Financial Analysis Package MT 3.0

for GAUSS™ Mathematical and Statistical System

Part Number: 007879
Version 3.0
Documentation Revision: 2173
October 07, 2013

Contents

1 Installation

If you are using **GAUSS 13** or later, there is an applications wizard available to install your application. Go to **Tools -> Install Applications** and follow the prompts to install applications from the CD or downloaded `.zip` file. For older versions of **GAUSS**, follow the instructions for the appropriate platform below.

1.1 Linux/Mac

If you are unfamiliar with Linux/Mac, see your system administrator or system documentation for information on the system commands referred to below. Note that if you have more than one version of **GAUSS** installed on your machine, you must install the application to each version where you want to use it. This will also be necessary when upgrading to a newer version of **GAUSS**.

1.1.1 Download

1. Copy the `.zip` file to `/tmp`.

2. `cd` to your **GAUSS** or **GAUSS Engine** installation directory. We are assuming `/usr/local/gauss` in this case.

```
cd /usr/local/gauss
```

3. Use `unzip` to extract the file.

   ```
   unzip /tmp/app_appname_vernum.revnum_WIN.zip
   ```

1.1.2 CD

1. Insert the CD into your machine's CD-ROM drive.

2. Open a terminal window.

3. `cd` to your current **GAUSS** or **GAUSS Engine** installation directory. We are assuming `/usr/local/gauss` in this case.

   ```
   cd /usr/local/gauss
   ```

4. Use `unzip`, depending on the file name extensions, to extract the files found on the CD. For example:

   ```
   unzip /cdrom/apps/app_appname_vernum.revnum_
   WIN.zip
   ```

   However, note that the paths may be different on your machine.

1.2 Windows

If you are unfamiliar with how to extract (unzip) files, see your system administrator or system documentation for information on the commands referred to below. Note that if you have more than one version of **GAUSS** installed on your machine, you must install the application to each

version where you want to use it. This will also be necessary when upgrading to a newer version of **GAUSS**.

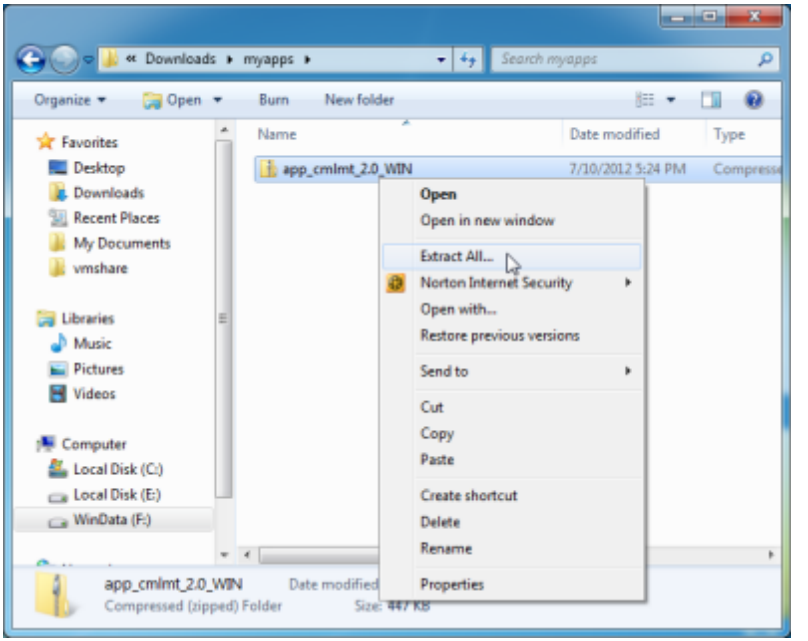
1.2.1 Download

GAUSS applications are a set of files that need to be placed in the /src, /lib and /examples directory from your GAUSSHOME directory. To install an application, you simply need to extract them to your GAUSSHOME directory. Internally the application zip file has the same file hierarchy as the **GAUSS** directory structure and so all files will automatically go to the right place.

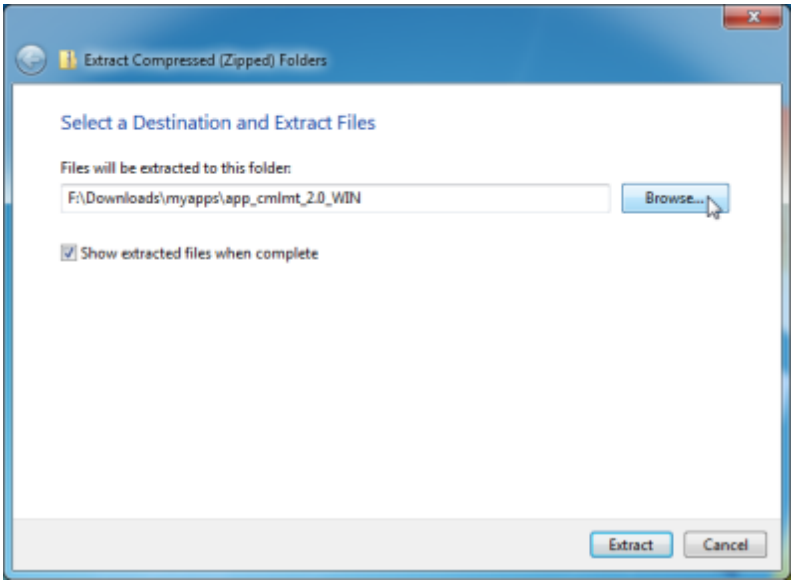
Step by step installation

Open the windows file explorer and browse to the location to which you have downloaded your application.

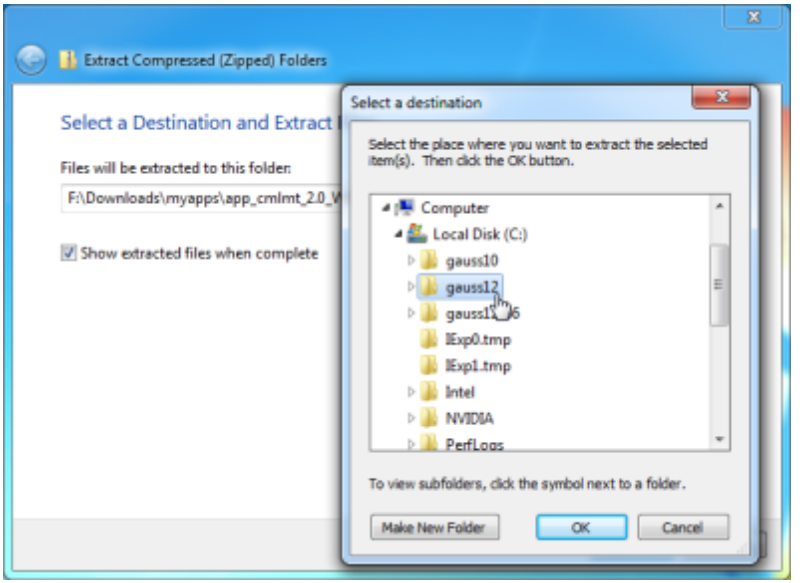
Right-click the application file and select **Extract All...** from the context menu.



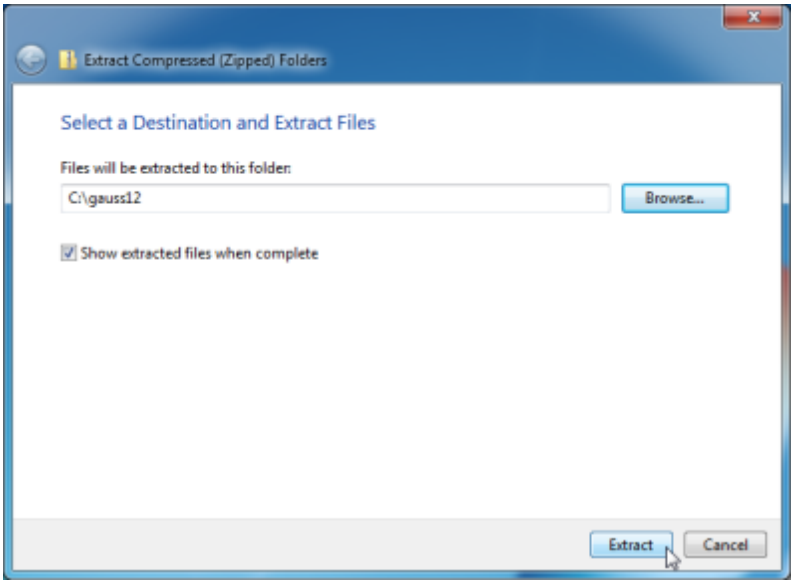
Select the **Browse** button.



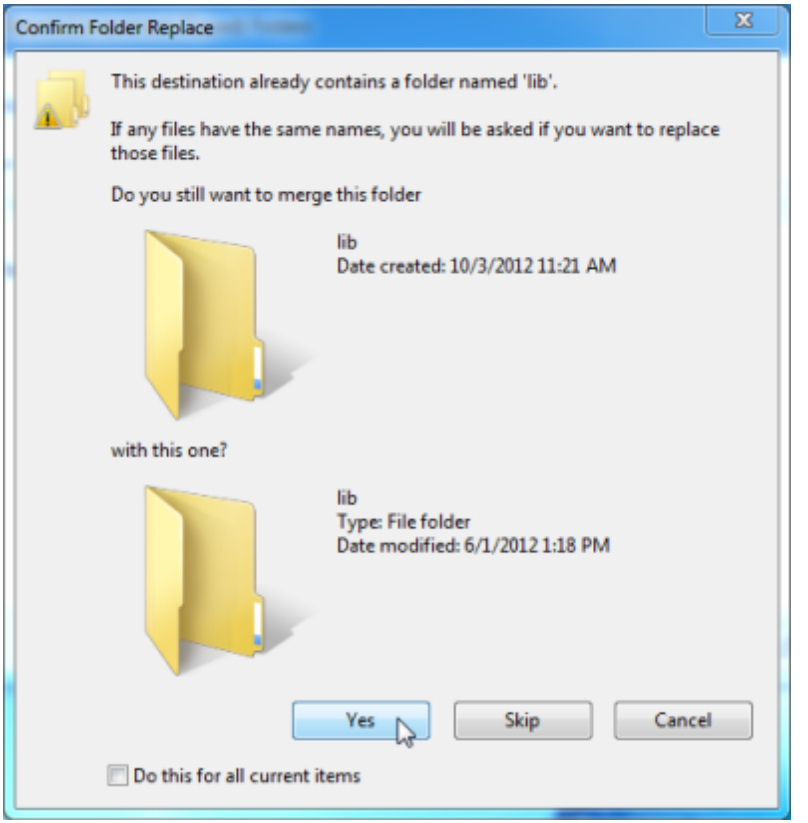
Browse to your **GAUSS** home directory.



Browse to the your **GAUSS** home directory. By default this will be `C:\gauss12`. Then press the **OK** button at the bottom of the window. If your **GAUSS** version is other than 12, the folder will end with that version number instead.



Click the **Extract** button.



Windows will warn you that some of the folders already exist. This is correct. Click the **Yes** button confirming that folder already exists.

Your application module is now installed.

1.2.2 CD

The process of installing an application from a CD is the same as that of installing from a downloaded file.

Insert the CD into your machine's CD-ROM drive.

Follow the steps listed previously to locate the file on the CD and extract the file into your **GAUSS** folder.

1.3 Difference Between the Linux/Mac and Windows Versions

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 Linux/Mac version.

2 Getting Started

GAUSS 10+is required to use these routines. See **_rtl_ver** in src/gauss.dec.

The **FANPACMT** version number is stored in a global variable:

_fan_ver	3×1 matrix, the first element contains the major version number, the second element the minor version number, and the third element the revision number.

If you call for technical support, you may be asked for the version of your copy of **FANPACMT**.

2.1 README Files

If there is a README.fan file, it contains any last minute information on the **FANPACMT** procedures. Please read it before using them.

2.2 Setup

In order to use the procedures in **FANPACMT**, the **FANPACMT** library must be active. This is done by including **fanpacmt** in the library statement at the top of your program or command file:

```
library fanpacmt;
```

This enables **GAUSS** to find the **FANPACMT** procedures. If you plan to make any right-hand references to the global variables (described in the Reference section), the statement

```
#include fanpacmt.ext
```

is also required in versions prior to **GAUSS** 13.1.

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.

3 Financial Analysis Package MT

Written by
Ronald Schoenberg

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

3.1 Modeling with FANPACMT

FANPACMT 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 modeling process; and (2) \gauss{} procedures, which can be called directly to perform the computations.

The **FANPACMT** 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 fanpacmt,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 \ref{:keyref} 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.
computeReturns	Computes returns from price data.
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.
setBoxcox	Indicates variables for Box-Cox transformation.
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 equationsi.
setIndLogElapsedPeriod	Creates independent variable measuring elapsed time between observations.
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 **FANPACMT** keyword commands do not precisely fit your needs, you may design your own command files using the **FANPACMT** 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 **FANPACMT** procedures discussed in Section \ref{:procdesc} in this chapter, and described in Chapter \ref{:procref}.

You might also want to write your own procedures for models not included in **FANPACMT**. To do this you will need to write a procedure for computing the log-likelihood and call the **SQPsolveMT** procedure for the estimation. These procedures are discussed in the **GAUSS** documention for the **Run-Time Library**.

When the **FANPACMT** 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 **FANPACMT**:

ols	Normal linear regression model
tols	t distribution linear regression model
arma(m,n)	Normal ARMA model
tarma(m,n)	t distribution ARMA model
garch(p,q)	Normal GARCH model
agarch(p,q)	Normal GARCH model with asymmetry parameters
tagarch(p,q)	t distribution GARCH model with asymmetry parameters
gtagarch(p,q)	Skew gen. t distribution GARCH model with asymmetry parameters
tgarch(p,q)	t distribution GARCH model
gtgarch(p,q)	Skew gen. t distribution GARCH model
igarch(p,q)	Normal integrated GARCH model
tigarch(p,q)	t distribution integrated GARCH model
gtigarch(p,q)	Skew gen. t distribution integrated GARCH model
egarch(p,q)	Exponential GARCH model
negarch(p,q)	Normal GARCH model with leverage parameters
tegarch(p,q)	t distribution GARCH model with leverage parameters
gtegarch(p,q)	Skew gen. t distribution GARCH model with leverageparameters

figarch(p,q)	Normal fractionally integrated GARCH model
fitgarch(p,q)	t distribution fractionally integrated GARCH model
figtgarch(p,q)	Skew gen. t distribution fractionally integrated GARCH model
varma(m,n)	Normal multivariate VARMA model
tvarma(m,n)	t distribution multivariate VARMA model
bkgarch(p,q,r,s)	Normal BEKK multivariate GARCH model
bktgarch(p,q,r,s)	t distribution BEKK multivariate GARCH model
bkstgarch(p,q,r,s)	Skew t distribution BEKK multivariate GARCH model
dvgarch(p,q,r,s)	Normal DVEC multivariate GARCH model
dvtgarch(p,q,r,s)	t distribution DVEC multivariate GARCH model
cccgarch(p,q,r,s)	Constant correlation Normal multivariate GARCH model
ccctgarch(p,q,r,s)	Constant correlation t distribution multivariate GARCH model
cccstgarch (p,q,r,s)	Constant correlation skew t distribution multivariate GARCH model
cccegarch(p,q,r,s)	Constant correlation Normal exponential multivariate GARCH model
ccctegarch (p,q,r,s)	Constant correlation t distribution exponential multivariate GARCH model
cccstegarch (p,q,r,s)	Constant correlation skew t distribution exponential multivariate GARCH model
cccfigarch (p,q,r,s)	Constant correlation Normal fractionally integrated multivariate GARCH model
ccctfigarch (p,q,r,s)	Constant correlation t distribution fractionally integrated multivariate GARCH model

cccsfiegarch (p,q,r,s)	Constant correlation skew t distribution fractionally integrated multivariate GARCH model
cccgjrgarch (p,q,r,s)	Constant correlation Normal multivariate GJR GARCH model
ccctgjrgarch (p,q,r,s)	Constant correlation t distribution multivariate GJR GARCH model
cccstgjrgarch (p,q,r,s)	Constant correlation skew t distribution multivariate GJR GARCH model
fmgarch(p,q,k,r,s)	Multivariate factor GARCH model Normal distribution
fmtgarch (p,q,k,r,s)	Multivariate factor GARCH model t distribution
fmstgarch (p,q,k,r,s)	Multivariate factor GARCH model skew t distribution
gogarch(p,q,r,s)	Generalized multivariate factor GARCH model Normal distribution
gotgarch(p,q,r,s)	Generalized multivariate factor GARCH model t distribution
gostgarch(p,q,r,s)	Generalized multivariate factor GARCH model skew t distribution
dccgarch(p,q,r,s)	Dynamic correlation Normal multivariate GARCH model
dcctgarch(p,q,r,s)	Dynamic correlation t distribution multivariate GARCH model
dccstgarch (p,q,r,s)	Dynamic correlation skew t distribution multivariate GARCH model
dccegarch(p,q,r,s)	Dynamic correlation Normal exponential multivariate GARCH model
dcctegarch (p,q,r,s)	Dynamic correlation t distribution exponential multivariate GARCH model

dccstegarch (p,q,r,s)	Dynamic correlation skew t distribution exponential multivariate GARCH model
dccfigarch (p,q,r,s)	Dynamic correlation Normal fractionally integrated multivariate GARCH model
dcctfigarch (p,q,r,s)	Dynamic correlation t distribution fractionally integrated multivariate GARCH model
dccsfiegarch (p,q,r,s)	Dynamic correlation skew t distribution fractionally integrated multivariate GARCH model
dccgjrgarch (p,q,r,s)	Dynamic correlation Normal multivariate GJR GARCH model
dcctgjrgarch (p,q,r,s)	Dynamic correlation t distribution multivariate GJR GARCH model
dccstgjrgarch (p,q,r,s)	Dynamic correlation skew t distribution multivariate GJR GARCH model

For an ARCH model set $p = 0$.

For ARMA-GARCH models use (p,q,m,n) where m is the order of the auto-regression parameters, and n the order of the moving average parameters.

3.2 Univariate Time Series Models

The following section describes the univariate time series models available in **FANPACMT**.

3.2.1 ARCH/GARCH/ARMAGARCH models

For the generalized autoregressive conditional heteroskedastic (GARCH) model let

$$\Theta(L)\epsilon_t = \Phi(L)y_t - x_t\beta - \delta\sigma_t$$

where $t - 1, 2, \ldots T$, and y_t an observed time series, x_t an observed time series of fixed exogenous variables including a column of ones, β a vector of coefficients, $\Theta(L) = 1 + \theta_2 L^2 \ldots + \theta_n L^n, \Phi(L) = 1 - \phi_1 L - \phi_2 L^2 \ldots - \phi_m L^m$, and $\sigma_t = E(\epsilon_t)$.

Also define

$$\epsilon_t \equiv n_t \sigma_t$$

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

Standard Model

The standard specification of the conditional variance is

$$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \ldots + \alpha_q \epsilon_{t-q}^2 + \beta_1 \sigma_{t-1}^2 + \ldots + \beta_p \sigma_{t-p}^2 + \Gamma Z_t$$

where Z_t is an observed time series of fixed exogenous variables.

There are two variations of the specification of the conditional variance, the "leverage" or "egarch" model (Nelson,1991) and the "asymmetry" or "agarch" model (Glosten, L.R., Jagannathan, R., and Runkle, D.E., 1993).

Leverage Model

$$logo_t^2 = \omega + \alpha_1 \left(|\epsilon_{t-1}| - E(|\epsilon_{t-1}|) + \zeta \epsilon_{t-1} \right) \ldots + \alpha_q \left(|\epsilon_{t-q}| - E(|\epsilon_{t-q}|) + \zeta \epsilon_{t-q} \right) \beta_1 \sigma_{t-1}^2 + \ldots + \beta_p \sigma_{t-p}^2 + \Gamma Z_t$$

Asymmetry Model

$$\sigma_t^2 = \omega + (\alpha_1 + \tau S_{t-1})\epsilon_{t-1}^2 + \ldots + (\alpha_q + \tau S_{t-q})\epsilon_{t-q}^2 + \beta_1\sigma_{t-1}^2 + \ldots + \beta_p\sigma_{t-p}^2 + \Gamma Z_t$$

where $S_t = 1$ if $\epsilon_t < 0$ and $S_t = 0$ otherwise.

Log-likelihood

For maximum likelihood estimation of all models, we provide two distributions for η_t, the Normal and Student's t, and in addition to these for the EGARCH there is also the generalized exponential distribution. For some univariate distributions the skew generalized t distribution is provided.

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

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

Student's t Distribution

The unit t distribution with v degrees of freedom and variance σ^2 for η_t is

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

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

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

Generalized Error Distribution

For the generalized error distribution

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

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

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

and

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

The log-likelihood is

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

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

Skew Generalized t Distribution

The skew generalized t distribution is described in Theodossiou (1998). This distribution is available only for univariate models. The log-likelihood for an observation is

$$logl_i = log\gamma - \frac{v+1}{\kappa}log\left[1 + \left(\frac{\left|\frac{u_i + \mu_i}{\sigma_i}\right|}{\theta\left(1 + \lambda sign\left(u_i + \mu_i\right)\right)}\right)\right]$$

where

$$\begin{aligned}
\gamma_i &= 0.5S\sigma_i^{-1}\kappa B(1/\kappa, v/\kappa)^{-\frac{3}{2}}B(3/\kappa, (v-2)/\kappa)^{\frac{1}{2}} \\
\theta &= S^{-1}B(1/\kappa, v/\kappa)^{\frac{1}{2}}B(3/\kappa, (v-2)/\kappa)^{-\frac{1}{2}} \\
S &= 1 + 3\lambda^2 + 4\lambda^2 B(2/\kappa, (v-1)/\kappa)B(1/\kappa, v/\kappa)^{-\frac{1}{2}}B(3/\kappa, (v-2)/\kappa)^{-\frac{1}{2}} \\
\mu_i &= 2\sigma_i\lambda B(2/\kappa, (v-1)/\kappa)B(1/\kappa, v/\kappa)^{-\frac{1}{2}}B(3/\kappa, (v-2)/\kappa)^{-\frac{1}{2}}
\end{aligned}$$

and where $-1 < \lambda < 1$ is a skew parameter, and $\kappa > 0$ and $v > 0$ are kurtosis parameters. For $\lambda = 0$ and $\kappa = 2$ we have the Student's t distribution, for $\lambda = 0, \kappa = 2, v = \text{inf}$ the Normal distribution, and for $\lambda = 0, \kappa = \text{inf}, v = \text{inf}$ the uniform distribution.

Multivariate Skew t Distribution

The multivariate skew t distribution is described in Gupta (2003). The p.d.f. is

$$p_v(y,a) = 2f_v(y)F\left(\left(\sim(v+k)a'y\right)/\left(v+y'\Sigma^{-1}y\right)^{1/2}\right)$$

where $f_v(y)$ is the central multivariate t p.d.f. with v df and F() is the central t c.d.f. with $v+k$ df. *alpha* is a $k \times 1$ vector of skew parameters.

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).

$$\omega \geq 0$$

$$\beta_1 \geq 0$$

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

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

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

Then

$$\omega / \left(1 - \Delta_1 - \Delta_2 + \Delta_1 \Delta_2\right) \geq 0$$

$$\beta 1^2 + 4\beta_2 \geq 0$$

$$\Delta_1 > 0$$

$$\sum_{j=0}^{q-1} \alpha_{j+1} \Delta^{-j} > 0$$

and,

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

where

$$\phi_0 = \alpha_1$$

$$\phi_1 = \beta_1 \phi_0 + \alpha_2$$

$$\phi_2 = \beta_1 \phi_1 + \beta_2 \phi_0 + \alpha_3$$

.

.

.

$$\phi_q = \beta_1 \phi_{q-1} - \beta_2 \phi_{q-2}$$

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

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

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

$$1 - \left(\alpha_1 + \beta_1\right) Z - \left(\alpha_2 + \beta_2\right) Z^2 - \ldots$$

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 **FANPACMT** 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.

3.2.2 IGARCH

The IGARCH(p,q) model is a GARCH(p,q) model with a unit root. This is accomplished in \fanmt{} by adding the equality constraint

$$\sum_i \alpha_1 + \sum_i \beta_1 = 1$$

3.2.3 FIGARCH

The IGARCH(p,q) model is a GARCH(p,q) model with a unit root. This is accomplished in \fanmt{} by adding the equality constraint

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

where $t = 1, 2, \ldots T$, 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

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

and

$$A(L) = \beta_1 L + \beta_2 L^2 + \ldots + \beta_q L^q$$

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)

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

where $v_t \equiv \epsilon_t^2 - \sigma_t^2$ is the "innovation"' at time t for the conditional variance process.

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

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

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

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

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.

Log-likelihood

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

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

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

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 **FANPACMT** 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

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

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

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

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 $0 \le d \le 1$ 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

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

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 **FANPACMT** 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.

3.2.4 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

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

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 v degrees of freedom and variance σ^2, the log-likelihood is

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

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

3.3 Multivariate Time Series Models

The following section describes the multivariate time series models available in **FANPACMT**.

3.3.1 Diagonal Vec ARCH/GARCH/VARGARCH

For the Diagonal Vec VARGARCH model let

$$\Theta\left(L\right)\epsilon_t = \Phi\left(L\right)y_t - x_t\beta - \delta\sigma_t$$

where $t-1, 2, \ldots T$, and y_t an observed time series, x_t an observed time series of fixed exogenous variables including a column of ones, β a vector of coefficients, $\Theta(L) = 1 + \theta_2 L^2 \ldots + \theta_n L^n$, $\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 \ldots - \phi_m L^m$, and $\sigma_t = E(\epsilon_t)$.

Each nonredundant element of Σ_t is a separate GARCH model

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

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

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

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

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

where

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

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

$$logL = -\frac{T-\mu}{2}(log\Gamma((v+k)/2) - log\Gamma(v/2) - log\Gamma((v-2)\pi)) - \frac{1}{2} \sum_{t+\mu+1}^{T} log \left|\Sigma_t\right| - \frac{v+k}{2} \sum_{t+\mu+1}^{T} log\left(1 + \epsilon_t' \sum_t^{-1} \epsilon_t / (v-2)\right)$$

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

$$\left|I - (A_1 + B_1)z - (A_2 + B_2)z^2 - \ldots\right|$$

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.

3.3.2 Constant Conditional Correlation GARCH Model

Define a vector of ℓ residuals

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

where $t = 1, 2, \ldots T$, and y_t an observed multiple time series, x_t an observed time series of exogenous variables including a column of ones, 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 modeled as a separate GARCH model

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

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

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

Constant Conditional Correlation GARCH-in-cv

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

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

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

Constant Conditional Correlation GARCH-in-mean

For the constant conditional correlation GARCH-in-mean (or CCCGARCHM) 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

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

where

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

and where

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

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

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

$$logL= -\frac{T-\mu}{2}\left(log\Gamma((v+k)/2) - log\Gamma\left(v/2\right) - \frac{1}{2}log((v-2)\pi)\right)$$
$$-\frac{1}{2}\sum_{t+\mu+1}^{T} log\left|\Sigma_t\right| - \frac{v+k}{2}\sum_{t+\mu+1}^{T} log\left(1+\epsilon_t'\sum_t^{-1}\epsilon_t/(v-2)\right)$$

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

$$\left|I-(A_1+B_1)z-(A_2+B_2)z^2-\ldots\right|$$

lie outside the unit circle (Gourieroux, 1997). Since the A_i and B_i are diagonal matrices, this amounts to determining the roots of *k* polynomials.

$$1-(A_{111}+B_{111})z-(A_{211}+B_{211})z^2-\ldots \quad (1)$$
$$1-(A_{122}+B_{122})z-(A_{2kk}+B_{2kk})z^2-\ldots \quad (2)$$
$$\vdots \quad (3)$$

$$1-(A_{1kk}+B_{1kk})z-(A_{2kk}+B_{2kk})z^2-\ldots \quad (4)$$

Stationarity in the constant conditional correlation 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.

3.3.3 BEKK GARCH

Define a vector of ℓ residuals

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

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

Further define the conditional variance Σ_t of ϵ_t

$$\Sigma_{t,ii} = \Omega + \Sigma_{k=1}^{K}\lambda_k\lambda_k'\left(W_k'\alpha_{k_1}^2\epsilon_{t-1}'\epsilon_{t-1}+\ldots+\alpha_{k_2}^2\epsilon_{t-q}'\epsilon_{t-q}\right)W_k$$
$$\Sigma_{k=1}^{K}\lambda_k\lambda_k'\left[W_k'\left(\beta_{k_2}^2+\ldots+\beta_{k_2}^2\Sigma_{t-p}\right)W_k\right.$$

where K is the number of factors, Ω is a symmetric matrix, λ, a $T\times K$ matrix, and W, a $T\times K$ matrix, are left and right eigenvectors, and α, $K\times q$, and β, $K\times p$ are arch and garch parameters respectively.

3.3.4 Factor GARCH

Define a vector of ℓ residuals (Bauwens, et al., 2006)

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

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

The conditional variance Σ_t of ϵ_t for an FMGARCH(1,1,2) would look like this

$$\begin{aligned}\Sigma_{t,ii} = \; & \Omega + \lambda_1 \lambda_1' a_1^2 w_1' \epsilon_{t-1}' \epsilon_{t-1} w_1 + b_1^2 w_1' \Sigma_{t-1} w_1 + \\ & \lambda_2 \lambda_2' a_2^2 w_2' \epsilon_{t-1}' \epsilon_{t-1} w_2 + b_2^2 w_2' \Sigma_{t-1} w_2\end{aligned}$$

where Ω is a symmetric matrix and λ_i and λ_i are a $T \times 1$ vectors. For identification, the columns of w are constrained to sum to one.

Positive Definiteness of Conditional Variances

The positive definiteness of the conditional variances is assured by the structure of the model.

3.3.5 Generalized Orthogonal GARCH

Define a vector of ℓ residuals (Bauwens, et al., 2006)

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

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

Further define the conditional variance σ_{it} of ϵ_{it} of the *i-th* series

$$\sigma_{it} = w_1 + a_1 \epsilon_{i,t-1}^2 + \ldots + a_q \epsilon_{i,t-q}^2 + b_i \sigma_{i,t-1} + \ldots + b_p \sigma_{i,t-p}$$

where $\omega = 1 - a_1 - \ldots - a_q - b_1 - \ldots - b_p$

Define the $N \times N$ matrix, Δ and the the $N \times N$ matrix, V, the i-th diagonal element of which is equal to the variance of ϵ. Then we have for the conditional variance covariance matrix, H',

$$H_1 = V^{\frac{1}{2}} \Lambda \Sigma \Lambda' \Sigma' V^{\frac{1}{2}}$$

where Σ_t is a diagonal $N \times N$ matrix with i-th diagonal element $\omega = 1 - a\left[1\right] - \ldots - a_q - b_1 - \ldots - b_p$.

3.3.6 Dynamic Conditional Correlation GARCH Model

Define a vector of ℓ residuals

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

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

Define, following Engle (2002),

$$R_t = \Lambda Q_t \Lambda$$

where Λ is a diagonal matrix with elements equal to the square root of the diagonal elements of Q_t, and

$$Q_t = (1-\alpha-\beta)Q_0 + \alpha\epsilon_{t-1}\epsilon_{t-1}' + \beta Q_{t-1}$$

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

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

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

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

Dynamic Conditional Correlation GARCH-in-cv

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

$$\Sigma_{t,ii} = \Omega_i + A_{i,1}\epsilon_{i,t-1}^2 + \ldots + A_{i,q}\epsilon_{i,t-q}^2 + B_{i,1}\Sigma_{i,t-1} + \ldots + B_{i,p}\Sigma_{i,t-p} + Z_i\Gamma_{ij}$$

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

Dynamic Conditional Correlation GARCH-in-mean

For the dynamic conditional correlation GARCH-in-mean (or DCCGARCHM) 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.

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)

$$\left| I - (A_1 + B_1)z - (A_2 + B_2)z^2 - \ldots \right|$$

lie outside the unit circle (Gourieroux, 1997). Since the A_i and B_i are diagonal matrices, this amounts to determining the roots of *k* polynomials.

$$1 - (A_{111} + B_{111})z - (A_{211} + B_{211})z^2 - \ldots \quad (5)$$

$$1 - (A_{122} + B_{122})z - (A_{2kk} + B_{2kk})z^2 - \ldots \quad (6)$$

. .

. (7)

. .

$$1 - (A_{1kk} + B_{1kk})z - (A_{2kk} + B_{2kk})z^2 - \ldots \quad (8)$$

Stationarity in the constant conditional correlation 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.

3.4 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 **FANPACMT** 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

$$JF\left(J, N-K; \alpha\right) = (\theta-\hat{\theta})' V^{-1}\left(\theta-\hat{\theta}\right) \tag{9}$$

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

$$\min\left\{\eta_k'\theta \middle| (\theta-\hat{\theta})' V^{-1}\left(\theta-\hat{\theta}\right) \geq JF\left(J, N-K; \alpha\right)\right\} \tag{10}$$

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

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 (10) is no longer (1.1) and second, (9) is not valid whenever the hyper-ellipsoid is on or near a constraint boundary.

(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 $\epsilon = \sqrt{\left(\sigma_e^2 / N\right) X^2_{(1-\alpha,k)}}$ of the constraint boundary.

Here, and in **FANPACMT**, 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.

3.4.1 Covariance Matrix of Parameters

FANPACMT 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

$$\hat{\theta} \rightarrow N\left(\theta, A^{-1}BA^{-1}\right)$$

where

$$A = E\left[\frac{\partial^2 L}{\partial\theta\,\partial\theta'}\right]$$

$$B = E\left[\left(\frac{\partial L}{\partial\theta}\right)'\left(\frac{\partial L}{\partial\theta}\right)\right]$$

Estimates of A and B are

$$\hat{A} = \frac{1}{N}\sum_{i}^{N}\frac{\partial^2 L_i}{\partial\theta\,\partial\theta'}$$

$$\hat{B} = \frac{1}{N}\sum_{i}^{N}\left(\frac{\partial L_i}{\partial\theta}\right)'\left(\frac{\partial L_i}{\partial\theta}\right)$$

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

$$\hat{\theta} \rightarrow N\left(\theta, \hat{A}^{-1}\right)$$

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

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

The likelihood function augmented by constraints is then

$$L_A = L + \lambda_1 g(\theta)_1 + \ldots + \lambda_1 g(\theta)^I + \lambda_1 h_{\oplus 1}(\theta) + \ldots + \lambda_1 h_{\oplus j}(\theta) + h_{\ominus 1}(\theta)_i - \zeta_1^2 + \ldots + h_{\ominus K}(\theta) - \zeta_K^2$$

and the Hessian of the augmented likelihood is

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

where the dot represents the Jacobian with respect to θ, $L = \sum_1^N \log P(Y_i; \theta)$, 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

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

Let Ξ be the orthonormal basis for the null space of $\widetilde{B}$, 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.

3.4.2 Quasi-Maximum Likelihood Covariance Matrix of Parameters

FANPACMT computes a QML covariance matrix of the parameters when requested. Define $B = (\partial L_A / \partial\theta)'\left(\partial L_A / \partial\theta\right)$ 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
```

3.4.3 Confidence Limits

FANPACMT 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.

3.5 FANPACMT 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

3.5.1 Initializing the Session

First, an analysis session must be established.

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

will start a new session, and

```
getsession ses1;
```

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`.

3.5.2 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;
```

FANPACMT 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 **FANPACMT** 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 **FANPACMT** keyword command, `setVarNames`:

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

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

3.5.3 The Date Variable

FANPACMT 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. **FANPACMT** will convert the dates to the standard format and the observations will be sorted. For example:

```
library fanpacmt,pgraph;
sessionnissan '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 **FANPACMT** 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.

3.5.4 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

$$LR_1 = \kappa \, log\left(\frac{P_t}{P_{t-1}}\right)$$

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

$$PCTR_1 = \kappa \frac{P_t - P_{t-1}}{P_{t-1}}$$

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.

```
computeLogReturns 100;
```

3.5.5 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 **FANPACMT** 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.

3.5.6 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;
```

3.5.7 Simulation

A keyword command is available for simulating data from the various models in **FANPACMT**. 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 fanpacmt;
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.

3.5.8 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 Section 3.2.1). 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 one. 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 **FANPACMT** for selecting these types of constraints:

```
setConstraintType standard;
```

selects the Nelson and Cao (1992) constraints (described in Section 3.2.1). 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.

3.5.9 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 the **GAUSS SQPsolveMT** procedure. In some cases, therefore, the iterations may be time consuming. **SQPsolveMT** permits you to monitor the iterations using keystrokes. To cause**SQPsolveMT** to print iteration information to the screen, press o. To force termination of the iterations press c.

The following models may be estimated:

ols	Normal linear regression model
tols	t distribution linear regression model
arma(m, n)	Normal ARMA model
tarma(m, n)	t distribution ARMA model
garch(p, q)	Normal GARCH model
agarch(p, q)	Normal GARCH model with asymmetry parameters
tagarch(p, q)	t distribution GARCH model with asymmetry parameters
gtagarch(p, q)	Skew gen. t distribution GARCH model with asymmetry parameters
tgarch(p, q)	t distribution GARCH model
gtgarch(p, q)	Skew gen. t distribution GARCH model
igarch(p, q)	Normal integrated GARCH model
tigarch(p, q)	t distribution integrated GARCH model
gtigarch(p, q)	Skew gen. t distribution integrated GARCH model
egarch(p, q)	Exponential GARCH model
negarch(p, q)	Normal GARCH model with leverage parameters
tegarch(p, q)	t distribution GARCH model with leverage parameters
gtegarch(p, q)	Skew gen. t distribution GARCH model with leverage parameters
figarch(p, q)	Normal fractionally integrated GARCH model
fitgarch(p, q)	t distribution fractionally integrated GARCH model
figtgarch(p, q)	Skew gen. t distribution fractionally integrated GARCH model

varma(m, n)	Normal multivariate VARMA model
tvarma(m, n)	t distribution multivariate VARMA model
bkgarch(p, q, r, s)	Normal BEKK multivariate GARCH model
bktgarch(p, q, r, s)	t distribution BEKK multivariate GARCH model
bkstgarch(p, q, r, s)	Skew t distribution BEKK multivariate GARCH model
dvgarch(p, q, r, s)	Normal DVEC multivariate GARCH model
dvtgarch(p, q, r, s)	t distribution DVEC multivariate GARCH model
cccgarch(p, q, r, s)	Constant correlation Normal multivariate GARCH model
ccctgarch(p, q, r, s)	Constant correlation t distribution multivariate GARCH model
cccstgarch(p, q, r, s)	Constant correlation skew t distribution multivariate GARCH model
cccegarch(p, q, r, s)	Constant correlation Normal exponential multivariate GARCH model
ccctegarch(p, q, r, s)	Constant correlation t distribution exponential multivariate GARCH model
cccstegarch(p, q, r, s)	Constant correlation skew t distribution exponential multivariate GARCH model
cccfigarch(p, q, r, s)	Constant correlation Normal fractionally integrated multivariate GARCH model
ccctfigarch(p, q, r, s)	Constant correlation t distribution fractionally integrated multivariate GARCH model
cccsfiegarch(p, q, r, s)	Constant correlation skew t distribution fractionally integrated multivariate GARCH model
cccgjrgarch(p, q, r, s)	Constant correlation Normal multivariate GJR GARCH model

ccctgjrgarch(p, q, r, s)	Constant correlation t distribution multivariate GJR GARCH model
cccstgjrgarch(p, q, r, s)	Constant correlation skew t distribution multivariate GJR GARCH model
fmgarch(p,q,k,r,s)	Multivariate factor GARCH model Normal distribution
fmtgarch (p,q,k,r,s)	Multivariate factor GARCH model t distribution
fmstgarch (p,q,k,r,s)	Multivariate factor GARCH model skew t distribution
fmtgarch (p,q,k,r,s)	Multivariate factor GARCH model t distribution
fmstgarch (p,q,k,r,s)	Multivariate factor GARCH model skew t distribution
gogarch(p,q,r,s)	Generalized multivariate factor GARCH model Normal distribution
gotgarch(p,q,r,s)	Generalized multivariate factor GARCH model t distribution
gostgarch(p,q,r,s)	Generalized multivariate factor GARCH model skew t distribution
dccgarch(p,q,r,s)	Dynamic correlation Normal multivariate GARCH model
dcctgarch(p,q,r,s)	Dynamic correlation t distribution multivariate GARCH model
dccstgarch (p,q,r,s)	Dynamic correlation skew t distribution multivariate GARCH model
dccegarch(p,q,r,s)	Dynamic correlation Normal exponential multivariate GARCH model
dcctegarch (p,q,r,s)	Dynamic correlation t distribution exponential multivariate GARCH model

dccstegarch (p,q,r,s)	Dynamic correlation skew t distribution exponential multivariate GARCH model
dccfigarch(p,q,r,s)	Dynamic correlation Normal fractionally integrated multivariate GARCH model
dcctfigarch (p,q,r,s)	Dynamic correlation t distribution fractionally integrated multivariate GARCH model
dccsfiegarch (p,q,r,s)	Dynamic correlation skew t distribution fractionally integrated multivariate GARCH model
dccgjrgarch (p,q,r,s)	Dynamic correlation Normal multivariate GJR GARCH model
dcctgjrgarch (p,q,r,s)	Dynamic correlation t distribution multivariate GJR GARCH model
dccstgjrgarch (p,q,r,s)	Dynamic correlation skew t distribution multivariate GJR GARCH model

For an ARCH model set p = 0.

For ARMA-GARCH models use (*p,q,m,n*) where m is the order of the auto-regression parameters, and n the order of the moving average parameters.

3.5.10 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;
```

3.5.11 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
------------------------------------------------
       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
   ------------------------------------------------
```


3.5.12 Conditional Variances and Standard Deviations

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

```
plotCV;
```

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;
```

3.5.13 Example

The following example analyzes daily data on Intel common stock. Two models are fitted, the results are printed, and the conditional variances are plotted.

```
library fanpacmt,pgraph;
session wilshire 'wilshire example';
setDataset wilshire;
/* capitalization weighted returns */
setSeries cwret;
setInferenceType simLimits;
estimate run1 garch(2,2);
estimate run2 garch(2,2,2,0);
showResults;
testSR;
plotCV;



==========================================================
=========
                      Session: wilshire

----------------------------------------------------------
--------
                      wilshire example
```

```
-------------------------------------------------------
--------
 FANPACMT Version 3.0.0  Data Set: wilshire  3/05/2003
12:07:02

=======================================================-
=========



~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-
~~~~~~~~~
                            run: run1

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

return code =    0
normal convergence


Model:  GARCH

Number of Observations     : 341
Observations in likelihood : 341
Degrees of Freedom         : 335

AIC         1991.58
BIC         2014.57
LRS         1979.58
```

```
          roots
      ______________
        1.0241991
       -1.7635304

        Abs(roots)
      ______________
        1.0241991
        1.7635304


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

Series: cwret


  Variance Equation


Variance Equation Constant(s)

Estimate
 1.11894
standard error
 0.07007

lower confidence limit
 0.95686
```

```
upper confidence limit
 1.23437

Garch Parameter(s)

Estimate

 0.38099
 0.43180
standard error

 0.04657
 7.49337

lower confidence limit

 0.22638
-24.66137

upper confidence limit

 0.44698
 12.25291

Arch Parameter(s)

Estimate

 0.02834
 0.12185
standard error
```

```
 0.08229
 0.17828

lower confidence limit

-0.26628
-0.25567

upper confidence limit

 0.10723
 0.31078


  Mean Equations


Constant(s)

Estimate
 1.1420
standard error
 0.0511

lower confidence limit
 0.9634

upper confidence limit
 1.1420
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-
~~~~~~~~~
                              run: run2

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

return code =    0
normal convergence


Model:  GARCH

Number of Observations     : 341
Observations in likelihood : 341
Degrees of Freedom         : 333

AIC         1994.36
BIC         2025.02
LRS         1978.36

          roots
     _______________
       1.0274493
      -1.8113846
      0.25899372 +        4.2630194i
      0.25899372 -        4.2630194i
```

```
        Abs(roots)

      ______________
        1.0274493
        1.8113846
        4.2708796
        4.2708796


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

Series: cwret


  Variance Equation


Variance Equation Constant(s)

Estimate
 1.11880
standard error
 0.05325

lower confidence limit
 0.97001

upper confidence limit
 1.16972

Garch Parameter(s)
```

```
Estimate

 0.38530
 0.44236
standard error

 0.05406
 10.72465

lower confidence limit

 0.18941
-39.06050

upper confidence limit

 0.39583
 5.74083

Arch Parameter(s)

Estimate

 0.03592
 0.09496
standard error

 0.10721
 0.21764

lower confidence limit
```

```
-0.36311
-0.65643

upper confidence limit

 0.12166
 0.21019


  Mean Equations


Constant(s)

Estimate
 1.1793
standard error
 180.5153

lower confidence limit
-608.9496

upper confidence limit
 1.1793

AR Parameter(s)

Estimate

 0.0284
-0.0548
```

```
standard error

 0.0198
 0.0211

lower confidence limit

-0.0248
-0.1295

upper confidence limit

 0.0485
-0.0548




=========================================================-
=========
                        Session: wilshire

-----------------------------------------------------------
--------
                        wilshire example

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

                    Time Series

================================================
```

```
                  Series: cwret
-------------------------------------------------
       skew      -360.8689   pr =      0.000
   kurtosis      9266.8838   pr =      0.000

   heteroskedastic-consistent
   Ljung-Box       42.4523   pr =      0.065
-------------------------------------------------


                   Residuals

=================================================
                 run1: GARCH(2,2)
-------------------------------------------------
       skew      -357.0129   pr =      0.000
   kurtosis      9280.9146   pr =      0.000

   heteroskedastic-consistent
   Ljung-Box       21.1015   pr =      0.885

=================================================
             run2: GARCH(2,2,2,0)
-------------------------------------------------
       skew      -335.4611   pr =      0.000
   kurtosis      9034.2040   pr =      0.000

   heteroskedastic-consistent
   Ljung-Box       20.7854   pr =      0.894
-------------------------------------------------
```

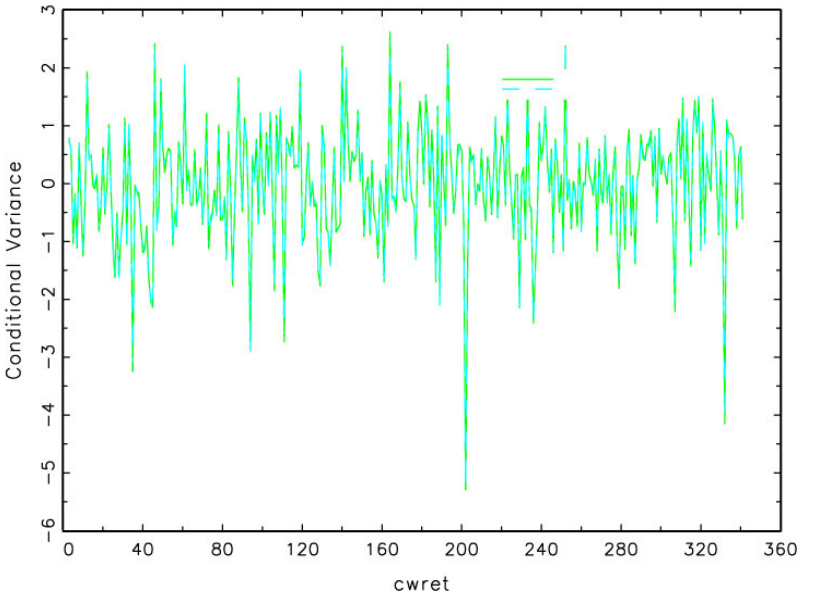


Figure 3.1: Plot of conditional variances for ARCH and GARCH models on a 27-year monthly series of the capitalization-weighted Wilshire 5000 index

3.5.14 Altering SQPSolveMT Control Variables

When an estimation is invoked (i.e., when **`estimate`** is called) **FANPACMT** a default **`SQPsolveMT`** control structure is constructed. If you wish to alter any of the SQPsolveMT control variables add a proc to the command file that takes an SQPsolveMTControl structure as an input argument and output argument. In the proc modify the control variables as desired.

```
proc sqpcont( struct sqpsolvemtcontrol c0 );
    c0.maxiters = 100;
```

```
        c0.printiters = 1;
        retp(c0);
    endp;
    struct fanControl f0;
    f0 = fanControlCreate;
    f0.sqpsolvemtControlProc = &sqpcont;
```

Be aware that some of the control variables are required for the estimation and their modification may produce unpredictable results. If you wish, for example, to add linear constraints to the model, you must test first whether the linear constraint matrices have already been set:

```
    proc sqpcont( struct sqpsolvemtcontrol c0 );

        if rows(c0.A) == 0;
            c0.A = 1~0~0~0~0~-1;
            c0.b = 1;
        else;
            c0.A = c0.A | (1~0~0~0~0~-1);
            c0.b = c0.B | 1;
        endif;

        retp(c0);
    endp;
    struct fanControl f0;
    f0 = fanControlCreate;
    f0.sqpsolvemtControlProc = &sqpcont;
```


3.5.15 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. \fanmt{} 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:\index {setIndVars@\commandname{setIndVars}}

```
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 \commandname{setIndVars} for each dependent variable as needed. Any equation for which \commandname {setIndVars} is not called will contain all the independent variables.

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

3.5.16 Example

The following example analyzes daily data on Intel common stock. Two models are fitted, the results are printed, and the conditional variances are plotted.

```
library fanpacmt,pgraph;

session mult 'May 15, 1997 to November 9, 1998';

setDataset stocks;

setSeries AMZN YHOO;
setIndVars *YHOOvol *AMZNvol;
```

```
setIndEqs AMZN lnAMZNvol;
setIndEqs YHOO lnYHOOvol;
setCVIndEqs AMZN lnAMZNvol;
setCVIndEqs YHOO lnYHOOvol;
setSqrtCV on;
setInferenceType simBound;
setStationarityConstraint roots;

computeLogReturns 251;

estimate run1 cccgarchcv(2,1);
showResults;
plotCV;
```

```
================================================================
=============
                              Session: mult

----------------------------------------------------------------
------------
                    May 15, 1997 to November 9, 1998

----------------------------------------------------------------
------------
 FANPACMT Version 3.0.0   Data Set:  stocks
3/06/2003 13:33:54

================================================================
=============
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~
                                run: run1

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

return code =    0
normal convergence


Model:  CDVGARCHV

Number of Observations     : 375
Observations in likelihood : 375
Degrees of Freedom         : 360

AIC         4505.97
BIC         4564.87
LRS         4475.97

          roots
     _______________
       9.4281112
               1
       46.223078
       1.0000341
```

```
          3.5668614
          3.5668614
          27.960257
           2.261289

          Abs(roots)

        _____________
          9.4281112
                  1
          46.223078
          1.0000341
          3.5668614
          3.5668614
          27.960257
           2.261289


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

  Series 1: AMZN
  Series 2: YHOO


    Variance Equation


  Variance Equation Constant(s)

  Estimate
```

```
 3.92334
 2.55377
standard error

 19.82638
 21.41505

lower confidence limit

-81.88559
-81.84615

upper confidence limit

 3.92334
 2.55377

Garch Parameter(s)

Estimate

 0.37878    0.34116
-0.04795   -0.01059
standard error

 0.04611    0.18736
 0.06490    0.25303

lower confidence limit

 0.17216   -0.53310
-0.30663   -1.16765
```

```
upper confidence limit

 0.37878    0.34116
-0.04795   -0.01059

Arch Parameter(s)

Estimate
 0.21711    0.18001
standard error
 0.03088    0.02506

lower confidence limit
 0.05836    0.05761

upper confidence limit
 0.21711    0.18001

Variance Equation Regression Coefficient(s)

Estimate

 0.00000    0.59439
 0.57870    0.00000
standard error

 0.00000    0.01721
 0.01530    0.00000

lower confidence limit
```

```
 0.00000    0.52353
 0.51587    0.00000

upper confidence limit

 0.00000    0.59439
 0.57870    0.00000


  Mean Equations


Constant(s)

Estimate

-6.2382
-6.5485
standard error

 0.0545
 0.0621

lower confidence limit

-6.4206
-6.6672

upper confidence limit

-6.2382
-6.4871
```

```
Regression coefficient(s)

Estimate

 0.0000       0.7642
 0.7222       0.0000
standard error

 0.0000       0.4687
 0.6427       0.0000

lower confidence limit

 0.0000      -0.9480
-1.3597       0.0000

upper confidence limit

 0.0000       0.7642
 1.3099       0.0000


  Miscellaneous Parameters


VC

Estimate

 1.0000       0.4433
 0.4433       1.0000
```

```
standard error

 1.0000       0.0147
 0.0147       1.0000

lower confidence limit

 1.0000       0.4416
 0.4416       1.0000

upper confidence limit

 1.0000       0.5055
 0.5055       1.0000
```

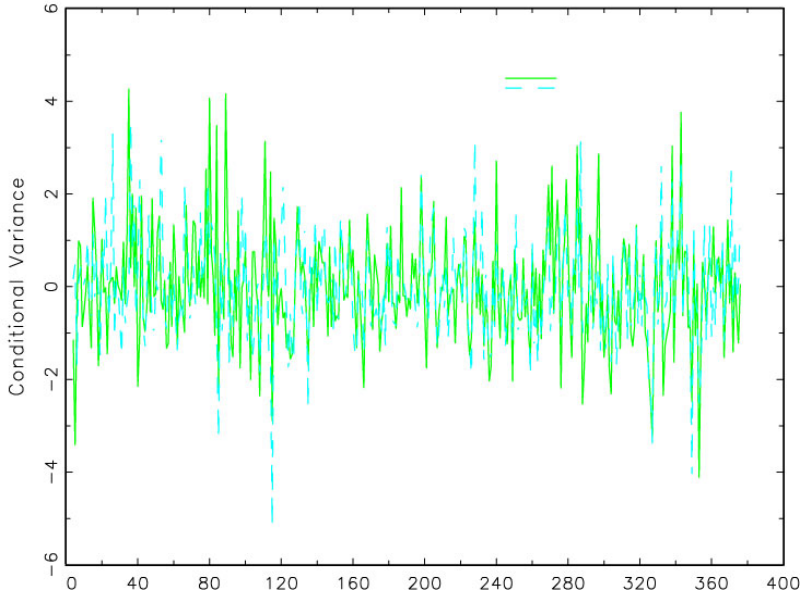


Figure 3.2: Plot of conditional variances for AMZN and YHOO using a Constant Correlation Diagonal Vec multivariate GARCH model

3.6 Data Transformations

Box-Cox

This transformation involves estimating additional parameters. The Box-Cox transformation is

$$f(x) = \frac{x^{\lambda} - 1}{\lambda}$$

where λ is an estimated parameter.

When using the **ugarch** or **mgarch** procedures for estimation, this transformation is specified by setting the *bxcx* member of the *fanControl* structure for independent variables, or the *seriesbxcx* member for the time series variables.

The keyword command **setBoxcox** can be used to set this transformation for selected variables in the model.

Computing Returns

Two keyword commands are available for computing returns from prices in time series. computeLogReturns computes

$$R_i = K\,ln\left(P_i / P_{i-1}\right)$$

where K is a scale factor. The scale factor is specified as an argument in the keyword call:

```
computeLogReturns 251;
```

The keyword command computePercentReturns computes

$$R_i = K\left(P_i - P_{i-1}\right) / P_{i-1}$$

where K is a scale factor. For a time series with interpretation as a proportion $K = 1$, or as a percent, $K = 100$.

Log Transformations on Independent Variables

The keyword command setIndVars can be used to specify a log transformation of selected independent variables. Insert an asterisk in front of the selected variables in the variable list. Then in later references to those

variables in keyword commands, append *ln* to its name. For an illustration see the example in Section 3.5.15.

Creating Elapsed Time Between Observations

Two keywords will generate independent variables measuring the elapsed time between observations in the time series, `setIndLogElapsedPeriod` and `setIndElapsedPeriod`. The variable names for the generated variablers are *lnEP* and *EP* respectively.

```
library fanpacmt,pgraph;
session wilshire 'wilshire example';
setDataset wilshire;
setSeries cwret;
/* capitalization weighted returns */setLo-
gElapsedPeriod;
setIndEqs lnEP;

estimate run1 garch(2,2);

showResults;
```


3.7 FANPACMT Session Structure

When using the keywords \fanmt{} saves results and various aspects of the problem in a *fanSession* structure. The following is the definition of that structure (from `fanpacmt.sdf`):

```
struct fanSession {
           string name;
           string title;
           string array runNames;
```

```
                string dataset;
                matrix series;
                matrix indvars;
                matrix range;
                matrix scale;
                matrix version;
                string date;
                matrix stddev;
                matrix mean;
                matrix mnind;
                matrix stdind;
                struct fanEstimation Runs;
                   };
```


The instance of a *fanSession* structure is saved to the disk with the name given in the session keyword command as session name. The member *runNames* contains the names of all of the runs in that session given in the estimation keyword command. Associated with each of those run names is an element of the *fanEstimation* instance with the results of those runs.

The *fanEstimation* structure has the following definition:

```
    struct fanEstimation {

                string name;
                scalar model;
                string title;
                struct fanControl control;
                scalar aic;
                scalar bic;
                scalar lrs;
                scalar numObs;
```

```
                scalar df;

                struct PV par;
                scalar retcode;
                matrix moment;
                matrix hessian;
                matrix climits;
                matrix tsforecast;
                matrix cvforecast;

                      };
```

A *PV* structure is nested in this structure containing the parameter estimates. The *PV* structure doesn't have any public members but rather there are a set of functions associated with the *PV* structure. Two of these are used to retrieve the parameters. **pvUnpack** retrieves them in their original form as matrices or arrays. **pvGetParvector** retrieves the entire vector of estimated parameters.

The parameter names stored in *par*, the instance of the *PV* structure in **fanSession** are

1. *beta0*, constants in mean equations
2. *beta*, regression coefficients
3. *lambda_y*, Box-Cox coefficients
4. *lambda_x*, Box-Cox coefficients
5. *omega*, constants in conditional variance equations
6. *garch*, garch coefficients
7. *arch*, arch coefficients

8. *ar*, auto-regression coefficients
9. *ma*, moving average coefficients
10. *gamma*, inCV coefficients
11. *nu*, "degrees of freedom" parameter for t-distribution
12. *rho*, shape parameter for exponential distribution
13. *dm*, fractional integration parameter
14. *zeta*, leverage parameter for egarch model
15. *delta*, in mean coefficient for conditional variance
16. *delta_s*, in mean coefficient conditional standard deviation
17. *s2*, covariance matrix for multivariate t
18. *tau*, assymetry parameter

The parameters may be unpacked using either the number or the name:

```
garch = pvUnpack(f0.runs[1].par,"garch");
arch = pvUnpack(f0.runs[1].par,7);
```

Example

```
>> run fanpacmt.sdf
>> struct fanSession f0
>> { f0, ret } = loadStruct("wilshire","fanSession");
>> print f0.runnames

       run1
```

```
        run2

>> print pvUnpack(f0.runs[1].par,"garch")

      0.3810
      0.4318

>> print pvUnpack(f0.runs[2].par,"garch")

      0.3853
      0.4424


>> print pvGetParVector(f0.runs[1].par);

      1.1420
      1.1189
      0.3810
      0.4318
      0.0283
      0.1218

>> print f0.runs[1].lrs

      1979.5753

>>  f0.runs[2].lrs;

      1978.3599
```

3.8 FANPACMT Procedures

The **FANPACMT** procedures used by the keyword commands can be called directly. The maximum likelihood procedures for each of the **FANPACMT** models can be put into command files and estimates generated using the **SQPsolveMT** 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 fanpacmt;
#include fanpacmt.sdf

Y = loadd("example");
struct DS d0;
d0.dataMatrix = Y;

struct fanControl c0;
c0 = fanControlCreate;

c0.p = 3;
c0.q = 2;

/* Nelson and Cao constraints  */
c0.cvConstType = 1;

struct fanEstimation f0;
f0 = ugarch(c0,d0);

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

```
print;

lbl = pvGetParnames(f0.par);
p = pvGetParVector(f0.par);

format /rd 12,4;
print "         Coefficients          lower cl
upper cl";
for i(1,rows(p),1);
    print lbl[i];;
    print p[i];;
    print f0.climits[i,.];
endfor;
```

```
    Nelson & Cao constraints

       Coefficients          lower cl    upper cl
 beta0[1,1]       0.4568       0.3796       0.5340
 omega[1,1]       0.4129       0.0229       0.8030
 garch[1,1]      -0.5595      -1.2238       0.1049
 garch[2,1]       0.1137      -0.0545       0.2820
 garch[3,1]       0.0647      -0.1385       0.2678
 arch[1,1]        0.4904       0.2653       0.7155
 arch[2,1]        0.4713       0.1297       0.8129
```


3.9 Bibliography

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

2. Baillie, Richard T., Bollerslev, Tim, and Mikkelsen, Hans Ole Æ., 1996. "Fractionally integrated generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics*, 74:3-28.
3. Bauwens, Luc, Laurent, Sebastien, and Rombouts, Jeroen V. K., 2006. "Multivariate GARCH models: a survey, *Journal of Applied Econometrics*, 21:79-109.
4. Engle, R.F., 2002. "Dynamic conditional correlation -- a simple class of multivariate GARCH models," *Journal of Business and Economic Statistics*, 20:339-350
5. Gallant, A. R., 1987. *Nonlinear Statistical Models*. New York: Wiley.
6. Geweke, John, 1995. "Posterior simulators in econometrics," Working Paper 555, Research Department, Federal Reserve Bank of Minneapolis.
7. Glosten, L.R., Jagannathan, R., and Runkle, D.E., 1993. "On the relation between the expected value and the volatility of the nominal excess return on stocks," *Journal of Finance*, 48(5):1779-1801.
8. Gouriéroux, Christian, 1997. **ARCH Models and Financial Applications**. New York: Springer-Verlag.
9. 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.
10. Gupta, A. K., 2003. "Multivariate skew t-distribution," *Statistics*, 37 (4):359-363.
11. 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.
12. Nelson, Daniel B., 1991. "Conditional Heteroskedasticity in Asset Returns: a New Approach," *Econometrica*, 59(2):347-370.

13. Nelson, Daniel B. and Cao, Charles Q., 1992. "Inequality constraints in the univariate GARCH model," *Journal of Business and Economic Statistics*, 10:229-235.
14. 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.
15. Schoenberg, Ronald J., 1997. "Constrained maximum likelihood," *Computational Economics*, 10:251-266.
16. Self, Steven G. and Liang, Kung-Yee, 1987. "Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under non-standard conditions," *Journal of the American Statistical Association*, 82:605-610.
17. Theodossiou, Panayiotis, 1998. "Financial data and the skewed generalized t distribution," *Management Science*, 44:1650-1661.
18. White, H., 1981. "Consequences and detection of misspecified nonlinear regression models." *Journal of the American Statistical Association*, 76:419-433.
19. White, H., 1982. "Maximum likelihood estimation of misspecified models," *Econometrica*, 50:1-25.
20. Wolak, Frank, 1991. The local nature of hypothesis tests involving inequality constraints in nonlinear models," *Econometrica*, 59:981-995.

4 FANPACMT Procedure Reference

The FANPACMT Procedure Reference chapter describes each of the commands, procedures and functions available in **FANPACMT**.

bkgarch

Purpose

Estimates parameters of BEKK garch model.

Library

fanpacmt

Format

out = **bkgarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.

C.p	scalar, order of the garch parameters.
C.q	scalar, order of the arch parameters.
C.ar	scalar, order of the autoregressive parameters.
C.ma	scalar, order of the moving average parameters.
C.density	scalar, density of error term, 0 - Normal, 1 - Student's t.
C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
C.bxcx	$1 \times K$ vector, mask indicating which independent variables are to be transformed via the boxcox function.
C.seriesbxcx	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
C.CVIndEquations	$L \times K$ matrix, a $1 \times K$ vector for

		each conditional variance equation indicating independent variables to be included.
	C.inMean	scalar, $M \times L$ matrix, a $1 \times L$ vector for each mean equation indicating which conditional variance equation is included in which mean equation.
	C.stConstType	scalar, type of enforcement of stationarity requirements, 1 - roots of characteristic polynomial constrained outside unit circle, 2 - arch, garch parameters constrained to sum to less than one and greater than zero, 3 - none.
	C.cvConstType	scalar, type of enforcement of nonnegative conditional variances, 0 - direct constraints, 1 - Nelson & Cao constraints.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	

`D.[1].dataMatrix`	$N \times 1$ vector, time series.
`D.[2].dataMatrix`	$N \times k$ matrix, independent variables (optional).

Output

`out`	instance of a `fanEstimation` structure.	
	`out.model`	scalar, set to 17.
	`out.control`	a copy of the input `fanControl` structure.
	`out.aic`	scalar, scalar, Akiake criterion.
	`out.bic`	scalar, Bayesian information criterion.
	`out.lrs`	scalar, likelihood ratio statistic.
	`out.numObs`	scalar, number of observations.
	`out.df`	scalar, degress of freedom.
	`out.par`	instance of `PV` structure containing parameter estimates.

`omega`	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
`garch`	$T \times p$ matrix of garch coefficients.
`arch`	$T \times q$ matrix of arch coefficients.
`AR`	$T \times T \times r$ array of autoregression parameters.
`MA`	$T \times T \times s$ array of moving average parameters.
`beta0`	$T \times 1$ vector, constants in mean equation.
`beta`	$T \times 1$ vector, constants in mean

		equation.
	nu	scalar, degrees of freedom for t or skew t distribution.
	skew	*T*×$1 vector, skew parameters for skew t distribution.
out.retcode	scalar, return code.	
	0	normal convergence.
	1	forced exit.
	2	maximum number of iterations exceeded.
	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian

		calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment		$K \times K$ matrix, moment matrix of parameter.
out.climits		$K \times 2$ matrix, confidence limits.
out.tsForecast		$M \times L$ matrix, time series forecast.
out.cvForecast		$M \times L \times L$ array, forecast of conditional covariance matrices.

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI;
load index[368,9] = index.asc;

x0 = index[.,6:7];
```

```
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = BKGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = out.climits;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;
```

```
format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];
endfor;

likelihood ratio  3604.4834
aic               3636.4834
bic               3698.9255

beta0[1,1]     0.9836     0.2369     1.7304
beta0[2,1]     0.3065    -0.1110     0.7241
omega[1,1]     0.4260    -0.3758     1.2278
omega[1,2]     0.4098     0.1309     0.6887
omega[2,2]     0.0026    -0.0671     0.0722
garch[1,1,1]   1.1589     1.0201     1.2977
garch[1,1,2]  -1.5521    -2.3178    -0.7864
garch[1,2,1]   0.2498    -0.0409     0.5406
garch[1,2,2]  -1.1321    -1.2647    -0.9995
arch[1,1,1]    0.1705     0.0795     0.2614
arch[1,1,2]   -0.1785    -0.3365    -0.0206
arch[1,2,1]   -0.0193    -0.0721     0.0336
arch[1,2,2]    0.2286     0.1257     0.3315
nu[1,1]       15.1652     1.5201    28.8102
skew[1,1]     -0.0340    -0.0930     0.0250
skew[2,1]     -0.0183    -0.1301     0.0935
```

Source

mgarchmt.src

cccegarch

Purpose

Estimates parameters of multivariate constant conditional correlation exponential garch model.

Library

fanpacmt

Format

out = **cccegarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.

	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
	C.CVIndEquations	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D		1×1 or 2×1 *DS* structure, data.
	D.[1].dataMatrix	$N \times 1$ vector, time series.
	D.[2].dataMatrix	$N \times k$ matrix, independent variables (optional).

Output

out		instance of a *fanEstimation* structure.
	out.control	a copy of the input *fanControl* structure.
	out.aic	scalar, scalar, Akiake criterion.
	out.bic	scalar, Bayesian information

criterion.

out.lrs	scalar, likelihood ratio statistic.
out.numObs	scalar, number of observations.
out.df	scalar, degress of freedom.
out.par	instance of *PV* structure containing parameter estimates.

omega	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
garch	$T \times p$ matrix of garch coefficients.
arch	$T \times q$ matrix of arch coefficients.
AR	$T \times T \times r$ array of autoregression parameters.
MA	$T \times T \times s$ array of moving average

	parameters.
`beta0`	$T\times 1$ vector, constants in mean equation.
`beta`	$T\times k$ vector, constants in mean equation.
`gamma`	$T\times m$ matrix, parameters of exogenous variables in conditional variance equations.
`lambda_y`	$R\times 1$ matrix, BoxCox parameters for time.
`lambda_x`	$S\times k$ matrix, BoxCox parameters for exogenous variables.
`delta`	$T\times r$ matrix,

	parameters of conditional variance in mean equations.
delta_s	$T \times s$ matrix, parameters of square root of conditional variance in mean equations.
cor	$T \times T$ matrix, constant conditional correlation matrix.
zeta	$T \times q$ matrix, leverage (egarch) parameters.
nu	scalar, degrees of freedom for t or skew t distribution.
skew	$T \times 1$ vector, skew parameters for skew t

distribution.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0	normal convergence.
1	forced exit.
2	maximum number of iterations exceeded.
3	function calculation failed.
4	gradient calculation failed.
5	Hessian calculation failed.
6	line search failed.
7	error with constraints.

	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI;
load index[368,9] = index.asc;

x0 = index[.,3:5];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 200;
    retp(c0);
endp;
```

```
struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = CCCGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = real(sqrt(diag(out.moment)));;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print s2[i,.];;
endfor;

likelihood ratio  5422.0195
aic               5466.0195
bic               5551.8774

beta0[1,1]    0.0806     0.0806     0.0806
beta0[2,1]    0.0789     0.0789     0.0789
beta0[3,1]    0.1869    -0.0204     0.3942
```

```
omega[1,1]     1.1092      1.1092      1.1092
omega[2,1]     1.1485     -0.2888      2.5859
omega[3,1]     1.2297      1.0514      1.4081
garch[1,1]     0.6597      0.6597      0.6597
garch[1,2]     0.6475      0.1062      1.1887
garch[1,3]     0.6784      0.6784      0.6784
arch[1,1]      0.0756     -0.1713      0.3225
arch[1,2]     -0.0300     -0.1684      0.1084
arch[1,3]      0.1845     -0.1983      0.5673
nu[1,1]        2.0100           .           .
skew[1,1]     -0.0116     -0.0116     -0.0116
skew[2,1]      0.0116      0.0116      0.0116
skew[3,1]     -0.0012     -0.0489      0.0464
zeta[1,1]     -0.0549     -0.0549     -0.0549
zeta[1,2]     -0.1216     -0.7281      0.4849
zeta[1,3]     -0.1223     -0.7763      0.5316
cor[2,1]       0.8756      0.8466      0.9046
cor[3,1]       0.4484      0.3498      0.5471
cor[3,2]       0.6601      0.5925      0.7276
```

Source

cccegarchmt.src

cccfigarch

Purpose

Estimates parameters of multivariate constantconditional correlation fractionally integrated garch model.

Library

fanpacmt

Format

```
out = cccfigarch(C,D);
```

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
	C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to

		be included.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	*N*×1 vector, time series.
	D.[2].dataMatrix	*N*×*k* matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.	
	out.control	a copy of the input *fanControl* structure.
	out.aic	scalar, scalar, Akiake criterion.
	out.bic	scalar, Bayesian information criterion.
	out.lrs	scalar, likelihood ratio statistic.
	out.numObs	scalar, number of observations.
	out.df	scalar, degress of freedom.

out.par	instance of *PV* structure containing parameter estimates.	
	omega	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
	garch	$T \times p$ matrix of garch coefficients.
	arch	$T \times q$ matrix of arch coefficients.
	AR	$T \times T \times r$ array of autoregression parameters.
	MA	$T \times T \times s$ array of moving average parameters.
	beta0	$T \times 1$ vector, constants in mean equation.
	beta	$T \times k$ vector,

	constants in mean equation.
gamma	$T \times m$ matrix, parameters of exogenous variables in conditional variance equations.
lambda_y	$R \times 1$ matrix, BoxCox parameters for time.
lambda_x	$S \times k$ matrix, BoxCox parameters for exogenous variables.
delta	$T \times r$ matrix, parameters of conditional variance in mean equations.
delta_s	$T \times s$ matrix,

	parameters of square root of conditional variance in mean equations.
cor	$T \times T$ matrix, constant conditional correlation matrix.
dm	scalar, fractional integration parameter.
nu	scalar, degrees of freedom for t or skew t distribution.
skew	$T \times 1$ vector, skew parameters for skew t distribution.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode	scalar, return code.	
	0	normal convergence.
	1	forced exit.
	2	maximum number of iterations exceeded.
	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;

x0 = index[.,3:5];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = CCCFIGarch(f0,d0);
```

```
lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = real(sqrt(diag(out.moment)));;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];;
endfor;

likelihood ratio  2832.8369
aic               2872.8369
bic               2950.8895

beta0[1,1]     0.1115    -0.2743     0.4973
beta0[2,1]     0.0906    -0.4529     0.6340
beta0[3,1]     0.2632    -0.4727     0.9991
omega[1,1]     0.6502    -0.3085     1.6089
omega[2,1]     0.6193    -0.3899     1.6284
omega[3,1]     0.8829    -0.5663     2.3322
garch[1,1]     0.0998    -0.6802     0.8797
garch[1,2]    -0.0176    -0.1544     0.1192
garch[1,3]     0.1413    -0.0234     0.3060
arch[1,1]     -0.0880    -0.1939     0.0179
arch[1,2]     -0.0880    -0.1773     0.0013
```

```
arch[1,3]     -0.0745     -0.1929      0.0438
nu[1,1]        2.0100           .           .
skew[1,1]     -0.0585     -0.3732      0.2562
skew[2,1]      0.0523     -0.3908      0.4953
skew[3,1]      0.0079     -0.1955      0.2112
dm[1,1]        0.1071      0.0592      0.1550
cor[2,1]       0.8593      0.8150      0.9036
cor[3,1]       0.2818      0.1252      0.4383
cor[3,2]       0.5888      0.4742      0.7034
```

Source

cccfigarchmt.src

cccgarch

Purpose

Estimates parameters of multivariate constant conditional correlation garch model.

Library

fanpacmt

Format

out = **cccgarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
	C.CVIndEquations	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	$N \times 1$ vector, time series.

`D.[2].dataMatrix`	$N \times k$ matrix, independent variables (optional).

Output

`out`	instance of a `fanEstimation` structure.		
	`out.control`	a copy of the input `fanControl` structure.	
	`out.aic`	scalar, scalar, Akiake criterion.	
	`out.bic`	scalar, Bayesian information criterion.	
	`out.lrs`	scalar, likelihood ratio statistic.	
	`out.numObs`	scalar, number of observations.	
	`out.df`	scalar, degress of freedom.	
	`out.par`	instance of `PV` structure containing parameter estimates.	
		`omega`	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional

	variance equation.
garch	$T \times p$ matrix of garch coefficients.
arch	$T \times q$ matrix of arch coefficients.
AR	$T \times T \times r$ array of autoregression parameters.
MA	$T \times T \times s$ array of moving average parameters.
beta0	$T \times 1$ vector, constants in mean equation.
beta	$T \times 1$ vector, constants in mean equation.
gamma	$T \times m$ matrix, parameters of exogenous variables in conditional variance

	equations.
`lambda_y`	$R \times 1$ matrix, BoxCox parameters for time.
`lambda_x`	$S \times k$ matrix, BoxCox parameters for exogenous variables.
`delta`	$T \times r$ matrix, parameters of conditional variance in mean equations.
`delta_s`	$T \times s$ matrix, parameters of square root of conditional variance in mean equations.
`cor`	$T \times T$ matrix, constant conditional

	correlation matrix.
tau	$T \times q$ matrix, asymmetry (GJR) parameters.
zeta	$T \times q$ matrix, leverage (egarch) parameters.
dm	scalar, fractional integration parameter.
nu	scalar, degrees of freedom for t or skew t distribution.
skew	$T \times 1$ vector, skew parameters for skew t distribution.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0	normal convergence.
1	forced exit.
2	maximum number of iterations exceeded.
3	function calculation failed.
4	gradient calculation failed.
5	Hessian calculation failed.
6	line search failed.
7	error with constraints.
8	function complex.

out.moment $K \times K$ matrix, moment matrix of parameter.

out.climits $K \times 2$ matrix, confidence limits.

Example

```
new;
cls;

library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;

x0 = index[.,3:5];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 200;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
```

```
out = CCCGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = real(sqrt(diag(out.moment)));;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print s2[i,.];;
  print ftos(x[i],"%*.*lf",10,5;
endfor;
```

```
likelihood ratio  5420.5525
aic               5464.5525
bic               5550.3502

beta0[1,1]    0.1221    -0.0813     0.3255
beta0[2,1]    0.1198    -0.1234     0.3630
omega[1,1]    0.1429    -0.1567     0.4426
omega[2,1]    0.8327     0.8327     0.8327
omega[3,1]    0.9557     0.9557     0.9557
garch[1,1]    0.9438     0.9438     0.9438
garch[1,2]    0.3972     0.3972     0.3972
garch[1,3]    0.3741     0.3741     0.3741
```

```
arch[1,1]      0.0453      0.0031      0.0875
arch[1,2]      0.0344     -0.0032      0.0720
arch[1,3]      0.0626      0.0626      0.0626
nu[1,1]        2.0100
skew[1,1]     -0.0126     -0.0126     -0.0126
skew[2,1]     -0.0047     -0.0047     -0.0047
skew[3,1]      0.0267     -0.0764      0.1299
cor[2,1]       0.8332      0.8106      0.8558
cor[3,1]       0.3511      0.2455      0.4567
cor[3,2]       0.6008      0.5244      0.6773
```

Source

cccgarchmt.src

cccgjrgarch

Purpose

Estimates parameters of multivariate constant conditional correlation GJR garch model.

Library

fanpacmt

Format

out = **cccgjrgarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
	C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.	$N \times 1$ vector, time series.

`[1].dataMatrix`	
`D.[2].dataMatrix`	$N \times k$ matrix, independent variables (optional).

Output

`out`	instance of a `fanEstimation` structure.	
`out.control`	a copy of the input `fanControl` structure.	
`out.aic`	scalar, scalar, Akiake criterion.	
`out.bic`	scalar, Bayesian information criterion.	
`out.lrs`	scalar, likelihood ratio statistic.	
`out.numObs`	scalar, number of observations.	
`out.df`	scalar, degress of freedom.	
`out.par`	instance of `PV` structure containing parameter estimates.	
	`omega`	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in

	conditional variance equation.
`garch`	$T \times p$ matrix of garch coefficients.
`arch`	$T \times q$ matrix of arch coefficients.
`AR`	$T \times T \times r$ array of autoregression parameters.
`MA`	$T \times T \times s$ array of moving average parameters.
`beta0`	$T \times 1$ vector, constants in mean equation.
`beta`	$T \times k$ vector, constants in mean equation.
`gamma`	$T \times m$ matrix, parameters of exogenous variables in conditional

	variance equations.
lambda_y	$R \times 1$ matrix, BoxCox parameters for time.
lambda_x	$S \times k$ matrix, BoxCox parameters for exogenous variables.
delta	$T \times r$ matrix, parameters of conditional variance in mean equations.
delta_s	$T \times s$ matrix, parameters of square root of conditional variance in mean equations.
cor	$T \times T$ matrix, constant

conditional correlation matrix.

tau — $T \times q$ matrix, asymmetry (GJR) parameters.

nu — scalar, degrees of freedom for t or skew t distribution.

skew — $T \times 1$ vector, skew parameters for skew t distribution.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0 — normal convergence.

1 — forced exit.

2 — maximum number

		of iterations exceeded.
	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI;
```

```
load index[368,9] = index.asc;

x0 = index[.,6:7];
x = 250 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.start = {
  -1.0  //  beta0[1,1]
  -1.0  //  beta0[2,1]
  2.0   //  omega[1,1]
  2.0   //  omega[2,1]
  0.01  //  ccc_alpha[1,1]
  0.01  //  ccc_beta[1,1]
  0.2   //  garch[1,1]
  0.3   //  garch[1,2]
  0.01  //  arch[1,1]
  0.01  //  arch[1,2]
  0.01  //  tau[1,1]
  0.01  //  tau[1,2]
  4.0   //  nu[1,1]
  2.0   //  skew[1,1]
  2.0   //  skew[2,1]
};

f0.SQPsolveMTControlProc = &sqpc;
```

```
proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = CCGJRGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = real(sqrt(diag(out.moment)));;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];;
endfor;

likelihood ratio  2275.2080
```

```
aic                2305.2080
bic                2363.7475

beta0[1,1]      -0.8410    -1.1504    -0.5316
beta0[2,1]      -1.1050    -1.4604    -0.7496
omega[1,1]       2.0563     0.8081     3.3044
omega[2,1]       2.0337    -0.9205     4.9880
ccc_alpha[1,1]   0.0760    -1.3200     1.4720
ccc_beta[1,1]    0.0979    -1.4330     1.6287
garch[1,1]       0.2167    -0.2582     0.6916
garch[1,2]       0.3457    -0.6831     1.3745
arch[1,1]       -0.0115    -0.0184    -0.0046
arch[1,2]       -0.0040    -0.0123     0.0043
tau[1,1]         0.0016    -0.0166     0.0197
tau[1,2]         0.0011    -0.0154     0.0176
nu[1,1]          2.0200           .          .
skew[1,1]        0.9327    -0.5909     2.4564
skew[2,1]        2.8457    -0.1397     5.8312
```

Source

cccgarchmt.src

dccegarch

Purpose

Estimates parameters of multivariate dynamic conditional correlation exponential garch model.

Library

fanpacmt

Format

```
out = dccegarch(C,D);
```

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
	C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to

		be included.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	*N*×1 vector, time series.
	D.[2].dataMatrix	*N*×*k* matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.	
	out.control	a copy of the input *fanControl* structure.
	out.aic	scalar, scalar, Akiake criterion.
	out.bic	scalar, Bayesian information criterion.
	out.lrs	scalar, likelihood ratio statistic.
	out.numObs	scalar, number of observations.
	out.df	scalar, degress of freedom.

out.par	instance of *PV* structure containing parameter estimates.	
	omega	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
	garch	$T \times p$ matrix of garch coefficients.
	arch	$T \times q$ matrix of arch coefficients.
	AR	$T \times T \times r$ array of autoregression parameters.
	MA	$T \times T \times s$ array of moving average parameters.
	beta0	$T \times 1$ vector, constants in mean equation.
	beta	$T \times k$ vector,

	constants in mean equation.
gamma	$T \times m$ matrix, parameters of exogenous variables in conditional variance equations.
lambda_y	$R \times 1$ matrix, BoxCox parameters for time.
lambda_x	$S \times k$ matrix, BoxCox parameters for exogenous variables.
delta	$T \times r$ matrix, parameters of conditional variance in mean equations.
delta_s	$T \times s$ matrix,

	parameters of square root of conditional variance in mean equations.
dcc_alpha	scalar conditional correlation coefficient.
dcc_beta	scalar conditional correlation coefficient.
zeta	$T \times q$ matrix, leverage (egarch) parameters.
nu	scalar, degrees of freedom for t or skew t distribution.
skew	$T \times 1$ vector, skew parameters for skew t distribution.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0	normal convergence.
1	forced exit.
2	maximum number of iterations exceeded.
3	function calculation failed.
4	gradient calculation failed.
5	Hessian calculation failed.
6	line search failed.
7	error with constraints.
8	function complex.

out.moment $K \times K$ matrix, moment matrix of parameter.

out.climits $K \times 2$ matrix, confidence limits.

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;

x0 = index[.,3:4];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 1;
    retp(c0);
endp;

f0.start = {
   -0.9 // beta0[1,1]
   -0.8 // beta0[2,1]
    0.9 // omega[1,1]
```

```
    1.0 // omega[2,1]
    0.4 // dcc_alpha[1,1]
    0.5 // dcc_beta[1,1]
    0.3 // garch[1,1]
    0.4 // garch[1,2]
    0.1 // arch[1,1]
    0.1 // arch[1,2]
    2.0 // nu[1,1]
    2.0 // skew[1,2]
    0.7 // skew[1,2]
    0.0 // zeta[1,1]
    0.0 // zeta[1,2]
};

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = DCCEGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = out.climits;;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
```

```
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];;
endfor;

likelihood ratio        1485.1463
aic                     1515.1463
bic                     1573.6858

beta0[1,1]     -0.9170    -1.2084    -0.6255
beta0[2,1]     -0.8253    -1.0997    -0.5509
omega[1,1]      0.9861     0.2408     1.7313
omega[2,1]      1.1008     0.3131     1.8884
dcc_alpha[1,1]  0.4941     0.1626     0.8256
dcc_beta[1,1]   0.5030     0.1684     0.8375
garch[1,1]     -0.3863    -1.4037     0.6310
garch[1,2]     -0.4458    -1.4737     0.5822
arch[1,1]      -0.0327    -0.0948     0.0295
arch[1,2]      -0.0314    -0.0954     0.0327
nu[1,1]         2.0200          .          .
skew[1,1]       2.0666     0.3651     3.7681
skew[2,1]       0.7818    -3.1152     4.6788
zeta[1,1]      -0.0060    -0.0463     0.0343
zeta[1,2]      -0.0082    -0.0494     0.0331
```

Source

dccegarchmt.src

dccfigarch

Purpose

Estimates parameters of multivariate dynamic conditional correlation fractionally integrated garch model.

Library

fanpacmt

Format

out = **dccfigarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.

	`C.density`	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
	`C.indEquations`	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
	`C.covType`	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
`D`	1×1 or 2×1 `DS` structure, data.	
	`D.[1].dataMatrix`	$N \times 1$ vector, time series.
	`D.[2].dataMatrix`	$N \times k$ matrix, independent variables (optional).

Output

`out`	instance of a `fanEstimation` structure.	
	`out.control`	a copy of the input `fanControl` structure.
	`out.aic`	scalar, scalar, Akiake criterion.
	`out.bic`	scalar, Bayesian information

criterion.

out.lrs	scalar, likelihood ratio statistic.
out.numObs	scalar, number of observations.
out.df	scalar, degress of freedom.
out.par	instance of *PV* structure containing parameter estimates.

omega	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
garch	$T \times p$ matrix of garch coefficients.
arch	$T \times q$ matrix of arch coefficients.
AR	$T \times T \times r$ array of autoregression parameters.
MA	$T \times T \times s$ array of moving average

	parameters.
beta0	$T\times 1$ vector, constants in mean equation.
beta	$T\times k$ vector, constants in mean equation.
dcc_alpha	scalar conditional correlation coefficient.
dcc_beta	scalar conditional correlation coefficient.
gamma	$T\times m$ matrix, parameters of exogenous variables in conditional variance equations.
lambda_y	$R\times 1$ matrix, BoxCox parameters for time.

`lambda_x`	*S*×*k* matrix, BoxCox parameters for exogenous variables.
`delta`	*T*×*r* matrix, parameters of conditional variance in mean equations.
`delta_s`	*T*×*s* matrix, parameters of square root of conditional variance in mean equations.
`dm`	scalar, fractional integration parameter.
`nu`	scalar, degrees of freedom for t or skew t distribution.
`skew`	*T*×1 vector, skew

parameters for skew t distribution.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0	normal convergence.
1	forced exit.
2	maximum number of iterations exceeded.
3	function calculation failed.
4	gradient calculation failed.
5	Hessian calculation failed.
6	line search failed.

	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;

x0 = index[.,3:4];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
```

```
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = DCCFIGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = out.climits;;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];;
endfor;

print;
print "unconditional correlation matrix";
print;
print mcor(out,d0);
```

```
likelihood ratio  2300.0465
aic               2338.0465
bic               2412.1965

beta0[1,1]     -0.5028    -2.6801     1.6745
beta0[2,1]     -0.3051    -3.2247     2.6145
beta0[3,1]      0.5814    -2.3355     3.4984
omega[1,1]      1.4429     0.1226     2.7632
omega[2,1]      0.6590    -0.2594     1.5775
omega[3,1]      1.3732     1.3732     1.3732
dcc_alpha[1,1] 0.0000           .          .
dcc_beta[1,1]  0.0101    -0.2177     0.2379
garch[1,1]      0.2532    -0.2529     0.7592
garch[1,2]      0.6333     0.0570     1.2096
garch[1,3]      0.4286     0.4286     0.4286
arch[1,1]      -0.0381    -0.1258     0.0495
arch[1,2]      -0.0106    -0.0677     0.0465
arch[1,3]       0.0033    -0.0379     0.0446
nu[1,1]         7.9489     2.0437    13.8541
skew[1,1]      -0.0173    -0.5023     0.4678
skew[2,1]       0.2310    -0.8910     1.3531
skew[3,1]      -0.1497    -0.5649     0.2654
dm[1,1]         0.0265    -0.0496     0.1025

unconditional correlation matrix

1.0000     0.8910     0.4664
0.8910     1.0000     0.6745
0.4664     0.6745     1.0000
```

Source

dccfigarchmt.src

dccgarch

Purpose

Estimates parameters of multivariate dynamic conditional correlation garch model.

Library

fanpacmt

Format

out = **dccgarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive

		parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
	C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D		1×1 or 2×1 *DS* structure, data.
	D.[1].dataMatrix	$N \times 1$ vector, time series.
	D.[2].dataMatrix	$N \times k$ matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.
	out.control a copy of the input *fanControl*

structure.

out.aic	scalar, scalar, Akiake criterion.
out.bic	scalar, Bayesian information criterion.
out.lrs	scalar, likelihood ratio statistic.
out.numObs	scalar, number of observations.
out.df	scalar, degress of freedom.
out.par	instance of *PV* structure containing parameter estimates.

omega	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
garch	$T \times p$ matrix of garch coefficients.
arch	$T \times q$ matrix of arch coefficients.
AR	$T \times T \times r$ array of autoregression

	parameters.
MA	$T \times T \times s$ array of moving average parameters.
beta0	$T \times 1$ vector, constants in mean equation.
beta	$T \times k$ vector, constants in mean equation.
gamma	$T \times m$ matrix, parameters of exogenous variables in conditional variance equations.
lambda_y	$R \times 1$ matrix, BoxCox parameters for time.
lambda_x	$S \times k$ matrix, BoxCox parameters for

	exogenous variables.
`delta`	$T \times r$ matrix, parameters of conditional variance in mean equations.
`delta_s`	$T \times s$ matrix, parameters of square root of conditional variance in mean equations.
`dcc_alpha`	scalar conditional correlation coefficient.
`dcc_beta`	scalar conditional correlation coefficient.
`tau`	$T \times q$ matrix, asymmetry (GJR) parameters.
`zeta`	$T \times q$ matrix, leverage (egarch)

parameters.

dm	scalar, fractional integration parameter.
nu	scalar, degrees of freedom for t or skew t distribution.
skew	$T \times 1$ vector, skew parameters for skew t distribution.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0	normal convergence.
1	forced exit.
2	maximum number of iterations

		exceeded.
	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Example

```
new;
cls;

library fanpacmt;
#include fanpacmt.sdf
```

```
// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;

x0 = index[.,3:5];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = DCCGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = real(sqrt(diag(out.moment)));;
```

```
print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];;
endfor;

likelihood ratio  4343.8907
aic               4379.8907
bic               4450.1381

beta0[1,1]     -0.0083    -0.0083    -0.0083
beta0[2,1]      0.0250    -0.1361     0.1861
beta0[3,1]      0.3141     0.3141     0.3141
omega[1,1]      1.1408    -0.1112     2.3929
omega[2,1]      1.0548     1.0548     1.0548
omega[3,1]      1.2001     1.2001     1.2001
dcc_alpha[1,1]  0.0000          .          .
dcc_beta[1,1]   0.0101    -0.1679     0.1880
garch[1,1]      0.4776    -0.1043     1.0595
garch[1,2]      0.5433     0.5433     0.5433
garch[1,3]      0.5075     0.5075     0.5075
arch[1,1]      -0.0010    -0.0061     0.0042
arch[1,2]      -0.0007    -0.0007    -0.0007
arch[1,3]      -0.0051    -0.0084    -0.0019
```

```
nu[1,1]          7.9251      4.6133     11.2369
skew[1,1]       -0.0181     -0.2354      0.1992
skew[2,1]        0.0918     -0.1334      0.3169
skew[3,1]       -0.0867     -0.0867     -0.0867


unconditional correlation matrix

1.0000      0.8910      0.4664
0.8910      1.0000      0.6745
0.4664      0.6745      1.0000
```

Source

dccgarchmt.src

dccgjrgarch

Purpose

Estimates parameters of multivariate dynamic conditional correlation GJR garch model.

Library

fanpacmt

Format

out = **dccgjrgarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
	C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.	$N \times 1$ vector, time series.

	[1].dataMatrix
D.[2].dataMatrix	$N \times k$ matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.		
	out.control	a copy of the input *fanControl* structure.	
	out.aic	scalar, scalar, Akiake criterion.	
	out.bic	scalar, Bayesian information criterion.	
	out.lrs	scalar, likelihood ratio statistic.	
	out.numObs	scalar, number of observations.	
	out.df	scalar, degress of freedom.	
	out.par	instance of *PV* structure containing parameter estimates.	
		omega	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in

	conditional variance equation.
garch	$T \times p$ matrix of garch coefficients.
arch	$T \times q$ matrix of arch coefficients.
AR	$T \times T \times r$ array of autoregression parameters.
MA	$T \times T \times s$ array of moving average parameters.
beta0	$T \times 1$ vector, constants in mean equation.
beta	$T \times k$ vector, constants in mean equation.
gamma	$T \times m$ matrix, parameters of exogenous variables in conditional

	variance equations.
`lambda_y`	$R \times 1$ matrix, BoxCox parameters for time.
`lambda_x`	$S \times k$ matrix, BoxCox parameters for exogenous variables.
`delta`	$T \times r$ matrix, parameters of conditional variance in mean equations.
`delta_s`	$T \times s$ matrix, parameters of square root of conditional variance in mean equations.
`dcc_alpha`	scalar conditional correlation

coefficient.

dcc_beta	scalar conditional correlation coefficient.
tau	$T \times q$ matrix, asymmetry (GJR) parameters.
nu	scalar, degrees of freedom for t or skew t distribution.
skew	$T \times 1$ vector, skew parameters for skew t distribution.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0	normal convergence.

	1	forced exit.
	2	maximum number of iterations exceeded.
	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Example

```
library fanpacmt;
#include fanpacmt.sdf
```

```
// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;

x0 = index[.,6:7];
x = 250 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.start = {
  -1.0  // beta0[1,1]
  -1.0  // beta0[2,1]
  2.0   // omega[1,1]
  2.0   // omega[2,1]
  0.01  // dcc_alpha[1,1]
  0.01  // dcc_beta[1,1]
  0.2   // garch[1,1]
  0.3   // garch[1,2]
  0.01  // arch[1,1]
  0.01  // arch[1,2]
  0.01  // tau[1,1]
  0.01  // tau[1,2]
  4.0   // nu[1,1]
  2.0   // skew[1,1]
  2.0   // skew[2,1]
};
```

```
f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = DCCGJRGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = out.climits;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];;
endfor;
```

```
likelihood ratio  2275.2080
aic               2305.2080
bic               2363.7475

beta0[1,1]      -0.8410    -1.1504    -0.5316
beta0[2,1]      -1.1050    -1.4604    -0.7496
omega[1,1]       2.0563     0.8081     3.3044
omega[2,1]       2.0337    -0.9205     4.9880
dcc_alpha[1,1]   0.0760    -1.3200     1.4720
dcc_beta[1,1]    0.0979    -1.4330     1.6287
garch[1,1]       0.2167    -0.2582     0.6916
garch[1,2]       0.3457    -0.6831     1.3745
arch[1,1]       -0.0115    -0.0184    -0.0046
arch[1,2]       -0.0040    -0.0123     0.0043
tau[1,1]         0.0016    -0.0166     0.0197
tau[1,2]         0.0011    -0.0154     0.0176
nu[1,1]          2.0200           .          .
skew[1,1]        0.9327    -0.5909     2.4564
skew[2,1]        2.8457    -0.1397     5.8312
```

Source

dccgarchmt.src

dvfigarch

Purpose

Estimates parameters of diagonal vec multivariate fractionally integrated garch model.

Library

fanpacmt

Format

```
out = dvfigarch(C,D);
```

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t.
	C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.

`C.bxcx`	$1 \times K$ vector, mask indicating which independent variables are to be transformed via the boxcox function.
`C.seriesbxcx`	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
`C.CVIndEquations`	$L \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
`C.inMean`	scalar, $M \times L$ matrix, a $1 \times L$ vector for each mean equation indicating which conditional variance equation is included in which mean equation.
`C.stConstType`	scalar, type of enforcement of stationarity requirements, 1 - roots of characteristic polynomial constrained outside unit circle, 2 - arch, garch parameters constrained to sum to less than one and greater than zero, 3 - none.

	C.cvConstType	scalar, type of enforcement of nonnegative conditional variances, 0 - direct constraints, 1 - Nelson & Cao constraints.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	N×1 vector, time series.
	D.[2].dataMatrix	N×k matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.	
	out.model	scalar, set to 17.
	out.control	a copy of the input *fanControl* structure.
	out.aic	scalar, scalar, Akiake criterion.
	out.bic	scalar, Bayesian information criterion.

out.lrs	scalar, likelihood ratio statistic.	
out.numObs	scalar, number of observations.	
out.df	scalar, degress of freedom.	
out.par	instance of *PV* structure containing parameter estimates.	
	omega	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
	garch	$T \times p$ matrix of garch coefficients.
	arch	$T \times q$ matrix of arch coefficients.
	AR	$T \times T \times r$ array of autoregression parameters.

`MA`	$T \times T \times s$ array of moving average parameters.
`beta0`	$T \times 1$ vector, constants in mean equation.
`beta`	$T \times 1$ vector, constants in mean equation.
`gamma`	$T \times m$ matrix, parameters of exogenous variables in conditional variance equations.
`lambda_y`	$R \times 1$ matrix, BoxCox parameters for time.
`lambda_x`	$S \times k$ matrix, BoxCox parameters for exogenous

	variables.
delta	*T*×*r* matrix, parameters of conditional variance in mean equations.
delta_s	*T*×*s* matrix, parameters of square root of conditional variance in mean equations.
dm	scalar, fractional integration parameter.
nu	scalar, degrees of freedom for t or skew t distribution.
skew	*T*×1 vector, skew parameters for skew t distribution.

out.retcode	scalar, return code.	
	0	normal convergence.
	1	forced exit.
	2	maximum number of iterations exceeded.
	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

out.tsForecast	$M \times L$ matrix, time series forecast.
out.cvForecast	$M \times L \times L$ array, forecast of conditional covariance matrices.

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;

x0 = index[.,6:7];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;
f0.STconstType = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
```

```
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = DVFIGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = out.climits;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];
endfor;
```

Source

mgarchmt.src

dvgarch

Purpose

Estimates parameters of diagonal vec multivariate garch model.

Library

fanpacmt

Format

out = **dvgarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 -

	Normal, 1 - Student's t.
C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
C.bxcx	$1 \times K$ vector, mask indicating which independent variables are to be transformed via the boxcox function.
C.seriesbxcx	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
C.CVIndEquations	$L \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
C.inMean	scalar, $M \times L$ matrix, a $1 \times L$ vector for each mean equation indicating which conditional variance equation is included in which mean equation.
C.stConstType	scalar, type of enforcement of stationarity requirements, 1 -

		roots of characteristic polynomial constrained outside unit circle, 2 - arch, garch parameters constrained to sum to less than one and greater than zero, 3 - none.
	C.cvConstType	scalar, type of enforcement of nonnegative conditional variances, 0 - direct constraints, 1 - Nelson & Cao constraints.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	N×1 vector, time series.
	D.[2].dataMatrix	N×k matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.	
	out.model	scalar, set to 17.
	out.control	a copy of the input

	fanControl structure.
out.aic	scalar, scalar, Akiake criterion.
out.bic	scalar, Bayesian information criterion.
out.lrs	scalar, likelihood ratio statistic.
out.numObs	scalar, number of observations.
out.df	scalar, degress of freedom.
out.par	instance of *PV* structure containing parameter estimates.
	omega — $T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
	garch — $T \times p$ matrix of garch coefficients.

`arch`	$T \times q$ matrix of arch coefficients.
`AR`	$T \times T \times r$ array of autoregression parameters.
`MA`	$T \times T \times s$ array of moving average parameters.
`beta0`	$T \times 1$ vector, constants in mean equation.
`beta`	$T \times 1$ vector, constants in mean equation.
`gamma`	$T \times m$ matrix, parameters of exogenous variables in conditional variance equations.
`lambda_y`	$R \times 1$ matrix, BoxCox parameters for

	time.
`lambda_x`	$S \times k$ matrix, BoxCox parameters for exogenous variables.
`delta`	$T \times r$ matrix, parameters of conditional variance in mean equations.
`delta_s`	$T \times s$ matrix, parameters of square root of conditional variance in mean equations.
`tau`	$T \times q$ matrix, asymmetry (GJR) parameters.
`zeta`	$T \times q$ matrix, leverage (egarch) parameters.
`dm`	scalar, fractional

		integration parameter.
	nu	scalar, degrees of freedom for t or skew t distribution.
	skew	$T \times 1$ vector, skew parameters for skew t distribution.
out.retcode	scalar, return code.	
	0	normal convergence.
	1	forced exit.
	2	maximum number of iterations exceeded.
	3	function calculation failed.
	4	gradient calculation failed.

	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	
out.tsForecast	$M \times L$ matrix, time series forecast.	
out.cvForecast	$M \times L \times L$ array, forecast of conditional covariance matrices.	

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;
```

```
x0 = index[.,6:7];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;
f0.STconstType = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = DVGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = out.climits;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
```

```
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];
endfor;

likelihood ratio  3300.8945
aic               3328.8945
bic               3383.5314

beta0[1,1]     0.4093    -0.4601     1.2786
beta0[2,1]     0.2961    -0.7386     1.3307
omega[1,1]     0.4309    -0.2435     1.1054
omega[2,1]     0.2514    -0.0698     0.5725
omega[3,1]     0.2536    -0.0807     0.5880
garch[1,1]     0.8732     0.7172     1.0293
garch[1,2]     0.9030     0.8154     0.9907
garch[1,3]     0.9337     0.8737     0.9937
arch[1,1]      0.0396    -0.0034     0.0826
arch[1,2]      0.0317     0.0026     0.0607
arch[1,3]      0.0246     0.0001     0.0491
nu[1,1]       11.1374     3.6854    18.5894
skew[1,1]     -0.1183    -0.4258     0.1892
skew[2,1]      0.0497    -0.2137     0.3130
```

Source

mgarchmt.src

dvgjrgarch

Purpose

Estimates parameters of diagonal vec multivariate GJR garch model.

Library

fanpacmt

Format

out = **dvgjrgarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.
	C.ma	scalar, order of the moving average parameters.
	C.density	scalar, density of error term, 0 -

	Normal, 1 - Student's t.
C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
C.bxcx	$1 \times K$ vector, mask indicating which independent variables are to be transformed via the boxcox function.
C.seriesbxcx	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
C.CVIndEquations	$L \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
C.inMean	scalar, $M \times L$ matrix, a $1 \times L$ vector for each mean equation indicating which conditional variance equation is included in which mean equation.
C.stConstType	scalar, type of enforcement of stationarity requirements, 1 -

		roots of characteristic polynomial constrained outside unit circle, 2 - arch, garch parameters constrained to sum to less than one and greater than zero, 3 - none.
	C.cvConstType	scalar, type of enforcement of nonnegative conditional variances, 0 - direct constraints, 1 - Nelson & Cao constraints.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	N×1 vector, time series.
	D.[2].dataMatrix	N×k matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.	
	out.model	scalar, set to 17.
	out.control	a copy of the input

	fanControl structure.
out.aic	scalar, scalar, Akiake criterion.
out.bic	scalar, Bayesian information criterion.
out.lrs	scalar, likelihood ratio statistic.
out.numObs	scalar, number of observations.
out.df	scalar, degress of freedom.
out.par	instance of *PV* structure containing parameter estimates.
	omega — $T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
	garch — $T \times p$ matrix of garch coefficients.

`arch`	$T \times q$ matrix of arch coefficients.
`AR`	$T \times T \times r$ array of autoregression parameters.
`MA`	$T \times T \times s$ array of moving average parameters.
`beta0`	$T \times 1$ vector, constants in mean equation.
`beta`	$T \times 1$ vector, constants in mean equation.
`gamma`	$T \times m$ matrix, parameters of exogenous variables in conditional variance equations.
`lambda_y`	$R \times 1$ matrix, BoxCox parameters for

	time.
`lambda_x`	$S \times k$ matrix, BoxCox parameters for exogenous variables.
`delta`	$T \times r$ matrix, parameters of conditional variance in mean equations.
`delta_s`	$T \times s$ matrix, parameters of square root of conditional variance in mean equations.
`tau`	$T \times q$ matrix, asymmetry (GJR) parameters.
`nu`	scalar, degrees of freedom for t or skew t distribution.

	skew	$T \times 1$ vector, skew parameters for skew t distribution.
out.retcode	scalar, return code.	
	0	normal convergence.
	1	forced exit.
	2	maximum number of iterations exceeded.
	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.

	8	function complex.
out.moment		$K \times K$ matrix, moment matrix of parameter.
out.climits		$K \times 2$ matrix, confidence limits.
out.tsForecast		$M \times L$ matrix, time series forecast.
out.cvForecast		$M \times L \times L$ array, forecast of conditional covariance matrices.

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI; lo
ad index[368,9] = index.asc;

x0 = index[.,6:7];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;
```

```
f0.STconstType = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = DVGJRGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = out.climits;

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print x[i];;
  print s2[i,.];
endfor;
```

```
likelihood ratio  1714.5147
aic               1744.5147
bic               1803.0542

beta0[1,1]    -0.2818    -1.0594     0.4958
beta0[2,1]    -1.3156    -2.1086    -0.5225
omega[1,1]     0.7937    -0.1859     1.7733
omega[2,1]     0.1834    -0.1052     0.4719
omega[3,1]     0.5619     0.5619     0.5619
garch[1,1]     0.7002     0.4409     0.9595
garch[1,2]     0.9036     0.7739     1.0332
garch[1,3]     0.8781     0.8781     0.8781
arch[1,1]     -0.2195    -0.4409     0.0019
arch[1,2]     -0.0647    -0.1551     0.0258
arch[1,3]     -0.0884    -0.0884    -0.0884
nu[1,1]        2.7784     1.9613     3.5954
skew[1,1]     -0.2341    -0.5016     0.0334
skew[2,1]      0.4145     0.2175     0.6116
dm[1,1]        0.9990          .          .
```

Source

mgarchmt.src

fmgarch

Purpose

Estimates parameters of multivariate factor garch model.

Library

fanpacmt

Format

```
out = fmgarch(C,D);
```

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.k	scalar, number of factors.
	C.density	scalar, density of error term, 0 - Normal, 1 - Student's t.
	C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.

D	1×1 or 2×1 *DS* structure, data.
	D.[1].dataMatrix — *N*×1 vector, time series.
	D.[2].dataMatrix — *N*×*k* matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.
	out.control — a copy of the input *fanControl* structure.
	out.aic — scalar, scalar, Akiake criterion.
	out.bic — scalar, Bayesian information criterion.
	out.lrs — scalar, likelihood ratio statistic.
	out.numObs — scalar, number of observations.
	out.df — scalar, degress of freedom.
	out.par — instance of *PV* structure containing parameter estimates.
	omega — *T*×*T* lower triangular matrix,

	Choleski factorization of constant matrix in conditional variance equation.
garch	$T \times p$ matrix of garch coefficients.
arch	$T \times q$ matrix of arch coefficients.
AR	$T \times T \times r$ array of autoregression parameters.
MA	$T \times T \times s$ array of moving average parameters.
beta0	$m \times 1$ constants in mean equation.
beta	$m \times n$ matrix, regression coefficients in mean equation if there are independent variables.

	lambda	$m \times k$ matrix, factor loadings.
	w	$m \times k$ matrix, factors.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0	normal convergence.
1	forced exit.
2	maximum number of iterations exceeded.
3	function calculation failed.
4	gradient calculation failed.
5	Hessian calculation failed.

	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Example

```
library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI;
load index[368,9] = index.asc;

x0 = index[.,3:4];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.k = 2;
f0.density = 3;
```

```
f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = FMGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = real(sqrt(diag(out.moment)));

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print s2[i,.];;
  print ftos(x[i],"%*.*lf",10,5);
endfor;
```

Source

fmgarchmt.src

gogarch

Purpose

Estimates parameters of multivariate generalized orthogonal garch model.

Library

fanpacmt

Format

out = **gogarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.density	scalar, density of error term, 0 -

		Normal, 1 - Student's t.
	`C.indEquations`	$M \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
	`C.covType`	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
`D`		1×1 or 2×1 `DS` structure, data.
	`D.[1].dataMatrix`	$N \times 1$ vector, time series.
	`D.[2].dataMatrix`	$N \times k$ matrix, independent variables (optional).

Output

`out`		instance of a `fanEstimation` structure.
	`out.control`	a copy of the input `fanControl` structure.
	`out.aic`	scalar, scalar, Akiake criterion.
	`out.bic`	scalar, Bayesian information criterion.

out.lrs	scalar, likelihood ratio statistic.
out.numObs	scalar, number of observations.
out.df	scalar, degress of freedom.
out.par	instance of *PV* structure containing parameter estimates.

omega	$T \times T$ lower triangular matrix, Choleski factorization of constant matrix in conditional variance equation.
garch	$T \times p$ matrix of garch coefficients.
arch	$T \times q$ matrix of arch coefficients.
AR	$T \times T \times r$ array of autoregression parameters.
MA	$T \times T \times s$ array of moving average parameters.

beta0 $m \times 1$ constants in mean equation.

beta $m \times n$ matrix, regression coefficients in mean equation if there are independent variables.

lambda $m \times k$ matrix, factor loadings.

To extract an estimated matrix use the **pvUnpack** function:

```
arch = pvUnpack(out.par,
    "arch");
```

out.retcode scalar, return code.

0 normal convergence.

1 forced exit.

2 maximum number of iterations exceeded.

	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Example

```
new;
cls;

library fanpacmt;
#include fanpacmt.sdf

// DATE COMPQ INDU SPX RUT FTSE DAX NIKKEI HSI;
```

```
load index[368,9] = index.asc;

x0 = index[.,3:4];
x = 200 * ln(x0[2:368,.]./x0[1:367,.]);

struct FanControl f0;
f0 = fanControlCreate;
f0.p = 1;
f0.q = 1;
f0.density = 3;

f0.SQPsolveMTControlProc = &sqpc;

proc sqpc(struct sqpsolvemtControl c0);
    c0.printIters = 100;
    retp(c0);
endp;

f0.start = {
  0.1, // beta0[1,1]
  0.1, // beta0[2,1]
  0.6, // omega[1,1]
  0.7, // omega[2,1]
  0.1, // garch[1,1]
  0.1, // garch[2,1]
  0.1, // arch[1,1]
  0.1, // arch[2,1]
  0.6, // Lambda[1,1]
  0.3, // Lambda[1,2]
  0.4, // Lambda[2,1]
```

```
  0.5  // Lambda[2,2]
};

struct DS d0;
d0 = dsCreate;
d0.dataMatrix = x;

struct fanEstimation out;
out = GOGarch(f0,d0);

lbl = pvGetParNames(out.par);
x = pvGetParVector(out.par);
s2 = real(sqrt(diag(out.moment)));

print "likelihood ratio " out.lrs;
print "aic              " out.aic;
print "bic              " out.bic;
print;

format /rd 10,4;
for i(1,rows(x),1);
  print lbl[i];;
  print s2[i,.];;
  print ftos(x[i],"%*.*lf",10,5);
endfor;

likelihood ratio        2839.1777
aic                     2863.1777
bic                     2910.0093
```

```
beta0[1,1]     0.1672     0.1233
beta0[2,1]     0.1857     0.1350
omega[1,1]     0.6637     0.1063
omega[2,1]     0.7423     0.1403
garch[1,1]     0.1611     0.0999
garch[2,1]     0.1403     0.1181
arch[1,1]      0.1751     0.0537
arch[2,1]      0.1174     0.0462
Lambda[1,1]    0.6379     0.0873
Lambda[1,2]    0.3039     0.0643
Lambda[2,1]    0.4411     0.0000
Lambda[2,2]    0.5682     0.1395
```

Source

gogarchmt.src

mcvar

Purpose

Computes conditional variances for the multivariate garch model.

Library

fanpacmt

Format

```
r = mcvar(F,D);
```

Input

F	instance of a *fanEstimation* structure.
D	1×1 or 2×1 *DS* structure, data.
D.[1].dataMatrix	*N*×*M* matrix, time series.
D.[2].dataMatrix	*N*×*k* matrix, independent variables (optional).

Output

r	*N*×1 vector, conditional variances .

Source

mgarchmt.src

mforecast

Purpose

Computes time series and conditional variance forecasts for multivariate time series.

Library

fanpacmt

Format

```
r = mforecast(F,D,period,xp);
```

Input

F	instance of a *fanEstimation* structure.
D	1×1 or 2×1 *DS* structure, data.
D.[1].dataMatrix	*N*×*M* matrix, time series.
D.[2].dataMatrix	*N*×*k* matrix, independent variables (optional).
period	scalar, number of periods to be forecast.
xp	*M*×*R* 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*×*M* matrix, *L* period forecast of times series.
s	*L*×*M* matrix or *L*×*M*×*M* array, period forecast of conditional variance.

Source

mgarchmt.src

mgarch

Purpose

Estimates parameters of multivariate time series.

Library

fanpacmt

Format

out = **mgarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.

C.ma	scalar, order of the moving average parameters.
C.density	scalar, density of error term, 0 - Normal, 1 - Student's t.
C.multModel	scalar, 0 - diagonal vec, 1 - constant correlation diagonal vec, 2 - BEKK.
C.fractional	scalar, if nonzero, fractional integrated model.
C.leverage	scalar, if nonzero leverage terms are added.
C.assymetry	scalar, if nonzero assymetry terms are added.
C.unitRoot	scalar, if nonzero a unit root is forced on the determinantal polynomial.
C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
C.bxcx	$1 \times K$ vector, ask indicating which independent variables are to be transformed via the

	boxcox function.
C.seriesbxcx	scalar, if nonzero the time series is transformed via the boxcox function.
C.CVIndEquations	$L \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
C.inMean	scalar, $M \times L$ matrix, a $1 \times L$ vector for each mean equation indicating which conditional variance equation is included in which mean equation.
C.stConstType	scalar, type of enforcement of stationarity requirements, 1 - roots of characteristic polynomial constrained outside unit circle, 2 - arch, garch parameters constrained to sum to less than one and greater than zero, 3 - none.
C.cvConstType	scalar, type of enforcement of nonnegative conditional variances, 0 - direct constraints,

		1 - Nelson & Cao constraints.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	N×1 vector, time series.
	D.[2].dataMatrix	N×k matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.		
	out.model	scalar.	
		0	OLS.
		1	ARIMA.
		2	GARCH.
		3	FIGARCH.
		4	IGARCH.
		5	EGARCH.
		10	SIMEQ.

	11	VARMA.
	12	DVGARCH.
	13	DVFIGARCH.
	14	CDVGARCH.
	15	CDVFIGARCH.
	16	CDVEGARCH.
	17	BKGARCH.
out.control	a copy of the input *fanControl* structure.	
out.aic	scalar, scalar, Akiake criterion.	
out.bic	scalar, Bayesian information criterion.	
out.lrs	scalar, likelihood ratio statistic.	
out.numObs	scalar, number of observations.	
out.df	scalar, degress of freedom.	
out.par	instance of *PV* structure containing parameter estimates.	
	omega	$T \times T$ lower triangular matrix, Choleski

	factorization of constant matrix in conditional variance equation.
garch	*T*×*p* matrix of garch coefficients.
arch	*T*×*q* matrix of arch coefficients.
AR	*T*×*T*×*r* array of autoregression parameters.
MA	*T*×*T*×*s* array of moving average parameters.
beta0	*T*×1 vector, constants in mean equation.
beta	*T*×1 vector, constants in mean equation.
gamma	*T*×*m* matrix, parameters of exogenous variables

	in conditional variance equations.
`lambda_y`	$R \times 1$ matrix, BoxCox parameters for time.
`lambda_x`	$S \times k$ matrix, BoxCox parameters for exogenous variables.
`delta`	$T \times r$ matrix, parameters of conditional variance in mean equations.
`delta_s`	$T \times s$ matrix, parameters of square root of conditional variance in mean equations.
`cor`	$T \times T$ matrix, constant conditional correlation matrix.
`dcc_alpha`	$T \times q$ matrix, parameter in dynamic conditional correlation model.

dcc_beta	$T \times q$ matrix, parameter in dynamic conditional correlation model.
Lambda	$T \times n$ matrix, loadings matrix.
W	$T \times$n matrix, factor moments matrix.
tau	$T \times q$ matrix, asymmetry (GJR) parameters.
zeta	$T \times q$ matrix, leverage (egarch) parameters.
dm	scalar, fractional integration parameter.
nu	scalar, degrees of freedom for t or skew t distribution.
skew	$T \times 1$ vector, skew parameters for skew t distribution.

out.retcode	scalar, return code.	
	0	normal convergence.
	1	forced exit.
	2	maximum number of iterations exceeded.
	3	function calculation failed.
	4	gradient calculation failed.
	5	Hessian calculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter.	
out.climits	$K \times 2$ matrix, confidence limits.	

Source

mgarchmt.src

mres

Purpose

Computes residuals for the multivariate garch model.

Library

fanpacmt

Format

r = **mres**(*F*,*D*,*s*);

Input

F	instance of a *fanEstimation* structure.	
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	*N*×*M* matrix, time series.
	D.[2].dataMatrix	*N*×*k* matrix, independent variables (optional).
s	scalar, if nonzero standardized residuals are computed, otherwise they are unstandardized. Default = 0.	

Output

r	*N*×1 vector, residuals.

Source

mgarchmt.src

mroots

Purpose

Computes residuals for the multivariate garch model.

Library

fanpacmt

Format

r = **mroots**(*F*);

Input

F	instance of a *fanEstimation* structure.

Output

r	$N \times 1$ vector, roots.

Remarks

For the diagonal vec model \commandname{mroot} computes roots of

$$1-\left[\left(\alpha_1+\beta_1\right)\right]Z-\left[\left(\alpha_2+\beta_2\right)\right]Z^2+\ldots$$
$$1-\left[\beta_1\right]Z-\left[\beta_2\right]Z^2+\ldots\left[\beta_p\right]Z^p$$

where α_i and β_i are $L(L+1)/2 \times 1$ vectors of the ARCH and GARCH parameters for the diagonal vec model and [ ] indicates expansion to symmetric matrices.

For the constant correlation diagonal vec model α_i and β_i $L \times 1$ vectors and [ ] indicates expansion to diagonal matrices.

For the BEKK model α_i and β_i are $L \times L$ matrices of parameters and [ ] indicates no change.

For all models, roots of the characteristic polynomial for the AR and MA parameters are also computed:

$$1-\phi_1 Z-\phi_2 Z^2+\ldots+\phi_p Z^p$$
$$1-\theta_1 Z-\theta_2 Z^2+\ldots+\theta_p Z^p$$

where the ϕ_i are the AR parameters, and where the θ_i are the MA parameters.

Source

mgarchmt.src

mSimulation

Purpose

Simulates time series.

Library

fanpacmt

Format

D = **mSimulation**(*S*);

Input

S	instance of a *fanSimulation* structure.	
	S.par	instance of *PV* structure containing packed parameter matrices.
		S.par.beta0 $L \times 1$ vector, constants in mean equations.

	`S.par.omega`	$L \times 1$ vector, constants in variance equations.
	`S.par.garch`	$P \times L$ matrix, garch parameters.
	`S.par.arch`	$Q \times L$ matrix, arch parameters.
	`S.par.beta0`	$L \times 1$ vector, constants in mean equations.
	`S.par.phi`	$R \times L \times L$ array, AR parameters.
	`S.par.theta`	$R \times L \times L$ array, MA parameters.
	`S.par.tau`	$L \times 1$ vector, asymmetry parameters.
	`S.par.delta`	$L \times 1$ vector, inmean coefficients, variance.
	`S.par.delta_s`	$L \times 1$ vector, inmean coefficients, standard dev.
`S.numObs`	scalar, number of observations.	
`S.seed`	scalar, seed for random number.	

Output

D	1×1 or 2×1 *DS* structure, data. *D.[1].dataMatrix* $N \times L$ vector, time series.

Remarks

Parameters are specified by packing the appropriate matrices into S.par using the \commandname{pvPackm} functions. For example,

```
struct PV p0;
struct fanSimulation s0;

p0 = pvPack(p0,.5~.6,"garch");
p0 = pvPack(p0,.3~.4,"arch");
p0 = pvPack(p0,.5|.4,"beta0");
p0 = pvPack(p0,1|1,"omega");

s0.par = p0;
s0.numObs = 100;

struct DS d0;
d0 = mSimulation(s0);
```

Source

mgarchmt.src

simlimits

Purpose

Statistical inference using Andrews simulation method.

Library

fanpacmt

Format

cl,vc = **simlimits**(*&fnct,S,D,F*);

Input

&fnct	scalar, pointer to procedure computing minus log-likelihood.
S	instance of a *sqpSolveMTout* structure containing results of estimation.
D	1×1 or 2×1 *DS* structure, data.
	D.[1].dataMatrix — *N*×1 vector, time series.
	D.[2].dataMatrix — *N*×*k* matrix, independent variables (optional).
F	instance of a *fanControl* structure containing control

variables used in estimation.

Output

cl	*N*×1 vector, lower and upper confidence limits.
vc	*K*×*K* matrix, covariance matrix.

Source

simlimitsmt.src

ucvar

Purpose

Statistical inference using Andrews simulation method.

Library

fanpacmt

Format

r = **ucvar**(*F*, *D*);

Input

F	instance of a *fanEstimation* structure.
D	1×1 or 2×1 *DS* structure, data.
D.[1].dataMatrix	*N*×1 vector, time series.
D.[2].dataMatrix	*N*×*k* matrix, independent variables (optional).

Output

r	*N*×1 vector, vector, conditional variances.

Source

ugarchmt.src

uforecast

Purpose

Statistical inference using Andrews simulation method.

Library

fanpacmt

Format

```
r,s = uforecast(F,D,period,xp);
```

Input

F	instance of a *fanEstimation* structure.
D	1×1 or 2×1 *DS* structure, data.
D.[1].dataMatrix	*N*×1 vector, time series.
D.[2].dataMatrix	*N*×*k* matrix, independent variables (optional).
period	scalar, number of periods to be forecast.
xp	*M*×*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	*M*×1 vector, *M* period forecast of times series.
s	*M*×1 vector, *M* period forecast of conditional variance.

Source

ugarchmt.src

ugarch

Purpose

Estimates parameters of univariate time series.

Library

fanpacmt

Format

out = **ugarch**(*C*,*D*);

Input

C	instance of a *fanControl* structure.	
	C.p	scalar, order of the garch parameters.
	C.q	scalar, order of the arch parameters.
	C.ar	scalar, order of the autoregressive parameters.

C.ma	scalar, order of the moving average parameters.
C.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 skew t distribution.
C.multModel	scalar, 0 - diagonal vec, 1 - constant correlation diagonal vec, 2 - BEKK.
C.fractional	scalar, if nonzero, fractional integrated model.
C.leverage	scalar, if nonzero leverage terms are added.
C.assymetry	scalar, if nonzero assymetry terms are added.
C.unitRoot	scalar, if nonzero a unit root is forced on the determinantal polynomial.
C.indEquations	$M \times K$ matrix, a $1 \times K$ vector for each mean equation indicating which independent variables are included.
C.bxcx	$1 \times K$ vector, mask indicating which independent variables are

	to be transformed via the boxcox function.
C.seriesbxcx	scalar, if nonzero the time series is transformed via the boxcox function.
C.CVIndEquations	$L \times K$ matrix, a $1 \times K$ vector for each conditional variance equation indicating independent variables to be included.
C.inMean	scalar, $M \times L$ matrix, a $1 \times L$ vector for each mean equation indicating which conditional variance equation is included in which mean equation.
C.stConstType	scalar, type of enforcement of stationarity requirements, 1 - roots of characteristic polynomial constrained outside unit circle, 2 - arch, garch parameters constrained to sum to less than one and greater than zero, 3 - none.
C.cvConstType	scalar, type of enforcement of nonnegative conditional

		variances, 0 - direct constraints, 1 - Nelson & Cao constraints.
	C.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
D	1×1 or 2×1 *DS* structure, data.	
	D.[1].dataMatrix	N×1 vector, time series.
	D.[2].dataMatrix	N×k matrix, independent variables (optional).

Output

out	instance of a *fanEstimation* structure.		
	out.model	scalar.	
		0	OLS.
		1	ARIMA.
		2	GARCH.
		3	FIGARCH.
		4	IGARCH.
		5	EGARCH.

	10	SIMEQ.
	11	VARMA.
	12	DVGARCH.
	13	DVFIGARCH.
	14	CDVGARCH.
	15	CDVFIGARCH.
	16	CDVEGARCH.
	17	BKGARCH.
out.control	a copy of the input *fanControl* structure.	
out.aic	scalar, scalar, Akiake criterion.	
out.bic	scalar, Bayesian information criterion.	
out.lrs	scalar, likelihood ratio statistic.	
out.numObs	scalar, number of observations.	
out.df	scalar, degress of freedom.	
out.par	instance of *PV* structure containing parameter estimates.	

out.retcode	scalar, return code.
	0 normal convergence.
	1 forced exit.
	2 maximum number of iterations exceeded.
	3 function calculation failed.
	4 gradient calculation failed.
	5 Hessian calculation failed.
	6 line search failed.
	7 error with constraints.
	8 function complex.
out.moment	$K \times K$ matrix, moment matrix of parameter estimates.
out.climits	$K \times 2$ matrix, confidence limits.

out.tsForecast	*M*×1 vector, time series forecast.
out.cvForecast	*M*×1×1 array, forecast of conditional variances.

Source

ugarchmt.src

uRes

Purpose

Computes residuals for the univariate garch model.

Library

fanpacmt

Format

r = **uRes**(*F*,*D*,*s*);

Input

F	instance of a *fanEstimation* structure.
D	1×1 or 2×1 *DS* structure, data.

`D.[1].dataMatrix`	$N \times 1$ vector, time series.
`D.[2].dataMatrix`	$N \times k$ matrix, independent variables (optional).
`s`	scalar, if nonzero standardized residuals are computed, otherwise they are unstandardized. Default = 0.

Output

`r`	$N \times 1$ vector, residuals.

Source

ugarchmt.src

uRoots

Purpose

Computes roots of the characteristic polynomial for univariate models.

Library

fanpacmt

Format

```
r = uRoots(F);
```

Input

F	instance of a *fanEstimation* structure.

Output

r	$L \times 1$ vector, roots.

Remarks

Computes roots of

$$1-\left(\alpha_1+\beta_1\right)-\left(\alpha_2+\beta_2\right)Z^2+\ldots$$
$$1-\beta_1 Z-\beta_2 Z^2+\beta_p Z^p$$

and

$$1-\phi_1 Z-\phi_2 Z^2+\ldots+\phi_p Z^p$$
$$1-\theta_1 Z-\theta_2 Z^2+\ldots+\theta_p Z^p$$

where the β_i are the GARCH parameters, where the α_i are the ARCH parameters, where the ϕ_i are the AR parameters, and where the θ_i are the MA parameters.

Source

ugarchmt.src

uSimulation

Purpose

Simulates univariate time series.

Library

fanpacmt

Format

D = **uSimulation**(*S*);

Input

S	instance of a *fanEstimation* structure.	
	S.par	instance of a *PV* structure containing packed parameter matrices.

`S.par.beta0`	foo.
`S.par.foo`	scalar, constant in mean equations.
`S.par.omega`	scalar, constant in variance equations.
`S.par.garch`	$P\times 1$ vector, garch parameters.
`S.par.arch`	$Q\times 1$ vector, arch parameters.
`S.par.phi`	$R\times 1$ vector, AR parameters.
`S.par.theta`	$S\times 1$ vector, MA parameters.
`S.par.tau`	scalar, asymmetry parameter.
`S.par.delta`	scalar, inmean coefficient, variance.

S.par.delta_s	scalar, inmean coefficient, standard dev.
S.numObs	scalar, number of observations.
S.seed	scalar, seed for random number generatorscalar, seed for random number generator.

Output

D	1×1 or 2×1 *DS* structure, data.
D.[1].dataMatrix	$N \times L$ vector, time series.

Remarks

Parameters are specified by packing the appropriate matrices into *S.par* using the **pvPackm** functions. For example,

```
struct PV p0;
struct fanSimulation s0;

p0 = pvPack(p0,.5,"garch");
p0 = pvPack(p0,.3,"arch");
p0 = pvPack(p0,.5,"beta0");
p0 = pvPack(p0,1,"omega");
```

```
s0.par = p0;
s0.numObs = 100;

struct DS d0;
d0 = uSimulation(s0);
```

Source

ugarchmt.src

5 FANPACMT 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.
`computeReturns`	Computes returns from price data.
`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 \substitute{_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.
`setBoxcox`	Indicates variables for Box-Cox transformation.
`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.
`setIndLogElapsedPeriod`	Creates independent variable measuring elapsed time between observations.
`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.

clearSession

Purpose

Resets globals to default values.

Library

fanpacmt

Format

```
clearSession;
```

Source

fankeymt.src

computeLogReturns

Purpose

Computes log returns from price data.

Library

fanpacmt

Format

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

Input

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

Global Input

_fan_Series	$N \times k$ matrix, time series.

Global Output

_fan_Series	$N \times k$ 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

fankeymt.src

computePercentReturns

Purpose

Computes log returns from price data.

Library

fanpacmt

Format

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

Input

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

Global Input

_fan_Series	$N \times k$ matrix, time series.

Global Output

_fan_Series	$N \times k$ matrix, time series.

Remarks

Computes the percent returns from price data.

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

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

fankeymt.src

constrainPDCovPar

Purpose

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

Library

fanpacmt

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 **fanpacmt** 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 **fanpacmt** keyword command **showResults**.

Source

fankeymt.src

estimate

Purpose

Generates estimates of parameters of a time series model.

Library

fanpacmt

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: If a GARCH model name is appended with an M, an inmean model is estimated, and if with a V, an inCV model is estimated.

OLS	Normal ordinary least squares.
TOLS	t distribution ordinary least squares.
ARIMA(r,d,s)	Normal ARIMA. If r, d, and s are not specified, an ARIMA(1,1,1) is estimated.
TARIMA (p,d,q)	t distribution ARIMA.
EGARCH	EGARCH with generalized error distribution.

NEGARCH	EGARCH with Normal distribution.
TEGARCH	EGARCH with t distribution.
GTEGARCH	EGARCH with skew generalized t distribution.
AGARCH (p,q,r,s)	Normal GARCH with asymmetry parameters.
TAGARCH (p,q,r,s)	Student's t distribution GARCH with asymmetry parameters.
GTAGARCH (p,q,r,s)	Skew Generalized t distribution GARCH with asymmetry parameters.
GARCH (p,q,r,s)	Normal GARCH.
TGARCH (p,q,r,s)	Student's t distribution GARCH.
GTGARCH (p,q,r,s)	Skew Generalized t distribution GARCH.
IGARCH (p,q,r,s)	Normal integrated GARCH.
ITGARCH (p,q,r,s)	t distribution integrated GARCH.
IGTGARCH	Skew Generalized t distribution

(p,q,r,s)	integrated GARCH.
FIGARCH (p,q,r,s)	Normal fractionally integrated GARCH.
FITGARCH (p,q,r,s)	t distribution fractionally integrated GARCH.
FIGTGARCH (p,q,r,s)	skew generalized t distribution fractionally integrated GARCH.
DVGARCH (p,q,r,s)	Normal diagonal vec multivariate GARCH.
DVTGARCH (p,q,r,s)	t distribution diagonal vec multivariate GARCH.
CDVGARCH (p,q,r,s)	Normal constant correlation diagonal vec multivariate GARCH.
CDVTGARCH (p,q,r,s)	t distribution constant correlation diagonal vec multivariate GARCH.
BKGARCH (p,q,r,s)	Normal BEKK multivariate GARCH..
BKTGARCH (p,q,r,s)	t distribution BEKK multivariate GARCH.
VARMA(r,s)	Normal VARMA.
TVARMA(r,s)	t distribution VARMA

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*times$1, character vector of labels of independent variables

Remarks

estimategenerates 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 **sqpSolveMT** optimization program. See the **sqpSolveMT** procedure in the **GAUSS Run-Time Library** documentation for details concerning the optimization.

Example

```
library fanpacmt,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

fankeymt.src

forecast

Purpose

\Purpose Generates forecasts of a time series model.

Library

fanpacmt

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	$L \times k$ matrix, L forecasts for K models.
fan CVforecast	$L \times k$ 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 **fanpacmt** 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

fankeymt.src

getCOR

Purpose

Computes conditional correlations and puts them into a global variable.

Library

fanpacmt

Format

getCOR [*list*];

Input

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

Global Output

_fan_CV	$N \times k$ 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

fankeymt.src

getCV

Purpose

Computes conditional variances and puts them into a global variable.

Library

fanpacmt

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

fankeymt.src

getEstimates

Purpose

Stores estimates in global variable.

Library

fanpacmt

Format

getEstimates [*list*];

Input

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

Global Output

fan Estimates	$K \times L$ matrix, global into which estimates are stored.

Source

fankeymt.src

getRD

Purpose

Computes unstandardized residuals and puts them into a global variable.

Library

fanpacmt

Format

getRD [*list*];

Input

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

Global Output

fan Residuals	$N \times k$ matrix, standardized residuals.

Source

fankeymt.src

getSeriesACF

Purpose

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

Library

fanpacmt

Format

getSeriesACF [*list*] *num diff*;

Input

list	list of runs. If omitted, standardized residuals will be produced for all runs.
num	scalar, maximum number of autocorrelations to compute.
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

fankeymt.src

getSeriesACF

Purpose

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

Library

fanpacmt

Format

getSeriesACF [*list*] *num diff*;

Input

list	list of runs. If omitted, standardized residuals will be produced for all runs.
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* 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 \substitute{diff} to the smaller number.

See Also

plotSeriesACF, plotSeriesPACF, getSeriesACF

Source

fankeymt.src

getSession

Purpose

Resets globals to default values.

Library

fanpacmt

Format

```
getSession;
```

Source

fankeymt.src

getSR

Purpose

Computes standardized residuals and puts them into a global variable.

Library

fanpacmt

Format

getSR [*list*];

Input

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

Global Output

_fan_Residuals	$N \times k$ matrix, standardized residuals.

See Also

plotSR

Source

fankeymt.src

plotCOR

Purpose

Plots conditional correlations.

Library

fanpacmt, 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	$N \times k$ matrix, conditional correlations.

Remarks

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

Source

fanplotmt.src

plotCSD

Purpose

Plots conditional standard deviations.

Library

fanpacmt, pgraph

Format

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

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.
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	$L \times K$ matrix, forecasts of conditional variances.

Global Output

_fan_CV	$N \times K$ 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

fanplotmt.src

plotCV

Purpose

Plots conditional standard deviations.

Library

fanpacmt, 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`	$N \times K$ matrix, conditional variances.
`_fan_CVforecast`	$L \times K$ 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

fanplotmt.src

plotQQ

Purpose

Plots quantile-quantile plot.

Library

fanpacmt, pgraph

Format

plotCV [*list*];

Input

list	list of runs. If no list, conditional variances will be plotted for all runs.

Global Output

_fan_SR	*N*×*K* matrix, standardized residuals.

Source

fanplotmt.src

plotSeries

Purpose

Plots time series.

Library

fanpacmt, pgraph

Format

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

Input

list	list of 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	$N \times 1$ vector, time series.
_fan_TSforecast	$L \times 1$ vector, forecasts.

Remarks

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

Source

fanplotmt.src

plotSeriesACF

Purpose

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

Library

fanpacmt, 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*×*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, getSeriesPACF

Source

fanplotmt.src

plotSeriesPACF

Purpose

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

Library

fanpacmt, 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, getSeriesACF, getSeriesPACF

Source

fanplotmt.src

plotSR

Purpose

Plots standardized residuals.

Library

fanpacmt

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	$N \times k$ matrix, standardized residuals.

Remarks

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

Source

fanplotmt.src

session

Purpose

Resets globals to default values.

Library

fanpacmt

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

Source

fankeymt.src

setAlpha

Purpose

Sets confidence level for statistical inference.

Library

fanpacmt

Format

```
setAlpha alpha;
```

Input

alpha	scalar, confidence level. Default = .05.

Source

fankeymt.src

setConstraintType

Purpose

Computes conditional correlations and puts them into a global variable.

Library

fanpacmt

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.	
	standard	standard constraints.
	bounds	bounds constraints on parameters.
	unconstrained	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

fankeymt.src

setCovParType

Purpose

Sets type of covariance matrix of parameters.

Library

fanpacmt

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

fankeymt.src

setCVIndEqs

Purpose

Declares independent variables for inclusion into conditional variance equation.

Library

fanpacmt

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	$L \times K$ 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 variablesby 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

fankeymt.src

setDataset

Purpose

Sets dataset name for analysis.

Library

fanpacmt

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 $K \times 1$ 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, **fanpacmt** assumes that *name* is a file containing the data.

If one of the columns in the **GAUSS** data set is labeled DATE, **fanpacmt** 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_dataset* is DATE, **fanpacmt** 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 **fanpacmt** 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, **fanpacmt** 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 fanpacmt,pgraph;
session nissan 'Analysis of Nissan daily log-returns';
setVarNames date nsany;
setDataset nsany.asc;
setSeries nsany;
estimate run1 garch(1,3);
showResults;
```

Source

fankeymt.src

setIndElapsedPeriod

Purpose

Causes an independent variable measuring elapsed time between observations to be generated.

Library

fanpacmt

Format

```
setIndElapsedPeriod;
```

Global Output

_fan_IndVars $N \times KL$ matrix, independent variables.

Remarks

Adding this keyword command to the command file causes **fanpacmt** to generate an independent variable measuring the elapsed time between observations. The name of this variable is *EP*. For example,

```
library fanpacmt,pgraph;
session wilshire 'wilshire example';
setDataset wilshire;
setSeries cwret; /* capitalization weighted return
s */
setIndElapsedPeriod;
setIndEqs EP;

estimate run1 garch(2,2);

showResults;
```

Source

fankeymt.src

setIndEqs

Purpose

Sets independent variable list for particular run.

Library

fanpacmt

Format

setIndEqs *name list*;

Input

name	name of time series for this set of independent variables, may be omitted for univariate runs.
list	list of names of independent variables.

Global Output

_fan_IndEquations	$L \times K$ matrix, indicator matrix for coefficients to be estimated..

Remarks

For multivariate models, call *setIndEqs* for each time series, listing the

independent variablesby name in each call:

```
setIndEqs nmsft  lnVol1  SandP
setIndEqs nmsft  lnVol1  SandP
```

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

Source

fankeymt.src

setIndLogElapsedPeriod

Purpose

Causes an independent variable measuring log elapsed time between observations to be generated.

Library

fanpacmt

Format

setIndLogElapsedPeriod;

Global Output

_fan_IndVars $N \times KL$ matrix, independent variables.

Remarks

Adding this keyword command to the command file causes **fanpacmt** to generate an independent variable measuring the elapsed time between observations. The name of this variable is *lnEP*. For example,

```
library fanpacmt,pgraph;
session wilshire 'wilshire example';
setDataset wilshire;
setSeries cwret; /* capitalization weighted returns */
setIndElapsedPeriod;
setIndEqs lnEP;

estimate run1 garch(1,1);

showResults;
```

Source

fankeymt.src

setIndVars

Purpose

Declares exogenous or independent variables.

Library

fanpacmt

Format

```
setIndVars list;
```

Input

list	list of names of independent variables for current session.

Global Output

_fan_IndvarNames	$L \times K$ character vector, names of independent variables for each equation.

Remarks

A variable in the list can be log-transformed by pre-pending an asterisk to the name in the list. For example

```
setIndVars *volume;
```

causes **fanpacmt** to generate the independent variable *ln(volume)*. The label for the new variable is the old label pre-pended by "ln." This to add it to the list of independent variables for a particular run add

```
setIndEqs lnvolume;
```

to the command file prior to the call to **estimate**.

Source

fankeymt.src

setInferenceType

Purpose

Sets type of statistical inference.

Library

fanpacmt

Format

setInferenceType *type*;

Input

type	if omitted, standard errors computed from covariance matrix of parameters are computed. Otherwise, set to.	
	NONE	no confidence limits computed.
	SIMLIMITS	confidence limits by Andrew's simulation method.
	WALD	confidence limits computed from covariance matrix of parameters.

Source

fankeymt.src

setInmean

Purpose

Declares inmean model.

Library

fanpacmt

Format

setInmean *depname list*;

Input

depname	name of dependent variable .
list	column of conditional covariance matrix to be added to *depname*'s inMean equation.

Global Output

_gg_inMeanEqs	$L \times L^*(L+1)/2$ matrix, an element of which is set to 1 if corresponding coefficient is to be estimated, otherwise

it is set to zero, where *L* is the number of time series variables.

Remarks

The columns of *_gg_inMeanEqs* correspond to the lower left nonredundant portions of the conditional covariance matrices. The columns associated with covariances will be ignored if a constant correlation model is being estimated. For univariate models, *_gg_inMeanEqs* is set to scalar 1.

Source

fankeymt.src

setLagInitialization

Purpose

Sets number of lags EXCLUDED in analysis for FIGARCH models.

Library

fanpacmt

Format

```
setLagInitialization num;
```

Input

num	number of lags excluded.

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 excluded.

Source

fankeymt.src

setLagTruncation

Purpose

Sets number of lags INCLUDED in analysis for FIGARCH models.

Library

fanpacmt

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

fankeymt.src

setLjungBoxOrder

Purpose

Sets order for Ljung-Box statistic.

Library

fanpacmt

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

fankeymt.src

setOutputFile

Purpose

Computes conditional correlations and puts them into a global variable.

Library

fanpacmt

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

fankeymt.src

setRange

Purpose

sets range of time series to be analyzed.

Library

fanpacmt

Format

```
setRange start end;
```

Input

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_Series	$N \times L$ matrix, time series.

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

Source

fankeymt.src

setSeries

Purpose

Declares time series to be analyzed.

Library

fanpacmt

Format

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

Input

list	list of names of time series.
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_Series	$N \times L$ matrix, time series.
_fan_SeriesNames	$L \times 1$ character vector, names of time series.
_fan_Date	$N \times 1$ vector, dates of observations in `yyyymmdd` format. This requires that the session dataset contain a variable in that same format with variable name "date."

Source

fankeymt.src

setVarNames

Purpose

Declares time series to be analyzed.

Library

fanpacmt

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_SeriesNames	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

fankeymt.src

showEstimates

Purpose

Displays estimates and their labels in a simple format.

Library

fanpacmt

Format

showEstimates [*list*];

Input

list	list of names of estimation runs. If no run names are provided, all runs are displayed.

Source

fankeymt.src

showResults

Purpose

Displays results of a run.

Library

fanpacmt

Format

showResults [*list*];

Input

list	list of names of estimation runs. If no run names are provided, all runs are displayed.

Example

```
library fanpacmt,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

fankeymt.src

showRuns

Purpose

Displays a list of current runs in a session..

Library

fanpacmt

Format

```
showRuns;
```

Source

fankeymt.src

simulate

Purpose

Simulates data with GARCH errors.

Library

fanpacmt

Format

simulate *starray*;

Input

starray	name of estimation run. It must come first and it cannot contain embedded blanks.	
run_title	*K*×1 string array, simulation parameters.	
	Model	model name (required).
	NumObs	number of observations.
	DatasetName	name of **GAUSS** data set into which simulated data will be

	put (required).
TimeSeriesName	variable label of time series.
Omega	ARCH 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 fanpacmt;

session test 'test session';

string ss = {

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

simulate  ss;
```

Source

fansimmt.src

testSR

Purpose

Computes skew and kurtosis statistics and a heteroskedastic-consistent Ljung-Box statistic for standardized residuals as well as time series.

Library

fanpacmt

Format

testSR *list*;

Input

list	list of runs.

Remarks

The Ljung-Box statistic is the heteroskedastic-consistent statistic described in Gouriéroux, 1997.

Source

fankeymt.src

Index