., 1982. "Maximum likelihood estimation of misspecified models," *Econometrica*, 50:1-25.

Wolak, Frank, 1991. "The local nature of hypothesis tests involving inequality constraints in nonlinear models," *Econometrica*, 59:981-995.

2.8 NLP

NLP solves the standard nonlinear programming problem

$$min\ F(\theta)$$

subject to the linear constraints,

$$A\theta = B$$
$$C\theta \geq D$$

the nonlinear constraints,

$$G(\theta) = 0$$
$$H(\theta) \geq 0$$

and bounds,

$$\theta_l \leq \theta \leq \theta_u$$

$G(\theta)$ and $H(\theta)$ are functions provided by the user and must be differentiable at least once with respect to θ.

$F(\theta)$ must have first and second derivatives with respect to the parameters, and the matrix of second derivatives must be positive semi-definite.

NLP uses the Sequential Quadratic Programming method. In this method, the parameters are updated in a series of iterations beginning with starting values that you provide. Let θ_t be the current parameter values. Then the succeeding values are

$$\theta_{t+1} = \theta_t + \rho\delta$$

where δ is a $k \times 1$ direction vector, and ρ a scalar step length.

direction

Define

$$\begin{aligned}\Sigma(\theta) &= \frac{\partial^2 L}{\partial\theta\partial\theta'} \\ \Psi(\theta) &= \frac{\partial L}{\partial\theta}\end{aligned}$$

and the Jacobians

$$\begin{aligned}\dot{G}(\theta) &= \frac{\partial G(\theta)}{\partial\theta} \\ \dot{H}(\theta) &= \frac{\partial H(\theta)}{\partial\theta}\end{aligned}$$

For the purposes of this exposition, and without loss of generality, we may assume that the linear constraints and bounds have been incorporated into G and H.

The direction δ is the solution to the quadratic program

$$\textit{minimize}\ \frac{1}{2}\delta'\Sigma(\theta_t)\delta + \Psi(\theta_t)\delta$$

$$\begin{aligned}\textit{subject to}\quad &\dot{G}(\theta_t)\delta + G(\theta_t) = 0 \\ &\dot{H}(\theta_t)\delta + H(\theta_t) \geq 0\end{aligned}$$

This solution requires that Σ be positive semi-definite.

In practice, linear constraints are specified separately from the G and H because their Jacobians are known and easy to compute. And, the bounds are more easily handled separately from the linear inequality constraints.

line search

Define the *merit* function

$$m(\theta) = F + \max \mid \kappa \mid \sum_j \mid g_j(\theta) \mid - \max \mid \lambda \mid \sum_\ell \min(0, h_\ell(\theta))$$

where g_j is the j-th row of G, h_ℓ is the ℓ-th row of H, κ is the vector of Lagrangean coefficients of the equality constraints, and λ the Lagrangean coefficients of the inequality constraints.

The line search finds a value of ρ that minimizes or decreases $m(\theta_t + \rho\delta)$.

2.8.1 Derivatives

The SQP method requires the calculation of a Hessian, Σ, and various gradients and Jacobians, Ψ, $\dot{G}(\theta)$, and $\dot{H}(\theta)$. **NLP** computes these numerically if procedures to compute them are not supplied.

If you provide a procedure for computing Ψ, the first derivative of L, **NLP** uses it in computing Σ, the second derivative of L; i.e., Σ is computed as the Jacobian of the gradient. This improves the computational precision of the Hessian by about four places. The accuracy of the gradient is improved, and thus the iterations converge in fewer iterations. Moreover, the convergence takes less time because of a decrease in function calls – the numerical gradient requires k function calls, while an analytical gradient reduces that to one.

2.8.2 The Secant Algorithms

The Hessian may be very expensive to compute at every iteration, and poor start values may produce an ill-conditioned Hessian. For these reasons alternative algorithms are provided in **NLP** for updating the Hessian rather than computing it directly at each iteration. These algorithms, as well as step length methods, may be modified during the execution of **NLP**.

Beginning with an initial estimate of the Hessian, or a conformable identity matrix, an update is calculated. The update at each iteration adds more "information" to the estimate of the Hessian, improving its ability to project the direction of the descent. Thus, after several iterations, the secant algorithm should do nearly as well as Newton iteration with much less computation.

There are two basic types of secant methods: the BFGS (Broyden, Fletcher, Goldfarb, and Shanno), and the DFP (Davidon, Fletcher, and Powell). They are both rank two updates; that is, they are analogous to adding two rows of new data to a previously computed moment matrix. The Cholesky factorization of the estimate of the Hessian is updated using the functions **CHOLUP** and **CHOLDN**.

Secant Methods (BFGS and DFP)

BFGS is the method of Broyden, Fletcher, Goldfarb, and Shanno; and DFP is the method of Davidon, Fletcher, and Powell. These methods are complementary (Luenberger, 1984, page 268). BFGS and DFP are like the NEWTON method in that they use both first and second derivative information. However, in DFP and BFGS the Hessian is approximated, reducing considerably the computational requirements. Because they do not explicitly calculate the second derivatives, they are sometimes called *quasi-Newton* methods. While it takes more iterations than the NEWTON method, the use of an approximation produces a gain because it can be expected to converge in less overall time (unless analytical second derivatives are available, in which case it might be a toss-up).

The secant methods are commonly implemented as updates of the *inverse* of the Hessian. This is not the best method numerically for the BFGS algorithm (Gill and Murray, 1972). This version of **NLP**, following Gill and Murray (1972), updates the Cholesky factorization of the Hessian instead, using the functions **CHOLUP** and **CHOLDN** for BFGS. The new direction is then computed using **CHOLSOL**, a Cholesky solve, as applied to the updated Cholesky factorization of the Hessian and the gradient.

2.8.3 Line Search Methods

Given a direction vector d, the updated estimate of the parameters is computed

$$\theta_{t+1} = \theta_t + \rho\delta$$

where ρ is a constant, usually called the *step length*, that increases the descent of the function given the direction. **NLP** includes a variety of methods for computing ρ. The value of the function to be minimized as a function of ρ is

$$m(\theta_t + \rho\delta)$$

Given θ and d, this is a function of a single variable ρ. Line search methods attempt to find a value for ρ that decreases m. STEPBT is a polynomial fitting method; BRENT and HALF are iterative search methods. A fourth method, called ONE, forces a step length of 1. The default line search method is STEPBT. If this, or any selected method, fails, then BRENT is tried. If BRENT fails, then HALF is tried.

STEPBT

STEPBT is an implementation of a similarly named algorithm described in Dennis and Schnabel (1983). It first attempts to fit a quadratic function to $m(\theta_t + \rho\delta)$ and computes an ρ that minimizes the quadratic. If that fails, it attempts to fit a cubic function. The cubic function more accurately portrays the F, which is not likely to be very quadratic but is, however, more costly to compute. STEPBT is the default line search method because it generally produces the best results for the least cost in computational resources.

BRENT

This method is a variation on the *golden section* method due to Brent (1972). In this method, the function is evaluated at a sequence of test values for ρ. These test values are determined by extrapolation and interpolation using the constant $(\sqrt{5}-1)/2 = .6180....$ This constant is the inverse of the so-called "golden ratio" $((\sqrt{5}+1)/2 = 1.6180...$ and is why the method is called a golden section method. This method is generally more efficient than STEPBT, but requires significantly more function evaluations.

HALF

This method first computes $m(x+d)$, i.e., sets $\rho = 1$. If $m(x+d) < m(x)$, then the step length is set to 1. If not, then it tries $m(x+.5d)$. The attempted step length is divided by one-half each time the function fails to decrease, and exits with the current value when it does decrease. This method usually requires the fewest function evaluations (it often only requires one), but it is the least efficient in that it is not very likely to find the step length that decreases m the most.

2.8.4 Active and Inactive Parameters

The **NLP** global **_nlp_Active** may be used to fix parameters to their start values. This allows estimation of different models without having to modify the function procedure. **_nlp_Active** must be set to a vector of the same length as the vector of start values. Elements of **_nlp_Active** set to zero will be fixed to their starting values, while nonzero elements will be estimated.

2.9 Managing Optimization

The critical elements in optimization are scaling, starting point, and the condition of the model. When the starting point is reasonably close to the solution and the model reasonably scaled, the iterations converge quickly and without difficulty.

For best results therefore, you want to prepare the problem so that the model is well-specified and properly scaled, and that a good starting point is available.

The tradeoff among algorithms and step length methods is between speed and demands on the starting point, and condition of the model. The less demanding methods are generally time consuming and computationally intensive, whereas the quicker methods (either in terms of time or number of iterations to convergence) are more sensitive to conditioning and quality of starting point.

2.9.1 Scaling

For best performance, the diagonal elements of the Hessian matrix should be roughly equal. If some diagonal elements contain numbers that are very large and/or very small with respect to the others, **NLP** has difficulty converging. How to scale the diagonal elements of the Hessian may not be obvious, but it may suffice to ensure that the constants (or "data") used in the model are about the same magnitude.

2.9.2 Condition

The specification of the model can be measured by the condition of the Hessian. The solution of the problem is found by searching for parameter values for which the gradient is zero. If, however, the Jacobian of the gradient (i.e., the Hessian) is very small for a particular parameter, then **NLP** has difficulty determining the optimal values since a large region of the function appears virtually flat to **NLP**. When the Hessian has very small elements, the inverse of the Hessian has very large elements and the search direction gets buried in the large numbers.

Poor condition can be caused by bad scaling. It can also be caused by a poor specification of the model or by bad data. Bad models and bad data are two sides of the same coin. If the problem is highly nonlinear, it is important that data be available to describe the features of the curve described by each of the parameters. For example, one of the parameters of the Weibull function describes the shape of the curve as it approaches the upper asymptote. If data are not available on that portion of the curve, then that parameter is poorly estimated. The gradient of the function with respect to that parameter is very flat, elements of the Hessian associated with that parameter are very small, and the inverse of the Hessian contains very large numbers. In this case, it is necessary to respecify the model in a way that excludes that parameter.

2.9.3 Singular Hessian

Alternatively, **NLP** can be requested to automatically estimate the linear dependency in the Hessian. The estimation of the linear dependency is conducted by **NLP** during the iterations. When the **NLP** Global **_nlp_constrainHess** is set to a nonzero value, a pivoted QR factorization of the Hessian or estimated Hessian is generated at each iteration. This factorization "pivots" small values on the diagonal to the end of the matrix. If the trailing values on the diagonal are sufficiently small, the R matrix is partitioned into that part with the diagonal values that are sufficiently large and that part where they are small. Suppose there are k elements that are sufficiently large, then

```
b = inv(R[1:k,1:k])*R[1:k,k+1:rows(R)]
```

describes the linear dependency between the first k columns and the last rows(R)-k columns of R. The pivot vector of the QR factorization stipulates the relationship of the columns of R to the columns of the Hessian.

From the `b` matrix and the pivoting vector, **NLP** constructs an equality constraint matrix and adds it to the other constraints on the model, if any. This constraint enhances the progress of the iterations that would otherwise have some difficulty because of the poor condition of the Hessian. The constraint is in the form $Ax = 0$, where x is the vector of parameters, zero is a conformable vector of zeros, and A is a coefficient matrix constructed from the `b` matrix.

NLP imposes the equality constraint only on the iterations where it is necessary, and removes it when it is not needed. If it is needed at convergence, the equality constraint is applied to the calculation of the covariance matrix of parameters; i.e., to the inversion of the Hessian. With the feature turned on, the covariance matrix of parameters, including the standard errors, will almost always be generated. If the equality constraints found by **NLP** describe a "structural" condition of the data generating process, i.e., it holds for all samples, the covariance matrix of the parameters computed in this manner is consistent (Gallant, 1987). If the equality constraint is stochastic, i.e., it is the second type, the statistical properties of this estimator aren't established.

2.9.4 Starting Point

When the model is not particularly well-defined, the starting point can be critical. When the optimization doesn't seem to be working, try different starting points. A closed form solution may exist for a simpler problem with the same parameters. For example, ordinary least squares estimates may be used for nonlinear least squares problems or nonlinear regressions like probit or logit. There are no general methods for computing start values, and it may be necessary to attempt the estimation from a variety of starting points.

2.9.5 Diagnosis

When the optimization is not proceeding well, it is sometimes useful to examine the function, the gradient Ψ , the direction δ, the Hessian Σ, the parameters θ_t, or the step length ρ, during the iterations. The current values of these matrices can be printed out or stored in the global **_nlp_Diagnostic** by setting **_nlp_Diagnostic** to a nonzero value. Setting it to 1 causes **NLP** to print them to the screen or output file, 2 causes **NLP** to store them in **_nlp_Diagnostic**, and 3 does both.

When you have selected **_nlp_Diagnostic** = 2 or 3, **NLP** inserts the matrices into **_nlp_Diagnostic** using the **VPUT** command. The matrices are extracted using the **VREAD** command. For example,

FANPAC

```
_nlp_Diagnostic = 2;
{ x,f,g,ret } = nlp(&fct,x0);
h = vread(_nlp_Diagnostic,"hessian");
d = vread(_nlp_Diagnostic,"direct");
```

The following table contains the strings to be used to retrieve the various matrices in the **VREAD** command:

θ	"params"
δ	"direct"
Σ	"hessian"
Ψ	"gradient"
ρ	"step"

2.10 Constraints

There are two general types of constraints: nonlinear equality constraints, and nonlinear inequality constraints. However, for computational convenience, they are divided into five types: linear equality, linear inequality, nonlinear equality, nonlinear inequality, and bounds.

2.10.1 Linear Equality Constraints

Linear equality constraints are of the form

$$A\theta = B$$

where A is an $m_1 \times k$ matrix of known constants, B an $m_1 \times 1$ vector of known constants, and θ the vector of parameters.

The specification of linear equality constraints is done by assigning the A and B matrices to the **NLP** globals **_nlp_A** and **_nlp_B**, respectively. For example, to constrain the first of four parameters to be equal to the third,

```
_nlp_A = { 1 0 -1 0 };
_nlp_B = { 0 };
```

2.10.2 Linear Inequality Constraints

Linear inequality constraints are of the form

$$C\theta \geq D$$

where C is an $m_2 \times k$ matrix of known constants, D an $m_2 \times 1$ vector of known constants, and θ the vector of parameters.

The specification of linear equality constraints is done by assigning the C and D matrices to the **NLP** globals **_nlp_C** and **_nlp_D**, respectively. For example, to constrain the first of four parameters to be greater than the third, and as well the second plus the fourth greater than 10

```
_nlp_C = { 1 0 -1 0,
           0 1  0 1 };
_nlp_D = { 0,
          10 };
```

FANPAC

2.10.3 Nonlinear Equality

Nonlinear equality constraints are of the form

$$G(\theta) = 0$$

where θ is the vector of parameters and $G(\theta)$ is an arbitrary, user-supplied function. Nonlinear equality constraints are specified by assigning the pointer to the user-supplied function to the **GAUSS** global **_nlp_EqProc**.

For example, suppose you wish to constrain the norm of the parameters to be equal to 1:

```
proc eqp(b);
    retp(b'b - 1);
endp;
_nlp_EqProc = &eqp;
```

2.10.4 Nonlinear Inequality

Nonlinear inequality constraints are of the form

$$H(\theta) \geq 0$$

where θ is the vector of parameters and $H(\theta)$ is an arbitrary, user-supplied function. Nonlinear equality constraints are specified by assigning the pointer to the user-supplied function to the **GAUSS** global **_nlp_IneqProc**.

For example, suppose you wish to constrain a covariance matrix to be positive definite, the lower left nonredundant portion of which is stored in elements r:r+s of the parameter vector:

```
proc ineqp(b);
    local v;
    v = xpnd(b[r:r+s]); /* r and s defined elsewhere */
    retp(minc(eigh(v)) - 1e-5);
endp;
_nlp_IneqProc = &ineqp;
```

This constrains the minimum eigenvalue of the covariance matrix to be greater than a small number (1e-5). This guarantees the covariance matrix to be positive definite.

2.10.5 Bounds

Bounds are a type of linear inequality constraint. For computational convenience, they may be specified separately from the other inequality constraints. To specify bounds, the lower and upper bounds, respectively, are entered in the first and second columns of a matrix that has the same number of rows as the parameter vector. This matrix is assigned to the **NLP** global **_nlp_Bounds**.

If the bounds are the same for all of the parameters, only the first row is necessary.

To bound four parameters

```
_nlp_Bounds = { -10 10,
                -10  0,
                  1 10,
                  0  1 };
```

Suppose all of the parameters are to be bounded between -50 and +50 then,

```
_nlp_Bounds = { -50 50 };
```

is all that is necessary.

2.10.6 Example

The calculation of an "efficient frontier" is a quadratic programming problem. **NLP** can solve this kind of problem in one iteration. However, **NLP** can also solve a more general efficient frontier problem; in particular, one in which there are general nonlinear constraints on parameters. In the following example, a standard efficient frontier is computed, and then a second one is computed in which the weights are constrained to not differ from a preselected set of weights by more than a given amount, in this case 30 percent.

The correlation matrix, standard deviations, and returns are taken from Marmer and Louis Ng (1993)

```
library fanpac, pgraph;
nlpset;
graphset;

corr = {
  1,
 .097,  1,
-.039, .231,  1,
 .159, .237, .672,  1,
-.035, .211, .391, .454,  1,
-.024, .247, .424, .432, .941, 1 };

s = { .94, 11.26, 19.21, 13.67, 17.73, 12.19 };
Sigma = xpnd(corr) .* s .* s';
Mu = { 10.67, 10.54, 12.76, 13.67, 17.73, 13.68 };


proc ObjectiveFunction(w);
     retp(w'*Sigma*w);   /* volatility */
endp;

/*
** Constraints
*/

_nlp_A = ones(1,6);
_nlp_B = 1;

_nlp_A = _nlp_A | Mu';
_nlp_B = _nlp_B | 0;

_nlp_Bounds = { 0 1 };


start = { 1, 0, 0, 0, 0, 0 };

MN = seqa(10.75,.025,20);

W = {};
SD = {};

i = 1;
do until i > 20;

    _nlp_B[2,1] = MN[i];
    { x,f,g,ret } = nlp(&ObjectiveFunction,start);
```

```
    w = w | x';
    SD = SD | sqrt(f);    /* portfolio volatility */

i = i + 1;
endo;


format /rd 8,4;
print "Unrestricted Weights and Efficient Frontier";
print;
print "  r_{k}      sd_{k}      w_{k}";
print mn~sd~w;
print;
print;

/*
**  Now an efficient frontier restricting change from previous weights
*/

PreviousWeights = {.6,.05,.1,.0,.2,.05 };

/*
** Inequality Constraints
*/

_nlp_C = -eye(6) | eye(6);
_nlp_D = (- PreviousWeights - .3) | (PreviousWeights - .3);


W1 = {};
SD1 = {};
i = 1;
do until i > 20;

    _nlp_B[2,1] = MN[i];

    { x,f,g,ret } = nlp(&ObjectiveFunction,start);
    w1 = w1 | x';
    SD1 = SD1 | sqrt(f);    /* portfolio volatility */

i = i + 1;
endo;

print "Restricted Weights and Efficient Frontier";
print;
print "  r_{k}      sd_{k}      w_{k}";
```

```
print mn~sd1~w1;

title("Efficient Frontier");
Xlabel("Variance");
Ylabel("Return");
_plegstr = "Unrestricted Solution\000Restricted Solution";
_plegctl = { 1 5 .95 11.1 };


xy(SD~SD1,MN~MN);



Unrestricted Weights and Efficient Frontier

  r_{k}      sd_{k}      w_{k}

 10.7500   0.9431   0.9872   0.0000   0.0021   0.0000   0.0107   0.0000
 10.7750   0.9534   0.9839   0.0000   0.0017   0.0000   0.0144   0.0000
 10.8000   0.9677   0.9807   0.0000   0.0012   0.0000   0.0181   0.0000
 10.8250   0.9857   0.9775   0.0000   0.0007   0.0000   0.0217   0.0000
 10.8500   1.0072   0.9743   0.0000   0.0003   0.0000   0.0254   0.0000
 10.8750   1.0321   0.9710   0.0000   0.0000   0.0000   0.0290   0.0000
 10.9000   1.0601   0.9674   0.0000   0.0000   0.0000   0.0326   0.0000
 10.9250   1.0911   0.9639   0.0000   0.0000   0.0000   0.0361   0.0000
 10.9500   1.1247   0.9603   0.0000   0.0000   0.0000   0.0397   0.0000
 10.9750   1.1608   0.9568   0.0000   0.0000   0.0000   0.0432   0.0000
 11.0000   1.1991   0.9533   0.0000   0.0000   0.0000   0.0467   0.0000
 11.0250   1.2394   0.9497   0.0000   0.0000   0.0000   0.0503   0.0000
 11.0500   1.2815   0.9462   0.0000   0.0000   0.0000   0.0538   0.0000
 11.0750   1.3253   0.9426   0.0000   0.0000   0.0000   0.0574   0.0000
 11.1000   1.3706   0.9391   0.0000   0.0000   0.0000   0.0609   0.0000
 11.1250   1.4173   0.9356   0.0000   0.0000   0.0000   0.0644   0.0000
 11.1500   1.4652   0.9320   0.0000   0.0000   0.0000   0.0680   0.0000
 11.1750   1.5141   0.9285   0.0000   0.0000   0.0000   0.0715   0.0000
 11.2000   1.5641   0.9246   0.0000   0.0000   0.0006   0.0748   0.0000
 11.2250   1.6149   0.9207   0.0000   0.0000   0.0012   0.0781   0.0000


Restricted Weights and Efficient Frontier

  r_{k}      sd_{k}      w_{k}

 10.7500   1.3106   0.9000   0.0684   0.0066   0.0000   0.0000   0.0249
 10.7750   1.2891   0.9000   0.0604   0.0068   0.0000   0.0000   0.0328
 10.8000   1.2771   0.9000   0.0528   0.0057   0.0027   0.0000   0.0388
```

FANPAC

10.8250	1.2734	0.9000	0.0454	0.0038	0.0073	0.0000	0.0435
10.8500	1.2778	0.9000	0.0379	0.0019	0.0118	0.0000	0.0483
10.8750	1.2903	0.9000	0.0305	0.0001	0.0164	0.0000	0.0530
10.9000	1.3077	0.9000	0.0300	0.0000	0.0172	0.0058	0.0470
10.9250	1.3265	0.9000	0.0299	0.0000	0.0177	0.0119	0.0405
10.9500	1.3466	0.9000	0.0298	0.0000	0.0183	0.0180	0.0340
10.9750	1.3679	0.9000	0.0297	0.0000	0.0189	0.0240	0.0274
11.0000	1.3904	0.9000	0.0296	0.0000	0.0195	0.0301	0.0209
11.0250	1.4140	0.9000	0.0294	0.0000	0.0200	0.0362	0.0143
11.0500	1.4387	0.9000	0.0293	0.0000	0.0206	0.0423	0.0078
11.0750	1.4643	0.9000	0.0292	0.0000	0.0212	0.0484	0.0012
11.1000	1.4914	0.9000	0.0264	0.0000	0.0212	0.0524	0.0000
11.1250	1.5209	0.9000	0.0230	0.0000	0.0212	0.0559	0.0000
11.1500	1.5526	0.9000	0.0195	0.0000	0.0211	0.0594	0.0000
11.1750	1.5863	0.9000	0.0161	0.0000	0.0211	0.0629	0.0000
11.2000	1.6220	0.9000	0.0126	0.0000	0.0210	0.0664	0.0000
11.2250	1.6596	0.9000	0.0092	0.0000	0.0209	0.0699	0.0000

2.11 Gradients

2.11.1 Analytical Gradient

To increase accuracy and reduce time, you may supply a procedure for computing the gradient $\Psi(\theta) = \partial L/\partial\theta$ analytically.

This procedure has two input arguments: a $K \times 1$ vector of parameters and an $N_i \times L$ submatrix of the input data set. The **NLP** global **_nlp_GradProc** is then set to the pointer to that procedure.

In practice, unfortunately, much of the time spent on writing the gradient procedure is devoted to debugging. To help in this debugging process, **NLP** can be instructed to compute the numerical gradient along with your prospective analytical gradient for comparison purposes. In the example above, this is accomplished by setting **_nlp_GradCheckTol** to a small nonzero value.

2.11.2 Analytical Hessian

You may provide a procedure for computing the Hessian $\Sigma(\theta) = \partial^2 F/\partial\theta\partial\theta'$. This procedure has one argument, the $K \times 1$ vector of parameters, and returns a $K \times K$ symmetric matrix of second derivatives of the objection function with respect to the parameters.

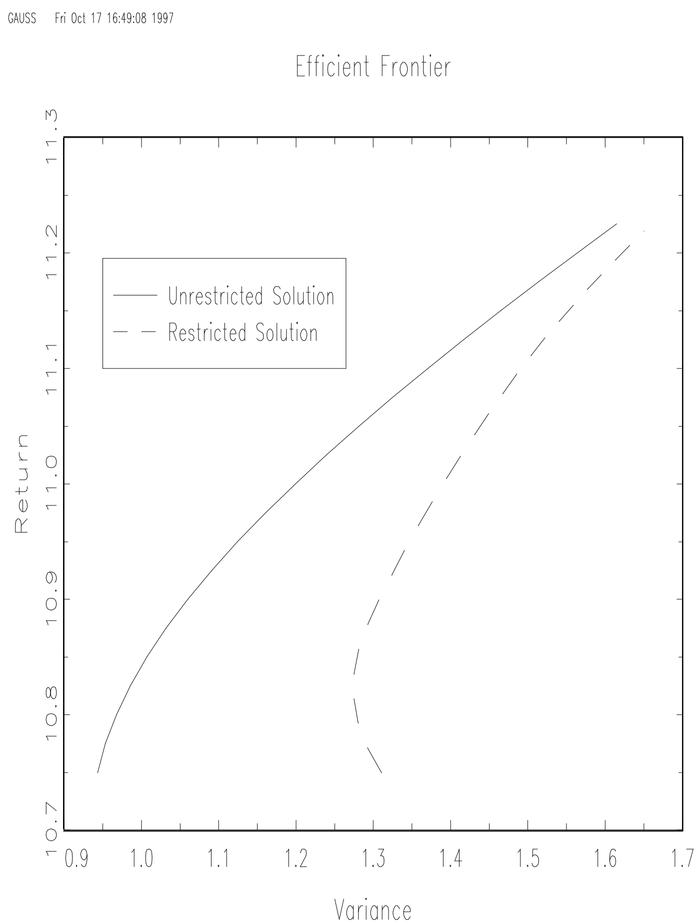

Figure 2.4: Comparison of efficient frontiers

The pointer to this procedure is stored in the global variable **_nlp_HessProc**.

In practice, unfortunately, much of the time spent on writing the Hessian procedure is devoted to debugging. To help in this debugging process, **NLP** can be instructed to compute the numerical Hessian along with your prospective analytical Hessian for comparison purposes. To accomplish, this **_nlp_GradCheckTol** is set to a small nonzero value.

2.11.3 Analytical Nonlinear Constraint Jacobians

When nonlinear equality or inequality constraints have been placed on the parameters, the convergence can be improved by providing a procedure for computing their Jacobians i.e., $\dot{G}(\theta) = \partial G(\theta)/\partial\theta$ and $\dot{H}(\theta) = \partial H(\theta)/\partial\theta$.

These procedures have one argument, the $K \times 1$ vector of parameters, and return an $M_j \times K$ matrix, where M_j is the number of constraints computed in the corresponding constraint function. Then the **NLP** globals **_nlp_EqJacobian** and **_nlp_IneqJacobian** are set to pointers to the nonlinear equality and inequality Jacobians, respectively.

2.11.4 Example

The following example illustrates furnishing procedures for computing gradients and Jacobians. It is taken from Hock and Schittkowski (1981, page 55). The function to be minimized is

$$F(\theta) = (\theta_1 + 3\theta 2 + \theta_3)^2 + 4(\theta_1 - \theta 2)^2$$

subject to the inequality constraint,

$$6\theta_2 + 4\theta_3 - \theta_1^3 - 3 \geq 0$$

and the equality constraint,

$$1 - \sum_{i=1}^{3} \theta_i = 0$$

and bounds

$$\theta_i \geq 0, \ \ i = 1, 2, 3$$

The starting point

$$\begin{bmatrix} .1 & .7 & .2 \end{bmatrix}$$

is feasible. The published solution is

$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

The procedure for solving this problem is

```
library co;
#include co.ext;
coset;

proc fct(x);
   retp( (x[1] + 3*x[2] + x[3])^2 + 4*(x[1] - x[2])^2 );
endp;

proc ineqp(x);
    retp(6*x[2] + 4*x[3] - x[1]^3 - 3);
endp;

proc ineqj(x);
   retp(-3*x[1]^2~6~4);
endp;

proc eqp(x);
    retp(1-sumc(x));
endp;

proc eqj(x);
   retp(-ones(1,3));
endp;


proc gp(x);
   local g, t;
   g = zeros(3,1);
   g[1] = 10*x[1] - 2*x[2] + 2*x[3];
   g[2] = -2*x[1] + 26*x[2] + 6*x[3];
   g[3] = 2*x[1] + 6*x[2] + 2*x[3];
   retp(g);
endp;

proc hsp(x);
   local h;
   h = zeros(3,3);
   h[1,1] = 10;
   h[1,2] = -2;
   h[1,3] = 2;

   h[2,1] = -2;
   h[2,2] = 26;
   h[2,3] = 6;

   h[3,1] = 2;
```

```
   h[3,2] = 6;
   h[3,3] = 2;
   retp(h);
endp;

_nlp_Bounds = { 0 1e256 };

start = { .1, .7, .2 };

_nlp_GradProc = &gp;
_nlp_HessProc = &hsp;
_nlp_IneqProc = &ineqp;
_nlp_IneqJacobian = &ineqj;
_nlp_EqProc = &eqp;
_nlp_EqJacobian = &eqj;

{ x,f,g,ret } = co( &fct,start );

call coprt(x,f,g,ret);

print;
print;
print "published solution";
print "     0       0      1";

print;
print "nonlinear equality Lagrangeans";
print vread(_nlp_Lagrange,"nlineq");
print;
print "nonlinear inequality Lagrangeans";
print vread(_nlp_Lagrange,"nlinineq");
print;
print "boundary Lagrangeans";
print vread(_nlp_Lagrange,"bounds");
```

The output from this run is

```
===============================================================
 NLP Version 1.0.0                          2/20/95   1:05 pm
===============================================================

return code =    0
normal convergence

Value of objective function        1.000000
```

```
Parameters    Estimates    Gradient
-----------------------------------------
P01             0.0000       2.0000
P02             0.0000       6.0000
P03             1.0000       2.0000

Number of iterations        3
Minutes to convergence     0.00267


published solution
     0       0      1

nonlinear equality Lagrangeans
-2.0000

nonlinear inequality Lagrangeans
 0.0000

boundary Lagrangeans
.
```

The missing value for the boundary Lagrangeans indicates that they are all inactive. The zero nonlinear inequality Lagrangean indicates that the nonlinear inequality boundary was encountered somewhere during the iterations, but is now inactive.

2.11.5 Run-Time Switches

If the user presses **H** on their keyboard during the iterations, a help table is printed to the screen which describes the run-time switches. By this method, important global variables may be modified during the iterations.

G	Toggle **_nlp_GradMethod**
V	Revise **_nlp_DirTol**
O	Toggle **_nlp_IterInfo**
M	Maximum Tries
I	Compute Hessian
E	Edit Parameter Vector
C	Force Exit
A	Change Algorithm
J	Change Line Search Method
T	Trust Region method
H	Help Table

The algorithm may be switched during the iterations either by pressing **A**, or by pressing one of the following:

1	Broyden-Fletcher-Goldfarb-Shanno (BFGS)
2	Davidon-Fletcher-Powell (DFP)
3	Newton-Raphson (NEWTON) or (NR)

The line search method may be switched during the iterations either by pressing **S**, or by pressing one of the following:

Shift-1	no search (1.0 or 1 or ONE)
Shift-2	cubic or quadratic method (STEPBT)
Shift-3	step halving method (HALF)
Shift-4	Brent's method (BRENT)

2.12 Error Handling

Return Codes

The fourth argument in the return from **NLP** contains a scalar number that contains information about the status of the iterations upon exiting **NLP**. The following table describes their meanings:

0	normal convergence
1	forced exit
2	maximum iterations exceeded
3	function calculation failed
4	gradient calculation failed
5	Hessian calculation failed
6	line search failed
7	function cannot be evaluated at initial parameter values
8	error with gradient
9	error with constraints
10	secant update failed
11	maximum time exceeded
12	error with weights
13	quadratic program failed
14	equality Jacobian failed
15	inequality Jacobian failed
20	Hessian failed to invert
34	data set could not be opened
99	termination condition unknown

Error Trapping

Setting the global **_nlp_IterInfo** = 1 turns on printing iteration information to the screen. Even if **_nlp_IterInfo** is set to zero, error codes are printed to the screen unless error trapping is also turned on. Setting the trap flag to 4 causes **NLP** to *not* send the messages to the screen:

```
trap 4;
```

Whatever the setting of the trap flag, **NLP** discontinues computations and returns with an error code. The trap flag in this case only affects whether messages are printed to the screen or not. This is an issue when the **NLP** function is embedded in a larger program, and you want the larger program to handle the errors.

2.12.1 Bibliography

Brent, R. P., 1972. *Algorithms for Minimization Without Derivatives.* Englewood Cliffs, NJ: Prentice-Hall.

Dennis, J. E. Jr., and Schnabel, R.B., 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Englewood Cliffs, NJ: Prentice-Hall.

Gallant, A. R., 1987, *Nonlinear Statistical Methods*, New York: Wiley.

Gill, P. E. and Murray, W., 1972. "Quasi-Newton methods for unconstrained optimization." *J. Inst. Math. Appl.*, 9, 91-108.

Han, S.P., 1977. "A globally convergent method for nonlinear programming." *Journal of Optimization Theory and Applications*, 22:297-309.

Hock, Willi and Schittkowski, Klaus, 1981. *Lecture Notes in Economics and Mathematical Systems.* New York: Springer-Verlag.

Jamshidian, Mortaza and Bentler, P.M., 1993. "A modified Newton method for constrained estimation in covariance structure analysis." *Computational Statistics & Data Analysis*, 15:133-146.

Marmer, Harry S. and Ng, F.K. Louis, 1993. "Mean-Semivariance Analysis of Option-Based Strategies: A Total Asset Mix Perspective," *Financial Analysts Journal*, May-June.

Chapter 3

FANPAC Keyword Reference

Summary of Keyword Commands

clearSession	clears session from memory, resets global variables
constrainPDCovPar	sets **NLP** global for constraining covariance matrix of parameters to be positive definite
computeLogReturns	computes log returns from price data
computePercentReturns	computes percent returns from price data
estimate	estimates parameters of a time series model
forecast	generates a time series and conditional variance forecast
getCV	puts conditional variances or variance-covariance matrices into global vector **_fan_CV**
getCOR	puts conditional correlations into global variable **_fan_COR**
getEstimates	puts model estimates into global variable **_fan_Estimates**
getResiduals	puts unstandardized residuals into global vector
getSeriesACF	puts autocorrelations into global variable **_fan_ACF**
getSeriesPACF	puts partial autocorrelations into global variable **_fan_PACF**
getSession	retrieves a data analysis session
getSR	puts standardized residuals into global vector
plotCOR	plots conditional correlations
plotCSD	plots conditional standard deviations
plotCV	plots conditional variances
plotQQ	generates quantile-quantile plot
plotSeries	plots time series
plotSeriesACF	plots autocorrelations
plotSeriesPACF	plots partial autocorrelations
plotSR	plots standardized residuals
session	initializes a data analysis session
setAlpha	sets inference alpha level
setConstraintType	sets type of constraints on parameters
setCovParType	sets type of covariance matrix of parameters

setCVIndEqs	declares list of independent variables to be included in conditional variance equations
setDataset	sets dataset name
setIndEqs	declares list of independent variables to be included in mean equations
setInferenceType	sets type of inference
setIndVars	declares names of independent variables
setLagTruncation	sets lags included for FIGARCH model
setLagInitialization	sets lags excluded for FIGARCH model
setLjungBoxOrder	sets order for Ljung-Box statistic
setOutputFile	sets output file name
setRange	sets range of data
setSeries	declares names of time series
setVarNames	sets variable names for data stored in ASCII file
showEstimates	displays estimates in simple format
showResults	displays results of estimations
showRuns	displays runs
simulate	generates simulation
testSR	generates skew, kurtosis, Ljung-Box statistics

Purpose

Resets globals to default values.

Library

`fanpac`

Format

clearSession;

Source

`fanpac.src`

Purpose

Sets **NLP** global for constraining covariance matrix of parameters to be positive definite

Library

`fanpac`

Format

constrainPDCovPar [*action*];

Input

action String. If absent, constraint feature is turned off, otherwise, set to

ON feature is turned on,

OFF feature is turned off,

Global Output

_gg_constPDCovPar Scalar, internal **FANPAC** global. If nonzero, the **NLP** global **_nlp_ConstrainHess** is set to a nonzero value, causing **NLP** to construct equality constraints to handle linear dependencies in the Hessian.

Remarks

If an equality constraint is so constructed by **NLP** at convergence, it will be used in calculating the covariance matrix of the parameters. This equality constraint is stored by **NLP** in the **NLP** global, **_nlp_PDA** and is reported by the **FANPAC** keyword command **showResults**.

Source

`fanpac.src`

Purpose

Computes log returns from price data.

Library

`fanpac`

Format

computeLogReturns [*list*] [*scale*];

Input

list	List of time series. Default, all time series.
scale	Scale factor. If omitted, scale factor is set to one.

Global Input

_fan_Series	N×k matrix, time series.

Global Output

_fan_Series	N×k matrix, time series.

Remarks

Computes the log returns from price data.

$$R_i = \kappa \, log\left(\frac{P_i}{P_{i-1}}\right)$$

where P_i is the price at time i and κ is the scale factor. For best numerical results, use a scale factor that scales the time units of the series to a year. Thus for monthly data, $\sigma = 12$, and for daily data, $\sigma = 251$.

Source

`fanpac.src`

Purpose

Computes percent returns from price data.

Library

`fanpac`

Format

computePercentReturns [*list***] [***scale***];**

Input

list	List of time series. Default, all time series.
scale	Scale factor. If omitted, scale factor is set to 100.

Global Input

_fan_Series	N×k matrix, time series.

Global Output

_fan_Series	N×k matrix, time series.

Remarks

Computes the percent returns from price data.

$$R_i = \kappa \left(\frac{P_i - P_{i-1}}{P_{i-1}} \right)$$

where P_i is the price at time i and κ is the scale factor. For interpretation as a "percent," use the default scale factor of 100.

Source

`fanpac.src`

Purpose

Generates estimates of parameters of a time series model.

Library

`fanpac`

Format

estimate *run_name* **[***run_title***]** *model***;**

Input

run_name	Name of estimation run. It must come first and it cannot contain embedded blanks.
run_title	Title of run, put in SINGLE quotes if title contains embedded blanks. May be omitted.
model	Type of time series model:

OLS Normal ordinary least squares.

TOLS t distribution ordinary least squares.

ARIMA(p,d,q) Normal ARIMA. If p, d, and q are not specified, an ARIMA(1,1,1) is estimated.

TARIMA(p,d,q) t distribution ARIMA.

EGARCH GARCH with generalized error distribution.

ARCH(p,q) Normal ARCH.

TARCH(p,q) Student's t distribution ARCH.

ARCHM(p,q) Normal ARCH-in-mean.

TARCHM(p,q) Student's t distribution ARCH-in-mean.

ARCHV(p,q) Normal ARCH-in-cv.

TARCHV(p,q) Student's t distribution ARCH-in-cv.

GARCH(p,q) Normal GARCH.

TGARCH(p,q) Student's t distribution GARCH.

GARCHM(p,q) Normal GARCH-in-mean.

TGARCHM(p,q) Student's t distribution GARCH-in-mean.

GARCHV(p,q) Normal GARCH-in-cv.

TGARCHV(p,q) Student's t distribution GARCH-in-cv.

IGARCH(p,q) Normal integrated GARCH.

ITGARCH(p,q) t distribution integrated GARCH.

FIGARCH(p,q) Normal fractionally integrated GARCH.

FITGARCH(p,q) t distribution fractionally integrated GARCH.

IGARCHV(p,q) Normal integrated GARCH-in-cv.

ITGARCHV(p,q) t distribution integrated GARCH-in-cv.

FIGARCHV(p,q) Normal fractionally integrated GARCH-in-cv.

FITGARCHV(p,q) t distribution fractionally integrated GARCH-in-cv.

IGARCHM(p,q) Normal integrated GARCH-in-mean.

ITGARCHM(p,q) t distribution integrated GARCH-in-mean.

FIGARCHM(p,q) Normal fractionally integrated GARCH-in-mean.

FITGARCHM(p,q) t distribution fractionally integrated GARCH-in-mean.

DVARCH(p,q) Normal diagonal vec multivariate ARCH.

DVTARCH(p,q) t distribution diagonal vec multivariate ARCH.

CDVARCH(p,q) Normal constant correlation diagonal vec multivariate ARCH.

CDVTARCH(p,q) t distribution constant correlation diagonal vec multivariate ARCH.

BKARCH(p,q) Normal BEKK multivariate ARCH.

BKTARCH(p,q) t distribution BEKK multivariate ARCH.

DVARCHM(p,q) Normal diagonal vec multivariate ARCH-in-mean.

DVTARCHM(p,q) t distribution diagonal vec multivariate ARCH-in-mean.

CDVARCHM(p,q) Normal constant correlation diagonal vec multivariate ARCH-in-mean.

CDVTARCHM(p,q) t distribution constant correlation diagonal vec multivariate ARCH-in-mean.

DVARCHV(p,q) Normal diagonal vec multivariate ARCH-in-cv.

DVTARCHV(p,q) t distribution diagonal vec multivariate ARCH-in-cv.

CDVARCHV(p,q) Normal constant correlation diagonal vec multivariate ARCH-in-cv.

CDVTARCHV(p,q) t distribution constant correlation diagonal vec multivariate ARCH-in-cv.

DVGARCH(p,q) Normal diagonal vec multivariate GARCH.

DVTGARCH(p,q) t distribution diagonal vec multivariate GARCH.

CDVGARCH(p,q) Normal constant correlation diagonal vec multivariate GARCH.

CDVTGARCH(p,q) t distribution constant correlation diagonal vec multivariate GARCH.

BKGARCH(p,q) Normal BEKK multivariate GARCH.

BKTGARCH(p,q) t distribution BEKK multivariate GARCH.

DVGARCHM(p,q) Normal diagonal vec multivariate GARCH-in-mean.

DVTGARCHM(p,q) t distribution diagonal vec multivariate GARCH-in-mean.

CDVGARCHM(p,q) Normal constant correlation diagonal vec multivariate GARCH-in-mean.

CDVTGARCHM(p,q) t distribution constant correlation diagonal vec multivariate GARCH-in-mean.

DVGARCHV(p,q) Normal diagonal vec multivariate GARCH-in-cv.

DVTGARCHV(p,q) t distribution diagonal vec multivariate GARCH-in-cv.

CDVGARCHV(p,q) Normal constant correlation diagonal vec multivariate GARCH-in-cv.

CDVTGARCHV(p,q) t distribution constant correlation diagonal vec multivariate GARCH-in-cv.

Global Input

_fan_Dataset Name of **GAUSS** data set containing time series being analyzed.

_fan_SeriesNames Name of time series being analyzed.

_fan_IndVarNames K×1, character vector of labels of independent variables.

Remarks

estimate generates estimates of the parameters of the specified model. The results are stored in a **GAUSS** .fmt file on the disk in the form of a vpacked matrix. These results are not printed by estimate. See **showResults** for displaying results.

All models except OLS are estimated using the **NLP** optimization program. See the **NLP** procedure in Section 2.8 for details concerning the optimization.

Example

```
library fanpac,pgraph;

session test 'test session';

setDataset stocks;
```

```
setSeries intel;
setOutputfile test.out reset;

estimate run1 garch;
estimate run2 garch(2,1);
estimate run3 arima(1,2,1);

showResults;
plotSeries;
plotCV;
```

Source

fanpac.src

Purpose

Generates forecasts of a time series model.

Library

`fanpac`

Format

forecast [*list*] [*periods*];

Input

list	Names of run for forecast. If none is specified, forecasts will be generated for all runs.
periods	Number of periods to be forecast. If not specified, the forecast is for one period.

Global Output

_fan_TSforecast L×K matrix, L forecasts for K models.

_fan_CVforecast L×K matrix, L forecasts for K models.

Remarks

If the model is a GARCH model, a forecast of the conditional variance is generated as well. The forecasts are written to a **FANPAC** global. The time series forecast is written to **_fan_TSforecast** and the conditional variance is written to **_fan_CVforecast**. If **plotCV** or **plotCSD** is called after the call to **forecast**, the forecasts are included in the plot. If **plotSeries** is called after the call to **forecast**, the time series forecast is plotted with the time series as well.

Source

`fanpac.src`

Purpose

Computes conditional variances and puts them into a global variable.

Library

`fanpac`

Format

getCV [*list*];

Input

list	List of runs. If omitted, conditional variances will be produced for all runs.

Global Output

_fan_CV	N×K matrix, conditional variances.

Remarks

Conditional variances are relevant only for ARCH/GARCH models. No results are generated for other models.

See also

plotCV

Source

`fanpac.src`

Purpose

Computes conditional correlations and puts them into a global variable.

Library

`fanpac`

Format

getCOR [*list*];

Input

list	List of runs. If omitted, conditional correlations will be produced for all runs.

Global Output

_fan_COR	N×K matrix, conditional correlations

Remarks

Conditional correlations are relevant only for multivariate ARCH/GARCH models. No results are generated for other models.

See also

plotCOR

Source

`fanpac.src`

Purpose

Stores estimates in global variable.

Library

`fanpac`

Format

getEstimates [*list*];

Input

list	List of runs. If omitted, estimates for all runs will be stored in global variable.

Global Output

_fan_Estimates	K×L matrix, global into which estimates are stored.

Remarks

Source

`fanpac.src`

■ Purpose

Computes unstandardized residuals and puts them into a global variable.

■ Library

`fanpac`

■ Format

getRD [*list*];

■ Input

list	List of runs. If omitted, standardized residuals will be produced for all runs.

■ Global Output

_fan_Residuals	N×K matrix, standardized residuals.

■ Source

`fanpac.src`

Purpose

Computes autocorrelation function and puts the vector into a global variable.

Library

`fanpac`

Format

getSeriesACF [*list***]** *num diff* **;**

Input

list	List of series. If omitted, will be produced for all series.
num	Scalar, maximum number of autocorrelations to compute. If omitted, set to number of observations.
diff	Scalar, order of differencing. If omitted, set to zero.

Global Output

_fan_ACF	*num*×K matrix, autocorrelations.

Remarks

If one number is entered as an argument, *num* will be set to that value. If two numbers are entered as arguments, *num* will be set to the larger number and *diff* to the smaller number.

See also

plotSeriesACF, **plotSeriesPACF**, **getSeriesPACF**

Source

`fanpac.src`

■ Purpose

Computes autocorrelation function and puts the vector into a global variable.

■ Library

`fanpac`

■ Format

getSeriesPACF [*list***]** *num diff* **;**

■ Input

list	List of series. If omitted, will be produced for all series.
num	Scalar, maximum number of autocorrelations to compute. If omitted, set to number of observations.
diff	Scalar, order of differencing. If omitted, set to zero.

■ Global Output

_fan_PACF	*num*×K matrix, autocorrelations.

■ Remarks

If one number is entered as an argument, *num* will be set to that value. If two numbers are entered as arguments, *num* will be set to the larger number and *diff* to the smaller number.

■ See also

plotSeriesPACF, **plotSeriesACF**, **getSeriesACF**

■ Source

`fanpac.src`

Purpose

Retrieves a data analysis session.

Library

`fanpac`

Format

getSession *session_name*;

Input

session_name Name of an existing session.

Remarks

getSession retrieves a session created by a previous analysis.

Source

`fanpac.src`

Purpose

Computes standardized residuals and puts them into a global variable.

Library

`fanpac`

Format

getSR [*list*];

Input

list	List of runs. If omitted, standardized residuals will be produced for all runs.

Global Output

_fan_SR	N×K matrix, standardized residuals.

See also

plotSR

Source

`fanpac.src`

Purpose

Plots conditional correlations.

Library

`fanpac, pgraph`

Format

plotCOR [*list***] [***start end***];**

Input

list	List of runs. If no list, conditional correlations will be plotted for all runs.
start	Scalar, starting row or date to be included in plot. If row number, it must be greater than 1 and less than *end*. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *start* to START is equivalent to first observation.
end	Scalar, ending row or date to be included in plot. If row number, it must be greater than *start* and less than or equal to the number of observations. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *end* to END is equivalent to last observation.

Global Output

_fan_COR	N×K matrix, conditional correlations.

Remarks

Conditional correlations are relevant only for multivariate ARCH/GARCH models. No plots or output are generated for other models.

Source

`fanplot.src`

Purpose

Plots conditional standard deviations.

Library

`fanpac, pgraph`

Format

plotCSD [*list*] [*start end*] [*scale*];

Input

list	List of runs. If no list, conditional variances will be plotted for all runs.
start	Scalar, starting row or date to be included in plot. If row number, it must be greater than 1 and less than *end*. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *start* to START is equivalent to first observation.
end	Scalar, ending row or date to be included in plot. If row number, it must be greater than *start* and less than or equal to the number of observations. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *end* to END is equivalent to last observation.
scale	Scalar, scale factor. The conditional standard deviations are multiplied by the square root of the scale factor before plotting. Default = 1.

Global Input

_fan_CVforecast L×K matrix, forecasts of conditional variances.

Global Output

_fan_CV N×K matrix, conditional variances.

Remarks

Conditional standard deviations are relevant only for ARCH/GARCH models. No plots or output are generated for other models.

plotCSD plots the square roots of the conditional variances times the scale factor, if any.

If **plotCSD** is called after a call to **forecast**, the square root of the forecasts of the conditional variances stored in **_fan_CVforecast** are plotted as well.

Source

`fanplot.src`

Purpose

Plots conditional variances.

Library

`fanpac, pgraph`

Format

plotCV [*list*] [*start end*];

Input

list	List of runs. If no list, conditional variances will be plotted for all runs.
start	Scalar, starting row or date to be included in plot. If row number, it must be greater than 1 and less than *end*. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *start* to START is equivalent to first observation.
end	Scalar, ending row or date to be included in plot. If row number, it must be greater than *start* and less than or equal to the number of observations. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *end* to END is equivalent to last observation.

Global Output

_fan_CV	N×K matrix, conditional variances.
_fan_CVforecast	L×K matrix, forecasts of conditional variances.

Remarks

Conditional variances are relevant only for ARCH/GARCH models. No plots or output are generated for other models.

If **plotCV** is called after a call to **forecast**, the forecasts of the conditional variance stored in **_fan_CVforecast** are plotted as well.

Source

`fanplot.src`

Purpose

Plots quantile-quantile plot.

Library

`fanpac, pgraph`

Format

plotQQ [*list*];

Input

list	List of runs. If no list, QQ plots will be generated for all runs.

Global Output

_fan_SR	N×K matrix, standardized residuals.

Source

`fanplot.src`

■ Purpose

Plots time series.

■ Library

`fanpac, pgraph`

■ Format

plotSeries [*list***] [***start end***];**

■ Input

list	List of series. If no list, all series will be plotted.
start	Scalar, starting row or date to be included in plot. If row number, it must be greater than 1 and less than *end*. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *start* to START is equivalent to first observation.
end	Scalar, ending row or date to be included in plot. If row number, it must be greater than *start* and less than or equal to the number of observations. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *end* to END is equivalent to last observation.

■ Global Input

_fan_Series N×1 vector, time series.

_fan_TSforecast L×1 vector, forecasts.

■ Remarks

If **forecast** is called before **plotSeries**, the time series forecast stored in **_fan_TSforecast** is included in the plot.

■ Source

`fanplot.src`

Purpose

Computes autocorrelation function and puts the vector into a global variable.

Library

`fanpac, pgraph`

Format

plotSeriesACF [*list*] [*num*] [*diff*];

Input

list	List of series. If omitted, will be produced for all series.
num	Scalar, maximum number of autocorrelations to compute. If omitted, set to number of observations.
diff	Scalar, order of differencing. If omitted, set to zero.

Global Output

_fan_ACF	$num \times$K matrix, autocorrelations.

Remarks

If one number is entered as an argument, *num* will be set to that value. If two numbers are entered as arguments, *num* will be set to the larger number and *diff* to the smaller number.

See also

plotSeriesPACF, **getSeriesACF**

Source

`fanplot.src`

■ Purpose

Computes autocorrelation function and puts the vector into a global variable.

■ Library

`fanpac, pgraph`

■ Format

plotSeriesPACF [*list*] [*num*] [*diff*];

■ Input

list	List of series. If omitted, will be produced for all series.
num	Scalar, maximum number of autocorrelations to compute. If omitted, set to number of observations.
diff	Scalar, order of differencing. If omitted, set to zero.

■ Global Output

_fan_PACF	*num*×K matrix, autocorrelations.

■ Remarks

If one number is entered as an argument, *num* will be set to that value. If two numbers are entered as arguments, *num* will be set to the larger number and *diff* to the smaller number.

■ See also

plotSeriesACF, **getSeriesPACF**

■ Source

`fanplot.src`

Purpose

Plots standardized residuals.

Library

`fanpac, pgraph`

Format

plotSR [*list***] [***start end***];**

Input

list	List of runs. If no list, standardized residuals will be plotted for all runs.
start	Scalar, starting row or date to be included in plot. If row number, it must be greater than 1 and less than *end*. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *start* to START is equivalent to first observation.
end	Scalar, ending row or date to be included in plot. If row number, it must be greater than *start* and less than or equal to the number of observations. If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date." Setting *end* to END is equivalent to last observation.

Global Output

_fan_SR	N×K matrix, standardized residuals.

Remarks

Standardized residuals are relevant only for ARCH/GARCH models. No plots or output are generated for other models.

Source

`fanplot.src`

Purpose

Initializes a data analysis session.

Library

`fanpac`

Format

session *session_name* **[***session_title***];**

Input

session_name Name of session; it must contain no more than 8 characters and no embeddded blanks.

session_title Title of run, put in SINGLE quotes if title contains embedded blanks. If no title entered, it is set to null string.

Source

`fanpac.src`

Purpose

Sets confidence level for statistical inference.

Library

`fanpac`

Format

setAlpha *alpha*;

Input

alpha	Scalar, confidence level. Default = .05.

Source

`fanpac.src`

Purpose

sets type of constraints for stationarity conditions in Garch models

Library

`fanpac`

Format

SetConstraintType *type*;

Input

type String, type of constraint

- **standard** standard constraints
- **bounds** bounds constraints on parameters
- **unconstrained** no constraints

Global Output

_gg_ConstType Scalar, type of constraints

- **1** standard constraints
- **2** bounds constraints on parameters
- **3** no constraints

Remarks

standard For garch(1,q) and garch(2,q) models, parameter are constrained using the Nelson & Cao specifications to ensure that conditional variances are nonnegative for all observations in and out of sample. Also, stationarity is assured by constraining roots to be outside unit circle. This involves a nonlinear constraint on parameters. These are the least restrictive constraints that satisfy the conditions of nonnegative conditional variances and stationarity.

bounds Nonnegativity of conditional variances is carried out by direct constraints on the conditional variances. This does not assure nonnegativity outside of the sample. Stationarity is imposed by placing bounds on parameters, that is, **arch** and **garch** coefficients are constrained to be greater than zero and sum to less than one. These constraints are more restrictive than the standard coefficients, and are the most commonly applied constraints.

unconstrained Conditional variances are directly constrained to be nonnegative as in the bounds method, but no constraints are applied to ensure stationarity.

Source

`fanpac.src`

Purpose

Sets type of covariance matrix of parameters.

Library

`fanpac`

Format

setCovParType *type*;

Input

type String, type of covariance matrix.

- **ML** Maximum likelihood.
- **XPROD** Cross product of first derivatives.
- **QML** Quasi-maximum likelihood.

Global Output

_fan_CovParType Scalar, type of covariance matrix of parameters.

- **ML** Maximum likelihood.
- **XPROD** Cross-product of first derivatives.
- **QML** Quasi-maximum likelihood.

Remarks

let $H = \partial logl/\partial\theta\partial\theta'$ be the Hessian and $G = \partial logl/\partial\theta$ the matrix of first derivatives. Then ML $= H^{-1}$, XPROD $= (G'G)^{-1}$, and $WML = H^{-1}(G'G)H^{-1}$.

Source

`fanpac.src`

Purpose

Declares independent variables for inclusion into conditional variance equation.

Library

fanpac

Format

setCVIndEqs *name list*;

Input

name Name of time series for this set of independent. variables

list List of names of independent variables.

Global Output

_fan_CVIndEquations L×K character vector, names of independent variables for each equation.

Remarks

An equation is associated with each time series. For multivariate models, call **setCVIndEqs** for each time series, listing the independent variables by name in each call:

```
setCVIndEqs msft  logVol1  SandP
setCVIndEqs intc  logVol2  SandP
```

If time series names are omitted, only one call is permitted and all independent variables are assumed to be entered in all equations.

```
setCVIndEqs  logVol1 logVol2 SandP
```

Source

fanpac.src

Purpose

Sets dataset name for analysis.

Library

`fanpac`

Format

setDataset *name* **[***newname***];**

Input

name Name of file containing data.

newname If *name* is not the name of a **GAUSS** data set, a **GAUSS** data set will be created with name *newname* from the data in *name*.

Global Input

_fan_VarNames Scalar or K×1 character vector, column numbers
– or –
variable names of the columns of the data in the data file. If *name* is not a **GAUSS** data set file, **_fan_VarNames** is required to name the variables in the data set.

If **_fan_VarNames** is set to scalar number of columns, the variables in the data file will be given labels X1, X2..... If **_fan_VarNames** is scalar missing (default), it is assumed that the data file contains a single column of data.

Global Output

_fan_dataset String, name of **GAUSS** data set.

Remarks

If *name* is not a **GAUSS** data set file or a DRI database, **FANPAC** assumes that *name* is a file containing the data.

If one of the columns in the **GAUSS** data set is labeled DATE, **FANPAC** will assume that this variable is a date variable in the format *yyyymmddhhmmss*.

If the data file is not a **GAUSS** data set file or DRI database, and one of the variable names in **_fan_VarNames** is DATE, **FANPAC** will assume that the associated column in the data on that file is a date variable. The format of the date in that file can be *mm/dd/yy* or *mm/dd/yyyy* or *yyyymmdd*, and it will be put by **FANPAC** into the *yyyymmddhhmmss* format.

If the data in the data file are in the nonstandard order, i.e., from most recent date at the top to the oldest date at the bottom, **FANPAC** reverses the order of the data in the **GAUSS** data set generated from the data. This will also occur if any of the dates are out of order. If the data are stored in a **GAUSS** data set, this check will not be made.

Example

```
library fanpac,pgraph;
session nissan 'Analysis of Nissan daily log-returns';
setVarNames date nsany;
setDataset nsany.asc;
setSeries nsany;
estimate run1 garch(1,3);
showResults;
```

Source

fanpac.src

Purpose

Declares independent variables.

Library

`fanpac`

Format

setIndEqs *name list*;

Input

name	Name of time series for this set of independent variables.
list	List of names of independent variables.

Global Output

_fan_IndEquations	L×K matrix, indicator matrix for coefficients to be estimated.

Remarks

An equation is associated with each time series. For multivariate models, call **setIndEqs** for each time series, listing the independent variables by name in each call:

```
setIndEqs msft  logVol1  SandP
setIndEqs intc  logVol2  SandP
```

If **setIndEqs** is not called for a particular dependent variable, coefficients for all independent variables will be estimated for that dependent variable.

Source

`fanpac.src`

Purpose

Sets type of statistical inference.

Library

`fanpac`

Format

setInferenceType [*type*];

Input

type	If omitted, standard errors computed from covariance matrix of parameters are computed. Otherwise, set to

WALD inversion of Wald statistic,

PFL inversion of LR statistic,

SE standard errors computed from covariance matrix of parameters.

Source

`fanpac.src`

Purpose

Declares exogenous or independent variables.

Library

`fanpac`

Format

setIndVars *list*;

Input

list	List of names of independent variables for current session.

Global Output

_fan_IndvarNames	L×K character vector, names of independent variables for each equation.

Source

`fanpac.src`

Keyword Reference

Purpose

Sets number of lags INCLUDED in analysis for FIGARCH models.

Library

`fanpac`

Format

setLagTruncation *num*;

Input

num	Number of lags included.

Remarks

The conditional variance in the FIGARCH(p,q) model is the sum of an infinite series of prior conditional variances. In practice, the log-likelihood is computed from available data; and this means that the calculation of the conditional variance will be truncated. To minimize this error, the log-probabilities for initial observations can be excluded from the log-likelihood. The default is one-half of the observations. To change this specification, **setLagTruncation** can be set to some other value that determines the number of observations to be included.

Source

`fanpac.src`

Purpose

Sets number of lags EXCLUDED in analysis for FIGARCH models.

Library

`fanpac`

Format

setLagInitialization *num*;

Input

num	Number of lags included.

Remarks

The conditional variance in the FIGARCH(p,q) model is the sum of an infinite series of prior conditional variances. In practice, the log-likelihood is computed from available data; and this means that the calculation of the conditional variance will be truncated. To minimize this error, the log-probabilities for initial observations can be excluded from the log-likelihood. The default is one-half of the observations. To change this specification **setLagInitialization** can be set to some other value that determines the number of observations to be excluded.

Source

`fanpac.src`

Purpose

Sets order for Ljung-Box statistic.

Library

`fanpac`

Format

setLjungBoxOrder *order*;

Input

order	Number of autocorrelations included in the Ljung-Box test statistic. It must be less than the total number of observations.

Source

`fanpac.src`

Purpose

Sets output file name and status.

Library

`fanpac`

Format

setOutputFile *filename* **[***action***];**

Input

filename	Output file is created with this name.
action	String. If absent, output file is turned on, otherwise, set to
	ON output file is turned on, **OFF** output file is turned off, **RESET** output file is reset.

Source

`fanpac.src`

■ Purpose

sets range of time series to be analyzed

■ Library

`fanpac`

■ Format

setRange *start end*;

■ Input

start Scalar, starting row or date to be included in series. If row number, it must be greater than 1 and less than *end*.

If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date."

Setting *start* to START is equivalent to first observation.

end scalar, ending row or date to be included in series. If row number, it must be greater than *start* and less than or equal to the number of observations.

If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date."

Setting *end* to END is equivalent to last observation.

■ Global Output

_fan_Series N×L matrix, time series.

_fan_Date N/*times*1 vector, dates of observations in yyyymmmdd format. This requires that the session dataset contain a variable in that same format with variable name "date."

■ Source

`fanpac.src`

Purpose

Declares time series to be analyzed.

Library

`fanpac`

Format

setSeries *list* **[***start end***];**

Input

list List of names of time series.

start Scalar, starting row or date to be included in series. If row number, it must be greater than 1 and less than *end*.

If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date."

Setting *start* to START is equivalent to first observation.

end Scalar, ending row or date to be included in series. If row number, it must be greater than *start* and less than or equal to the number of observations.

If date, it may be in one of the formats, *yyyymmdd*, *yyyymmddhhmmss*, *mm/dd/yy*, *mm/dd/yyyy*, where if yy the 20th century is assumed. The session dataset must also have included a variable with the variable name "date."

Setting *end* to END is equivalent to last observation.

Global Output

_fan_Series N×L matrix, time series.

_fan_SeriesNames L×1 character vector, names of time series.

_fan_Date N/*times*1 vector, dates of observations in yyyymmmdd format. This requires that the session dataset contain a variable in that same format with variable name "date."

Source

`fanpac.src`

Keyword Reference

Purpose

Sets variable names for ASCII file containing data.

Library

`fanpac`

Format

setVarNames *list*;

Input

list Variable names of the columns of an ASCII file containing data.
– or –
scalar number of columns of data in ASCII file

Global Output

_fan_VarNames K×1 character vector, variable names of data in the ASCII data file.

Remarks

If list is a scalar number of columns, variables in data file will be given labels X1, X2,....

Source

`fanpac.src`

Purpose

Displays estimates and their lables in a simple format

Library

`fanpac`

Format

showEstimates *list*;

Input

list	List of names of estimation runs. If no run names are provided, all runs are displayed.

Source

`fanpac.src`

■ Purpose

Displays results of a run.

■ Library

`fanpac`

■ Format

showResults *list*;

■ Input

list	List of names of estimation runs. If no run names are provided, all runs are displayed.

■ Example

```
library fanpac,pgraph;

session test 'test session';

setDataset stocks;
setSeries intel;
setOutputfile test.out reset;

estimate run1 garch;
estimate run2 garch(2,1);
estimate run3 arima(1,2,1);

showResults;
```

■ Source

`fanpac.src`

Purpose

Displays a List of current runs in a session.

Library

`fanpac`

Format

showRuns;

Source

`fanpac.src`

Purpose

Simulates data with GARCH errors.

Library

`fanpac`

Format

simulate *starray*;

Input

starray K×1 string array, simulation parameters

Model model name (required).

NumObs number of observations (required).

DatasetName name of **Gauss** data set into which simulated data will be put (required).

TimeSeriesName variable label of time series.

Omega GARCH process constant, required for GARCH models.

GarchCoefficients GARCH coefficients, required for GARCH models.

ArchCoefficients ARCH coefficients, required for GARCH models.

ARCoefficients AR coefficients, required for ARIMA models.

MACoefficients MA coefficients, required for ARIMA models.

RegCoefficients Regression coefficients, required for OLS models.

DFCoefficient degrees of freedom parameter for t-density. If set, t-density will be used; otherwise Normal density.

Constant constant (required).

Seed seed for random number generator (optional).

Example

```
library fanpac;

string ss = {

  "Model garch(1,2)",
  "NumObs 300",
  "DatasetName example",
  "TimeSeriesName Y",
```

```
    "Omega .2",
    "GarchParameter .5",
    "ArchParameter .4 -.1",
    "Constant .5",
    "Seed 7351143"
  };

  simulate ss;
```

Source

`fansim.src`

Purpose

Computes skew and kurtosis statistics and a heteroskedastic-consistent Ljung-Box statistic for standardized residuals as well as time series.

Library

`fanpac`

Format

testSR *list*;

Input

list	List of runs.

Remarks

The Ljung-Box statistic is the heteroskedastic-consistent statistic described in Gouriéroux, 1997.

Source

`fanpac.src`

Chapter 4

FANPAC Procedure Reference

Purpose

Computes time series and conditional variance forecasts.

Library

`fanpac`

Format

{ r,s } = **arch_forecast(**b,q,$period$,xp**);**

Input

b	$K \times 1$ vector, coefficients.
q	Scalar, order of ARCH parameters.
$period$	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r	$L \times 1$ vector, L period forecast of times series.
s	$L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if ARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if ARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ARCH-in-mean coefficient, if any,

ARCH-in-cv coefficients, if any.

Source

`arch.src`

Purpose

Computes Normal density ARCH log-likelihood.

Library

`fanpac`

Format

y = **arch_n(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times k$ matrix, independent variables. If none, set to missing value.

_fan_q Scalar, order of ARCH parameters.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if ARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if ARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ARCH-in-mean coefficient, if any,

ARCH-in-cv coefficients, if any.

The ARCH model cannot be both ARCH-in-mean and ARCH-in-CV.

Source

`arch.src`

Purpose

Computes gradient of Normal density ARCH log-likelihood.

Library

`fanpac`

Format

y = **arch_n_grd(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times K$ matrix, gradient matrix.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_q Scalar, order of ARCH parameters.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any.

Source

`arch.src`

Purpose

Computes t-density ARCH log-likelihood.

Library

`fanpac`

Format

y = **arch_t**(b);

Input

b — $K \times 1$ vector, coefficients.

Output

y — $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series — $N \times 1$ vector, time series.

_fan_IndVars — $N \times k$ matrix, independent variables. If none, set to missing value.

_fan_q — Scalar, order of ARCH parameters.

_fan_IndEquations — $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations — $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean — Scalar, nonzero if ARCH-in-mean model, else zero.

_fan_inCV — Scalar, nonzero if ARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

Procedure Reference

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ARCH-in-mean coefficient, if any,

ARCH-in-cv coefficients, if any.

residual variance,

ν.

Source

`arch.src`

Purpose

Computes gradient of t-density ARCH log-likelihood.

Library

`fanpac`

Format

y = **arch_t_grd(**b**);**

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times K$ matrix, gradient matrix.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_q Scalar, order of ARCH parameters.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

residual variance,

ν.

Source

`arch.src`

Procedure Reference

Purpose

Computes ARCH model Nelson and Cao constraints.

Library

`fanpac`

Format

y = **arch_ineq(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$L \times 1$ vector, roots.

Global Input

_fan_q	Scalar, order of ARCH parameters.

Remarks

Computes Nelson and Cao (1992) constraint function. When the statement

```
_nlp_IneqProc = &arch_ineq;
```

the appropriate constraints are placed on the ARCH model such that the parameters satisfy the constraints described in Nelson and Cao (1992).

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any.

Source

`arch.src`

Purpose

Computes ARCH conditional variances.

Library

`fanpac`

Format

h = **arch_cv**(b,q);

Input

b $K \times 1$ vector, coefficients.

q scalar, order of ARCH parameters.

Output

h $N \times 1$ vector, conditional variances.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times k$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if ARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if ARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_q ARCH parameters,

Procedure Reference

constant in time series equation,

regression coefficients, if any,

ARCH-in-mean coefficient, if any,

ARCH-in-cv coefficients, if any.

The ARCH model cannot be both ARCH-in-mean and ARCH-in-CV.

Source

`arch.src`

Purpose

Computes ARCH standardized residuals.

Library

`fanpac`

Format

h = **arch_sr(**b,q**);**

Input

b $K \times 1$ vector, coefficients.

q scalar, order of ARCH parameters.

Output

h $N \times 1$ vector, standardized residuals.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if ARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if ARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ARCH-in-mean coefficient, if any,

ARCH-in-cv coefficients, if any.

The ARCH model cannot be both ARCH-in-mean and ARCH-in-CV.

Source

`arch.src`

Purpose

Computes roots of ARCH model.

Library

`fanpac`

Format

r = **arch_roots(**b,q**);**

Input

b $K \times 1$ vector, coefficients.

q scalar, order of ARCH parameters.

Output

r $L \times 1$ vector, roots.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if ARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if ARCH-in-cv model, else zero.

Remarks

Computes roots of

$$1 - \alpha_1 Z - \alpha_2 Z^2 + \cdots + \alpha_q Z^q$$

where the α_i are the ARCH parameters.

The parameters in b are expected in the following order:

Procedure Reference

ω, constant in conditional variance equation,

p GARCH parameters,

q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ARCH-in-mean coefficient, if any,

ARCH-in-cv coefficients, if any.

The ARCH model cannot be both ARCH-in-mean and ARCH-in-CV.

Source

`arch.src`

Purpose

Computes time series and conditional variance forecasts.

Library

`fanpac`

Format

f = arima_forecast(b**,**p**,**d**,**q**,**$period$**,**xp**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of AR parameters.
d	Scalar, order of differencing.
q	Scalar, order of MA parameters.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

f	$L \times 3$ matrix, column 1 gives the lower forecast confidence limit, column 2 the forecasts, and column 3 the upper forecast confidence limits.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

p MA parameters,

q AR parameters,

constant in time series equation,

regression coefficients, if any.

Source

`fanarima.src`

Purpose

Computes Normal density ARIMA log-likelihood.

Library

`fanpac`

Format

y = **arima_n(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_p	Scalar, order of AR parameters.
_fan_d	Scalar, order of differencing.
_fan_q	Scalar, order of MA parameters.

Remarks

The parameters in b are expected in the following order:

_fan_p AR parameters,

_fan_q MA parameters,

a constant,

regression coefficients, if any.

Source

`fanarima.src`

Purpose

Computes t-density ARIMA log-likelihood.

Library

`fanpac`

Format

y = **arima_t**(b);

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_p	Scalar, order of AR parameters.
_fan_d	Scalar, order of differencing.
_fan_q	Scalar, order of MA parameters.

Remarks

The parameters in b are expected in the following order:

_fan_p AR parameters,

_fan_q MA parameters,

a constant,

regression coefficients, if any,

residual variance,

ν.

Source

`fanarima.src`

Procedure Reference

- **Purpose**

Computes ARIMA model constraints.

- **Library**

`fanpac`

- **Format**

y = **arima_ineq(**b**);**

- **Input**

b	$K \times 1$ vector, coefficients.

- **Output**

y	$L \times 1$ vector, roots.

- **Global Input**

_fan_p	Scalar, order of AR parameters.
_fan_d	Scalar, order of differencing.
_fan_q	Scalar, order of MA parameters.

- **Remarks**

Constrains the roots of the characteristic polynomials

$$1 - \beta_1 Z - \beta_2 Z^2 + \cdots + \beta_p Z^p$$

where the β_i are the MA parameters and

$$1 - \alpha_1 Z - \alpha_2 Z^2 + \cdots + \alpha_q Z^q$$

where the α_i are the AR parameters, to be outside the unit circle.

- **Remarks**

The parameters in b are expected in the following order:

_fan_p AR parameters,

_fan_q MA parameters,

regression coefficients, if any,

a constant.

- **Source**

`fanarima.src`

Purpose

Computes Normal density ARIMA standardized residuals.

Library

`fanpac`

Format

s = **arima_n_sr**(b,p,d,q);

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of AR parameters.
d	Scalar, order of differencing.
q	Scalar, order of MA parameters.

Output

s	$N \times 1$ vector, standardized residuals.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

p AR parameters,

q MA parameters,

a constant,

regression coefficients, if any.

Source

`fanarima.src`

Purpose

Computes t-density ARIMA standardized residuals.

Library

`fanpac`

Format

s = **arima_t_sr**(b,p,d,q);

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of AR parameters.
d	Scalar, order of differencing.
q	Scalar, order of MA parameters.

Output

s	$N \times 1$ vector, standardized residuals.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

p AR parameters,

q MA parameters,

a constant,

regression coefficients, if any,

residual variance,

ν.

Source

`fanarima.src`

Purpose

Computes roots of ARIMA model.

Library

`fanpac`

Format

r = **arima_roots(**b,p,d,q**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of AR parameters.
d	Scalar, order of differencing.
q	Scalar, order of MA parameters.

Output

r	$L \times 1$ vector, roots.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

Computes roots of

$$1 - \beta_1 Z - \beta_2 Z^2 + \cdots + \beta_p Z^p$$

where the β_i are the MA parameters and

$$1 - \alpha_1 Z - \alpha_2 Z^2 + \cdots + \alpha_q Z^q$$

where the α_i are the AR parameters.

Remarks

The parameters in b are expected in the following order:

p AR parameters,

q MA parameters,

a constant,

regression coefficients, if any.

Source

`fanarima.src`

Purpose

Computes time series and conditional variance forecasts for the multivariate diagonal vec ARCH model.

Library

`fanpac`

Format

{ r,s } = **bkarch_forecast**(b,q,$period$,xp);

Input

b	$K \times 1$ vector, coefficients.
q	Scalar, order of ARCH parameters.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r	$L \times 1$ vector, L period forecast of times series.
s	$L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series	$N \times L$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_IndEquations	$L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The parameters in b are expected in the following order:

ω, $L\times$ 1 vector of constants in conditional variance-covariance equation,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

Source

`march.src`

Purpose

Computes Normal density multivariate diagonal vec ARCH log-likelihood.

Library

`fanpac`

Format

y = **bkarch_n(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times L$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_q	Scalar, order of BKARCH parameters.
_fan_IndEquations	$L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any.

Source

`march.src`

Purpose

Computes t-density multivariate diagonal vec ARCH log-likelihood.

Library

`fanpac`

Format

y = **bkarch_t**(b);

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_q Scalar, order of BKARCH parameters.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The parameters in b are expected in the following order:

L ω, $L\times$ 1 vector of constants in conditional variance-covariance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

nonredundant portion of residual variance-covariance matrix,

ν.

Source

`march.src`

Purpose

Computes multivariate diagonal vec ARCH conditional variance-covariance matrices.

Library

`fanpac`

Format

h = **bkarch_cv(**b,q**);**

Input

b	$K \times 1$ vector, coefficients.
q	Scalar, order of ARCH parameters.

Output

h	$N \times L * (L+1)/2$ vector, conditional variance-covariance matrices.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_IndEquations	$L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The variance-covariance matrix for the t-th observation is stored in transposed vech-ed form in the t-th row of h.

The parameters in b are expected in the following order:

ω, $L \times 1$ vector of constants in conditional variance-covariance equation,

q ARCH parameters,

constant in time series equation,

regression coefficients, if any.

Source

`march.src`

Procedure Reference

Purpose

Computes ARCH standardized residuals.

Library

`fanpac`

Format

s = **bkarch_sr(**b,q**);**

Input

b — $K \times 1$ vector, coefficients.

q — Scalar, order of ARCH parameters.

Output

s — $N \times L$ vector, standardized residuals.

Global Input

_fan_Series — $N \times 1$ vector, time series.

_fan_IndVars — $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations — $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ vector of constants in conditional variance-covariance equation,

q ARCH parameters,

constant in time series equation,

regression coefficients, if any.

Source

`march.src`

Purpose

Computes time series and conditional variance forecasts for the multivariate diagonal vec GARCH model.

Library

`fanpac`

Format

{ r,s } = **bkgarch_forecast(**b,p,q,$period$,xp**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r	$L \times 1$ vector, L period forecast of times series.
s	$L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series	$N \times L$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_IndEquations	$L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The parameters in b are expected in the following order:

Procedure Reference

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

Source

`bkgarch.src`

Purpose

Computes Normal density multivariate diagonal vec GARCH log-likelihood.

Library

`fanpac`

Format

y = **bkgarch_n(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times L$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_p	Scalar, order of GARCH parameters.
_fan_q	Scalar, order of ARCH parameters.
_fan_IndEquations	$L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

Source

`bkgarch.src`

Purpose

Computes t-density multivariate diagonal vec GARCH log-likelihood.

Library

`fanpac`

Format

y = **bkgarch_t**(b);

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_p	Scalar, order of GARCH parameters.
_fan_q	Scalar, order of ARCH parameters.
_fan_IndEquations	$L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

nonredundant portion of residual variance-covariance matrix,

ν.

Source

`bkgarch.src`

Purpose

Computes multivariate diagonal vec GARCH conditional variance-covariance matrices.

Library

`fanpac`

Format

h = **bkgarch_cv(**b,p,q**);**

Input

b	$K \times 1$ vector, coefficients,
p	Scalar, order of GARCH parameters,
q	Scalar, order of ARCH parameters.

Output

h	$N \times L * (L+1)/2$ vector, conditional variance-covariance matrices.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_IndEquations	$L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.
_fan_p	Scalar, order of GARCH parameters.
_fan_q	Scalar, order of ARCH parameters.

Remarks

The variance-covariance matrix for the t-th observation is stored in transposed vech-ed form in the t-th row of h.

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

Source

`bkgarch.src`

Procedure Reference

Purpose

Computes GARCH standardized residuals.

Library

`fanpac`

Format

s = **bkgarch_sr**(b,p,q);

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

s	$N \times L$ vector, standardized residuals.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_IndEquations	$L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

Source

`bkgarch.src`

Purpose

Computes time series and conditional variance forecasts for the multivariate constant correlation diagonal vec ARCH model.

Library

`fanpac`

Format

{ *r*,*s* } = **cdvarch_forecast(***b*,*q*,*period*,*xp***);**

Input

b — $K \times 1$ vector, coefficients.

q — Scalar, order of ARCH parameters.

period — Scalar, number of periods to be forecast.

xp — $M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r — $L \times 1$ vector, L period forecast of times series.

s — $L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series — $N \times L$ vector, time series.

_fan_IndVars — $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations — $L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations — $L \times K$ matrix, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean — Scalar, nonzero if CDVARCH-in-mean model, else zero.

_fan_inCV — Scalar, nonzero if CDVARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVARCH-in-cv coefficients, if any.

Source

`cdvarch.src`

Purpose

Computes Normal density multivariate constant correlation diagonal vec ARCH log-likelihood.

Library

`fanpac`

Format

y = **cdvarch_n(**b**);**

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times L$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_q Scalar, order of CDVARCH parameters.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean nonzero if CDVARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVARCH-in-cv coefficients, if any.

The CDVARCH model cannot be both CDVARCH-in-mean and CDVARCH-in-CV.

Source

`cdvarch.src`

Purpose

Computes t-density multivariate constant correlation diagonal vec ARCH log-likelihood.

Library

`fanpac`

Format

y = **cdvarch_t(**b**);**

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_q Scalar, order of CDVARCH parameters.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVARCH-in-cv coefficients, if any,

nonredundant portion of residual variance-covariance matrix,

ν.

■ Source

`cdvarch.src`

Purpose

Computes multivariate constant correlation diagonal vec ARCH conditional variance-covariance matrices.

Library

`fanpac`

Format

h = **cdvarch_cv(**b,q**);**

Input

b	$K \times 1$ vector, coefficients.
q	Scalar, order of ARCH parameters.

Output

h	$N \times L * (L+1)/2$ vector, conditional variance-covariance matrices.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVARCH-in-cv model, else zero.

Remarks

The variance-covariance matrix for the t-th observation is stored in transposed vech-ed form in the t-th row of h.

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L \times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVARCH-in-cv coefficients, if any.

The CDVARCH model cannot be both CDVARCH-in-mean and CDVARCH-in-CV.

Source

`cdvarch.src`

Purpose

Computes standardized residuals for constant correlation diagonal vec ARCH model.

Library

`fanpac`

Format

s = **cdvarch_sr(**b,q**);**

Input

b $K \times 1$ vector, coefficients.

q Scalar, order of ARCH parameters.

Output

s $N \times L$ vector, standardized residuals.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVARCH-in-cv coefficients, if any.

The CDVARCH model cannot be both CDVARCH-in-mean and CDVARCH-in-CV.

Source

`cdvarch.src`

Purpose

Computes time series and conditional variance forecasts for the multivariate constant correlation diagonal vec GARCH model.

Library

`fanpac`

Format

{ r,s } = **cdvgarch_forecast(**b,p,q,$period$,xp**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r	$L \times 1$ vector, L period forecast of times series.
s	$L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series $N \times L$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ matrix, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVGARCH-in-cv coefficients, if any.

Source

`mgarch.src`

Purpose

Computes Normal density multivariate constant correlation diagonal vec GARCH log-likelihood.

Library

`fanpac`

Format

y = **cdvgarch_n(**b**);**

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times L$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_p Scalar, order of GARCH parameters.

_fan_q Scalar, order of ARCH parameters.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean nonzero if CDVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

Procedure Reference

ω, $L \times 1$ constant vector in conditional variance equation,

$L \times$ **_fan_p** GARCH parameters,

$L \times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVGARCH-in-cv coefficients, if any.

The CDVGARCH model cannot be both CDVGARCH-in-mean and CDVGARCH-in-CV.

Source

`mgarch.src`

Purpose

Computes t-density multivariate constant correlation diagonal vec GARCH log-likelihood.

Library

`fanpac`

Format

y = **cdvgarch_t**(b);

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_p Scalar, order of GARCH parameters.

_fan_q Scalar, order of ARCH parameters.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVGARCH-in-cv coefficients, if any,

nonredundant portion of residual variance-covariance matrix,

ν.

Source

`mgarch.src`

Purpose

Computes multivariate constant correlation diagonal vec GARCH conditional variance-covariance matrices.

Library

`fanpac`

Format

h = **cdvgarch_cv(**b,p,q**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

h	$N \times L * (L+1)/2$ vector, conditional variance-covariance matrices.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The variance-covariance matrix for the t-th observation is stored in transposed vech-ed form in the t-th row of h.

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVGARCH-in-cv coefficients, if any.

The CDVGARCH model cannot be both CDVGARCH-in-mean and CDVGARCH-in-CV.

Source

`mgarch.src`

Purpose

Computes standardized residuals for constant correlation diagonal vec GARCH model.

Library

`fanpac`

Format

s = **cdvgarch_sr(**b,p,q**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

s	$N \times L$ vector, standardized residuals.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

Procedure Reference

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, CDVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, CDVGARCH-in-cv coefficients, if any.

The CDVGARCH model cannot be both CDVGARCH-in-mean and CDVGARCH-in-CV.

Source

`mgarch.src`

Purpose

Computes time series and conditional variance forecasts for the multivariate diagonal vec ARCH model.

Library

`fanpac`

Format

{ *r*,*s* } = **dvarch_forecast(***b*,*q*,*period*,*xp***);**

Input

b	$K \times 1$ vector, coefficients.
q	Scalar, order of ARCH parameters.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r	$L \times 1$ vector, L period forecast of times series.
s	$L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series $N \times L$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ matrix, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if DVARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if DVARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVARCH-in-cv coefficients, if any.

Source

`dvarch.src`

Purpose

Computes Normal density multivariate diagonal vec ARCH log-likelihood.

Library

`fanpac`

Format

y = **dvarch_n(**b**);**

Input

b — $K \times 1$ vector, coefficients.

Output

y — $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series — $N \times L$ vector, time series.

_fan_IndVars — $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_q — Scalar, order of dvarch parameters.

_fan_IndEquations — $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations — $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean — nonzero if DVARCH-in-mean model, else zero.

_fan_inCV — Scalar, nonzero if DVARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVARCH-in-cv coefficients, if any.

The DVARCH model cannot be both DVARCH-in-mean and DVARCH-in-CV.

Source

`dvarch.src`

Purpose

Computes t-density multivariate diagonal vec ARCH log-likelihood.

Library

`fanpac`

Format

y = **dvarch_t**(b);

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_q Scalar, order of dvarch parameters.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if DVARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if DVARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_q** ARCH parameters,

Procedure Reference

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVARCH-in-cv coefficients, if any.

nonredundant portion of residual variance-covariance matrix,

ν.

Source

`dvarch.src`

Purpose

Computes multivariate diagonal vec ARCH conditional variance-covariance matrices.

Library

`fanpac`

Format

h = **dvarch_cv(**b,q**);**

Input

b	$K \times 1$ vector, coefficients.
q	Scalar, order of ARCH parameters.

Output

h	$N \times L * (L+1)/2$ vector, conditional variance-covariance matrices.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if DVARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if DVARCH-in-cv model, else zero.

Remarks

The variance-covariance matrix for the t-th observation is stored in transposed vech-ed form in the t-th row of h.

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVARCH-in-cv coefficients, if any.

The DVARCH model cannot be both DVARCH-in-mean and DVARCH-in-CV.

Source

`dvarch.src`

Purpose

Computes ARCH standardized residuals.

Library

`fanpac`

Format

s = **dvarch_sr(**b,q**);**

Input

b — $K \times 1$ vector, coefficients.

q — Scalar, order of ARCH parameters.

Output

s — $N \times L$ vector, standardized residuals.

Global Input

_fan_Series — $N \times 1$ vector, time series.

_fan_IndVars — $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations — $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations — $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean — Scalar, nonzero if DVARCH-in-mean model, else zero.

_fan_inCV — Scalar, nonzero if DVARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L \times$ **_fan_q** ARCH parameters,

Procedure Reference

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVARCH-in-cv coefficients, if any.

The DVARCH model cannot be both DVARCH-in-mean and DVARCH-in-CV.

Source

`dvarch.src`

Purpose

Computes time series and conditional variance forecasts for the multivariate diagonal vec GARCH model.

Library

`fanpac`

Format

{ r,s } = **dvgarch_forecast(**b,p,q,$period$,xp**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r	$L \times 1$ vector, L period forecast of times series.
s	$L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series $N \times L$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ matrix, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if DVGARCH-in-mean model, else zero.

Procedure Reference

_fan_inCV Scalar, nonzero if DVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVGARCH-in-cv coefficients, if any.

Source

`mgarch.src`

Purpose

Computes Normal density multivariate diagonal vec GARCH log-likelihood.

Library

`fanpac`

Format

y = **dvgarch_n(**b**);**

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times L$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_p Scalar, order of GARCH parameters.

_fan_q Scalar, order of ARCH parameters.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean nonzero if DVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if DVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVGARCH-in-cv coefficients, if any.

The DVGARCH model cannot be both DVGARCH-in-mean and DVGARCH-in-CV.

Source

`mgarch.src`

Purpose

Computes t-density multivariate diagonal vec GARCH log-likelihood.

Library

`fanpac`

Format

y = **dvgarch_t(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_p Scalar, order of GARCH parameters.

_fan_q Scalar, order of ARCH parameters.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if DVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if DVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

Procedure Reference

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVGARCH-in-cv coefficients, if any.

nonredundant portion of residual variance-covariance matrix,

ν.

Source

`mgarch.src`

Purpose

Computes multivariate diagonal vec GARCH conditional variance-covariance matrices.

Library

`fanpac`

Format

h = **dvgarch_cv(**b,p,q**);**

Input

b $K \times 1$ vector, coefficients.

p Scalar, order of GARCH parameters.

q Scalar, order of ARCH parameters.

Output

h $N \times L * (L+1)/2$ vector, conditional variance-covariance matrices.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_p Scalar, order of GARCH parameters.

_fan_q Scalar, order of ARCH parameters.

_fan_inMean Scalar, nonzero if DVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if DVGARCH-in-cv model, else zero.

Remarks

The variance-covariance matrix for the t-th observation is stored in transposed vech-ed form in the t-th row of h.

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVGARCH-in-cv coefficients, if any.

The DVGARCH model cannot be both DVGARCH-in-mean and DVGARCH-in-CV.

Source

`mgarch.src`

Purpose

Computes GARCH standardized residuals.

Library

`fanpac`

Format

s = **dvgarch_sr(**b,p,q**);**

Input

b $K \times 1$ vector, coefficients.

p Scalar, order of GARCH parameters.

q Scalar, order of ARCH parameters.

Output

s $N \times L$ vector, standardized residuals.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if DVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if DVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, $L \times 1$ constant vector in conditional variance equation,

$L\times$ **_fan_p** GARCH parameters,

$L\times$ **_fan_q** ARCH parameters,

$L \times 1$ constant vector in time series equation,

regression coefficients, if any.

$L \times 1$ vector, DVGARCH-in-mean coefficients, if any,

$L \times K$ matrix, DVGARCH-in-cv coefficients, if any.

The DVGARCH model cannot be both DVGARCH-in-mean and DVGARCH-in-CV.

Source

`mgarch.src`

Purpose

Computes log-likelihood for exponential GARCH model with generalized error density.

Library

`fanpac`

Format

y = **garch_e(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_p	Scalar, order of GARCH parameters.
_fan_q	Scalar, order of ARCH parameters.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ρ,

ϕ.

Source

`egarch.src`

Purpose

Computes time series and conditional variance forecasts.

Library

`fanpac`

Format

{ r,s } = **garch_e_forecast(**b,p,q,$period$,xp**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r	$L \times 1$ vector, L period forecast of times series.
s	$L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ρ,

ϕ.

Source

`egarch.src`

Purpose

Computes gradient of log-likelihood for exponential GARCH model with generalized error density.

Library

`fanpac`

Format

y = **garch_e_grd(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times K$ matrix, gradient matrix.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_p	Scalar, order of GARCH parameters.
_fan_q	Scalar, order of ARCH parameters.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ρ,

ϕ.

Source

`egarch.src`

Purpose

Computes EGARCH conditional variances.

Library

`fanpac`

Format

h = **garch_e_cv**(b,p,q);

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

h	$N \times 1$ vector, conditional variances.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

p GARCH parameters,

q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ρ,

ϕ.

Source

`egarch.src`

Purpose

Computes EGARCH standardized residuals.

Library

`fanpac`

Format

s = **garch_e_sr**(b,p,q);

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

s	$N \times 1$ vector, standardized residuals.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

p GARCH parameters,

q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

ρ,

ϕ.

Source

`egarch.src`

Purpose

Computes time series and conditional variance forecasts.

Library

`fanpac`

Format

{ r,s } = **garch_fi_forecast**(b,p,q,$period$,xp);

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r	$L \times 1$ vector, L period forecast of time series.
s	$L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ matrix, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

d, dimension parameter,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

Source

`figarch.src`

Purpose

Computes Normal density FIGARCH log-likelihood.

Library

`fanpac`

Format

y = **garch_fi_n(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.
_fan_p	Scalar, order of GARCH parameters.
_fan_q	Scalar, order of ARCH parameters.
_fan_Init	Scalar, number of lags not included in likelihood.
_fan_IndEquations	$L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.
_fan_CVIndEquations	$L \times K$ matrix, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.
_fan_inMean	Scalar, nonzero if CDVGARCH-in-mean model, else zero.
_fan_inCV	Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

Procedure Reference

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

d, dimension parameter,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

Source

`figarch.src`

Purpose

Computes t-density FIGARCH log-likelihood.

Library

`fanpac`

Format

y = **garch_fi_t(**b**);**

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_p Scalar, order of GARCH parameters.

_fan_q Scalar, order of ARCH parameters.

_fan_Init Scalar, number of lags not included in likelihood

_fan_IndEquations $L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ matrix, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

d, dimension parameter,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

ν.

Source

`figarch.src`

Purpose

Computes FIGARCH conditional variances.

Library

`fanpac`

Format

h = **garch_fi_cv**(b,p,q);

Input

b $K \times 1$ vector, coefficients.

p Scalar, order of GARCH parameters.

q Scalar, order of ARCH parameters.

Output

h $N \times 1$ vector, conditional variances.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $L \times K$ matrix, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $L \times K$ matrix, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if CDVGARCH-in-mean model, else zero.

_fan_inCV Scalar, nonzero if CDVGARCH-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

p GARCH parameters,

q ARCH parameters,

d, dimension parameter,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

Source

`figarch.src`

Purpose

Computes FIGARCH standardized residuals.

Library

`fanpac`

Format

s = **garch_fi_sr**(b,p,q);

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

s	$N \times 1$ vector, standardized residuals.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

p GARCH parameters,

q ARCH parameters,

d, dimension parameter,

constant in time series equation,

regression coefficients, if any.

Source

`figarch.src`

Purpose

Computes time series and conditional variance forecasts.

Library

`fanpac`

Format

{ *r*,*s* } = **garch_forecast**(*b*,*p*,*q*,*period*,*xp*);

Input

b — $K \times 1$ vector, coefficients.

p — Scalar, order of GARCH parameters.

q — Scalar, order of ARCH parameters.

period — Scalar, number of periods to be forecast.

xp — $M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.

Output

r — $L \times 1$ vector, L period forecast of time series.

s — $L \times 1$ vector, L period forecast of conditional variance.

Global Input

_fan_Series — $N \times 1$ vector, time series.

_fan_IndVars — $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations — $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations — $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean — Scalar, nonzero if Garch-in-mean model, else zero.

_fan_inCV	Scalar, nonzero if Garch-in-cv model, else zero.
_fan_p	Scalar, order of GARCH parameters.
_fan_q	Scalar, order of ARCH parameters.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

The Garch model cannot be both Garch-in-mean and Garch-in-CV.

Source

`garch.src`

Purpose

Computes Normal density GARCH log-likelihood.

Library

`fanpac`

Format

y = **garch_n(**b**);**

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_p Scalar, order of GARCH parameters.

_fan_q Scalar, order of ARCH parameters.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if Garch-in-mean model, else zero.

_fan_inCV Scalar, nonzero if Garch-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

The Garch model cannot be both Garch-in-mean and Garch-in-CV.

Source

`garch.src`

Purpose

Computes t-density GARCH log-likelihood.

Library

`fanpac`

Format

y = **garch_t(**b**);**

Input

b $K \times 1$ vector, coefficients.

Output

y $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_p Scalar, order of GARCH parameters.

_fan_q Scalar, order of ARCH parameters.

_fan_inMean Scalar, nonzero if Garch-in-mean model, else zero.

_fan_inCV Scalar, nonzero if Garch-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

ν.

The Garch model cannot be both Garch-in-mean and Garch-in-CV.

Source

`garch.src`

Purpose

Computes GARCH model Nelson and Cao constraints.

Library

`fanpac`

Format

y = **garch_ineq(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$L \times 1$ vector, roots.

Global Input

_fan_p	Scalar, order of GARCH parameters.
_fan_q	Scalar, order of ARCH parameters.

Remarks

Computes Nelson and Cao (1992) constraint function. When the statement

```
_nlp_IneqProc = &garch_ineq;
```

the appropriate constraints are placed on the GARCH model such that the parameters satisfy the constraints described in Nelson and Cao (1992).

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

_fan_p GARCH parameters,

_fan_q ARCH parameters,

constant in time series equation,

regression coefficients, if any.

Source

`garch.src`

Purpose

Computes GARCH conditional variances.

Library

`fanpac`

Format

h = **garch_cv(**b,p,q**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

h	$N \times 1$ vector, conditional variances.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if Garch-in-mean model, else zero.

_fan_inCV Scalar, nonzero if Garch-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

p GARCH parameters,

q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

The Garch model cannot be both Garch-in-mean and Garch-in-CV.

Source

`garch.src`

Purpose

Computes GARCH standardized residuals.

Library

`fanpac`

Format

s = **garch_sr(**b,p,q**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

s	$N \times 1$ vector, standardized residuals.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

_fan_IndEquations $1 \times K$ vector, specification matrix for independent variables. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_CVIndEquations $1 \times K$ vector, specification matrix for independent variables in conditional variance equation. If element is nonzero, a coefficient is estimated, otherwise not.

_fan_inMean Scalar, nonzero if Garch-in-mean model, else zero.

_fan_inCV Scalar, nonzero if Garch-in-cv model, else zero.

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

p GARCH parameters,

q ARCH parameters,

constant in time series equation,

regression coefficients, if any,

Garch-in-mean coefficient, if any,

Garch-in-cv coefficients, if any.

The Garch model cannot be both Garch-in-mean and Garch-in-CV.

■ Source

`garch.src`

Purpose

Computes roots of GARCH model.

Library

`fanpac`

Format

r = **garch_roots(**b,p,q**);**

Input

b	$K \times 1$ vector, coefficients.
p	Scalar, order of GARCH parameters.
q	Scalar, order of ARCH parameters.

Output

r	$L \times 1$ vector, roots.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

Computes roots of

$$1 - (\alpha_1 + \beta_1)Z - (\alpha_2 + \beta_2)Z^2 + \cdots$$

where the β_i are the GARCH parameters and where the α_i are the ARCH parameters, and

$$1 - \beta_1 Z - \beta_2 Z^2 + \cdots + \beta_p Z^p$$

Remarks

The parameters in b are expected in the following order:

ω, constant in conditional variance equation,

p GARCH parameters,

q ARCH parameters,

constant in time series equation,

regression coefficients, if any.

Source

`garch.src`

Purpose

Computes time series and conditional variance forecasts.

Library

`fanpac`

Format

r = **ols_forecast(** b,*period*,*xp*,*vc*,*density* **);**

Input

b	$K \times 1$ vector, coefficients.
period	Scalar, number of periods to be forecast.
xp	$M \times K$ matrix, forecast independent variables. If there are independent variables but no forecast independent variables, set $xp = 0$, and the means of the independent variables will be used for forecast.
vc	K×K matrix, covariance matrix of parameters.
density	Scalar, if 0, Normal density, else t-density.

Output

r	$L \times 1$ vector, L period forecast of time series.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

regression coefficients,

residual variance, if t-density,

ν, if t-density.

Source

`fanols.src`

Purpose

Computes t-density ordinary least squares log-likeihood.

Library

`fanpac`

Format

y = **ols_t(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

regression coefficients,

residual variance,

ν.

Source

`fanols.src`

Purpose

Computes gradient of t-density ordinary least squares log-likeihood.

Library

`fanpac`

Format

y = **ols_t_grd(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

y	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

regression coefficients,

residual variance,

ν.

Source

`fanols.src`

Purpose

Computes standardized residuals from Normal density ordinary least squares model.

Library

`fanpac`

Format

s = **ols_n_sr(**b**);**

Input

b	$K \times 1$ vector, coefficients.

Output

s	$N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series	$N \times 1$ vector, time series.
_fan_IndVars	$N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

regression coefficients.

Source

`fanols.src`

Purpose

Computes standardized residuals from t-density ordinary least squares model.

Library

`fanpac`

Format

s = **ols_t_sr**(b);

Input

b $K \times 1$ vector, coefficients.

Output

s $N \times 1$ vector, minus log-likelihood.

Global Input

_fan_Series $N \times 1$ vector, time series.

_fan_IndVars $N \times K$ matrix, independent variables. If none, set to missing value.

Remarks

The parameters in b are expected in the following order:

regression coefficients,

residual variance,

ν.

Source

`fanols.src`

Chapter 5

NLP Reference

Purpose

Minimizes a function subject to general constraints on parameters.

Library

`fanpac`

Format

{ x,f,g,*retcode* } = **NLP(&***fct*,*start***);**

Input

&*fct*	A pointer to a procedure that returns the function evaluated at the parameters.
start	$K \times 1$ vector, start values.

Output

x	$K \times 1$ vector, estimated parameters.
f	Scalar, function at minimum.
g	$K \times 1$ vector, gradient evaluated at x.
retcode	Scalar, return code. If normal convergence is achieved, then $retcode = 0$, otherwise a positive integer is returned indicating the reason for the abnormal termination:

0	normal convergence
1	forced exit
2	maximum iterations exceeded
3	function calculation failed
4	gradient calculation failed
5	Hessian calculation failed
6	line search failed
7	function cannot be evaluated at initial parameter values
8	error with gradient
9	error with constraints
10	secant update failed
11	maximum time exceeded

13 quadratic program failed

20 Hessian failed to invert

99 termination condition unknown

Globals

_nlp_A $M_1 \times K$ matrix. Linear equality constraint coefficient matrix **_nlp_A** is used with **_nlp_B** to specify linear equality constraints:

```
_nlp_A * X = _nlp_B
```

where X is the $K \times 1$ unknown parameter vector.

_nlp_Active Vector, defines fixed/active coefficients. This global allows you to fix a parameter to its starting value. This is useful, for example, when you wish to try different models with different sets of parameters without having to re-edit the function. When it is to be used, it must be a vector of the same length as the starting vector. Set elements of **_nlp_Active** to 1 for an active parameter, and to zero for a fixed one.

_nlp_Algorithm Scalar, selects optimization method:

1 BFGS – Broyden, Fletcher, Goldfarb, Shanno method

2 DFP – Davidon, Fletcher, Powell method

3 NEWTON – Newton-Raphson method

4 scaled BFGS

5 scaled DFP

Default = 3

_nlp_Delta Scalar, floor for eigenvalues of Hessian in the NEWTON algorithm. When nonzero, the eigenvalues of the Hessian are augmented to this value.

_nlp_B $M_1 \times 1$ vector. Linear equality constraint constant vector **_nlp_B** is used with **_nlp_A** to specify linear equality constraints:

```
_nlp_A * X = _nlp_B
```

where X is the $K \times 1$ unknown parameter vector.

_nlp_Bounds $K \times 2$ matrix, bounds on parameters. The first column contains the lower bounds, and the second column the upper bounds. If the bounds for all the coefficients are the same, a 1x2 matrix may be used. Default = { -1e256 1e256 }.

_nlp_C $M_3 \times K$ matrix. Linear inequality constraint coefficient matrix **_nlp_C** is used with **_nlp_D** to specify linear inequality constraints:

```
_nlp_C * X = _nlp_D
```

where X is the $K \times 1$ unknown parameter vector.

_nlp_D $M_3 \times 1$ vector. Linear inequality constraint constant vector . **_nlp_D** is used with **_nlp_C** to specify linear inequality constraints:

```
_nlp_C * X = _nlp_D
```

where X is the $K \times 1$ unknown parameter vector.

_nlp_Diagnostic scalar:

0 Nothing is stored or printed.

1 Current estimates, gradient, direction function value, Hessian, and step length are printed to the screen.

2 The current quantities are stored in **_nlp_Diagnostic** using the **VPUT** command. Use the following strings to extract from **_nlp_Diagnostic** using **VREAD**:

function	"function"
estimates	"params"
direction	"direct"
Hessian	"hessian"
gradient	"gradient"
step	"step"

_nlp_DirTol Scalar, convergence tolerance for gradient of estimated coefficients. When this criterion has been satisifed, **NLP** exits the iterations. Default = 1e-5.

_nlp_EqJacobian Scalar, pointer to a procedure that computes the Jacobian of the nonlinear equality constraints with respect to the parameters. The procedure has one input argument, the $K \times 1$ vector of parameters, and one output argument, the $M_2 \times K$ matrix of derivatives of the constraints with respect to the parameters. For example, if the nonlinear equality constraint procedure was

```
proc eqproc(p);
    retp(p[1]*p[2]-p[3]);
endp;
```

the Jacobian procedure and assignment to the global would be

```
proc eqj(p);
    retp(p[2]~p[1]~-1);
endp;

_nlp_EqJacobian = &eqj;
```

_nlp_EqProc Scalar, pointer to a procedure that computes the nonlinear equality constraints. For example, the statement

```
_nlp_EqProc = &eqproc;
```

tells **NLP** that nonlinear equality constraints are to be placed on the parameters and where the procedure computing them is to be found. The procedure must have one input argument, the $K \times 1$ vector of parameters, and one output argument, the $M_2 \times 1$ vector of computed constraints that are to be equal to zero. For example, suppose that you wish to place the following constraint:

```
P[1] * P[2] = P[3]
```

The procedure for this is:

```
proc eqproc(p);
    retp(p[1]*[2]-p[3]);
endp;
```

_nlp_FeasibleTest Scalar. If nonzero, testing for feasibility in the line search is turned off.

_nlp_FinalHess $K \times K$ matrix. The Hessian used to compute the covariance matrix of the parameters is stored in **_nlp_FinalHess**. This is most useful if the inversion of the Hessian fails, which is indicated when **NLP** returns a missing value for the covariance matrix of the parameters. An analysis of the Hessian stored in **_nlp_FinalHess** can then reveal the source of the linear dependency responsible for the singularity.

_nlp_GradCheckTol Scalar. Tolerance for the deviation of numerical and analytical gradients when procedures exist for the computation of analytical gradients, Hessians, and/or Jacobians. If set to zero, the analytical gradients will not be compared to their numerical versions. When adding procedures for computing analytical gradients, it is highly recommended that you perform the check. Set **_nlp_GradCheckTol** to some small value (1e-3, say) when checking. It may have to be set larger if the numerical gradients are poorly computed to make sure that **NLP** doesn't fail when the analytical gradients are being properly computed.

_nlp_GradMethod Scalar, method for computing numerical gradient:

0 central difference

1 forward difference (default)

_nlp_GradProc Scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. For example, the statement

```
_nlp_GradProc=&gradproc;
```

tells **NLP** that a gradient procedure exists, as well as, where to find it. The user-provided procedure has one input argument, the $K \times 1$ vector of parameter values. The procedure returns a single output argument, the $K \times 1$ vector of gradients of the function with respect to the parameters evaluated at the vector of parameter values.

For example, suppose the function is $b_1 \exp -b_2$, then the following would be added to the command file:

```
proc lgd(b);
    retp(exp(-b[2])|-b[1]*b[2]*exp(-b[2]));
endp;

_nlp_GradProc = &lgd;
```

Default = 0; i.e., no gradient procedure has been provided.

_nlp_GradStep Scalar, or 1×2, or $K \times 1$, or $K \times 2$ matrix, increment size for computing gradient and/or Hessian. If scalar, step size will be value times parameter estimates for the numerical gradient. If 1×2, the first element is multiplied times parameter value for gradient, and second element the same for the Hessian. If $K \times 1$, the step size for the gradient will be the elements of the vector; i.e., it will not be multiplied times the parameters. If $K \times 2$, the second column sets the step sizes for the Hessian.

When the numerical gradient is not performing well, set to a larger value (1e-3, say). Default is the cube root of machine precision.

_nlp_HessProc Scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. For example, the instruction

```
_nlp_HessProc = &hessproc;
```

tells **NLP** that a procedure has been provided for the computation of the Hessian and where to find it. The procedure that is provided by the user must have one input argument, the $K \times 1$ vector of parameter values. The procedure returns a single output argument, the $K \times K$ symmetric matrix of second order derivatives of the function evaluated at the parameter values.

_nlp_IneqJacobian Scalar, pointer to a procedure that computes the Jacobian of the nonlinear equality constraints with respect to the parameters. The procedure has one input argument, the $K \times 1$ vector of parameters, and one output argument, the $M_4 \times K$ matrix of derivatives of the constraints with respect to the parameters. For example, if the nonlinear equality constraint procedure was

```
proc ineqproc(p);
    retp(p[1]*p[2]-p[3]);
endp;
```

the Jacobian procedure and assignment to the global would be

```
proc ineqj(p);
    retp(p[2]~p[1]~-1);
endp;

_nlp_IneqJacobian = &ineqj;
```

_nlp_IneqProc Scalar, pointer to a procedure that computes the nonlinear inequality constraints. For example, the statement

```
_nlp_IneqProc = &ineqproc;
```

tells **NLP** that nonlinear equality constraints are to be placed on the parameters and where the procedure computing them is to be found. The procedure must have one input argument, the $K \times 1$ vector of parameters, and one output argument, the $M_4 \times 1$ vector of computed constraints that are to be equal to zero. For example, suppose that you wish to place the following constraint:

```
P[1] * P[2] >= P[3]
```

The procedure for this is:

```
proc ineqproc(p);
    retp(p[1]*[2]-p[3]);
endp;
```

_nlp_IterInfo 2x1 vector, contains information about the iterations. The first element contains the number of iterations, the second element contains the elapsed time in minutes of the iterations.

_nlp_Lagrange Vector, created using **VPUT**. Contains the Lagrangean coefficients for the constraints. They may be extracted with the **VREAD** command using the following strings:

"lineq"	linear equality constraints
"nlineq"	nonlinear equality constraints
"linineq"	linear inequality constraints
"nlinineq"	nonlinear inequality constraints
"bounds"	bounds

When an inequality or bounds constraint is active, its associated Lagrangean is nonzero. The linear Lagrangeans preceed the nonlinear Lagrangeans in the covariance matrices.

_nlp_LineSearch Scalar, selects method for conducting line search. The result of the line search is a *step length*; i.e., a number that reduces the function value when multiplied times the direction..

1 step length = 1

- **2** cubic or quadratic step length method (STEPBT)
- **3** step halving (HALF)
- **4** Brent's step length method (BRENT)

Default = 2.

Usually **_nlp_LineSearch** = 2 is best. If the optimization bogs down, try setting **_nlp_LineSearch** = 1, 4, or 5. **_nlp_LineSearch** = 3 generates slower iterations but faster convergence, and **_nlp_LineSearch** = 1 generates faster iterations but slower convergence.

_nlp_MaxIters Scalar, maximum number of iterations.

_nlp_MaxTime Scalar, maximum time in iterations in minutes. Default = 1e+5, about 10 weeks.

_nlp_MaxTry Scalar, maximum number of tries to find step length that produces a descent.

_nlp_Options Character vector, specification of options. This global permits setting various **NLP** options in a single global using identifiers. For example,

```
_nlp_Options = { newton brent trust central file };
```

sets the line search method to BRENT, the descent method to NEWTON, trust region on, the numerical gradient method to central differences.

The following is a list of the identifiers:

Algorithms BFGS, DFP, NEWTON, BFGS-SC, DFP-SC

Line Search ONE, STEPBT, HALF, BRENT

Trust Method TRUST

Gradient method CENTRAL, FORWARD

Output method NONE, FILE, SCREEN

_nlp_ParNames $K \times 1$ character vector, parameter labels.

_nlp_Switch 4×1 or 4×2 vector, algorithm switching. If 4×1, row number

- **1** algorithm number to switch to
- **2** **NLP** switches if function changes less than this amount
- **3** **NLP** switches if this number of iterations is exceeded
- **4** **NLP** switches if line search step changes less than this amount

else if 4×2, **NLP** switches between the algorithm defined in row 1, column 1, and that defined in row 1, column 2.

_nlp_Trust Scalar. If nonzero, the trust region method is turned on. Default = 0.

_nlp_TrustRadius Scalar. The trust region if the trust region method is turned on. Default = .01.

_nlp_title String. Title of run

Remarks

Specifying Constraints.

There are five types of constraints: linear equality, linear inequality, nonlinear equality, nonlinear inequality, and bounds. Linear constraints are specified by initializing the appropriate **NLP** globals to known matrices of constants. The linear equality constraint matrices are **_nlp_A** and **_nlp_B**, and they assume the following relationship with the parameter vector:

```
_nlp_A * x = _nlp_B
```

where `x` is the parameter vector.

Similarly, the linear *in*equality constraint matrices are **_nlp_C** and **_nlp_D**, and assume the following relationship with the parameter vector:

```
_nlp_C * x >= _nlp_D
```

The nonlinear constraints are specified by providing procedures and assigning their pointers to **NLP** globals. These procedures take a single argument, the vector of parameters, and return a column vector of evaluations of the constraints at the parameters. Each element of the column vector is a separate constraint.

For example, suppose you wish to constrain the product of the first and third coefficients to be equal to 10, and the squared second and fourth coefficients to be equal to the squared fifth coefficient:

```
proc eqp(x);
    local c;
    c = zeros(2,1);
    c[1] = x[1] * x[3] - 10;
    c[2] = x[2] * x[2] + x[4] * x[4] - x[5] * x[5];
    retp(c);
endp;

 _nlp_EqProc = &eqp;
```

The nonlinear equality constraint procedure causes **NLP** to find estimates for which its evaluation is equal to a conformable vector of zeros.

The nonlinear *in*equality constraint procedure is similar to the equality procedure. **NLP** finds estimates for which the evaluation of the procedure is greater than or equal to zero. The nonlinear inequality constraint procedure is assigned to the global **_nlp_IneqProc**. For example, suppose you wish to constrain the norm of the coefficients to be greater than one:

```
proc ineqp(x);
    retp(x'x-3);
endp;

 _nlp_IneqProc = &ineqp;
```

Bounds are a type of linear inequality constraint. They are specified separately for computational and notational convenience. To declare bounds on the parameters, assign a two column vector with rows equal to the number of parameters to the **NLP** global **_nlp_Bounds**. The first column is the lower bounds and the second column the upper bounds. For example,

```
_nlp_Bounds = {  0 10,
              -10  0
              -10 20 };
```

If the bounds are the same for all of the parameters, only the first row is required.

Writing the Function to be Minimized

The user must provide a procedure for computing the function. The procedure has one input argument, a vector of parameters. The output is the scalar value of the function evaluated at the current value of the parameters. Suppose that the function procedure has been named *pfct*. The format of the procedure is

$f = fct(x)$**;**

where x is a column vector of parameters.

Supplying an Analytical GRADIENT Procedure

To decrease the time of computation, the user may provide a procedure for the calculation of the gradient of the function. The global variable **_nlp_GradProc** must contain the pointer to this procedure. Suppose the name of this procedure is *gradproc*. Then

$g = gradproc(x)$**;**

where the input argument is the vector of parameters and the output argument is g is column vector of gradients of the function with respect to coefficients

Providing a procedure for the calculation of the first derivatives also has a significant effect on the calculation time of the Hessian. The calculation time for the numerical computation of the Hessian is a quadratic function of the size of the matrix. For large matrices, the calculation time can be very significant. This time can be reduced to a linear function of size if a procedure for the calculation of analytical first derivatives is available. When such a procedure is available, **NLP** automatically uses it to compute the numerical Hessian.

The major problem one encounters when writing procedures to compute gradients and Hessians is in making sure that the gradient is being properly computed. **NLP** checks the gradients and Hessian when **_nlp_GradCheckTol** is nonzero. **NLP** generates both numerical and analytical gradients, and viewing the discrepancies between them can help in debugging the analytical gradient procedure.

Supplying an Analytical HESSIAN Procedure

Selection of the NEWTON algorithm becomes feasible if the user supplies a procedure to compute the Hessian. If such a procedure is provided, the global variable **_nlp_HessProc** must contain a pointer to this procedure. Suppose this procedure is called *hessproc*. The format is

$h = hessproc(x)$;

The input argument is the $K \times 1$ vector of parameter values. The output argument is the $K \times K$ matrix of second order partial derivatives evaluated at the coefficients in x.

Supplying Analytical Jacobians of the Nonlinear Constraints

At each iteration the Jacobians of the nonlinear constraints, if they exist, are computed numerically. This is time-consuming and generates a loss of precision. For models with a large number of inequality constraints a significant speed-up can be achieved by providing analytical Jacobian procedures. The improved accuracy can also have a significant effect on convergence.

The Jacobian procedures take a single argument, the vector of parameters, and return a matrix of derivatives of each constraint with respect to each parameter. The rows are associated with the constraints and the columns with the parameters. The pointer to the nonlinear equality Jacobian procedure is assigned to **_nlp_EqJacobian**. The pointer to the nonlinear *in*equality Jacobian procedure is assigned to **_nlp_IneqJacobian**.

For example, suppose the following procedure computes the equality constraints:

```
 proc eqp(x);
     local c;
     c = zeros(2,1);
     c[1] = x[1] * x[3] - 10;
     c[2] = x[2] * x[2] + x[4] * x[4] - x[5] * x[5];
     retp(c);
 endp;
_nlp_EqProc = &eqp;
```

Then the Jacobian procedure would look like this:

```
 proc eqJacob(x);
    local c;
    c = zeros(2,5);
    c[1,1] = x[3];
    c[1,3] = x[1];
    c[2,2] = 2*x[2];
    c[2,4] = 2*x[4];
    c[3,5] = -2*x[5];
    retp(c);
endp;
_nlp_EqJacobian = &eqJacob;
```

The Jacobian procedure for the nonlinear inequality constraints is specified similarly, except that the associated global containing the pointer to the procedure is **_nlp_IneqJacobian**.

■ Source

nlp.src

Purpose

Resets **NLP** global variables to default values.

Library

`fanpac`

Format

NLPSet;

Input

None.

Output

None.

Remarks

Putting this instruction at the top of all command files that invoke **NLP** is generally good practice. This prevents globals from being inappropriately defined when a command file is run several times or when a command file is run after another command file has executed that calls **NLP**.

Source

`nlp.src`

Purpose

Computes covariance matrix of parameters.

Library

`fanpac`

Format

h = **NLPCovPar(**x**,&**fct**,&**grd**,**$nobs$**,**ind**);**

Input

x — $K \times 1$ vector, maximum likelihood parameter estimates.

&*fct* — A pointer to a procedure that returns minus the log-likelihood evaluated for each observation at the parameter estimates.

&*grd* — A pointer to a procedure that returns the gradient of minus the log-likelihood evaluated for each observation at the parameters. If set to zero, a numerical gradient is computed.

nobs — Scalar, number of observations in dataset.

ind — Scalar:

- *1* — Maximum likelihood covariance matrix of parameters from inverse of Hessian with correction made for constraints, if any. Requires Lagrangeans stored in **_nlp_Lagrange**.
- *2* — covariance matrix of parameters from inverse of crossproduct of Jacobian with correction made for constraints, if any. Requires Lagrangeans stored in **_nlp_Lagrange**.
- *3* — Quasi-Maximum Likelihood covariance matrix of parameters from inverse of Hessian with correction made for constraints, if any. Requires Lagrangeans stored in **_nlp_Lagrange**.
- *-1* — Maximum likelihood covariance matrix of parameters from inverse of Hessian with no correction made for constraints.
- *-2* — Covariance matrix of parameters from inverse of crossproduct of Jacobian with no correction made for constraints.
- *-3* — Quasi-Maximum Likelihood covariance matrix of parameters from inverse of Hessian with no correction made for constraints.

Output

h K × K matrix, covariance matrix of parameters

Global Input

_nlp_Lagrange Vector, created by **NLP** using **VPUT**. Contains the Lagrangean coefficients for the constraints.

Source

`nlp.src`

Purpose

Computes confidence limits of parameters by inversion of the Wald statistic.

Library

`fanpac`

Format

cl = **NLPClimits(***x*,*vc*,*nobs*,*alpha*,*sel***);**

Input

x	$K \times 1$ vector, maximum likelihood parameter estimates.
vc	$K \times K$ matrix, covariance matrix of parameter estimates.
nobs	Scalar, number of observations in dataset.
alpha	(1-alpha)% two-tailed limits are computed. Default = .95.
sel	$L \times 1$ vector, selection of parameters. If set to zero, all parameters are selected.

Output

cl	$L \times 2$ matrix, lower (first column) and upper (second column) limits of the selected parameters.

Global Input

_nlp_Lagrange	Vector, created by **NLP** using VPUT. Contains the Lagrangean coefficients for the constraints.

Source

`nlpclim.src`

Index

O

P

Q

R

S

T