Time Series MT 2.1

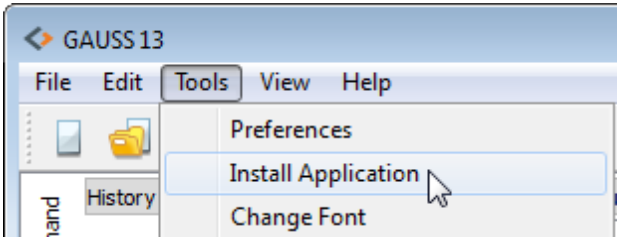
for GAUSS™ Mathematical and Statistical System

Part Number: 008624
Version 2.1
Documentation Revision: 2648
December 03, 2013

Contents

1 Installation

If you are using **GAUSS 13** or later, there is an applications installation 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 `.tar.gz` or `.zip` file to `/tmp`.

2. If the file has a `.tar.gz` extension, unzip it using `gunzip`. Otherwise skip to step 3.

   ```
   gunzip app_appname_vernum.revnum_UNIX.tar.gz
   ```

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

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

4. Use `tar` or `unzip`, depending on the file name extension, to extract the file.

   ```
   tar xvf /tmp/app_appname_vernum.revnum_UNIX.tar
   ```

   – or –

   ```
   unzip /tmp/app_appname_vernum.revnum_UNIX.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 `tar` or `unzip`, depending on the file name extensions, to extract the files found on the CD. For example:

   ```
   tar xvf /cdrom/apps/app_appname_vernum.revnum_
   UNIX.tar
   ```
   – or –
   ```
   unzip /cdrom/apps/app_appname_vernum.revnum_
   UNIX.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

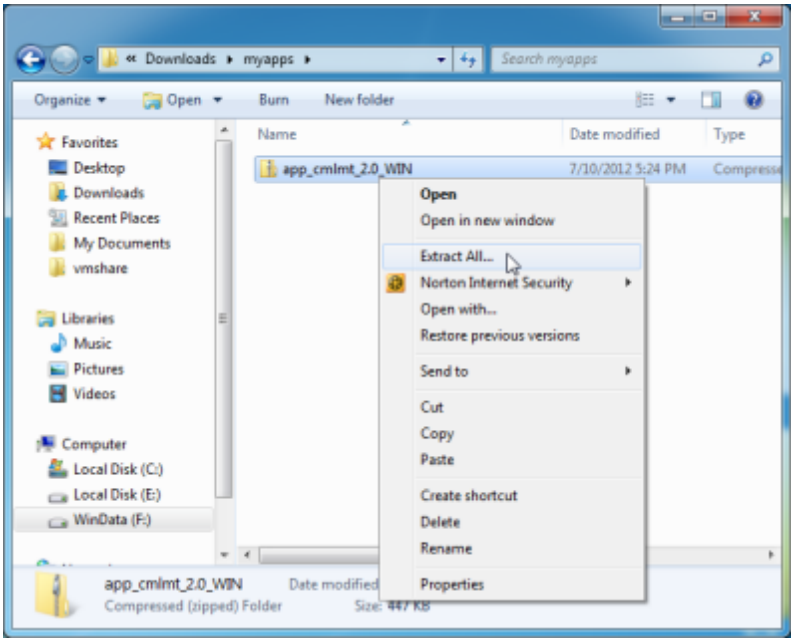
GAUSS applications are a set of files that need to be placed in the /src, /lib and /examples directory from your GAUSSHOME directory. To install an application, you simply need to extract them to your GAUSSHOME directory.

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.

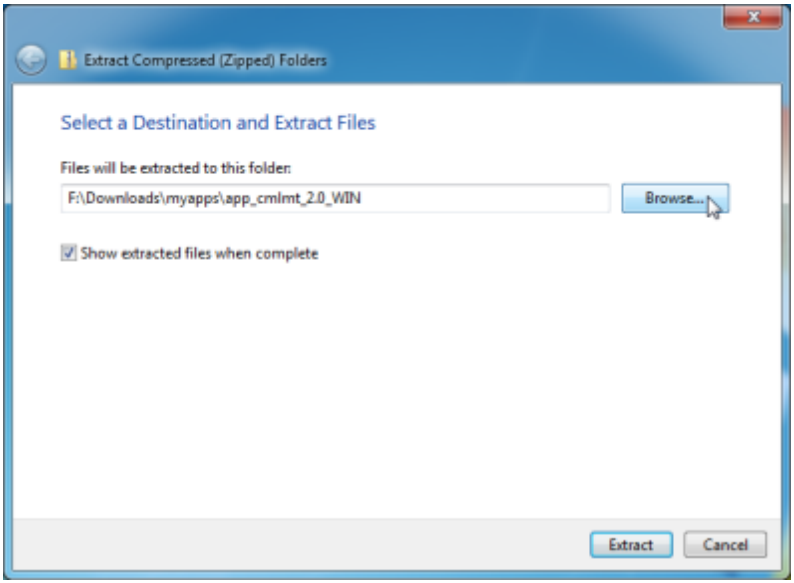
Step by step installation

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

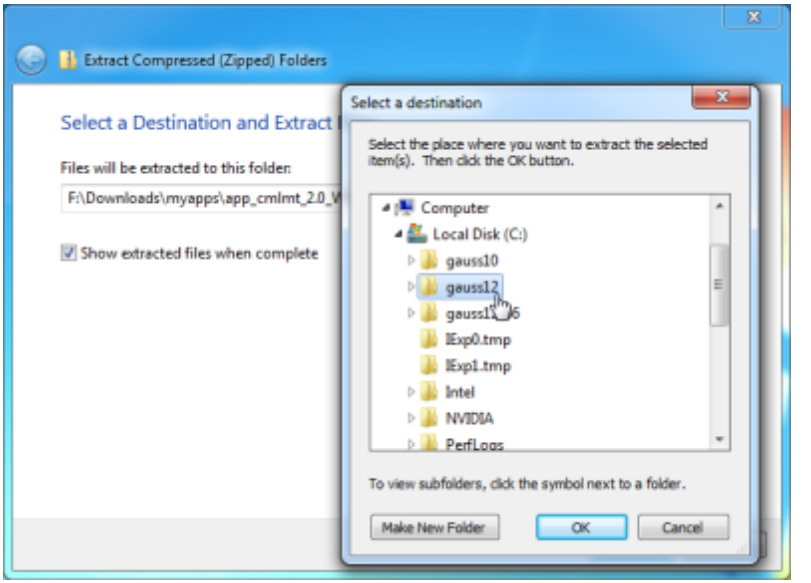
Right-click the application file and select **Extract All...** from the context menu.



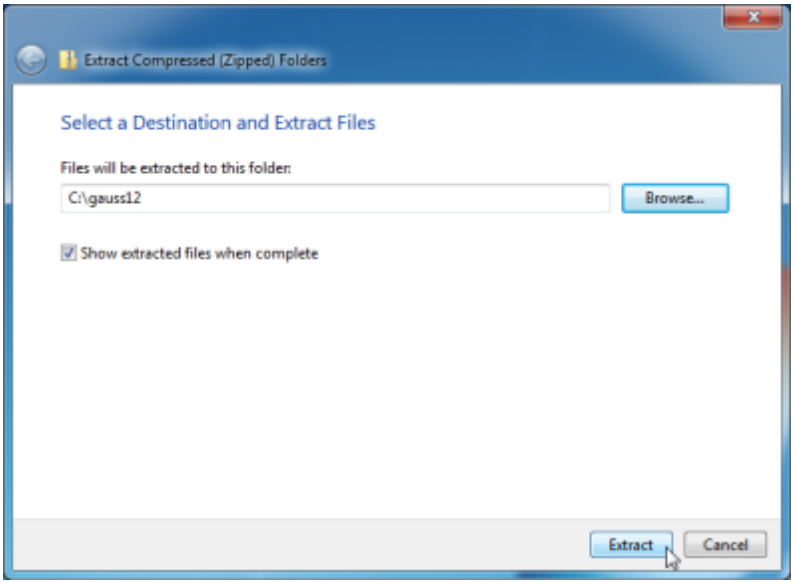
Select the **Browse** button.



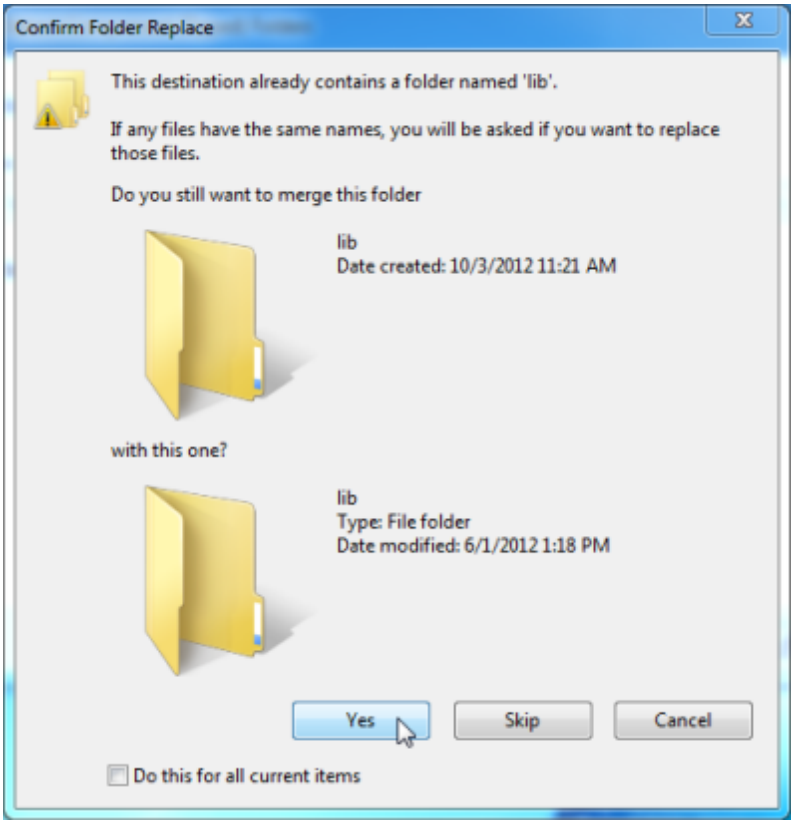
Browse to your **GAUSS** home directory.



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, then 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 13.1+ is required to use these routines.

The **Time Series MT** version number is stored in a global variable:

_tsmt_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 **Time Series MT**.

2.1 README Files

The file `README.tsmt` contains any last minute information on the **Time Series MT** procedures. Please read it before using them.

2.2 Setup

There are four essential steps to using the procedures in this module. These must be specified in any programs that call these procedures.

1. **Header:**

The header consists of up to five statements:

1. A `library` statement which activates the **TSMT** library, making its functions available

```
library tsmt;
```

2. Declarations of instances of any needed *Control* structures.

```
struct varmamtControl vmc;
```

3. Declarations of instances of any needed *Out* structures.

```
struct varmamtOut vmo;
```

4. A call to the appropriate **ControlCreate** procedure for any needed *Control* structures, which sets its members to the default values.

```
vmc = varmamtControlCreate();
```

There are four types of *Control* and *Out* structures included with **TSMT**, corresponding to four groups of procedures:

1. **arimamt** uses *arimamtControl* and *arimamtOut* structures.
2. The **autoreg** procedures use *automtControl* and *automtOut* structures.
3. **tscsmt** uses *tscsmtControl* and *tscsmtOut* structures.

4. The **varmamt** and **ecmmt** procedures use *varmamtControl* and *varmamtOut* structures.

Here is an example of a full **TSMT** header section needed to call the **arimamt** function (for GAUSS 13.1 and newer):

```
library tsmt;
struct  arimamtControl a_ctl;
struct  arimamtOut a_out;
a_ctl = arimamtControlCreate();
```

2. **Data Setup:**

Next, the user must specify the data to be passed to the procedures. For example, the format for **varmaxmt** is:

```
vmo = varmaxmt(vmc,y,x);
```

Here is an example of the data setup for that procedure call:

```
load y[62,3] = minkmt.asc;
x = y[.,1];
y = y[.,2 3];
```

3. **Specify Options:**

Options are controlled by setting the members of the appropriate *Control* structure:

```
vmc.ar = 3;          //Estimate 3 AR matrices
vmc.printIters = 1; //Print iteration
                     //information
```

4. **Calling the Procedure:**

   Each estimation procedure can print results to the screen and send output to the specified output file and/or return a global output structure to memory. If all you need is the printed results, you can `call` the procedure. Placing the **GAUSS** keyword `call` in front of a function call tells **GAUSS** to execute the function, but disgard the return values. For example:

```
call varmaxmt(vmc,y,x);
```

   In this case, you would not need to declare an instance of a *varmamtOut* structure at the top of the program. However, if you want information returned to memory, you must assign the result to a *varmamtOut* structure.

```
vmo = varmaxmt(vmc,y,x);
```

   The resulting structure, *vmo*, stores all of the return statistics in an efficient manner. See Chapter 10 for more information on the statistics returned by each procedure.

3 VARMAX, ECM, SVARMAX

The **TSMT** module contains procedures for estimating and analyzing VARMA, VARMAX, ARMA, ARMAX, SVARMA, SARMA, SVARMAX, and ECMMT models.

3.1 Introduction

The procedure **varmaxmt** estimates VARMA, VARMAX, ARMA, and ARMAX models. Linear and nonlinear equality and inequality constraints may be placed on the parameter estimates, calling the **sqpsolvemt** procedure. **varmaxmt** calls a number of subordinate procedures that enable identification, estimation, diagnostic checking, and forecasting. These are described in the sections below. The **varmaxmt** procedures return parameter estimates, residuals, and various summary statistics.

The procedure **svarmaxmt** estimates the seasonal effects models SVARMA, SVARMAX, and SARMA.

The procedure **ecmmt** estimates ECM models. It calls a number of subordinate procedures that enable the recovery and analysis of long-run and short-run parameters and cointegrating vectors. The **ecmmt** procedures return parameter estimates (including cointegration coefficients), eigenvalues and eigenvectors

(computed using full information maximum likelihood), residuals, and summary statistics.

The procedures **varmaxmt**, **svarmaxmt** and **ecmmt** all use a full information maximum likelihood (FIML, exact, unconditional) estimation procedure adapted from code developed by Jose Alberto Mauricio of the Universidad Complutense de Madrid. The code was published as Algorithm AS311 in Applied Statistics. It is also described in "Exact maximum likelihood estimation of stationary vector ARMA models," *JASA*, 90:282-291. The estimation algorithm assumes that a covariance stationary process is passed to it. Sample means are removed from all data prior to estimation. Further discussion of the estimation method and requirements is contained in Section 3.7.

The **sqpsolvemt** procedure in the **GAUSS Run-Time Library** links Mauricio's FIML (exact, unconditional) estimation to constraints. **sqpsolvemt** uses Newton's method to minimize the negative of a log-likelihood function subject to different types of constraints.

The following procedures are included in the **TSMT** library to enable estimating and analyzing VARMA and ECM models:

ecmmt	Estimates an Error Correction Model.
lsdvmt	Estimates Least Squares Dummy Variable with bias correction.
svarmaxmt	Estimates seasonal VARMAX.
switchmt	Estimates switching-regression model.
macfmt	Returns an ACF matrix for multivariate models.
nwmt	Returns the Newey-West Covariance matrix.
varmamtControlCreate	Resets the VARMA global variables.
varmaxmt	Estimates a VARMAX model.

vmadfmt	Computes the Augmented Dickey Fuller statistic, allowing for deterministic polynomial time trends of an arbitrary order.
vmcadfmt	Computes the Augmented Dickey Fuller statistic applied to the residuals of a cointegrating regression, allowing for deterministic polynomial time trends of an arbitrary order.
vmc_sjamt	Returns critical values for Johansen's Maximum Eigenvalue statistic.
vmc_sjtmt	Returns critical values for Johansen's Trace statistic.
vmdetrendmt	Returns residuals from regressing on a time trend polynomial.
vmdiffmt	Differences a time series matrix.
vmsdiffmt	Differences a seasonal time series matrix.
vmforecastmt	Forecasts VARMAXMT models.
vmppmt	Performs Phillips-Perron unit root tests.
vmptrendmt	Creates a polynomial matrix of time trends of order *p*.
vmrztcritmt	Returns τ critical values for the Augmented Dickey-Fuller statistic, derived from the residuals of a cointegrating regression. Depends on *p*, the AR order in the fitted regression, the number of observations, and the number of explanatory variables.
vmsjmt	Calculates Johansen's Trace and Maximum Eigenvalue test statistics.

vmztcritmt	Returns τ critical values for the Augmented Dickey-Fuller test statistic, depending on the number of observations and *p*, the AR order in the fitted regression.

3.1.1 The varmamtControl Structure

The table below contains a list of the numerous control variables, which are members of the *varmamtControl* structure. They give the user considerable control over the model's specification and estimation. A more complete description of their use is in the following sections.

A	matrix, linear equality constraint coefficients.
adforder	scalar, number of AR lags in the ADF test statistic.
altdepvarNames	string array, alternate names for dependent variables. ar scalar, order of autoregression.
ar	scalar, order of autoregression.
covparType	scalar, specifies ML or QML covariance matrix of parameters.
critl	scalar, the significance level for ACF indicator matrices.
ctl	*sqpsolvemtControl* structure, control parameters for **sqpsolvemt** optimization.
diff	scalar, order of differencing to achieve stationarity.
IndEquations	matrix, set zero restrictions on *x* coefficients.
lags	scalar, lags over which the ACF and Diagnostics are defined.
ma	scalar, order of moving average.

maxvec	scalar, maximum number of elements allowed in any one matrix.
NoDet	scalar, controls the constant term in the Johansen tests.
nwtrunc	scalar, the Newey-West truncation lag.
olsqtol	scalar, tolerance used by **olsqrmt**.
output	scalar, determines the output to be printed.
PrintIters	scalar flag, to print each iteration's information.
rho	scalar, order of cointegration.
row	scalar, number of rows to read at a time from dataset.
s	scalar, seasonal parameter.
sar	scalar, order of seasonal autoregression.
sdiff	scalar, order of seasonal differencing.
sma	scalar, order of seasonal moving average.
scale	vector, used to scale time series.
SetConstraints	scalar flag, impose stationarity and invertibility.
Start	an instance of a *PV* structure containing start values.
title	string, title to be printed at top of header.

3.1.2 Printing Output

Two members of the *varmamtControl* structure, *output* and *PrintIters*, determine the output that is displayed from **ecmmt**, **varmaxmt**, **sqpsolvemt**, and other subordinate procedures.

1. Set *output* = 0 to suppress all printing from the **varmaxmt** and **ecmmt** procedures.

2. Set *output* > 0 to print results from the **varmaxmt** and **ecmmt** procedures.

3. Set *PrintIters* = 0 (*output* is not equal to zero) to suppress printing while starting values are calculated for each dependent variable during the **sqpsolvemt** operation.

4. Set *PrintIters* >0 (*output* is not equal to zero) to print **sqpsolvemt** iteration information. This information includes the value of the objective function and the gradient at each estimated coefficient. It is useful in finding where and why convergence might fail.

3.2 VARMAX Models

A stationary and centered (means-removed) VARMAX model may be written as:

$$Y_t - \sum_{j=1}^{p} \Phi_j Y_{t-j} + \beta X_t = \varepsilon_t - \sum_{i=1}^{q} \Theta_j \varepsilon_{t-i}$$

for $t = 1 \cdots T$ where Y_t has dimension $L \times 1$, ϵ_t is a zero mean covariance stationary process that is normally distributed with positive definite covariance matrix $\sum$, and X_t is a $K \times 1$ vector of fixed explanatory variables. The Φ and Θ matrices have dimension $L \times L$. The β coefficients have dimension $L \times K$.

Another way to write the same system is using the backshift operator, B:

$$\Phi_p(B) Y_t + \beta X_t = \Theta(B) \varepsilon_t \qquad (1)$$

where $\Phi_p(B) = \Phi_0 - \Phi_1 B - \ldots - \Phi_p B^p$ and $\Theta_q(B) = \Theta_0 - \Theta_1 B - \ldots - \Theta_q B^q$ are matrix polynomials and Φ_0 and Θ_0 are nonsingular matrices of dimension $L \times L$ (often assumed to be the identity matrices).

3.2.1 Stationarity and Invertibility

The VARMAX process is stationary if the roots of $\det(\Phi_p(B))$ are greater than one in modulus. The VARMAX process is invertible if the roots of $\det(\Theta(B))$ are greater than one in modulus. The **vmroots** procedure finds the AR and MA characteristic roots and their moduli. The roots and their moduli are printed if the output member of the `varmamtControl` structure is nonzero.

3.3 Seasonal SVARMAX Models

The SVARMAX model includes parameters for describing seasonal effects in a VARMAX model. The SARIMA model is a special case that can also be estimated with **svarmaxmt**.

In widely used terminology, the SVARMAX model can be described as SVARMAX(p,d,q,P,D,Q,s) where

`p`	autoregression order
`d`	differencing parameter
`q`	moving average order
`P`	seasonal autoregressive order
`D`	seasonal differencing parameter
`Q`	seasonal moving average order
`s`	seasonal order

The model represented by

$$(1-L)^d(1-L^s)^D\phi(L)\Phi(L^s)Y_t = \theta(L)\Theta(L^s)\epsilon_t$$

where

$$\begin{aligned}
\phi(L) &= 1-\phi_1 L-\phi_2 L^2 \ldots \phi_p L^p \\
\Phi(L^s) &= 1-\Phi_1 L^s-\Phi_2 L^{2s} \ldots \Phi_p L^{Ps} \\
\theta(L) &= 1+\theta_1 L+\theta_2 L^2 \ldots \theta_q L^q \\
\Theta(L) &= 1+\Theta_1 L^s+\Theta_2 L^{2s} \ldots \Theta_Q L^{Qs}
\end{aligned}$$

Errors are assumed to be distributed N(0,Q). The estimation procedure assumes that all series are stationary. Setting `vmc.SetConstraints` a nonzero value enforces stationarity, by constraining the roots of the characteristic equations

$$1-\phi_1 z-\phi_2 z^2 \ldots \phi_p z^p$$

and

$$1-\Phi_1 z^s-\Phi_2 z^{2s} \ldots \phi_P z^{Ps}$$

are constrained to be outside the unit circle.

Example

```
library tsmt;
#include tsmt.sdf

load y[62,3] = minkmt.asc;

struct  DS d0;
d0 = dsCreate();
d0.dataMatrix = y[.,2 3];

struct  varmamtControl vmc;
vmc = varmamtControlCreate();
vmc.ar = 1;
vmc.sar = 1;
vmc.s = 2;

vmc.output = 1;

struct  varmamtOut vout;
vout = svarmaxmt(vmc,d0);

phi = pvUnpack(vout.par,"phi");
sphi = pvUnpack(vout.par,"sphi");
vc = pvUnpack(vout.par,"vc");

print;
print;
print;
print;
format /ld 12,8;
print "PHI";
```

```
print phi;
print;
print "Seasonal PHI";
print sphi;
print;
print "residual covariance matrix";
print vc;
```

with results:

```
PHI

Plane [1,.,.]

 0.67647089  0.45828681
-0.51253821  0.80645045

Seasonal PHI

Plane [1,.,.]

 0.34567984 -0.66783469
-0.13131616  0.05201072

residual covariance matrix

 0.05126324  0.02088442
 0.02088442  0.07058108
```

3.4 Error Correction Models

Error Correction Models are often used to estimate long-run and short-run relationships and to test for cointegration.

A stationary (means-removed) VAR(p) model is written as:

$$\phi(L)Y_t = Y_t - \sum_{j=1}^{p} \phi_j Y_{t-j} + \beta X_t = \epsilon_t \qquad (2)$$

where Y_t is an L dimensioned covariance stationary time series process, the ε_t are i.i.d. $N(0, \Omega_n)$, Ω_n is a positive definite matrix of order L, and X_t is a $K\times$ 1 vector of fixed explanatory variables, has the error correction form:

$$\Delta Y_t = \Pi Y_{t-1} + \sum_{i=1}^{k} \Gamma_i \Delta Y_{t-i} + \beta X_t + \varepsilon_t, t = 1, \ldots, T \qquad (3)$$

where the $\prod$ and Γ matrices have dimension $L\times L$. The β coefficients have dimension $L\times K$.

ecmmt estimates this model using FIML (exact, unconditional - Mauricio's procedure, discussed in Section 3.7). They both return a *varmamtOut*structure, which includes the following members:

aa	$L\times r$ matrix of coefficients, such that *(aa)*(bb)* = Π.
bb	$r\times L$ matrix, eigenvectors spanning the cointegrating space of dimension r.
va	$r\times 1$ vector, eigenvalues.
par	An instance of a *PV* structure containing parameter estimates, which can be retrieved with **pvUnpack**. *par* includes:

`Pi`	$L \times L$ matrix of cointegration coefficients.

Note that `pi` *is a reserved word in* ***GAUSS****. Users will need to assign this to a different variable name.*

3.4.1 Cointegration Tests

Given the above ECMMT model, the degree of cointegration (the dimension of the cointegrating space) may be examined using Johansen's likelihood ratio Trace and Maximum Eigenvalue statistics, returned by the **`vmsj`** procedure. The first set is based on Johansen's estimation procedure, specifically on his method for calculating eigenvalues of the Π matrix. The second set is based on the Π eigenvalues returned from Full Information Maximum Likelihood estimation of the ECMMT model.

If Π has full rank then all the variables in Y_t are stationary. If Π has less than full rank, say *r*, then *r* of the variables are cointegrated. The Trace statistic tests the null hypothesis that the rank of Π is less than or equal to *r* versus the alternative that it is greater than *r*. The Maximum Eigenvalue statistic tests the null hypothesis that the rank of Π is equal to *r* versus the alternative that the rank of Π is $r + 1$. These statistics are given in Johansen (1995):

$$\begin{aligned} Trace &= -T \sum_{i=r+1}^{L} \ln(1 - \hat{\lambda}_i) \\ \lambda_{\max} &= -T \ln(1 - \hat{\lambda}_{r+1}) \end{aligned}$$

where $\hat{\lambda}_{r+1}, \cdots, \hat{\lambda}_L$ are the smallest $L - r$ eigenvalues of $S_{11}^{-1}S_{10}S_{00}^{-1}S_{01}$ and the S_{ij} matrices represent sums of square from two regressions, ΔY_t on $\Delta Y_{t-1}, \ldots \Delta Y_{t-p+1}$ (returning residuals R_{0t}) and Y_{t-1} on

$\Delta Y_{t-1}, \ldots, \Delta Y_{t-p+1}$ (returning residuals R_{1t}).

Asymptotic critical values for the Trace and Maximum Eigenvalue statistics, based on Johansen's method of calculating eigenvalues and given that the correlations are estimated rather than observed, are returned by **vmc_sja** and **vmc_sjt**. The former returns Maximum Eigenvalue critical values and the latter returns Trace critical values.

3.4.2 Cointegration Coefficients

Occasionally the (*aa*)*(*bb*) calculation will not match the returned Π matrix. This is because the eigenvalues close to zero are associated with eigenvectors not in the cointegrating space. (*aa* * *bb*) will always equal Π if *r* equals zero, i.e., if all eigenvectors are in the cointegrating space. If *r* equals one, (*aa*)*(*bb*) will equal Π only if the eigenvalue associated with the removed eigenvector is zero. If the eigenvalue is close to zero, (*aa*)*(*bb*) will almost equal Π. If the eigenvalue is not close to zero, (*aa*)*(*bb*) will be quite different from Π.

3.5 Unit Root and Cointegration Tests

Much applied research tests whether theoretically predicted relationships among variables are confirmed in the real world. Other research involves forecasting, whether from a naive time series model or using a structural model based on behavior. In all cases it is important to work with stationary or cointegrated variables. Spurious correlation may result if the relationships between nonstationary series are examined. In addition, forecast variances for nonstationary series increase without bound.

Model building involves first testing for unit roots. The **vmadfmt** procedure performs Dickey-Fuller (DF) and Augmented Dickey-Fuller (ADF) unit root tests. The **vmppmt** procedure performs Phillips-Perron (PP) unit root tests.

Cointegration tests follow to ward off spurious estimated relationships. The **vmadfmt** procedure performs ADF cointegration tests. The **vmsjmt** procedure performs Johansen's Trace and Maximum Eigenvalue cointegration tests.

The **COINT** module, written by Sam Ouliaris and Peter C.B. Phillips and sold by Aptech Systems, Inc., contains numerous other unit root and cointegration tests, including Park-Choi (1988) G(p,q) and J(p,q) tests, the Stock and Watson (1988) common trends test, and the Phillips-Ouliaris (1990) P(u) and P(z) tests. The **COINT** module also contains numerous (time and frequency domain) methods for estimating the cointegrating vector, ARMA models, various model selection criteria, spectral density estimation, and long-run variance estimation.

3.5.1 Univariate Root Tests

The **unitroots** procedure calls a variety of unit root and cointegration tests and prints the results. The univariate unit root test statistics calculated are the Dickey-Fuller, Augmented Dickey-Fuller (both called with the **vmadfmt** procedure) and Phillips-Perron (called with **vmppmt**) statistics.

DF and ADF Unit Root Tests

The **vmadfmt** procedure calculates Dickey-Fuller and Augmented Dickey-Fuller unit root test statistics, returning the statistic, its τ statistic, and a 6×1 vector of critical values.

Three specifications are typically analyzed, a random walk with drift and trend, a random walk with drift, and a random walk:

$$\Delta Y_t = \alpha + \beta t + (\rho - 1)Y_{t-1} + \sum_{i-1,2,\ldots} \rho_i \Delta Y_{t-i} + \varepsilon_t \qquad (4)$$

$$\Delta Y_t = \alpha + (\rho - 1)Y_{t-1} + \sum_{i-1,2,\ldots} \rho_i \Delta Y_{t-i} + \varepsilon_t \quad (5)$$

$$\Delta Y_t = (\rho - 1)Y_{t-1} + \sum_{i-1,2,\ldots} \rho_i \Delta Y_{t-i} + \varepsilon_t \quad (6)$$

The time polynomial input argument to **vmadfmt** determines which of the above models will be estimated.

The Dickey-Fuller test assumes independent and identically distributed errors. This assumption precludes models with lagged dependent variables, (i.e., the lagged dependent variable terms in specifications (4) to (6) are not estimated) since lags induce dependency in the errors.

The Augmented Dickey-Fuller test eliminates serial correlation in the residuals by including lagged dependent variables in the specification. The *adforder* member of the *varmamtControl* structure sets the the number of AR terms to include in the Augmented Dickey-Fuller test statistic calculations. The default is 2.

The **GAUSS dfgls** unit root test follows the methodology proposed by Elliot, Rothenberg, and Stock (1996). Given y_a such that

$$y_a = (y_1, y_2 - \alpha y_1, ..., y_T - \alpha y_{T-1})'$$

Z_a such that

$$Z_a = (1, 2 - \alpha(2 - 1), ..., t - \alpha(t - 1))'$$

and x_a such that

$$x_a = (1, 1 - \alpha, ..., 1 - \alpha)'$$

The testing parameter is given as for the case including a time trend and as for the case including only a constant.

GLS detrends the data uses the estimators from the OLS regression

$$y_a = \beta_0 x_t + \beta_1 z_t + \epsilon_t$$

such that

$$y^d = y_t - \left(\hat{\beta}_0 + \hat{\beta}_1 t\right)$$

After detrending, the augmented Dickery-Fuller test is performed on the detrended data using a standard ADF testing regression

$$\Delta y_t^d = \alpha + \gamma y_{t-1}^d + \sum_{j=1}^{k} \zeta_j \Delta y_{t-j}^d + \epsilon_t$$

to test the null hypothesis using the ERS critical values.

Example

For this example we will use an autoregressive time series created using the **simarmamt** data generating function (included in the **TSMT** library). The series will include both a trend and constant:

```
b = 0.95;
p = 1;
q = 0;
const = 0.9;
trend = 0;
n = 500;
```

```
k = 1;
std = 1;
seed = 10191;
yt = simarmamt(b,p,q,const,trend,n,k,std,seed);
```

The results in a single AR(1) time series vector with 500 observations, a constant equal to 0.9, and an autoregressive coefficient equal to 0.5. If a trend in the data is desired it must be added to the generated series

```
yt = 0.05*seqa(1,1,rows(yt)) + yt;
```

The **dfgls** can be directly called to test the null hypothesis of a unit root

```
{ adf_stat, crit_mat } = dfgls(yt,0,1);
print "The DFGLS stats:";
adf_stat;
print "The ERS critical values:";
crit_mat;
```

where the second input specifies that no lags are included in the ADF regression and the third input specifies that the model includes a trend.

The command produces the following output:

```
The DFGLS stat:
-3.3137976
The ERS critical values:
-3.4800000  -3.1500000  -2.8900000  -2.5700000
```

These results indicate the rejection of the null hypothesis of a unit root at the 1% level.

References

1. Elliott, G., T. J. Rothenberg, and J. H. Stock. 1996. "Efficient tests for an autoregressive unit root." *Econometrica* 64: 813–836.
2. Enders, W. (2010). **Applied Econometric Time Series, 3e**. Hoboken, NJ: John Wiley & Sons, Inc.

Phillips-Perron Unit Root Tests

Phillips (1987) and Phillips and Perron (1988) test for unit roots by adjusting the OLS estimate of an AR(1) coefficient for serial correlation in the OLS residuals. Three specifications are considered, an AR(1) model without a drift, an AR(1) with a drift, and an AR(1) model with a drift and linear trend:

$$Y_t = \rho Y_{t-1} + \varepsilon_t \quad (7)$$

$$Y_t = \alpha + \rho Y_{t-1} + \varepsilon_t \quad (8)$$

$$Y_t = \alpha + \delta t + \rho Y_{t-1} + \varepsilon_t \quad (9)$$

The unit root null hypothesis is $H_0: (\rho - 1) = 0$.

Hamilton (1994, pp. 506-511) tests this hypothesis using two statistics that are analogs of the Phillips and Perron (1988) Z_α and Z_t statistics. Hamilton's statistics are based on OLS estimation of (7) to (9). They allow an identical formula for each statistic to be used for all three cases.

The **vmppmt** procedure returns the Z_t statistic as calculated by Hamilton and critical values. Suppose any one of the above equations is estimated by OLS, returning $\widehat{\rho}_T$ and $\hat{\sigma}_{\widehat{\rho}_T}$ (the OLS estimates of ρ and the standard error of $\widehat{\rho}_T$

respectively), $t_T = \left(\rho - 1\right) / \hat{\sigma}_{\hat{\rho}_T}$ (the usual OLS *t* statistic for testing H_0), $\hat{\varepsilon}_t$ (the OLS residuals), and s_T (the estimated standard error of the regression).

Hamilton's Z_t statistic is:

$$Z_t = \left(\hat{\gamma}_0 / \hat{\lambda}^2\right)\frac{1}{2}t_T - \left\{\frac{1}{2}\left(\hat{\lambda}^2 - \hat{\gamma}_0\right) / \hat{\gamma}\right\}\left\{T\left(\hat{\sigma}_{\rho_T} / s_T\right)\right\}$$

$\hat{\lambda}^2$ is an estimate of the asymptotic variance of the sample mean of ε_t. In the **vmppmt** procedure $\hat{\lambda}^2$ is estimated using the Newey-West (1987) estimator,

$$\hat{\lambda}^2 = \hat{\gamma}_0 + 2\sum_{j=1}^{q}\left[1 - j / \left(q + 1\right)\right]\hat{\gamma}_j$$

where $\hat{\gamma}_j = T^{-1}\sum_{t=j+1}^{T}\hat{\varepsilon}_t\hat{\varepsilon}_{t-j}$ are the sample autocovariances of ε_t.

The **nwtrunc** member of the *varmamtControl* structure sets the number of autocorrelations to use in calculating the Newey-West correction (*q* in the above equation). The default setting, 0, causes **GAUSS** to use a trunctation lag given by Newey and West, $q = 4(T / 100)^{2/9}$.

Under the null hypothesis, the Z_t statistics have the same asymptotic distribution as Dickey-Fuller statistics.

KPSS

The KPSS test uses a Lagrange-Multiplier type test to the null hypothesis of stationarity.

The test statistic γ is given by

$$\gamma = \frac{\left(T^{-2}\Sigma_t^T \hat{S}_t^2\right)}{\hat{\sigma}^2}$$

The KPSS test requires OLS detrending or demeaning data (if necessary) and is formulated using the residuals from the OLS regression

$$y_t = \boldsymbol{D}_t'\delta + u_t$$

where y_t is the test variable and is a matrix of deterministic components, including the constant and trend. Using the residuals from the regression the partial sum series and its squares are constructed

$$\sum_{t=1}^{T} \hat{S}_t^2$$

where

$$\hat{S}_t = \sum_{j=1}^{T} \hat{u}_j$$

Calculation of the KPSS test statistic requires a consistent estimator of the long-run variance, $\hat{\sigma}^2$. This is achieved using the nonparametric heteroskedasticity and autocorrelation consistent estimation approach discussed in Hobijn,et al. (1998). For full details of the estimation methodology refer to the help section for the **`getlrv`** function.

The **`kpss`** function requires several inputs allowing the user to control estimation of the long-run variance. First, providing a non-negative, non-zero

integer for *max_lags* allows the user to directly specify the maximum lag autocovariance used during estimation. If the *max_lags* input is zero, the maximum number of lags is determined using the Schwert criterion.

In addition, the user must specify whether to use the Newey-West automatic, data dependent procedure to estimate bandwidth. The alternative option is to choose the bandwidth as a deterministic function of the sample size *T*. Users must also choose between the Bartlett kernel and the Quadratic Spectral Kernel.

Example

This example uses a simulated trend-stationary, AR(1) series which follows the data generation process $y_t = 1 + 0.95y_{t-1} + 0.05 + \varepsilon_t$.

This is simulated using the **simarmamt** function

```
b = 0.95;
p = 1;
q = 0;
const = 1;
trend = 0.05;
n = 500;
k = 1;
std = 1;
seed = 10191;
yt = simarmamt(b,p,q,const,trend,n,k,std,seed);
```

The **kpss** function is used to test null hypothesis of stationarity. The **kpss** function requires a number of inputs which allow users to control the estimation of the long run variance used in calculating the long-run variance.

```
max_lags = 0;
trend = 1;
```

```
qsk = 1;
auto = 1;
print_out = 1;
{ mat, crit } = kpss(yt,max_lags, trend, qsk, auto,
print_out);
print "The t-stats for all possible lags:";
mat;
print "Critical values:";
crit;
```


This call of the **kpss** function specifies that the KPSS test statistic will use the Quadratic Spectral Kernel and will automatically calculate the data-specific bandwidth used in estimating the long-run variance. The code above produces the following results:

```
Trend stationary
Maxlag       6.0000000 chosen by Schwert criterion
Automatic bandwidth selection (max_lags)
7.0000000
```

The t-stats for all possible lags:

```
0.44832631
0.44798139
0.44326471
0.42492583
0.38668238
0.33610897
0.28911678
0.25789383
Critical values:
```

```
0.21600000        0.17600000        0.14600000
0.11900000
```

Zivot and Andrews

The Zivot and Andrews (1992) unit root tests the null hypothesis of a unit root against the alternative of trend stationarity with a break in the intercept, trend, or both intercept and trend. The proposed test statistic is the minimum t-test across all possible breakpoint. Hence, the Zivot and Andrews (1992) test modifies the standard ADF unit root test with dummy variables to implement breaks in the intercept and trend.

Under the null hypothesis the data generating process is

$$y_t = \propto + y_{t-1} + u_t$$

and y_t is I(1) with no structural break. The testing regression depends on whether the user specifies a break in the trend, the intercept or both.

Test A of the alternative hypothesis that y_t is stationary with a break in the intercept requires OLS estimation of

$$\Delta y_t = \propto + \theta_1 D_{1t}(\gamma) + \beta_t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta \Delta y_{t-i} + u_t$$

and

$$D_{1t}(\lambda) = \begin{cases} 1, & t > n\lambda \\ 0, & otherwise \end{cases}$$

with p specifying the lags used for the ADF unit root testing and λ specifying the break fraction (n^{break}/n).

Test B of the alternative hypothesis that y_t is stationary with a break in the trend requires OLS estimation of

$$\Delta y_t = \propto + \theta_2 D_{2t}(\gamma) + \beta_t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta \Delta y_{t-i} + u_t$$

and

$$D_{2t}(\lambda) = \begin{cases} t - n\lambda, & t > n\lambda \\ 0, & otherwise \end{cases}$$

with p specifying the lags used for the ADF unit root testing and λ specifying the break fraction (n^{break}/n).

Test C of the alternative hypothesis that y_t is stationary with a break in the intercept and trend requires OLS estimation of

$$\Delta y_t = \propto + \theta_1 D_{1t}(\gamma) + \beta_t + \theta_2 D_{2t}(\gamma) z + \gamma y_{t-1} + \sum_{i=1}^{p} \delta \Delta y_{t-i} + u_t$$

In all cases, the testing procedure performs OLS regressions for all possible break point and computes the t-test statistic that γ=0. The minimum of all t-tests is used for testing the null hypothesis using the critical values specified in table below.

Zivot-Andrews test statistic critical values

	Significance Level	
Test	5%	1%
A	-4.80	-5.34
B	-4.42	-4.93
C	-5.08	-5.57

Example

First consider the AR(1) time series, yt, generated using the **simarmamt** data generating function (included in the **TSMT** library):

```
b = 0.5;
p = 1;
q = 0;
const = 0.9;
trend = 0;
n = 500;
k = 8;
std = 1;
seed = 10191;
yt = simarmamt(b,p,q,const,trend,n,k,std,seed);
```


This reproduces the following data series, pulling error terms randomly from the standard normal distribution:

$$y_t = 0.9 + 0.5y_{t-1} + u_t$$
$$u_t \sim NIID(0, 1)$$

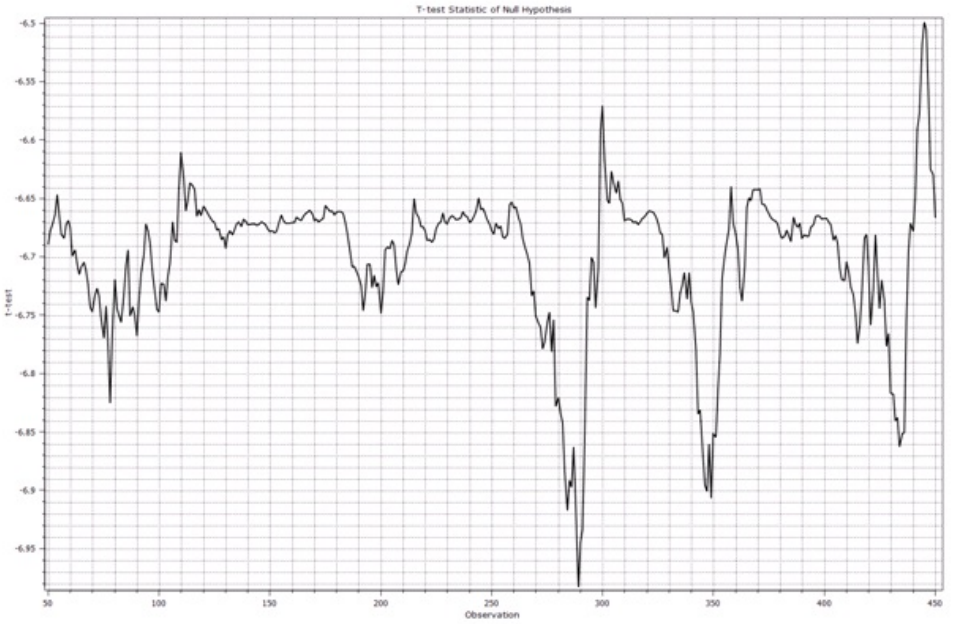
Suppose we wish to test the null hypothesis of unit root against the alternative hypothesis that yt is stationary with a break in the intercept using the Zivot-Andrews unit root test:

```
{ t_test, break_pt } = zandrews(yt,4,0.10,0,2);
```

This command implements the Zivot-Andrews using a maximum of 4 lags in the ADF test regression. The bottom and top 10% of data points are excluded from possible break points with the specification that tr = 0.10. A graph of the t-stats is produced, because the last input, *draw_graph*, is set to 1. The command produces the following output:

```
The min t-stat is:                        -8.310
The break point occurs at observation:   404.00
The five percent critical value is:       -4.800
The one percent critical value is:        -5.430
```


In addition, the specification that *which_output*is equal to 2, results in the following graph:



References

1. Hobijn, B. Franses, P.H., & Ooms, M. (2004). "Generalizations of the KPSS-test for stationarity." *Statistica neerlandixa*, 58(4), 483-502.
2. Kwiatkowski, D. Phillips, C.B., Schmidt, P., & Shin, Y. (1992). "Testing the null hypothesis of stationarity against the alternative of a unit root." *Journal of Econometrics*, 54, 159-178.
3. Zivot, E. & Andrews, D. (1992). "Further evidence on the great crash, the oil-price shock, and the unit-root hypothesis." *Journal of Business & Economic Statistics*, 10, 251-270.

3.5.2 Cointegration Tests

Residual Based Cointegration Tests

Cointegration tests fit into two categories, those based on single-equation estimation methods and those based on estimating systems of equations. Single equation tests involve testing for a unit root in the residuals that result from regressing one series on another. The Augmented Dickey-Fuller (ADF) test may be used for this purpose. The **`vmcadf`** procedure implements the ADF test for cointegration. The **`vmrztcrit`** procedure returns critical values for ADF cointegration tests.

System Based Cointegration Tests

Maddala and Kim (1998, p 211) note that single equation cointegration test results depend on the variable used to normalize the cointegrating relationship. In addition, the number of cointegrating relationships cannot be determined using single equation tests. These problems are avoided using tests based on systems of equations. System based cointegration tests examine the dimension of the cointegrating space across two or more variables.

The **vmsjmt** procedure implements Johansen's (1988) Trace and Maximum Eigenvalue system-based cointegration tests, using an ECMMT model. The null hypothesis under the Trace test is that the cointegrating space has dimension less than or equal to *r*. The alternative hypothesis is that there are more than *r* cointegrating vectors. The null hypothesis under the Maximum Eigenvalue test is that there are *r* + 1 cointegrating vectors versus the alternative that there are *r* cointegrating vectors.

The **vmsjmt** procedure returns the Trace and Maximum Eigenvalue test statistics. The **vmc_sjt** procedure returns Trace critical values at the 1%, 5%, 10%, 90%, 95%, and 99% levels. The **vmc_sja** procedure returns Maximum Eigenvalue critical values at the 1%, 5%, 10%, 90%, 95%, and 99% levels.

3.6 Identification

The first step in time series analysis is the identification of a stationary time series process. For univariate models, ACF, PACF functions and Ljung-Box statistics are returned. The ACF and PACF functions examine individual autocorrelations across different lags while the Ljung-Box statistics summarize all autocorrelations over a given number of lags. All are calculated across the number of lags specified in the `lags` member of the `varmamtControl` .structure. The default is 12.

The Ljung-Box statistics (see Ljung and Box (1978)) test:

$$H_0 : \rho_1 = \rho_2 \ldots = \rho_s = 0$$

where ρ_j is the population correlation between the ARMAX disturbances at time *t* and the ARMAX disturbances at time *t* - *j*. The statistics are defined by:

$$Q_s = T(T+2)\sum_{j=1}^{s}\left[\frac{r_j^2}{T-j}\right] \qquad (10)$$

where r_j is the sample correlation between the ARMAX residuals at time *t* and the ARMAX residuals at time *t* - *j*. Under H_0, Q_s has a chi-squared distribution with (s - the number of parameters estimated) degrees of freedom.

Six members of the *varmamtOut* structure are set: *aic*, *bic*, *lrs*, *mse*, *sse*, and *ssy*, containing the following information for each dependent variable:

aic	The Akaike Information Criterion (AIC) = 2 * (*F* + the number of estimated parameters).
bic	The Schwarz Bayesian Information Criterion (BIC) = 2 * *F* + (the number of estimated parameters)**ln* (number of observations).
lrs	The Likelihood Ratio Statistic (LRS) = 2 * *F*.
mse	Mean sum of error squares.
sse	Error sum of squares.
ssy	Sum of squares $Y(SS_{yy})$.

Identification information, summary statistics and univariate model output are printed if the *output* member of the *varmamtControl* structure is nonzero. For univariate model output, ACF and PACF values are appended with * and ** symbols to indicate significance at the 5% and 1% levels (using Bartlett's large sample approximation for the standard errors, $1 / \sqrt{T}$).

ACF and PACF Lag Report

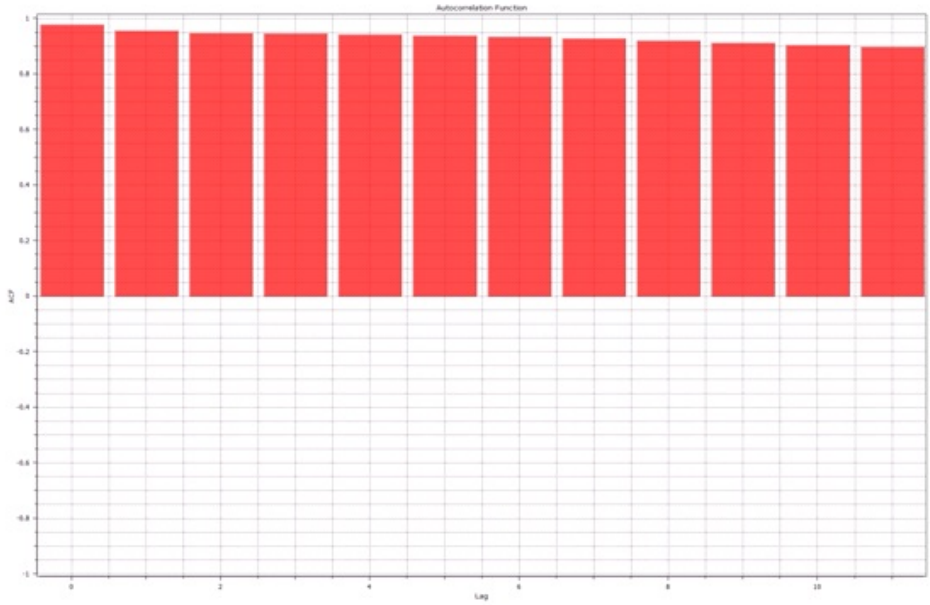
The **lagreport** function will create a full graphical report and return the acf and pacf for the specified number of lags. The following code

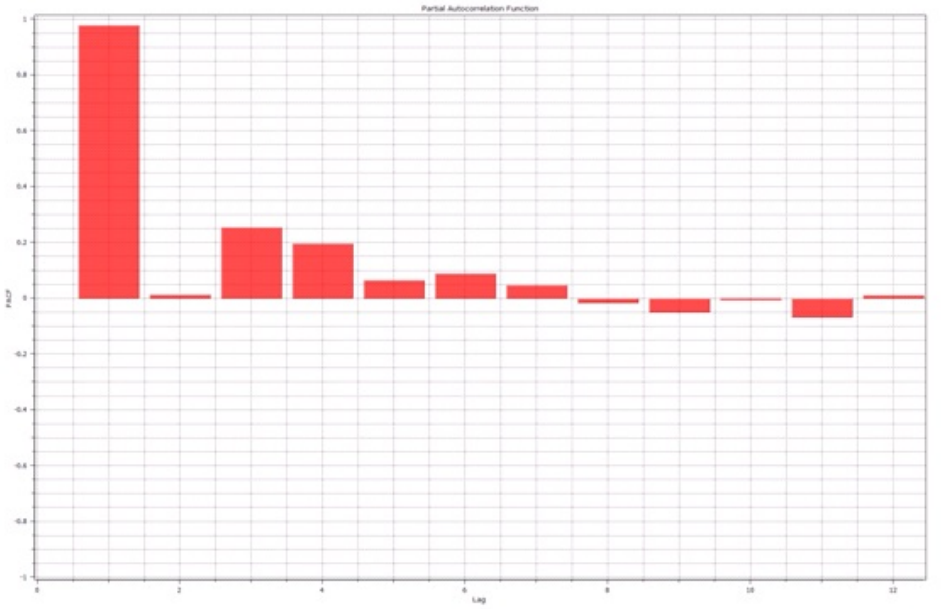
```
diff = 0;
lags=12;
{acf1, pacf1} = lagreport(yt,lags,diff);
```

Produces the following output

Lags	ACF
0.00	0.98
1.00	0.96
2.00	0.95
3.00	0.95
4.00	0.94
5.00	0.94
6.00	0.93
7.00	0.93
8.00	0.92
9.00	0.91
10.00	0.90
11.00	0.90
Lags	PACF
1.00	0.98
2.00	0.01
3.00	0.25
4.00	0.20
5.00	0.06
6.00	0.09
7.00	0.05
8.00	-0.02
9.00	-0.05
10.00	-0.01
11.00	-0.07
12.00	0.01

along with the following graphs:


Autocorrelation Function
ACF
Lag



3.6.1 Multivariate Identification

For multivariate processes, ACF matrices are returned as well as a multivariate portmanteau lack of fit statistic, Q_s. PACF values are returned for univariate processes, but are not returned for multivariate models. Sums of squares and the information vector are returned for multivariate models.

Multivariate Portmanteau Statistic

A multivariate portmanteau statistic (see Hosking (1980), Poksitt and Tremayne (1982), Li and McLeod (1981)) described in Reinsel (1993, p 133) is used to examine the residual autocorrelation matrices en toto.

Let the residuals be ε_t. Define the residual covariance matrix as:

$$C_\varepsilon(l) = \frac{1}{T}\sum_{1}^{T-1} \varepsilon_t \varepsilon_{t+l}' \quad l = 0, \ldots, s$$

$$C_\varepsilon(0) = \frac{1}{T}\sum_{1}^{T-1} \varepsilon_t \varepsilon_t'$$

The residual autocorrelation matrix is:

$$\widehat{\rho}_t(l) \equiv \widehat{V}_t^{-1/2} C\varepsilon(l)\, \widehat{V}_t^{-1/2} = \widehat{\rho}_{ij}(l)$$

$$Q_s = T^2 \sum_{l=1}^{s} (T-l)^{-1} \sum_{i=1}^{k} \sum_{j=1}^{k} C_{ij}(l) r_{ji}(-l)$$

$$= T^2 \sum_{l=1}^{s} (T-l)^{-1} tr\left\{ C_\varepsilon(l)\, \widehat{\Sigma}^{-1} C_\varepsilon(l)\, \widehat{\Sigma}^{-1} \right\}$$

$$= T^2 \sum_{l=1}^{s} (T-l)^{-1} tr\left\{ \widehat{\rho}_\varepsilon(l) \widehat{\rho}_\varepsilon(0)^{-} \widehat{\rho}_\varepsilon(-l) \widehat{\rho}_\varepsilon(0)^{-1} \right\}$$

Under the null hypothesis:

$$H_0 : Y_t \text{ is an } ARMA(p,q) \text{ process}$$
$$H_l : \textit{not } H_0$$

and assuming that s is large, the Q_s statistic has approximately a $X^2(L^2(s-p-q))$ distribution.

Wei (1990) notes that without further information a VARMAMT process may not be uniquely identified from its ACF function. Hannan (1969, 1970, 1976, and 1979) describes additional restrictions needed to identify a VARMAMT process.

If output is requested by the user, ACF and indicator matrices are printed, together with the portmanteau statistic. The indicator matrices contain + and - symbols, depending whether the individual autocorrelations are significantly positive or negative at the level specified in the `critl` member of the `varmamtControl` structure, using Bartlett's approximation, $1 / \sqrt{T}$, as the large sample standard error of each autocorrelation. The **macfmt** procedure calculates the ACF matrices.

3.7 Estimation

The **varmaxmt** and **ecmmt** procedures use a full information maximum likelihood (FIML, exact, unconditional) estimation procedure adapted from code developed by Jose Alberto Mauricio of the Universidad Complutense de Madrid. The code was published as Algorithm AS311 in Applied Statistics. It is also described in "Exact maximum likelihood estimation of stationary vector ARMA models," *JASA*, 90:282-291. Sample means are removed from all data prior to estimation and errors are assumed to be distributed $N(0, \Sigma)$.

Linear and non-linear constraints may be imposed on the coefficient estimates, invoking the **sqpsolvemt** procedure. For example, setting the `SetConstraints` member of the `varmamtControl` structure to a nonzero value enforces the stationarity required by the estimation procedure, by constraining the roots of the characteristic equation

$$I - \Phi_1 z - \Phi_2 z^2 - \ldots - \Phi_p z^p$$

to be outside the unit circle (where $\Phi_i, i = 1, \ldots, p$ are the AR coefficient matrices).

As noted earlier, the **vmroots** procedure returns roots of the AR and MA characteristic equations. The roots are printed if output is requested.

If any estimated parameters are on a constraint boundary, the Lagrangeans associated with these parameters will be nonzero. These Lagrangeans are stored in the `lagr` member of the `varmamtOut` structure. Standard errors are generally not available for parameters on constraint boundaries.

3.7.1 Quasi-Maxiumum Likelihood Covariance Matrix of Parameters

The **varmaxmt** and **ecmmt** procedures compute a QML covariance matrix of the parameters when requested. Let F be the log-likelihood function. Define $B = (\partial F_A / \partial \theta)'(\partial F_A / \partial \theta)$ evaluated at the estimates. The covariance matrix of the parameters is $\Omega^{-1} B \Omega^{-1}$ where Ω is $\left(\partial F_A^2 / \partial \theta' \partial \theta\right)$.

To request the QML covariance matrix, set the `covparType` member of the `varmamtControl` structure equal to one. The default, `covparType` = 0, is ML estimation of the covariance matrix.

3.7.2 Starting Values

The time that **sqpsolvemt** needs to reach a solution is often reduced significantly when starting values are specified. **ecmmt** and **varmaxmt** fit univariate ARMA models to generate starting values for each Y variable in the model, unless the user supplies their own starting values in the start member of the `varmamtControl` structure. Starting values must be specified by the user when the computed starting point fails or when there are inequality constraints.

The latter case requires a starting point which is feasible, i.e., one that satisfies the inequality constraints.

Starting values are entered into *start* by creating and initializing a *PV* structure. The parameter matrices are entered into the *PV* instance using **pvPacki**. The third argument in the call to **pvPacki** is which parameter matrix is being specified. This argument is as follows:

1	phi, $L \times p \times p$ array of autoregression coefficients
2	theta, $L \times q \times q$ array of moving average coefficients
3	vc, $L \times L$ residual covariance matrix
4	beta, $L \times K$ regression coefficient matrix
5	beta0, $L \times 0$ constant vector
6	zeta, $L \times p \times p$ array of ECM coefficients
7	pi, $L \times L$ matrix

All models require the constant vector (5), and if there are independent variables, a starting regression coefficient matrix (4).

The univariate ARMA model requires, in addition, either the AR coefficient array (1) or the MA coefficient array (2), or both.

The multivariate VARMA model requires either the AR coefficient array (1) or the MA coefficient array (2), or both, and, in addition, the residual covariance matrix (3).

The ECM model requires only the *phi* array (1) and the residual covariance matrix (3), in addition to the constant vector (5), and the regression coefficient matrix (4) if there are any independent variables.

For example, for a VARMA model,

```
library tsmt;
#include tsmt.sdf

load y[62,3] = minkmt.asc;

struct DS d0;
d0 = dsCreate();
d0.dataMatrix = y[.,2 3];

struct varmamtControl vmc;
vmc = varmamtControlCreate();
vmc.ar = 3;
vmc.output = 1;

phi0 = arrayalloc(3|2|2,0);
setArray phi0, 1, (.708~.469|-.699~1.151);
setArray phi0,2,(.277~-.68|.316~-.380);
setArray phi0,3,(-.504~.332|.050~.108);

vmc.start = pvCreate();
vmc.start = pvPacki(vmc.start, phi0,"phi",1);
vmc.start = pvPacksi(vmc.start, (.0499~.0188)|
    (.0188~.0565),"vc",3);

struct varmamtOut vout;
vout = varmaxmt(vmc,d0);

phi = pvUnpack(vout.par,"phi");
vc = pvUnpack(vout.par,"vc");

print;
print;
```

```
print;
print;
format /ld 12,8;
print "PHI";
print phi;

print "residual covariance matrix";
print vc;
```

And for the ECM model,


```
library tsmt;
#include tsmt.sdf

load y[362,3] = ecmmt.asc;

print "ECM Model";

struct varmamtControl vmc;
vmc = varmamtControlCreate();
vmc.ar = 1;
vmc.setConstraints = 0;

vmc.ctl.trustRadius = .01;
vmc.ctl.dirtol = 1e-6;

phi = { .05 -.05, 0 0.01, .1 -.07, .05 -.04 };

vmc.start = pvcreate;
vmc.start = pvPacki(vmc.start,areshape(phi,2|2|2),
    "phi",1);
vmc.start = pvPacksi(vmc.start,
```

```
        xpnd(15.9521|14.2525|15.9908),"vc",3);

    struct DS d0;
    d0[1].dataMatrix = y[.,2:3];

    struct varmamtOut vout;
    vout = ecmmt(vmc,d0);

    format /ld 12,4;
    print;
    print;
    print "eigenvalues";
    print vout.va;
```


3.8 sqpsolvemt and Newton's Method

ecmmt and **varmaxmt** minimize a log-likelihood function. When constraints exist (see Section 3.9 for a discussion of constraints and how to place them), **sqpsolvemt** uses Newton's method to minimize the log-likelihood function. This section provides a summary of the **sqpsolvemt** method. The reader is referred to Han (1977) for further details.

The *varmamtControl* structure contains a *sqpsolvemtControl* structure instance. The parameters of the optimization may be controlled by setting members of the *sqpsolvemtControl* structure instance appropriately. For details on the *sqpsolvemtControl* structure see documentation for **sqpsolvemt** in the **Run-Time Library** or in the header portion of the `sqpsolvemt.src` file in the `src` subdirectory. For examples of its use see Section 3.7.2 and Section 3.9.

Newton's method minimizes functions iteratively. Each iteration involves evaluating the function and determining the direction to move in the domain of

the function that results in the greatest increase in the function's value. Given the direction, the STEPBT line search method determines the **step length** that results in a lower objective function. See Dennis and Schnabel (1983) for a discussion of the STEPBT line search method.

Initial values for the unknown coefficients are required for the first iteration. These are generated automatically by the **ecmmt** and **varmaxmt** procedures if the start member of the *varmamtControl* structure is a missing value (the default).

They may also be set by the user as described in Section 3.7.2.

Let F be the log-likelihood function. **sqpsolvemt** minimizes F within the context of a standard nonlinear programming problem:

$$minF(\theta)$$

subject to the linear constraints,

$$\begin{aligned} A\theta &= B \\ C\theta &\geq D \end{aligned}$$

the nonlinear constraints,

$$\begin{aligned} G(\theta) &= 0 \\ H(\theta) &\geq 0 \end{aligned}$$

and bounds,

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

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

Hessian, Σ below) must be positive semi-definite.

Without loss of generality, we assume that the linear constraints and bounds have been incorporated into *G* and *H*. However, in practice, linear constraints are specified separately from *G* and *H* because their Jacobians are known and easy to compute. Bounds constraints are also more easily handled separately from the linear inequality constraints.

Successive parameter values are defined by:

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

where θ_t are the parameter values at time *t*, *d*, the direction, is an $NP \times 1$ vector (*NP* is the number of coefficients) and ρ is the step length, a scalar that applies equally to each element of *d*.

The direction, *d*, solves the quadratic program

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

$$subjectto \quad \dot{G}(\theta_t) d + G(\theta_t) = 0$$
$$\dot{H}(\theta_t) d + H(\theta_t) \geq 0$$

where $\sum$ is positive semi-definite. The $\Sigma(\theta)$ and $\Psi(\theta)$ matrices are given by:

$$\Sigma(\theta) = \frac{\partial^2 F}{\partial \theta \partial \theta'}$$
$$\Psi(\theta) = \frac{\partial F}{\partial \theta}$$

and the Jacobians are:

$$\dot{G}(\theta) = \frac{\partial G(\theta)}{\partial \theta}$$
$$\dot{H}(\theta) = \frac{\partial H(\theta)}{\partial \theta}$$

sqpsolvemt computes the Hessian Σ), Ψ, various gradients, and the Jacobians, $\dot{G}(\theta)$, and $\dot{H}(\theta)$ using numerical methods.

Given θ_t and *d*, the STEPBT line search method finds the **step length**, ρ, by minimizing the merit function:

$$m(\theta_t + \rho d) = F + \max|K| \sum_j |g_j(\theta_t + \rho d)| - \max|\lambda| \sum_\ell \min(0, h_\ell(\theta_t + \rho d))$$

as a scalar function of ρ, where g_j is the j-th row of *G*, h_ℓ is the ℓ-th row of *H*, K is the vector of Lagrangean coefficients of the equality constraints, and λ the Lagrangean coefficients of the inequality constraints.

STEPBT first approximates m as a quadratic function, and computes ρ to minimize the quadratic. If a feasible ρ does not exist, it attempts to fit a cubic function. If the cubic function fails and the `RandRadius` member of the `sqpsolvemtControl` structure is set to 0, **sqpsolvemt** stops iterating, without a solution.

Set `RandRadius` > 0 to have **sqpsolvemt** enter a random search in case the cubic loss function fitting fails. In a random search, **sqpsolvemt** chooses a random direction from the current point, within the radius set by `RandRadius`. If the `RandRadius` member of the `varmamtControl` structure is set to zero, a random search will not be attempted and the iterations will terminate.

A poor starting point and an excessively large direction can often put the **sqpsolvemt** iterations into ill-defined regions, from which the iterations cannot escape. To avoid this, a "trust region" can be defined to limit the direction (see Fletcher (1985)).

Setting the *TrustRadius* member of the *varmamtControl* structure imposes boundary constraints on the direction, relative to the starting position of the iterations. The direction is constrained to be no greater than *TrustRadius* in absolute value.

3.9 Setting Constraints

General constraints may be placed on parameters of VARMAMT models. There are five types of constraints: linear equality, linear inequality, nonlinear equality, nonlinear inequality, and bounds. These are not exclusive categories (i.e., there are several ways most constraints can be placed.) Below we give examples of specifying constrained parameters.

3.9.1 Constraints and the Coefficient Vector

Log-likelihood optimization is conducted by the **sqpsolvemt** function. **sqpsolvemt** "sees" all parameters in the model as a single vector. This parameter vector must be used to place constraints. The best way to look at this vector is to first run an unconstrained model and call **pvGetParNames**. The names will tell you which parameter is where in the parameter vector.

```
library tsmt;
#include tsmt.sdf

y = y[.,2 3];

struct  varmamtControl vmc;
```

```
vmc = varmamtControlCreate();
vmc.ar = 3;
vmc.output = 1;

load y[62,3] = minkmt.asc;

struct  varmamtOut vout;
vout = varmaxmt(vmc,y[.,2:3],0);

print pvGetParNames(vout.par);

     phi[1,1,1]
     phi[1,1,2]
     phi[1,2,1]
     phi[1,2,2]
     phi[2,1,1]
     phi[2,1,2]
     phi[2,2,1]
     phi[2,2,2]
     phi[3,1,1]
     phi[3,1,2]
     phi[3,2,1]
     phi[3,2,2]
        vc[1,1]
        vc[2,1]
        vc[2,2]
     beta0[1,1]
     beta0[2,1]
```


Now suppose that you want to estimate AR parameters for the first and third lags only, i.e., the second lag parameters are to be set to zero. First estimate the unconstrained model and then determine where the parameters for the second

lag from the call to **pvGetParnames**. In the above example they are in rows 5 through 8 in the parameter vector. You would then set up the constraint matrices *a* and *b* in the *varmamtControl* structure in the following way, where *vmc.ctl* is the *sqpsolvemtControl* structure instance:

```
struct varmamtControl vmc;
vmc = varmamtControlCreate();
vmc.ctl.a = { 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0,
              0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0,
              0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0,
              0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 };

vmc.ctl.b = { 0, 0, 0, 0 };
```

Some attention may have to be paid to the starting point when there are inequality constraints placed on the parameters. In general, **sqpsolvemt** requires a starting point that satisfies inequality constraints. You may need to provide starting values if the ones computed by **varmaxmt** do not satisfy the inequality constraints. It is not necessary for starting points to satisfy equality constraints. See Section 3.7.2 for a discussion about setting a starting point.

3.9.2 Linear Equality Constraints

For computational convenience linear equality constraints are treated separately from general nonlinear constraints. Let θ be the coefficient vector. Linear constraints are described as:

$$A\theta = B$$

To place a linear equality constraint, *A* and *B* are assigned to the `ctl.A` and `ctl.B` members of the `varmamtControl`structure, respectively, where `vmc.ctl` is the `sqpsolvemtControl` structure instance.

For example, suppose we wish to constrain the first AR coefficient matrix of a bivariate AR(2) model to equal zero. The coefficient vector looks like this

ϕ_{111}

ϕ_{121}

ϕ_{211}

ϕ_{221}

ϕ_{112}

ϕ_{122}

ϕ_{212}

ϕ_{222}

σ_{11}

σ_{21}

σ_{22}

Then in the command file we define the control variables:

```
vmc.ctl.A = { 1 0 0 0 0 0 0 0 0 0 0 0,
              0 1 0 0 0 0 0 0 0 0 0 0,
              0 0 1 0 0 0 0 0 0 0 0 0,
```

```
                0 0 0 1 0 0 0 0 0 0 0 0 };
vmc.ctl.B = { 0, 0, 0, 0 };
```

This constrains the first four elements of the parameter vector to zero.

3.9.3 Linear Inequality Constraints

Linear inequality constraints are described as:

$$C\theta \geq D$$

To place a linear inequality constraint, C and D are assigned to the `C` and `D` members of the `varmamtControl`structure, respectively, where `vmc.ctl` is the `sqpsolvemtControl` structure instance.

For example, suppose a bivariate AR(1) model is specified. We wish to constrain the diagonal elements of the AR coefficient matrix to be greater than the off diagonal elements. The coefficient vector looks like this:

ϕ_{11}

ϕ_{12}

ϕ_{21}

ϕ_{22}

σ_{11}

σ_{21}

σ_{22}

In the command file we define the control variables:

```
vmc.ctl.C = { 1 -1  0  0  0  0  0
              1  0 -1  0  0  0  0,
              0 -1  0  1  0  0  0,
              0  0 -1  1  0  0  0 };
vmc.ctl.D = { 0, 0, 0, 0 };
```

3.9.4 Nonlinear Equality Constraints

Nonlinear equality constraints are described as:

$$G(\theta) = 0$$

i.e., values for θ are found such that $G(\theta) = 0$.

Nonlinear constraints are placed by supplying a **GAUSS** procedure for *G*.

sqpsolvemt finds parameter estimates, $\hat{\theta}$ such that $G(\hat{\theta}) = 0$.

To place a nonlinear equality constraint, write a procedure taking the parameter vector as an input argument and returning a vector. Each element of the return vector represents a different constraint.

The following code, added to the command file, constrains the singular values of a bivariate AR(2) model coefficient matrices to be equal:

```
proc eqp(struct PV p, struct  DS d);
    local phi,s1,s2;
    phi = pvUnpack(p,"phi");
    s1 = svd(getMatrix(phi,1));
    s2 = svd(getMatrix(phi,2));
    retp(s1-s2);
endp;
```

```
vmc.ctl.EqProc = &eqp;
```

3.9.5 Nonlinear Inequality Constraints

Nonlinear inequality constraints are described as:

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

Nonlinear inequality constraints are placed by providing a **GAUSS** procedure for H. **sqpsolvemt** finds parameter estimates, $\widehat{\theta}$ such that $H(\widehat{\theta}) \geq 0$.

To place a nonlinear inequality constraint, write a procedure taking the parameter vector as an input argument and returning a vector. Each element of the return vector represents a different constraint.

For example, for a bivariate AR(2) model, the following constrains the absolute value of the eigenvalues of the first coefficient matrix to be greater than the eigenvalues of the second coefficient matrix:

```
proc ineqp(struct  PV p, struct  DS d);
    local phi,l1,l2;
    phi = pvUnpack(p,"phi");
    l1 = abs(eig(getMatrix(phi,1)));
    l2 = abs(eig(getMatrix(phi,2)));
    retp(l1-l2);
endp;

vmc.ctl.IneqProc = &ineqp;
```

3.9.6 Bounds Constraints

Bounds are a type of linear inequality constraint but are treated separately by **sqpsolvemt** for computational convenience. To place bounds on parameters, lower and upper values are entered into the *bounds* member of the *varmamtControl* structure. For example, to bound the coefficients of an AR (1) model to be between -.5 and +.5 define

```
vmc.ctl.bounds = { -.5 .5,
                   -.5 .5,
                   -.5 .5,
                   -.5 .5,
                -1e256 1e256,
                -1e256 1e256,
                -1e256 1e256 };
```

The first column of *bounds* corresponds to the lower boundaries and the second column the upper boundaries. The first four rows correspond to the AR coefficients in the parameter vector and the last three rows to the elements of the covariance matrix of the residuals which we choose not to constrain.

3.9.7 Start Values

The *varmamtControl* structure contains a member for setting start values. This member is an instance of a *PV* structure with the name *start*. The *PV* instance contains the following parameter matrices:

1	phi	$p \times L \times L$ array, autoregression coefficients
2	theta	$q \times L \times L$ array, moving average coefficients
3	vc	$L \times L$ matrix, residual covariance matrix
4	beta	$L \times K$ matrix, regression coefficients
5	beta0	$L \times 1$ vector, constants

6	sphi	$P \times L \times L$ array, seasonal autoregression coefficients
7	stheta	$Q \times L \times L$ array, seasonal moving average coefficients

Depending on the model several or all of these parameters must be specified for a starting point:

Model	**Parameters**
AR	phi
MA	theta
ARMA	phi, theta
ARMAX	phi, theta, beta
VAR	phi, vc
VARMA	phi, theta, vc
VARMAX	phi, theta, beta, vc
ECM	phi, vc
SAR	phi, sphi
SARMA	phi, theta, sphi, stheta
SVAR	phi, sphi, vc
SVARMA	phi, sphi, theta, stheta, vc
SVARMAX	phi, sphi, theta, stheta, beta, vc

The constant vector is not required for start values because the time series is centered for the estimation and the constants estimated afterwards. The covariance matrix, vc, is required only for multivariate models.

Example

The starting point for a VARMAX model might look like this:

```
struct  PV p0;
p0 = pvCreate();

a = areshape(0.2*eye(2),2|2|2);
p0 = pvPack(p0,a,"phi",1);
p0 = pvPack(p0,a,"theta",2);
p0 = pvPack(p0, 0.8*eye(2),"vc",3);
```

3.10 sqpsolvemt and Managing Optimization

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

3.10.1 Scaling

For best performance, the diagonal elements of the Hessian matrix should be roughly equal (the *Hessian* member of the *varmamtOut* structure contains the estimated Hessian). **sqpsolvemt** has difficulty converging when some diagonal elements contain numbers that are very large and/or very small with respect to the others. It may not be obvious how to scale the diagonal elements of the Hessian. However, ensuring that the data are of the same magnitude may help.

The *scale* member of the *varmamtControl* structure, used to scale the data, is either a scalar or an $L \times 1$ vector. If *scale* is a scalar, the data in all series are multiplied by the value. If *scale* is an $L \times 1$ vector, each series is

multiplied by the corresponding element of *scale*. The default scale value is 4/standard deviation of each series (found to be best by experimentation).

3.10.2 Condition

A well-conditioned problem has a Hessian for which the columns are linearly independent and the diagonal elements are roughly the same size, i.e., the data are properly scaled. In this case, the condition number of the Hessian, the ratio of the largest eigenvalue to the smallest eigenvalue, is close to unity. The condition number will be large when the data are improperly scaled or when the columns of the Hessian are linearly dependent. Users may examine the estimated Hessian. It's in the *Hessian* member of the *varmamtOut* structure.

The **sqpsolvemt** solution is found by searching for parameter values for which the gradient is zero. However, **sqpsolvemt** has difficulty determining optimal values when the Jacobian of the gradient (i.e., the Hessian) is very small for a particular parameter. In this case, a large region of the function appears virtually flat to **sqpsolvemt**. When the Hessian has very small elements, the inverse of the Hessian has very large elements and the search direction gets buried in the large numbers.

Poor condition can be caused by bad scaling, poor model specification, or bad data. Bad models and bad data are two sides of the same coin. If the problem is highly nonlinear, it is important that data are available to describe the curve over all relevant regions. For example, one of the parameters of the Weibull function describes the shape of the curve as it approaches the upper asymptote. If data are not available for that portion of the curve, the corresponding parameter is poorly estimated. The gradient of the function with respect to the parameter is very flat; elements of the Hessian associated with that parameter are very small and the inverse of the Hessian contains very large numbers. In this case, if the

underlying behavioral theory allows it, the model should be respecified to exclude the parameter.

3.10.3 Starting Point

For a starting point **ecmmt** and **varmaxmt** fit univariate ARMA models to generate starting values for each *Y* variable in the model, unless the user supplies their own starting values in the `start` member of the `varmamtControl` structure. User-defined `start` values are required when the automatically generated starting values fail or when there are inequality constraints in the model. The latter case requires a starting point that satisfies the inequality constraints. See Section 3.7.2 with regard to setting a user-defined starting point.

The starting point can be critical in finding a solution to a model that is not well-defined. Try different starting points when the optimization does not seem to work. If the underlying behavioral theory allows it, a simpler problem with the same parameters might be specified. This could lead to a closed form solution. For example, ordinary least squares estimates might be used for nonlinear least squares problems or nonlinear regressions like probit or logit. There are no general methods for computing start values and it may be necessary to attempt the estimation from a variety of starting points.

3.11 Diagnostic Checking

Identification and diagnostic checking go hand in hand. The earlier Identification section discussed the ACF, PACF, portmanteau, and information criteria.

3.12 Forecasting

The **vmforecast** procedure calculates *t* step ahead forecasts for a VARMAX model. Users must specify the coefficients involved, the dependent variable

dataset, residuals from the VARMAX estimation, the AR and MA orders, and the number of periods to forecast. A $t \times K$ matrix of fixed explanatory variables, covering only the forecast horizon, is also entered if beta coefficients were estimated.

vmforecastmt returns a $t \times (L + 1)$ matrix. The first column is the forecast horizon, i.e., the t in $T + t$. Subsequent columns contain the forecast Y values.

3.13 References

1. Ansely, Craig F. (1979). "An Algorithm for the Exact Likelihood of a Mixed Autoregressive-Moving Average Process," *Biometrika*, 66, 59-65.

2. Fletcher, R. (1985). "An ℓ_1 penalty method for nonlinear constraints," in P.T. Boggs, R.H. Byrd, and R.B. Schnabel, **Numerical Optimization 1984**, SIAM, 26-40.

3. Granger, C.W.J. and Newbold, Paul. (1986). **Forecasting Economic Time Series**. Second Edition, San Diego: Academic Press.

4. Hamilton, James D. (1994). **Time Series Analysis**, Princeton University Press.

5. Han, S.P. (1977). "A Globally Convergent Method for Nonlinear Programming," *Journal of Optimization Theory and Applications*, 22, 297-309.

6. Hannan, E.J. (1969). "The Identification of Vector Mixed Autoregressive Moving Average System," *Biometrika*, 56, 223-225.

7. Hannan, E.J. (1970). **Multiple Time Series**, John Wiley, New York.

8. Hannan, E.J. (1976). "Review of Multiple Time Series," *SIAM Reviews*,

18, 132.

9. Hannan, E.J. (1979). "The Estimation of the Order of an ARMA Process," *Ann. Statist*, 8, 1071-1081.

10. Hosking, J.R.M. (1980). "The Multivariate Portmanteau Statistic," *Journal of the American Statistical Association*, 75, 602-608.

11. Johansen, S.J. (1988). "Statistical Analysis of Cointegration Vectors," *Journal of Economic Dynamics and Control*, 12, 231-254.

12. Johansen, S.J. and Juselius, K. (1990). "Maximum Likelihood Estimation and Inference on Cointegration-with Applications to the Demand for Money," *Oxford Bulletin of Economics and Statistics*, 52, 169-2 10.

13. Li, W.K. and McLeod, A.I. (1981). "Distribution of the Residual Autocorrelations in Multivariate ARMA Time Series Models," *Journal of the Royal Statistical Society Series B*, 43, 231-239.

14. Ljung, G. and Box, G.E.P. (1978). "On a Measure of Lack of Fit in Time Series," *Biometrika*, 65, 297-303.

15. Maddala, G.S. and Kim, In-Moo. (1998). **Unit Roots, Cointegration, and Structural Change**, Cambridge University Press.

16. Newey, W.K. and West, K.D. (1987). "A Simple Positive Semi-Definite Heteroskedasticity and Autocorrelation-Consistent Covariance Matrix," *Econometrica*, 55, 703-708.

17. Osterwald-Lenum, M. (1992). "A Note with Fractiles of the Asymptotic Distribution of the Maximum Likelihood Cointegration Rank Test Statistics: Four Cases," *Oxford Bulletin of Economics and Statistics*, 54,

461-72.

18. Park, J.Y. and Choi, B. (1988). "A New Approach to Testing for a Unit Root," working paper 88-23, Department of Economics, Cornell University.

19. Phillips, P.C.B. and Ouliaris, S. (1990). "Asymptotic Properties of Residual Based Tests for Cointegration," *Econometrica* 58, 165-193.

20. Poskitt, D.S., and Tremayne, A.R. (1982). "Diagnostic Tests for Multiple Time Series Models," *Annals of Statistics*, 10, 114-120.

21. Reinsel, Gregory. (1993). **Elements of Multivariate Time Series Analysis**, Springer-Verlag.

22. Wei, William, W.S. (1990). **Time Series Analysis: Univariate and Multivariate Methods**, Addison-Wesley Publishing.

4 Nonlinear Time Series Models

There are a number of circumstances under which time series may include nonlinearities. When nonlinearities arise, traditional estimation techniques for linear time series are invalid. As such a number of techniques for testing for nonlinearities, as well as estimating models which include nonlinearities have developed. The **GAUSS TSMT** module contains a family of tools for both testing for nonlinearities. In addition, it includes procedures for estimating three fundamental nonlinear models: regime switching models, structural break models, and threshold autoregressive models.

4.1 Parameter Instability Tests

Thorough time series modeling often requires diagnostic testing for sources of nonlinearities. In the face of nonlinearities, model parameters and/or estimates change, such that the assumption of constant parameters across all time periods is invalid. The **GAUSS TMST** module provides several pre-programmed procedures for parameter stability analysis.

4.1.1 Rolling Regressions

The **GAUSS** `rolling` procedure performs rolling OLS regressions for a provided vector of dependent data and matrix of independent regressors. The `rolling` procedure estimates a rolling or expanding window OLS model. For a fixed window width, n, the OLS model for estimation is given by

$$y_t(n) = X_t(n)\beta_t(n) + \varepsilon_t(n), \qquad t = n, \ldots, T$$

The dependent data vector, $y_t(n)$, and the dependent data vector, $X_t(n)$, include the most recent observations for time $t - n - 1$ to t.

The procedure allows the implementation of a forward expanding regression window of size τ such that the regression window including observations 1 …(τ+1), 1…(τ+2), etc.

Similarly, users may utilize a backward expanding regression window such that the window includes the *last* τ, τ+1, τ+2,…, τ+(T- τ) observations.

The **rolling** procedure produces time series plots of rolling estimates for clear and quick visualization of the dynamic behavior of estimates.

Example

This example uses 400 observations of generated data with a break in the intercept after 120 observations. The error terms are standard normal. The data is replicated using the code below:


```
//This generates 400 observations of a
//linear time series with a break in the
//constant at observation 120

b1 = { 1.2, -2, 0.75 };
b2 = { 5, -2, 0.75 };

n1 = 120;
n_tot = 400;
xt = ones(n_tot,1)~rndn(n_tot,2);
```

```
et = rndn(n_tot,1);

//Create series with break
y1 = xt[1:n1,.]*b1 + et[1:n1,.];
y2 = xt[n1+1:n_tot,.]*b2 + et[n1+1:n_tot,.];
yt_break = y1|y2;
```

Three rolling techniques are available for estimating the model, pure rolling static windows, forward expanding windows, and backward expanding window. To first estimate the model using a static rolling window, set the input window to a positive integer value less than the number of observations. This sets the fixed estimation window size. When using the static window rolling regression the input *add* is irrelevant. Finally, the indicator input *gr* is set to one to produce time series graphs of the coefficient estimates:

```
//First set parameters to run a
//rolling window regression.
//Set-up expanding window size
wind = 15;

//Specify increment to increase window size
//Irrelevant for rolling window regression
add = 15;

//gr is an indicator for graphing
gr = 1;
```

Finally call the **rolling** procedure

```
{ beta, res, w } = rolling(yt_break, xt, wind, add,
gr);
```

To estimate the same model using a forward expanding window, the input parameter window is set to any negative integer. The input *add* is the used to specify the initial window size. A positive value for *add* indicates a forward expanding window and a negative value for *add* indicates a backward expanding window. The window size is always incremented by one observation for both forward and backward expanding window.

```
//Set parameters for a forward expanding
//window regression
//Set-up expanding window size
wind = -15;

//Specify increment to increase window size
//Irrelevant for rolling window regression
add = 15;

//gr is an indicator for graphing
gr = 1;

{ beta_fwd,res_fwd,w_fwd } = rolling(yt_break, xt,
wind, add, gr);
```

This produces time series estimates of the model coefficients, *beta_fwd*, one-step-ahead prediction residuals, *res_fwd*, and standardized one-step-ahead prediction residuals, *w_fwd*. In addition, it plots the rolling estimates in *beta_fwd*.

4.1.2 Chow Forecast

The Chow (1960) forecast test is a more statistically formal test for parameter stability and uses out-of-sample forecasts to test parameter constancy. It is built on the theory that if parameters are constant then out-of-sample forecasts should be unbiased.

The test considers a linear model such that

$$y_t = x_t'\beta + u_{t,}$$
$$u_t \sim \left(0, \sigma^2\right),$$
$$t = 1, \ldots, n$$

The Chow (1960) procedure for testing coefficient stability first splits the data sample into estimation and forecast subsamples using an estimation window, $n_1 > k$ such that

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

where sample one includes the first n_1 observations and sample two contains the last $T - n_1$ observations.

This OLS fitted model yields

$$\widehat{\beta}_1 = \left(X_1'X_1\right)^{-1} X_1'y_1$$
$$\widehat{u}_1 = y_1 - X_1\widehat{\beta}_1$$
$$\widehat{\sigma}_1^2 = \frac{1}{(n_1 - k)}\widehat{u}_1'\widehat{u}_1$$

Out-of-sample forecasts for the last $T - n_1$ observations are constructed using

$$\hat{y}_2 = X_2\hat{\beta}_1$$

with out-of-sample prediction errors and error variance given by

$$\hat{u}_2 = y_2 - \hat{y}_2 = y_2 - X_2\hat{\beta}_1$$

and

$$var(\hat{u}_2) = \sigma^2\left(I_{T-n_1} + X_2\left(X_1'X_1\right)^{-1}X_2'\right)$$

Given this and the assumption of Gaussian errors, *u*, the Chow F-test of the null hypothesis of coefficient stability is

$$Chow_{FCST}(n_2) = \frac{\hat{u}_2'\left(I_{n_2} + X_2(X_1'X_1)^{-1}X_2'\right)\hat{u}_2}{\hat{\sigma}_1^2} * \frac{n_1 - k}{n_2} \sim F(n_2, n_1 - k)$$

Example

This example uses the Chow forecast test to test for structural breaks in two different simulated data series, one with a break in the constant and one without any breaks. The first step is to generate the data using the **simarmamt** function. The series with a break is generated in two segments, one with a constant equal to 0.5 and a second with a constant equal to 3. Both series are AR(1) series with an autoregressive coefficient equal to 0.6:

```
library tsmt;

b = 0.6;
```

```
q = 0;
p = 1;
tr = 0;

//Set first constant
const1 = 0.5;

// Number of observations before break
N1 = 90;

//Total observation in complete series
n_tot = 300;
k = 1;
std = 0.5;
seed = 19786;

//First series with constant=0.3
y1 = simarmamt(b,p,q,const1,tr,n1,k,std,seed);

//Second series with constant=1.7
const2 = 3;
y2 = simarmamt(b,p,q,const2,tr,n_tot-n1,k,std,seed);

//Full time series with break
t_break = y1|y2;
```

The series without a break can be similarly generated using **simarmamt**

```
//Full time series without break
yt = simarmamt(b,p,q,const1,tr,n_tot,k,std,seed);
```

Next, compare and visualize the data using the **plotXY** function

```
//Plot time series data
//Declare the structure
struct plotControl myPlot;

//Initialize the structure
myplot = plotGetDefaults("xy");

//Set graph appearance
plotSetLegend(&myPlot, "off");

//Plot Series with break in first element of 2x1
layout
plotLayout(2,1,1);
plotSetTitle(&myPlot, "Simulated time series with
                       break at observation 90");
plotXY(myPlot,seqa(1,1,rows(yt_break)),yt_break);

//Plot Series without break
plotLayout(2,1,2);
plotSetTitle(&myPlot, "Simulated time series
                       without break");
plotXY(myPlot,seqa(1,1,rows(yt)),yt);

//Turn off 2x1 layout for future plots
plotClearLayout();
```


Finally, call the **chowfcst** procedure to test for a structural break. This requires first constructing the independent regressor matrix. This matrix should include a vector of ones for estimating the constant, and the lagged dependent variable:

```
//Generate xt regressors
//These should include a constant
xt_const = ones(n_tot,1);

//Lagged dependent variable
yt_lag1 = lag1(yt_break);
yt_lag2 = lag1(yt);

//Concat both into one data matrix
xt1 = xt_const~yt_lag1;
xt1 = xt_const~yt_lag2;

//Trim the first missing observation due to lag-
ging
xt1 = trimr(xt1,1,0);
xt2 = trimr(xt2,1,0);
```

In addition to the regressor matrix, the **chowfcst** function requires an input specifying the size of the prediction window. In the case, we will use a prediction window of our known break, *n1*. To test for a structural break using the **chowfcst** function:

```
//Call chowfcst using data with break
{chow_br, prob_br} = chowfcst(yt1,xt1,n1);
print "The Chow test statistic for series with
break:"; chow_br;
print "The p-value for series with break:"
prob_br;

//Call chowfcst using data without break
```

```
{chow, prob} = chowfcst(yt2,xt2,n1);
print "The Chow test statistic for series with
break:";
chow;
print "The p-value for series with break:";
prob;
```

Produces the following output:

```
The Chow test statistic for series with break:
       272.14587
The p-value for series with break:  1.3671052e-086
The Chow test statistic for series with break:
       2.9906912
The p-value for series with break:  1.4451492e-008
```

4.1.3 CUSUM Test of Coefficient Equality

In contrast to the CHOW test, the CUSUM test of coefficient equality compares subsample specific estimates to test for parameter equality across all subsamples.The Brown, Durbin, and Evans (1975) CUSUM test considers the empirical fluctuation process of the cumulative sums of standardized residuals. Under the null hypothesis of constant coefficients the residuals should have zero mean. Hence, significant deviation from zero at time, *t*, indicates possible structural change at time *t*.

The CUSUM test for instability is most appropriate for testing for parameter instability in the intercept term. It is best described as a test for instability of the variance of post-regression residuals. The model considers the linear model

$$y_t = x_t'\beta + u_t,$$
$$u_t \sim \left(0, \sigma^2\right),$$
$$t = 1, \ldots, n$$

The CUSUM test utilizes the standardized one-step ahead recursive estimation residuals. The estimation residual are given by

$$w_t = \frac{y_t - \hat{B}_{t-1}' x_y}{\sqrt{f_t}}$$

where

$$f_t = \hat{\sigma}^2 \left[1 + x_t' \left(x_t' x_t \right)^{-1} x_t \right]$$

The resulting CUSUM test statistic utilizes the cumulative sum of these residuals such that

$$CUSUM_t = \sum_{j=k+1}^{t} \frac{\hat{w}_j}{\hat{\sigma}_w}$$

where

$$\hat{\sigma}_w^2 = \frac{1}{n-k} \sum_{t=1}^{n} (w_t - \overline{w})^2$$

Example

This example uses the cusum residual test to test for structural breaks in two different simulated data series, one with a break in the constant and one without any breaks. Similar to the previous example, the first step is to generate the linear data. The series with a break is generated in two segments, one with a constant equal to 0.6 and a second with a constant equal to 0.95. The series without a break has a stable constant equal to 0.6.

```
b1 = { 0.6,0.25,0.75 };
b2 = { 0.95,0.25,0.75 };

n1 = 90;
n_tot = 300;
xt = ones(n_tot,1)~rndn(n_tot,2);
et = rndn(n_tot,1);

//Create series with break
y1 = xt[1:n1,.]*b1 + et[1:n1,.];
y2 = xt[n1+1:n_tot,.]*b2 + et[n1+1:n_tot,.];
yt_break = y1|y2;

//Create series without break
yt = xt*b1 + et;
```


The next step is to set the parameter inputs for the cusum test, which include an indicator variable for graphing the test statistics, and the minimum window length for the recursive regression:

```
//Next set the cusum parameters
```

```
minwin = 30;
gr=1;
```

Finally, call the cusum procedure to test both series for parameter instability.

```
//Next test residuals using cusum
{ cst1, cst1_2 } = cusum(yt_break,xt,gr,minwin);
{ cst2, cst2_2 } = cusum(yt,xt,gr,minwin);
```

Calling the cusum procedure for the nonlinear series produces the following graphical output of the cusum and cusum squared test statistics, respectively.

4.1.4 The Hansen-Nymblom Test

The Hansen-Nyblom (1989) parameter stability tests uses a Lagrange multiplier test to test the null hypothesis of constant parameters against the alternative of parameter instability. The **GAUSS** procedure tests for joint stability of all parameters, as well as individual parameter stability. The test should not be used for unit root processes or processes with deterministic trends. Under the null hypothesis the test statistic follows the Cramer-von Mises distribution.

The Hansen-Nyblom test for parameter instability is a locally most powerful, Lagrange multiplier test. The test considers the null hypothesis of parameter constancy against the alternative that the series follows a martingale. Though it is appropriate for testing the joint stability of all parameters as well as the stability of individual parameters, it should not be used for determining the timing of structural breaks. Consider a linear model such that

$$y_t = x_t'\beta + \varepsilon_t$$

Under the assumption of time varying parameters

$$\beta = \beta_t = \beta_{t-1} + \eta_t$$
$$\eta_t \sim \left(0, \sigma_\eta^2\right)$$

The null hypothesis

$$H_0: \quad \beta_t = \beta \; for \; all \; t$$

is equivalently given as

$$H_0: \quad \sigma_\eta^2 = 0$$

The **GAUSS** procedure first estimates the model parameters and the regression error such that

$$\widehat{\beta} = (X'X)^{-1}X'y$$
$$\hat{\varepsilon}_t = y_t - x_t'\widehat{\beta}$$

The least squares estimates follow directly from two first order conditions

$$0 = \sum_t^n x_{it}\hat{e}_t, \; i = 1, \ldots, m$$
$$0 = \sum_t^n \left(\hat{e}_t^2 - \hat{\sigma}^2\right)$$

These conditions are rewritten by defining a score variable, f_{it} , such that

$$f_{it} = \begin{Bmatrix} x_{it}\hat{e}_t & i = 1, \ldots, m \\ \hat{e}_t^2 - \hat{\sigma}^2 & i = m+1 \end{Bmatrix}$$

Using the score variable the Nyblom-Hansen test for individual parameters are given by

$$L_i = \frac{1}{nV_i} \sum_{i=1}^{n} S_{it}^2$$

where

$$S_i = \sum_{i=1}^{n} f_{it}^2$$

The joint test statistic is

$$L_c = \frac{1}{n} \sum_{i=1}^{n} S_t' V^{-1} S_t,$$

where

$$V = \sum_{i=1}^{n} f_t f_i',$$
$$f_t = \left(f_{1t}, \ldots, f_{m+1,t}\right),$$
$$S_t = \left(S_{1t}, \ldots, S_{m+1,t}\right).$$

The Nyblom-Hansen test statistic follows the Cramer-von Mises distribution.

Example

This example uses the Nyblom-Hansen LR test to test for structural breaks a series with a break in the constant. Similar to the previous example, the first step is to generate the linear data. The series with a break is generated in two

segments, one with an intercept equal to 1.2 and one with an intercept equal to 5.

```
/*****************************************/
//This generates 400 observations of an
//linear time series with a break in the con-
stant
//at observations 120

b1 = { 1.2, -2, 0.75 };
b2 = { 5, -2, 0.75 };

n1 = 120;
n_tot = 400;
xt = ones(n_tot,1)~rndn(n_tot,2);
et = rndn(n_tot,1);

//Create series with break
y1 = xt[1:n1,.]*b1 + et[1:n1,.];
y2 = xt[n1+1:n_tot,.]*b2 + et[n1+1:n_tot,.];
yt_break = y1|y2;
/*****************************************/

//Set parameters for a rolling window regression
//printOut is an indicator for graphing
printOut = 1;

{ ny, crit } = hansen(yt_break,xt,printOut);
```

With the *printOut* indicator input set to one, the following output is printed to the screen

```
Hansen test statistic for constancy of B_1  21.67736
Hansen test statistic for constancy of B_2   0.09693
Hansen test statistic for constancy of B_3   0.02684
Hansen test statistic for variance constancy 10.85039
Hansen test for joint parameter stability   -10.70504
Critical values 1% 2.5% 5% 7.5% 10% 20%
    2.12000  1.89000  1.68000  1.58000  1.49000
1.28000
```

4.2 Threshold Autoregressive Models

Threshold autoregressive models (TAR) are a family of models for dealing with structural breaks in which regime-switches occur when threshold variable, q_t, reaches a threshold value, c. Rather than assuming that regime-switches occur as a function of time, or as a function of an unobservable underlying mechanism, TAR models consider regime switches that are dependent on observable data. The assumed threshold variable and nature of the regime adjustment results in a variety of possible TAR models. The standard TAR model assumes that the threshold variable is a known function of the data. Estimation of the TAR model requires determining the optimal parameter estimates, as well as the optimal threshold value.

Self-Exciting TAR model assumes that the threshold variable is a lagged value of the dependent variable, and requires estimating both the threshold value and the threshold lag, along with the standard model parameters.

The general two-regime TAR model is given by

$$y_t = \left(\alpha_0 + \alpha_1 y_{t-1} + \cdots + \alpha_p y_{t-p}\right) 1\left(q_{t-1} \le \gamma\right)$$
$$+\left(\beta_0 + \beta_1 y_{t-1} \cdots + \beta_p y_{t-p}\right) 1\left(q_{t-1} > \gamma\right) + e_t$$

where 1(.) is the indicator function and $q_{t-1} = q\left(y_{t-1}, \cdots, y_{t-p}\right)$ is a function of the data. The model can be more clearly represented for least squares estimation by define

$$x_t = \left(1 y_{t-1} \cdots y_{t-p}\right)',$$
$$x_t(\gamma) = \left(x_t' 1\left(q_{t-1} \le \gamma\right) x_t' 1\left(q_{t-1} > \gamma\right)\right)',$$
$$y_t = x_t' \alpha 1\left(q_{t-1} \le \gamma\right) + x_t' \beta 1\left(q_{t-1} \le \gamma\right) + e_t,$$

or

$$y_t = x_t(y)'\theta + e_t$$

where $\theta = (\propto' \beta')$.

Under LS estimation the parameter vector θ then becomes

$$\theta(\gamma) = \left(\sum_{t=1}^{n} x_t(\gamma) x_t(\gamma)'\right)^{-1} \left(\sum_{t=1}^{n} x_t(\gamma) y_t\right)$$

The estimation residuals and residual variance are given by

$$\sigma_n^2(\gamma) = \frac{1}{n} \sum_{t=1}^{n} \hat{e}_t(\gamma)^2$$

The optimal LS estimate of the threshold value is then the value that minimizes the residual variance,

$$\hat{\gamma} = \underset{\gamma \in \Gamma}{\operatorname{argmin}}\, \hat{\sigma}_n^2(\gamma)$$

While both the TAR and SETAR models consider abrupt regime switches, the model can be adjusted to allow for smoothed transitions between regimes. The family of Smooth Transition Autoregressive model (STAR) assumes that regime switches follow a continuous process which is a function of the threshold variable. This results in a family of models such that

$$x_t = \left(1\, y_{t-1} \cdots y_{t-p}\right)'$$

$$y_t = x_t'\alpha\left[1 - G(q_{t-1} : \gamma, c)\right] + x_t'\beta(q_{t-1} : \gamma, c) + e_t$$

The **GAUSS TSMT** module provides the **tarTest** procedure for testing for and estimating TAR models. In addition, it includes the **starTest** procedure for testing for STAR dynamics.

4.2.1 TAR Model

The **GAUSS tarTest** procedure estimates a p'th order threshold autoregression and tests the hypothesis of a linear autoregression, using the statistics described in"Inference when a nuisance parameter is not identified under the null hypothesis."(Hansen, 1996).

4.2.2 TAR Model Control Structure

The TAR control structure allows users to specify and pass model parameters efficiently to the **tarTest** procedure. The first step to TAR modeling is to declare and initialize an instance of the TAR control structure. The **GAUSS** procedure **tarControlCreate** initializes an instance of the TAR control

structure with preset default values. The following example code sets the **GAUSS** workspace for TAR modeling:

```
//Declare the structure
struct tarControl TAR0;

//Initialize the structure
TAR0 = tarControlCreate();
```

Once initialize, the TAR control structure contains default values for 8 parameters used in TAR estimation and testing:

omit	Scalar or vector, lags (below p) to omit from.
p	Autoregression [0 implies an AR(p)].
lowerQuantile	Scalar, the lower quantile.
upperQuantile	Scalar, the upper quantile.
rep	Scalar, the number of simulation replications.
out	Scalar, 0 or 1, 1 indicates screen output.
graph	Scalar, 0 or 1, 1 indicates screen output.
dstart	Start date of the time series (follows format used by **plotTS**).
freq	Data frequency, "12" for monthly, "4" for quarterly, "1" for annual.

Parameters are changed from the default values using standard structure "dot" referencing:

```
//CHANGE PARAMETER VALUE
TAR0.p = 3;
```

4.2.3 Estimating the TAR Model

Once the TAR control structure is initialized it is passed to directly to the **tarTest** procedure, along with a Nx1 vector of time series data,for estimating the TAR model. The output from the **tarTest** procedure are then returned to TAR output structure:

```
//DECLARE OUTPUT STRUCTURE
struct tarOutput TARout;

//ESTIMATE MODEL
TARout = tarTest(y,p,TAR0);
```

4.2.4 Example

This example follows the empirical example found in Hansen (1996) and estimates a threshold model for quarterly GNP growth rates. The data file "gnp.dat" contains seasonable adjusted GNP for 1947-1990 and is transformed to annualized quarterly growth rates:

```
load gnp[175,2] = gnp.asc;
yg = ln(gnp[.,1]);
y = (yg[2:175] - yg[1:174]) * 400;
```

Next all parameter values for the TAR estimation must be set

```
//Declare the structure
struct tarControl TAR0;

//Initialize the structure
TAR0 = tarControlCreate();
```

```
//Maximum number of lags considered
TAR0.p = 5;

//Lags to omit from the test
TAR0.omit = 3 4;

//Trimming from top (lowerQuantile) and bottom
(upperQuantile) of data
TAR0.lowerQuantile = .15;
TAR0.upperQuantile = .85;

//Number of replications for Monte Carlo
TAR0.rep = 5000;

//Output and graph reporting
TAR0.printOutput = 1;
TAR0.graph = 1;

//Data start date and frequency
TAR0.dstart = 1947;
TAR0.freq = 4;
```


Finally, call the **GAUSS** procedure **tarTest**

```
//DECLARE OUTPUT STRUCTURE
struct tarOutput TARout;

//ESTIMATE MODEL
TARout = tarTest(y,p,TAR0)
```

This produces the following output to the command/program window:

```
OLS Estimation of Null Linear Model

Variable     Estimate         S.E.
C            1.99225488       0.59341810
Y(t-1)       0.31753696       0.08929921
Y(t-2)       0.13197878       0.08801236
Y(t-5)      -0.08696297       0.06763670

Residual Variance        15.960496


Searching over Threshold Variable:               1
Searching over Threshold Variable:               2
Searching over Threshold Variable:               3
Global Estimates
Threshold Variable Lag     2.0000000
Threshold Estimate       0.012572093
Error Variance             14.548361

Regime 1: Y(t-2) < 0.012572

Variable     Estimate         S.E.
C           -3.21255539       2.12039565
Y(t-1)       0.51278104       0.24699822
Y(t-2)      -0.92692272       0.30831951
Y(t-5)       0.38445656       0.24603002


Regime 1 Error Variance          23.533054

Regime 2: Y(t-2) > 0.012572
```

```
Variable     Estimate         S.E.
C            2.14186153       0.77389336
Y(t-1)       0.30085440       0.10132777
Y(t-2)       0.18484356       0.10131018
Y(t-5)      -0.15813482       0.07335517


Regime 2 Error Variance          12.143010

Test Statistics and Estimated Asymptotic P-Values

Robust LM Statistics
SupLM      14.06847762       0.16940000
ExpLM       3.96481133       0.16620000
AveLM       4.68986250       0.27380000

Standard LM Statistics
SupLMs     18.24477743       0.94380000
ExpLMs      4.77627149       0.94320000
AveLMs      4.57209118       0.87960000
```

In addition, the following graphs are sent to the graphics window:

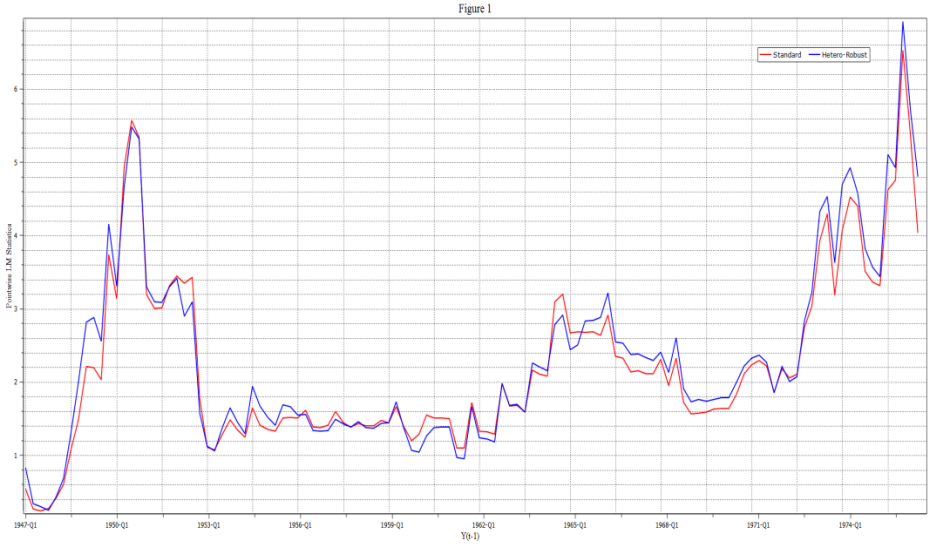
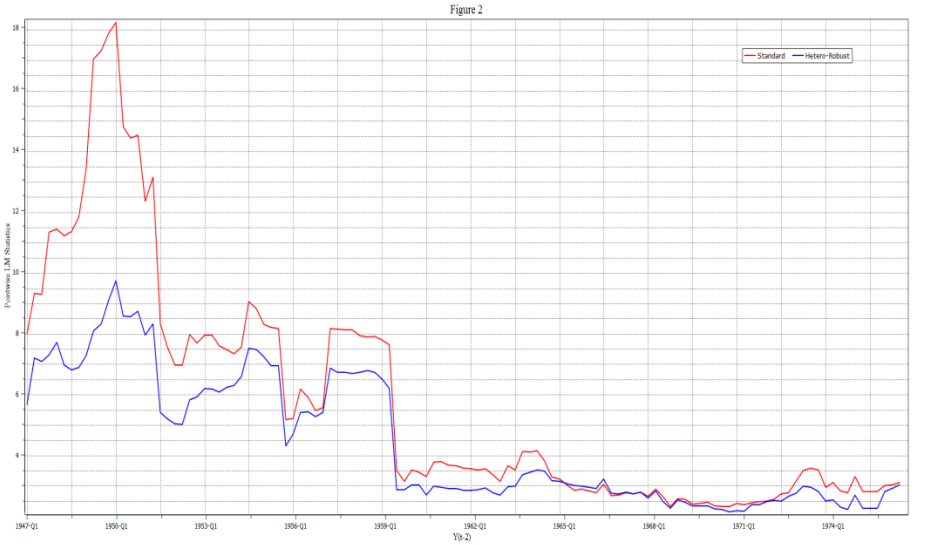


Figure 1
Standard
Hetero-Robust
Y(t-1)



4.3 Switching Regression

The **TSMT** module contains a procedure for estimating the regime-switching model of Hamilton (1994, 2005).

4.3.1 switchmtControl

This table contains a list of the members of the `switchmtControl` structure:

`constVariance`	scalar, if nonzero, error variances are constant across states.
`relevantStates`	scalar, if nonzero, lagged states are relevant for time series variable.
`aBayes, bBayes,`	scalars, if nonzero, parameters that

cBayes	control the Bayesian prior as described in Hamilton (1991).
userTransEqp	scalar, pointer to user-provided function for setting equality constraints on transition probability matrix.
start	instance of a *PV* structure containing start values.
ctl	instance of an *sqpsolvemtControl* structure.
output	scalar, controls printing. The default is print output, otherwise no output is printed.
title	string, title of run, default = "".

Start

Switching models can be difficult to estimate and may require a good starting point. This can be done by assigning a *PV* structure containing the starting point to the *start* member of the *switchmtControl* structure passed in as the first argument in the call to **switchmt**.

Only those members of the *PV* structure need to be filled that are relevant to the model being estimated. It contains the following members:

beta0	1, num_states by 1 vector, constants.
beta	2, num_states by K, coefficients on K independent variables if any.
phi	3, num_lags by 1 vector, autoregression coefficients.
sigma	4, scalar or num_states by 1 vector, error variances. If vmc.constVariance is zero, it is a scalar, otherwise it's a vector.

p	5, num_states by num_states matrix, transition probabilities.

For example,

```
struct  switchmtControl vmc;
vmc.start = pvPacki(pvCreate(),3|-3, "beta0", 1);
vmc.start = pvPacki(vmc.start,.1|.01,"Phi",3);
vmc.start = pvPacki(vmc.start, 1, "Sigma",4);
vmc.start = pvPacki(vmc.start,(.8~.1)|(.2~.9),"P",5);
```

Controlling Optimization

The *switchmtControl* structure contains an *sqpsolvemtControl*structure, the elements of which can be set to control the optimization. For example,

```
struct  switchmtControl vmc;
vmc.ctl.maxIters = 1000;
```

will set the maximum number of iterations to 1000. See the documentation for **sqpsolvemt** for discussion of the members of its control structure.

4.3.2 References

1. Engel, Charles and Hamilton, James, D., 1990, "Long Swings in the Exchange Rate: Are They in the Data and Do Markets Know It?" *American Economic Review*, 80(4):689.

2. Hamilton, James D., 1991, "A quasi-Bayesian approach to estimating

parameters for mixtures of Normal distributions," *Journal of Business and Economic Statistics*, 9:27-39.

3. Hamilton, James D., 1994, **Time Series Analysis**, Princeton University Press.

4. Hamilton, James D., 2005, "Regime-switching models," Department of Economics, University of San Diego, California.

4.4 Structural Break Models

The **sbreak** procedure estimates models with multiple structural breaks following the Bai and Perron (1998) global estimation method. The procedure may be used with complete or partial structural break models.

The Bai and Perron (1998) methodology utilizes least squares to estimate all structural breaks up to a user specified maximum number of breaks. Consider the model:

$$y_t = x_t'\beta + z_t'\delta_t + u_t$$
$$\left(t = T_{j-1}+1, \ldots, T_j\right)$$

for j 1 ,…, m+1. This model allows the observed dependent variable,y_t , to be modeled as a linear combination of regressors with both time invariant coefficients,x_t ,and time variant regressors, z_t .

Both the model parameters and all break points are estimated such that the sum of squares across all possible sample splits is minimized:

$$\hat{k} = argmin_k\left(\min_{\mu_1,\mu_2}\left\{\sum_{t=1}^{k}\left(Y_{t,1}-x_{t,1}'\beta+z_{t,1}'\delta_1\right)^2 + \sum_{t=k+1}^{T}\left(Y_{t,2}-x_{t,2}'\beta+z_{t,2}'\delta_2\right)^2\right\}\right)$$

4.5 Structural Break Control Structure

The structural break control structure allows users to specify and pass model parameters efficiently to the **sbreak** procedure. The first step to modeling structural breaks is to declare and initialize an instance of the structural break control structure. The **GAUSS** procedure **sbControlCreate** initializes an instance of the structural break control structure with preset default values. The following example code sets the **GAUSS** workspace for structural break modeling:

```
//Declare the structure
struct SBControl sbc0;

//Initialize the structure
sbc0 = sbControlCreate();
```

Once initialized, the structural break control structure contains default values for 10 parameters used in structural break estimation and testing:

q	Scalar, number of regressors subject to change.
p	Scalar, number of time-invariant regressors (x).
m	Scalar, maximum number of breaks.
trim	Scalar, trimming value.
h	Scalar, minimum length of segment (h>p+q), if less **GAUSS** automatically sets at default value of trim*T.
initialBeta	Scalar, initial values for beta
maxIters	Scalar, maximum number of iterations.
printOutput	Scalar, 0 or 1, 1 prints output to screen.
eps	Scalar, conversion criterion.

Parameters are changed from the default values using standard structure "dot" referencing:

```
//CHANGE PARAMETER VALUE
sbc0.q = 1;
sbc0.m = 5;
sbc0.trim = 0.15;
sbc0.h = 0;
sbc0.printOutput = 1;
sbc0.initialBeta = 0.5;
sbc0.maxIters = 40;
```

4.6 Estimating the Structural Break

Once the structural break control structure is initialized it is passed to directly to the **sbreak** procedure, along with a *T*x1 vector of time series data, *T*x*q* variable of regressors subject to structural change, and a *T*x*p* vector of regressors with time invariant coefficients.The output from the **sbreak** procedure is then returned to **sbreak** output structure:

```
//Declare the structure
struct sbOutput SBout;

//ESTIMATE MODEL
SBout = sbreak(y,z,x,sbc0);
```

The **GAUSS** procedure utilizes the dynamic programming methodology of Bai and Perron (2003), incorporating matrix functions and structures for improved efficiency.

4.6.1 Example

This example follows the empirical application of Bai and Perron (2003). Below we use the **GAUSS sbreak** procedure to estimate breaks in the mean of the US ex-post real interest rate. The first step is to directly load the quarterly series of the three-month treasury bill provided in the file `real.out`:

```
//Load y data
load path = c:\gauss14\examples
load y[] = real.dat

//Specify regressors
//Time varying coefficients in z
z = ones(rows(y),1);

//No time invariant regressors
x = 0;
```

Next we declare and initialize the structural break control structure and set model specific parameters:

```
//Declare sbControl structure
struct sbControl sbc0;

//Initialize instance of structure
sbc0 = sbControlCreate();

//Set individual model parameters
sbc0.q = 1;
sbc0.m = 5;
sbc0.trim = 0.15;
```

```
h = 0;
sbc0.printOutput = 1;
sbc0.initialBeta = 0.5;
sbc0.maxIters = 40;
```

Finally we declare the output structure and call the **sbreak** procedure:

```
//DECLARE OUTPUT STRUCTURE
struct sbOutput SBout;

//ESTIMATE MODEL
SBout = sbreak(y,z,x,sbc0);
```

4.7 References

1. Bai, J and Perron, P. (1998) Estimating and Testing Linear Models with Structural Change, *Econometrica, 66(1)*, 47-78.

2. Bai, J and Perron, P. (2003) Computation and analysis of multiple structural change models, *Journal of Applied Econometrics, 18(1)*, 1-22.

3. Brown, R.L., Durbin, J., and Evans, J.M. (1975). Techniques for testing the constancy of regression relationships over time, *Journal of Royal Statistical Society, Series B, 35*, 149-192.

4. Chow, G.C. (1960). Tests of equality between sets of coefficients in two linear regressions, *Econometrica, 52*, 211-22.

5. Engel,C. and Hamilton, J. (1990). Long swings in the exchange rate: Are they in the data and do markets know it?, *American Economic Review, 80 (4)*, 689.

6. Franses, P.H. and Dijk, D. (2000) **Non-linear Time Series Models in**

Empirical Finance. Cambridge University Press, New York.

7. Hamilton, J. (1991) A quasi-bayesian approach to estimating parameters for mixtures of normal distributions, *Journal of Business and Economic Statistics, 9*, 27-39.

8. Hamilton, J. (1994) **Time Series Analysis**, Princeton University Press, New York.

9. Hamilton, J. (2005), Regime-switching models, *Mimeo Department of Economics, University of San Diego, California*.

10. Hansen, B.E. (1996). Inference when a nuisance parameter is not identified under the null hypothesis, *Econometrica, 64(2)*, 413-430.

11. Hansen, B.E. (1992). Testing for parameter instability in linear models, *Journal of Policy Modeling*, 14(4): 517-533.

12. Nyblom, J. (1989). Testing for the constancy of parameters over time, *Journal of American Statistical Association*, 84(405), 223-230.

13. Zivot, E., and Wang, J. (2002). **Modeling Financial Time Series with S-PLUS**. Springer- Verlag, New York.

5 LSDV

The **TSMT** module contains a procedure for estimating and LSDV models for unbalanced data with correlation for bias. **`lsdvmt`** is the main procedure for estimating LSDV models. This model analyzes data that are both time series and cross-sectional..

5.1 LSDV Model

Following Bruno (2005) a set of time-series/cross-sectional is estimated by the LSDV model:

$$y_{it} = \gamma_1 y_{i,\,t-1} + \gamma_2 y_{i,t-2} + \cdots + \gamma_m y_{i,\,t-m} + x'_{it}\beta + \eta_i + \epsilon_{i,\,t}$$

where $i = 1,\cdots, N$, and $t = 1,\cdots, T$,$x_{i,t}$ is the $(k-1)\times 1$ vector of strictly exogenous explanatory variables, η_i is the unobserved individual effect, and $\epsilon_{i,t}$ is the unobserved disturbance.

Bruno's model allows for missing observations. Define the selection rule for the *it* observation where

$$s_{i,t} = \begin{cases} 1 & if\left(y_{i,t}, x_{i,t}\right), \ldots, (y_{i,t-m}, x_{i,t-m}) observed \\ 0 & otherwise \end{cases}$$

In matrix form we have

$$Sy = SD\eta + SW\delta + S\epsilon$$

where S is the $NT \times NT$ diagonal matrix with $s_{i,t}$ on the diagonal, $D = I_N \otimes l_T$ is the $NT \times N$ matrix of individual dummies, l_T is a $T \times 1$ vector of ones $yandW = \left[y_{-1} \ldots y_{-m} X\right]$ are the $NT \times 1$ and $NT \times K$ matrices of observations, $\delta = [\gamma \beta']'$ is the vector of coefficients, and η is the $NT \times 1$ vector of individual effects, ϵ is the $NT \times 1$ vector of disturbances,

The LSDV estimator is

$$\delta_{lsdv} = (W' A_s w)^{-1} W' A_s y$$

where

$$A_s = S(1 - D(D'SD)^{-1} D')S$$

5.1.1 Bias Correction

The lsdv model is well-known to be biased. Bruno (2005) reports several corrections for this bias. The one implemented in LSDVMT is

$$
\begin{aligned}
c1 &= \sigma_\epsilon^2 tr(\Pi) q_m \\
c2 &= -\sigma_\epsilon^2 [Q\overline{W}'\Pi A_s \overline{W} + tr(Q\overline{W}'\Pi A_s \overline{W}) I_{k+m} + 2\sigma_\epsilon^2 q_{11} tr(\Pi'\Pi\Pi) I_{k+m}] q_1 \\
c3 &= \sigma_\epsilon^4 tr(\Pi) \{ 2q_{mm} Q\overline{W}'\Pi\Pi'\Pi\overline{W} q_m + [(q'_m \overline{W}'\Pi\Pi'\overline{W} q_m) \\
&\quad + q_{11} tr(Q\overline{W}'\Pi\Pi'\overline{W}) + 2tr(\Pi'\Pi\Pi'\Pi) q_{mm}^2] qm \}
\end{aligned}
$$

where $Q = [E(W'A_s W)]^{-1} = \left[\overline{W}'A_s \overline{W} + \sigma_\epsilon^2 tr(\Pi'\Pi) e_m em' \right]^{-1}$; $\overline{W} = E(W)$; $e_m = (1, 1, \ldots 0, 0,)'$ is a $k \times 1$ vector with *m* ones; $q_m = Qe_m$; $q_{mm} = e'_m q_m$; L_T is the $T \times T$ matrix with *m* until lower subdiagonals and all other elements set to zero; $L = I_N \otimes L_T$; $\Gamma_T = (I_T - \gamma L_T)^{-1}$; $\Gamma = I_N \otimes \Gamma_T$; and $\Pi = A_s L \Gamma$.

The correction for bias is

$$B = c1 + c2 + c3$$

5.1.2 Example

The data must be organized in the following manner,

individual 1	T1
	T2
	.
	.
	.
individual 2	T1
	T2
	.

```
                              .
                              .
.                             T1
.                             T2
.                             .
.                             .
.                             .
```

LSDVMT

```
library tsmt;

struct  lsdvmtControl c0;
c0 = lsdvmtControlCreate();

series_len = 5;
num_lags = 2;

struct  DS d0;
d0 = reshape(dsCreate(),2,1);
d0[1].dname = "lsdvmt";
d0[1].vnames = "Y";
d0[2].vnames = "X1" $| "X2" $| "x3";

struct  lsdvmtOut out1;
out1 = lsdvmt(c0,d0,series_len,num_lags);
```

5.1.3 Special Features

Constraints

By default, **lsdvmt** constrains the coefficients of the lagged dependent variable to (-1, 1). This can be disabled by setting the *Constrain* member of the *lsdvmtControl* structure to zero.

Additional linear constraints can be placed on any of the coefficients by setting the *A*, *B* members of the *lsdvmtControl* structure for linear equality constraints, and the *C* and *D* members for linear inequality constraints. You may also set the *Bounds* member for setting bounds on the coefficients. For example:

```
struct lsdvmtControl c0;
c0 = lsdvmtControlCreate();

c0.A = { 0 -1 1 0 0 };
c0.B = 0;
```

will constrain the second coefficient to be equal to the third coefficient.

The following:

```
struct lsdvmtControl c0;
c0 = lsdvmtControlCreate();

c0.C = { 0 0 1 1 0 };
c0.D = 0;
```

constrains the sum of the third and fourth coefficients to be greater than zero. And this:

```
struct lsdvmtControl c0;
c0 = lsdvmtControlCreate();

c0.bounds = { 0 1e256,
              0 1e256,
              0 1e256,
              0 1e256,
              0 1e256  };
```

constrains all coefficients to be greater than zero.

Scaling

By default the data are scaled before the estimation and the results unscaled. This can be turned off by setting the *scale* member of the *lsdvmtControl* structure to zero.

5.1.4 The lsdvmtControl Structure

The table below contains a list of the control variables that are members of the *lsdvmtControl* structure. They give the user control over the model's specification and estimation.

Constrain	scalar, if nonzero, constraints applied to autoregression coefficients.
A	matrix, linear equality coefficient matrix.
B	matrix, linear equality constant vector.
Bounds	matrix, upper and lower bounds on parameter estimates.
C	matrix, linear inequality coefficients matrix.
D	matrix, linear inequality constant vector.
scale	scalar, if nonzero, data are scaled.

output	scalar, determines the output to be printed.
title	string, title to be printed at top of header.

5.1.5 The lsdvmtOut Structure

This table contains a list of the members of the *lsdvmtOut*structure.

AutoCoefficients	vector, uncorrected autoregression coefficients.
RegCoefficients	vector, uncorrected regression coefficients.
AutoCoefficientsCorr	vector, bias corrected autoregression coefficients.
RegCoefficientsCorr	vector, bias corrected regression coefficients.
AutoStderrs	vector, autoregression coefficient standard errors.
RegStderrs	vector, regression coefficient standard errors.
CovPar	matrix, covariance matrix of parameters.
SSresidual	scalar, residual sums of squares.
SStotal	scalar, total sums of squares.
SSexplained	scalar, explained sums of squares.
SSpooledResidual	scalar, pooled residual sums of squares.
biasCorr	vector, bias corrections.
Lagrange	vector, Lagrangeans for constraints.
numCases	scalar, number of cases.
numObservations	scalar, number of observations.
numMissing	scalar, number of observations with

	missing data.
numDF	scalar, number of degrees of freedom.
numPeriods	scalar, number of periods.
numParameters	scalar, number of parameters.
numPeriods	scalar, number of periods in time series.

5.2 References

1. Bruno, Giovanni S .F. (2005). "Approximating the bias of the LSDVMT estimator for dynamic unbalanced panel data models," *Economic Letters*, 87, 361-366.

2. Greene, William H. (2000). **Econometric Analysis**, 4th Ed., Prentice-Hall, New Jersey.

6 Univariate GARCH Models

For the generalized autoregressive conditional heteroskedastic (GARCH) model let

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

where $t = 1, 2, \ldots T$, and y_t is an observed time series, x_t is an observed time series of fixed exogenous variables including a column of ones, β a vector of coefficients, and $\sigma_t = E(\epsilon_t)$.

Also define

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

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

6.1 Standard Model

The standard specification of the conditional variance is

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

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

6.2 Log-likelihood

For maximum likelihood estimation of all models three distributions may be specified for η_t, the Normal, Student's t, and the skew generalized t distribution.

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

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

where

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

Student's t Distribution

The student's t distribution with ν degrees of freedom and variance σ^2 for η_t is

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

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

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

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

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

where

$$\gamma_i = 0.55\sigma_i^{-1}\kappa B(1/\kappa, v/\kappa)^{-\frac{3}{2}} B(3/\kappa, (v-2)/\kappa)^{\frac{1}{2}}$$

$$\theta = S^{-1} B(1/\kappa, v/\kappa)^{\frac{1}{2}} B(3/\kappa, (v-2)/\kappa)^{-\frac{1}{2}}$$

$$S = \left(1 + 3\lambda^2 + 4\lambda^2 B(2/\kappa, (v-1)/\kappa)^2 B(1/\kappa, v/\kappa)^{-1} B(3/\kappa, (v-2)/\kappa)^{-1}\right)^{\frac{1}{2}}$$

$$\mu_i = 2\sigma_i \lambda B(2/\kappa, (v-1)/\kappa) B(1/\kappa, v/\kappa)^{-\frac{1}{2}} B(3/\kappa, (v-2)/\kappa)^{-\frac{1}{2}}$$

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

6.3 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

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

and,

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

where

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

GARCH(p,q)

General constraints for $p > 2$ haven't been worked out. For such models the conditional variances are directly constrained 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.

6.3.1 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 ensure that you have the best fitting solution that satisfies the conditions of stationarity and nonnegative of conditional variances.

6.3.2 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.

6.4 IGARCH

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

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

6.5 GJRGARCH

Glosten, L.R., Jagannathan, R., and Runkle, D.E., 1993, proposed a model with asymmetrical effects of the conditional variance:

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

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

6.6 GARCHM

For the GARCHM, or GARCH-in-mean, model the time series equation is modified to include the square root of the conditional variance

$$\epsilon_t = y_t - x_t\beta - \delta\sigma_t^{1/2}$$

6.7 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 **TSMT** procedures don't compute t-statistics, but rather compute and report confidence limits.

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

$$JF(J, N-K; \alpha) = (\theta - \hat{\theta})' V^{-1} (\theta - \hat{\theta}) \tag{11}$$

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

$$\min \left\{ \eta_k' \theta \middle| (\theta - \hat{\theta})' V^{-1} (\theta - \hat{\theta}) \geq JF(J, N-K; \alpha) \right\} \tag{12}$$

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 (12) is no longer (5.7) and second, (11) is not valid whenever the hyper-ellipsoid is on or near a constraint boundary.

(11) 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 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.

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 0.8. 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 may produce correct limits (Geweke, 1995), but

they are considerably more computationally intensive. This method also assumes that the likelihood can be computed for parameters outside constraint boundaries. This is not the case for most constrained models and in this case the sampling method used for Bayesian limits will be biased.

6.7.1 Testing Against Inequality Constraints

Constraints of the form

$$H\theta \geq 0 \tag{13}$$

where H is a matrix of constants, arise in various empirical studies. There is a large literature on statistical inference under such linear inequality constraints, and more generally under nonlinear inequality constraints as well. An up-to-date account of these developments may be found in Silvapulle and Sen (2005). In what follows, we shall provide an introduction to tests against inequality constraints and indicate how **GAUSS** may be used for implementing a simple score test against inequality constraints.

Let ψ denote a $q \times 1$ subvector of θ and λ denote the remaining components of θ. For simplicity, let us write $\theta = \begin{pmatrix} \lambda \\ \psi \end{pmatrix}$ where $\psi = \left(\psi_1, \ldots, \psi_q\right)'$ and $\lambda = \left(\lambda_1, \ldots, \lambda_q\right)'$.

Suppose that we wish to test

$$H_0: \psi = 0 \text{ against } H_1: R\psi \geq 0, \psi = 0 \tag{14}$$

where R is a given matrix of constants; thus, R does not depend on θ and it is nonstochastic. If our objective were to test $\psi = 0$ against $\psi \neq 0$, then a simple and easy to apply test is the Rao's Score test, or equivalently the Lagrange

Multiplier test. This test is also a valid for the inequality constrained testing problem in (14), but it may not be the best because it ignores the inequality constraint $R\psi \geq 0$ in the alternative hypothesis. Various tests of (14), including likelihood ratio and score tests, have been developed. Now, we provide the essential details for testing (14) using a *one-sided score test*.

First, it is convenient to introduce the so called chi-bar square distribution that plays an important role in constrained statistical inference. The asymptotic null distribution of the likelihood ratio/Wald/Score test of $\psi = 0$ against $\psi \neq 0$ is a chi-square. When there are inequality constraints, such as $R\psi \geq 0$, in the null or the alternative hypothesis, the role of the chi-square distribution is replaced by a chi-bar square distribution; this is defined in the next paragraph.

Let $Z \sim N(0, V)$, where *Z* is a $q \times 1$ random vector and *V* is a $q \times q$ positive definite matrix. Let

$$\overline{\chi}^2(V, R) Z' V^{-1} Z - \min_{Ra \geq 0} (Z - a)' V^{-1} (Z - a) \qquad (15)$$

in the second term, it is implicit that *a* is a vector of the same length as *Z*.

We shall use the notation $\overline{\chi}^2(V, R)$ is used for the random variable on the RHS of (15) and also for its distribution. The random variable, $\overline{\chi}^2(V, R)$ is said to have a *chi-bar square distribution* and it can be expressed as follows:

$$\Pr\left\{\overline{\chi}^2(V, R) \leq c\right\} = \sum_{i=0}^{q} w_i \Pr\left(\chi_i^2 \leq c\right)$$

for some non-negative numbers, $w_i, i = 0, \ldots q,$ that are functions of (*q*, *V*, *R*); these quantities are known as *chi-bar square weights* and also as *level probabilities*. Except in some very special cases

$\Pr\left\{\bar{\chi}^2(V,R) \le c\right\}$ is difficult to compute exactly. However, it can be estimated by simulation to a desired degree of precision as follows:

1. Generate Z from $N(0, V)$.
2. Compute $\bar{\chi}^2(V,R)$.
3. Repeat the first two steps M times, say M=10000.
4. Estimate $\Pr\left\{\bar{\chi}^2(V,R) \le c\right\}$ by the proportion of times $\bar{\chi}^2(V,R)$ turned out to be less than or equal to c.

This is the method employed by **GAUSS**; for a similar method for estimating $\{w_i\}$ see Wolak (1987). When the number of repeated samples M is 10000, the standard error of the estimate of the probability obtained by this simulation method does not exceed 0.005; if c is large so that $\Pr\left\{\bar{\chi}^2(V,R) \le c\right\}$ is less than 0.1, then the standard error is less than 0.003. Thus, the precision in the estimation can be controlled by adjusting the number of repeated samples, M.

The asymptotic null distributions of several statistics for testing (14) turns out to be a chi-bar square distribution. Therefore, the **`chibarsq()`** procedure plays an important role in the implementation of tests against inequality constraints.

6.7.2 One-sided Score Test

As in (14) let $\psi = \left(\psi_1, \ldots, \psi_q\right)'$ denote a $q \times 1$ subvector of θ and let λ denote the remaining components of θ, $\theta = \begin{pmatrix}\lambda \\ \psi\end{pmatrix}$ where $\psi = \left(\psi_1, \ldots, \psi_q\right)$ and $\lambda = \left(\lambda_1, \ldots, \lambda_{p-q}\right)'$. Suppose that we wish to test

$$H_0: \psi = 0 \text{ against } H_1: R\psi \ge 0, \psi \ne 0 \qquad (16)$$

where R is a given matrix of constants. A generalized version of Rao's Score test can be applied for testing H_0 vs H_1. Let us first introduce the following: Let $L(\theta)$ denote the log-likelihood and

$$s\left(\theta\right) = \frac{\partial L(\theta)}{\partial \theta} \text{ : score function.} \tag{17}$$

Let $s(\theta)$ be partitioned as follows to conform with (λ, ψ):

$$\begin{pmatrix} s_\lambda \\ s_\psi \end{pmatrix} = \begin{pmatrix} \frac{\partial L}{\partial \gamma} \\ \frac{\partial L}{\partial \psi} \end{pmatrix} \tag{18}$$

Similarly, let us introduce the following notation for partitioning any given matrix P of the same order as θ, to conform with the partition, (λ, θ):

$$P = \begin{pmatrix} P_{\lambda\lambda} & P_{\lambda\psi} \\ P_{\psi\lambda} & P_{\psi} \end{pmatrix} \tag{19}$$

Let $\tilde{\lambda}$ denote the *mle* of λ under $H_0: \psi = 0$, and let

$$\tilde{\theta} = \begin{pmatrix} \tilde{\lambda} \\ 0 \end{pmatrix} \tag{20}$$

denote the *mle* of θ under $H_0: \psi = 0$.

Let

$$A(\theta) = -E\left[n^{-1} \frac{\partial}{\partial \theta'} s(\theta) \right] - n^{-1} E\left[\frac{\partial^2}{\partial \theta' \partial \theta} L(\theta) \right] \tag{21), (22}$$

Let $\tilde{s}$, $\tilde{A}$ and $\tilde{B}$ denote the corresponding quantities evaluated at $\tilde{\theta}$. These three quantities can be obtained by calling the constrained maximum likelihood procedure under the constraint $H\theta = 0$ where

$$H = (0I)$$

and I is the identity matrix of the same order as the dimension of ψ; note that $H\theta = \psi$ and hence $H\theta = 0$ is equivalent to $\psi = 0$.

Now, the *one-sided score statistic* of Silvapulle and Silvapulle (1995) [SS, hereafter], which is a generalized version of Rao's Score statistic, for testing $H_0: \psi = 0$ against the one-sided alternative $H_1: R\psi \geq 0, \psi \neq 0$ is

$$T_S = \tilde{u}' \tilde{D}^{-1} \tilde{u} - \min_{Ra \geq 0} (\tilde{u} - a)' \tilde{D}^{-1} (\tilde{u} - a) \tag{23}$$

where

$$\tilde{D} = \left[\left(\tilde{A} \tilde{B}^{-1} \tilde{A}' \right)^{-1} \right]_{\psi\psi} \tag{24}$$

and

$$\tilde{u} = n^{-1/2} \left[\tilde{A}_{\psi\psi} - \tilde{A}_{\psi\lambda} \tilde{A}_{\lambda\lambda}^{-1} \tilde{A}_{\lambda\psi} \right]^{-1} \left[\tilde{s}_{\psi} - \tilde{A}_{\psi\lambda} \left(\tilde{A}_{\psi\psi} \right)^{-1} \tilde{s}_{\lambda} \right] \tag{25}$$

An attractive feature of this one-sided score test of SS is that it does not require estimation of the model under the inequality constraints in the alternative hypothesis, and further, the test is applicable for methods based on estimating equations such as Generalized Estimating Equations (GEE) of Liang and Zeger (1986).

The asymptotic distribution of T_S under the null hypothesis is $\overline{\chi}^2(\widetilde{D}, R)$ where

$$D = \left[\left(AB^{-1}A' \right)^{-1} \right]_{\psi\psi}$$

Therefore, if t_S denotes the sample value of T_S and D does not depend on λ then an approximate large sample p-value is

$$\Pr\left\{ \overline{\chi}^2(D, R) \geq t_S \right\}$$

Further, if the exact form of D is unknown, then an estimate of the p-value is obtained by substituting an estimate for D.

Usually D depends on λ. In this case, it is customary to define the asymptotic p-value as

$$\sup_{\lambda} \Pr\left\{ \overline{\chi}^{-2}(D_\lambda, R) \geq t_S \right\}$$

where the suffix λ is used to indicate the D matrix depends on λ. This can be computed approximately by evaluating

$$\Pr\left\{ \overline{\chi}^{-2}(D_\lambda, R) \geq t_S \right\}$$

over a grid of λ values and finding the maximum over that grid; if the dimension q of λ is large, this may be computing intensive. Alternatively, some authors have suggested to estimate the large sample p-value by

$$\widetilde{p} = \Pr\left\{ \overline{\chi}^{-2}(\widetilde{D}, R) \geq t_S \right\} \tag{26}$$

where $\widetilde{D}$ is treated as nonstochastic; its suitability would depend on the particular case, and hence should be used with caution.

An upper bound for the large sample *p*-value is

$$p_u = 0.5\left[\Pr\left(\chi^2_{q-1} \geq t_s\right) + \Pr\left(\chi^2_q \geq t_s\right)\right]$$

where q is the number of components in ψ.

6.7.3 Likelihood Ratio Test

The likelihood ratio statistic is defined as

$$LRT = 2\left[\max_{H_1} L(\theta) - \max_{H_0} L(\theta)\right] \tag{27}$$

The asymptotic null distribution of LRT is $\overline{\chi}^2\left(H\widetilde{A}^{-1}H', I\right)$ where I is the identity matrix (see Theorem 4.3.1 in Silvapulle and Sen, 2005). Therefore, an estimate of the *p*-value, corresponding to (26), for the likelihood ratio test is

$$\Pr\left\{\overline{\chi}^2\left(H\widetilde{A}^{-1}H', I\right) \geq LRT\right\}.$$

An upper bound for the *p*-value of LRT is

$$0.5\left[\Pr\left(\chi^2_2 \geq LRT\right) + \Pr\left(\chi^2_3 \geq LRT\right)\right].$$

6.7.4 Covariance Matrix of Parameters

The computed covariance matrix of the parameters 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 an argument based on a Taylor-series approximation to the likelihood function (e.g., Amemiya, 1985, page 111) shows that

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

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

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

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

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

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

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

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

where ζ is a conformable vector of slack parameters.

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

The likelihood function augmented by constraints is then

$$\begin{aligned} L_A = \; & L + \lambda_1 g(\theta)_1 + \ldots \lambda_1 g(\theta)^1 + \gamma_1 h_{\oplus 1}\left(\theta\right) + \ldots + \gamma_J h_{\oplus J}\left(\theta\right) \\ & + h_{\ominus 1}(\theta)_i - \zeta_1^2 + \ldots + h_{\ominus 1}\left(\theta\right) - \zeta_K^2 \end{aligned} ,$$

and the Hessian of the augmented likelihood is

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

where the dot represents the Jacobian with respect to $\theta, L=\Sigma_{i-1}^{N}\log P\left(Y_i;\theta\right)$, 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.

6.7.5 Quasi-Maximum Likelihood Covariance Matrix of Parameters

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

6.7.6 Confidence Limits

TSMT 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.

6.7.7 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 6.3). These are the least restrictive constraints for these models.

For constraints on the conditional variances set

```
struct garchControl f0;
f0 = garchControlCreate();
f0.cvConstraintsType = 1;
```

to select the Nelson and Cao (1992) constraints (described in Section 6.3). These are the least restrictive constraints that ensure stationarity and nonnegativity of the conditional variances, and are imposed by default.

For direct constraints on the conditional variances set

```
f0.cvConstraintsType = 0;
```

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.

```
f0.stConstraintsType = 0;
```

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.

The least restrictive enforcement of stationarity is to constrain the characteristic polynomial to be outside the unit circle. Set

```
f0.stConstraintsType = 1;
```

```
f0.stConstraintsType = 2
```

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.

6.7.8 Altering sqpsolvemt Control Variables

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  garchControl f0;
f0 = garchControlCreate();
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 garchControl f0;
    f0 = garchControlCreate();
    f0.sqpsolvemtControlProc = &sqpcont;
```

6.7.9 Bibliography

1. Geweke, John, 1995. "Posterior simulators in econometrics," Working Paper 555, Research Department, Federal Reserve Bank of Minneapolis.
2. 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.
3. Gouriéroux, Christian, 1997. **ARCH Models and Financial Applications**. New York: Springer-Verlag.
4. Nelson, Daniel B. and Cao, Charles Q., 1992. "Inequality constraints in the univariate GARCH model," *Journal of Business and Economic Statistics*, 10:229-235.
5. Wolak, Frank, 1991. "The local nature of hypothesis tests involving inequality constraints in nonlinear models," *Econometrica*, 59:981-995.

7 Panel Data

The **TSMT** module includes procedures for the computation of estimates for the "pooled times-series cross-section" (TSCS) regression model.

7.1 Pooled Time-Series Cross-Section Regression Model

The assumption is that there are multiple observations over time on a set of cross-sectional units (e.g., people, firms, countries). For example, the analyst may have data for a cross-section of individuals each measured over 10 time periods. While these models were devised to study a cross-section of units over multiple time periods, they also correspond to models in which there are data for groups such as schools or firms with measurements on multiple observations within the groups (e.g., students, teachers, employees).

The specific model that can be estimated with this program is a regression model with variable intercepts, i.e., a model with individual-specific effects. The regression parameters for the exogenous variables are assumed to be constant across cross-sectional units. The intercept varies across individuals.

This program provides three estimators:

- The fixed-effects OLS estimator (analysis of covariance estimator).
- The constrained OLS estimator (individual-specific effects are excluded from the equation).
- The random effects estimator using GLS.

A Hausman test is computed to show whether the error components (random effects) model is the correct specification.

In addition to providing the analysis of covariance and GLS estimates, two multiple partial-squared correlations are computed. The first partial correlation (squared correlation) shows the percentage of variation in the dependent variable that can be explained by the set of independent variables while holding constant the group variable. The second estimate shows the extent to which variation in the dependent variable can be accounted for by the group variable after the other independent variables have been statistically held constant.

A feature of this program is that it allows for a variable number of time-series observations per cross-sectional unit. For instance, there might be 5 time-series observations for the first individual, 10 for the second, and so on. This is useful, for example, if there are missing values.

7.1.1 Panel Series Unit Root Tests

Im, Pesaran, and Shin

Assuming that y_{it} is an autoregressive process such that

$$y_{it} = \left(1 - \phi_i\right)\mu_i + \phi_i y_{i,t-1} + \varepsilon_{it}$$
$$for \quad i = 1, \ldots, N, t = 1, \ldots, T$$

The Im, Pesaran, and Shin (2003) test considers the null hypothesis of a unit root across in all panels. When written in first difference form such that

$$\Delta y_{it} = \alpha_i + \beta_i y_{i,t-1} + \varepsilon_{it}$$

The null and alternative hypothesis become

$$\begin{aligned} H_0 &: \beta_i = 0 \qquad\qquad\qquad\qquad for\ \ all\ \ i, \\ H_1 &: \beta_i < 0 \qquad i = 1, 2, \ldots, N_1, \ \ \beta_i = 0, i = N_1 + 1, N_1 + 2, \ldots, N \end{aligned}$$

Individual ADF tests are performed for each panel, resulting in individual t-ratios for β_i in each panel. For each individual panel the t-ratio is

$$t_{i,T} = \frac{\widehat{\beta}_{iT}\left(y'_{i,-1} M_\tau y_{i,-1}\right)^{1/2}}{\widehat{\sigma}_{iT}} = \frac{\Delta y'_{i,-1} M_\tau y_{i,-1}}{\widehat{\sigma}_{iT}\left(y'_{i,-1} M_\tau y_{i,-1}\right)^{1/2'}}$$

where $\widehat{\beta}_{iT}$ is the OLS estimator of

$$\begin{aligned} \beta_i, X_i &= \left(\tau_T, y_{i,-1}\right) \\ \tau_T &= (1, 1, \ldots, 1)' \\ y_{i,-1} &= \left(y_{i,0}, y_{i,1}, \cdots, y_{i,T-1}\right)', \end{aligned}$$

and

$$M_{X_i} = I_T - X_i\left(X'_i X_i\right)X'_i$$

The IPS panel unit root test statistic is the average t-ratio across all panels

$$t\_bar_{NT} = \frac{1}{N}\sum_{i=1}^{N} t_{i,T}$$

This is modified such that

$$Z_{t\_bar} = \frac{\sqrt{N}\left\{t\_bar_{NT} - E(\tau_T)\right\}}{\sqrt{Var(\tau_T)}} \sim N(0, 1)$$

where $E(\tau_T)$ and $Var(\tau_T)$ are extrapolated from the tables provided in Im, Pesaran, Shin (2003).

Example

This example follows the Im, Pesaran, and Shin discussion found in Enders (2010). The example uses the PANEL.XLS file of quarterly values of the real effective exchange rates (CPI-based) for Australia, Canada, France, Germany, Japan, Netherlands, United Kingdom, and United States, as provided online by Enders, (2010).

The first step is loading the data, note that the first column is a time column.

```
load data[] = "panel_g.csv";
N = 8;
yt = reshape(data,rows(data)/(N+1),N+1);
log_yt = log(yt[.,2:9]);
```

The parameters for input into **ips** are set following Enders (2010)

```
//Input the lags used in Enders, Table 4.8

lags = {5,6,3,1,3,1,1,3};
demean = 0;
```

```
prt = 1;
trend = 0;
```

In this case a 8 x 1 vector specifies the lags to be used in the ADF tests, indicating the individually determined lags for each panel. As an alternative, a single integer could be entered for lags, in which case the same number of lags would be used for each panel. The additional parameters indicate that the data will not demeaned across time periods (*demean*), that no trend is included in the data (*trend*), and that all output should be printed to screen (*prt*). The final step is the call to the **ips** function:

```
//Run IPS test without a trend and without
//demeaning
{z_1,pcrt1} = ips(log_yt,trend,demean,lags,prt);
```

Running all parts of code together prints the following output:

```
Individual t-stats:
-2.46627
-2.00066
-3.51045
-3.42360
-1.86847
-1.94953
-2.06635
-2.37869
Average t-stat:
-2.458
The adjusted z-stat is:  -2.85025
The critical values are: -1.90700 -2.01086 -2.20629
```

Comparing these results to the case of cross panel demeaned data requires an adjustment to the demean parameter

```
//Run IPS test with time demeaning:

demean = 1;
{z_2,pcrt2} = ips(log_yt,trend,demean,lags,prt);
```

Demeaning the effective exchange rate data and maintaining the same lag specification results in the following output:

```
Individual t-stats:
-2.466
-2.001
-3.511
-3.424
-1.869
-1.950
-2.066
-2.379
Average t-stat:
-2.458
The adjusted z-stat is:  -3.067
The critical values are: -1.907 -2.011 -2.206
```

References

Enders, W. (2010). **Applied Econometric Time Series, 3e**. Hoboken, NJ: John Wiley & Sons, Inc.

Im, S.K. Pesaran, M.H., & Shin, Y. (2003). "Testing for unit roots in heterogeneous panels." *Journal of Econometrics*, 115, 53-74.

Levin-Lin-Chu Panel Unit Root Test

The Levin-Lin-Chu (2002) (LLC) panel unit root test is applicable to three different autoregressive models, varying only in the deterministic components included in the model:

$$Model\ 1: \Delta y_{it} = \delta y_{i,t-1} + \zeta_{it}$$
$$Model\ 2: \Delta y_{it} = \alpha_{0,i} + \delta y_{i,t-1} + \zeta_{it}$$
$$Model\ 3: \Delta y_{it} = \alpha_{0,i} + \alpha_{1i} + \delta y_{i,t-1} + \zeta_{it}$$

where $-2 < \delta \leq 0$ for $i = 1, \ldots, N$.

This test assumes a homogenous autoregressive parameter across all panels but does allow for heterogeneous error processes such the for each individual

$$\zeta_{it} = \sum_{j=1}^{\infty} \theta_{ij} \zeta_{i,t-j} + \varepsilon_{it}$$

The table below summarizes the LLC hypotheses tested given the assumed model.

LLC Testing Hypotheses by Model

Model	H_0	H_1
1	$\delta = 0$	$\delta < 0$
2	$\delta = 0$ and $\alpha_{0,i} = 0$ *for all i*	$\delta < 0$ *and* $\alpha_{0,t} \epsilon R$
3	$\delta = 0$ and $\alpha_{1,i} = 0$ *for all i*	$\delta < 0$ *and* $\alpha_{1,t} \epsilon R$

The LLC test utilizes a three step testing procedure. The first step of the procedure performs standard ADF regression for each panel

$$\Delta y_{it} = \delta y_{i,t-1} + \sum_{L=1}^{p_i} \theta_{iL} \Delta y_{i,t-L} + \alpha_{mi} d_{mt} + \varepsilon_{it}$$
$$m = 1, 2, 3$$

The LLC testing procedure allows for panel specific serial correlation correction lags, p_i , in each individual panel ADF test. The **GAUSS `llc`** function allows the user to provide a vector of non-negative integer lags directly specifying the values of p_i .

Once the optimal lag length is determined, $\Delta y_{i,t} and y_{i,t-1}$ are regressed against $\Delta y_{i,t-L} for \left(L = 1, \ldots, p_i\right)$ and the deterministic variables d_{mt} . The regressions are used to construct two vectors of residuals

$$\hat{e}_{it} = \Delta y_{i,t} - \sum_{L=1}^{p_i} \hat{\pi}_{iL} \Delta y_{i,t-L} - \hat{\alpha}_{mi} d_{mt}$$

and

$$\hat{v}_{it} = y_{i,t-1} - \sum_{L=1}^{p_i} \tilde{\pi}_{iL} \Delta y_{i,t-L} - \tilde{\alpha}_{mi} d_{mt}$$

These are normalized such that

$$\tilde{e}_{it} = \frac{\hat{e}_{it}}{\hat{\sigma}_{\varepsilon i}}, \quad \tilde{v}_{it} = \frac{\hat{v}_{it}}{\hat{\sigma}_{\varepsilon i}}$$

where $\hat{\sigma}_{\varepsilon i}$ is the panel specific regression standard error from the panel specific ADF regression.

The second step of testing is to calculate the ratio of the long-run to short-run standard deviations. The **llc** function uses the **getlrv** function to calculate the long-run variance. The methodology and kernel used for determining the long-run function are specified using the input *lag_method* and *kernel*, respectively. Once the long-run variance is calculated, panel specific deviation ratios are constructed such that

$$\hat{s}_i = \frac{\hat{\sigma}_{yi}}{\hat{\sigma}_{\varepsilon i}}$$

with the average standard deviation ratio given by

$$\hat{S}_N = \frac{1}{N}\sum_{i=1}^{N}\hat{s}_i$$

The final step of the procedure is to calculate the panel test statistics. First, all series are pooled to estimate the common autoregressive parameter

$$\tilde{e}_{it} = \delta\tilde{v}_{i,t-1} + \tilde{\varepsilon}_{it}$$

From this regression a standard t –ratio is constructed for δ given by

$$t_\delta = \frac{\hat{\delta}}{STD(\hat{\delta})'}$$

where

$$\delta = \frac{\sum_{i=1}^{N}\sum_{t=2+p_i}^{N}\tilde{v}_{i,t-1}\tilde{e}_{i,t}}{\sum_{i=1}^{N}\sum_{t=2+p_i}^{N}\tilde{v}_{i,t-1}},$$

$$STD(\hat{\delta}) = \hat{\delta}_{\varepsilon}\left[\sum_{i=1}^{N}\sum_{t=12+p_i}^{N}\tilde{v}_{i,t-1}^{2}\right]^{1/2}$$

$$\hat{\delta}_{\varepsilon}^{2} = \frac{1}{N\tilde{T}}\sum_{i=1}^{N}\sum_{t=12+p_i}^{N}\left(e_{i,t} - \delta v_{i,t-1}\right)^{2} \quad and$$

$$\tilde{T} = T - \overline{p} - 1$$

These calculations use $\overline{p}$, the average ADF serial correlation lag length across all panels. For Model 1, the test statistic t_{δ} has a standard normal distribution. For Models 2 and 3, an adjusted test statistic, t_{δ}^{*} ,

$$t_{\delta}^{*} = \frac{t_{\delta} - N\tilde{T}\hat{S}_{N}\hat{\delta}_{\tilde{\varepsilon}}^{-2}STD\left(\tilde{\delta}\right)\mu_{m\tilde{T}}^{*}}{\delta_{m\tilde{T}}^{*}}$$

follows a standard normal distribution. Both the mean adjustment parameter, $\mu_{m\tilde{T}}^{*}$, and the standard deviation adjustment parameter, $\delta_{m\tilde{T}}^{*}$, are extrapolated from Table 2 in Levin, et al. (2003).

References

Levin, A., Lin, C., & Chu, C.J. (2002). "Unit root tests in panel data: asymptotics and finite-sample properties." *Journal of Econometrics*, 108, 1-24.

Breitung and Das Panel Root Test

The Breitung and Das (2005) unit root tests the null hypothesis that y_{it} is nonstationary. The test estimates a model similar to that of the Levin-Lin-Chu test

$$y_{it} = D'_{it}\gamma_i + x_{it}$$

where

$$x_{it} = \alpha_1 x_{i,t-1} + \alpha_2 x_{i,t-2} + \epsilon_{it}$$

The **GAUSS** implementation of the Breitung test assumes what is uncorrelated across panels. Note that this model assumes that the autoregressive parameters are constant across all panels. Hence, the alternative hypothesis that y_{it} is stationary is equivalent to the alternative hypothesis that the common autoregressive parameters sum to less than one, $\alpha_1 + \alpha_2 < 1$.

No trend

The testing procedure depends directly on the deterministic terms included in the model, D'_{it}. If the deterministic terms include a constant and no trend, the procedure pre-whitens the dependent variable such that

$$y^l_{i,t} = y_{i,t-1} - y_{i,t-p}$$

where *p* is the predetermined or user specified, optimal number of lags for removing serial correlation.

If the deterministic model does not include a constant or trend, the data is pre-whitened such that

$$y_{i,t}^{l} = y_{i,t-1}$$

In these two cases, the test statistic is formulated using the residuals from two testing regressions, the regression of Δy_{it} and $y_{i,t}^{l}$ on $\Delta y_{i,t-1}, \cdots, \Delta y_{it-p}$. The test statistic λ is given by

$$\lambda = \frac{\Sigma_{i=1}^{N}\Sigma_{t=p+2}^{T} y_{it}^{l} \cdot \frac{\Delta y_{it}}{\sigma_i^2}}{\sqrt{\Sigma_{i=1}^{N}\Sigma_{t=p+2}^{T}\left(y_{it}^{l}\right)^2 / \sigma_i^2}}$$

where

$$\sigma_i^2 = \frac{1}{T-p-2}\sum_{t=p+2}^{T}\left(\Delta y_{it}\right)^2$$

The test statistic λ is asymptotically distributed N(0,1). The null hypothesis is rejected at small values of λ.

With trend

When a trend is included in the deterministic terms, the testing procedure requires additional prewhitening to remove the trend. Prewhitening with a trend involves the construction of four modified residual vectors, Δu_i, u_i^{l}, Δv_i, and v_i^{l}. The procedure first estimates the regression

$$\Delta y_{it} = \alpha_{i0} + \sum_{j=1}^{p}\alpha_{ij}\Delta y_{i,t-j} + v_{it}$$

From this regression, two vectors, Δu_i and v_i^l are constructed such that

$$\Delta u_{is} = \Delta y_{is} - \sum_{j=1}^{p} \hat{\alpha}_{ij} \Delta y_{is-j}$$

and

$$u_{is}^l = y_{i,s-1} - \sum_{j=1}^{p} \hat{\alpha}_{ij} \Delta y_{is-j-1}$$

From these let

$$\sigma_i^2 = \frac{1}{T-p-2} \sum_{s=1}^{T-p-1} (\Delta u_{is} - \overline{\Delta u_i}) u_{is}$$

The two additional, vectors and ,are constructed such that

$$\Delta v_{is} = \sqrt{\frac{T-p-s-1}{T-p-2}} \left(\Delta u_{is} - \frac{1}{T-p-s-1} \sum_{j=s+1}^{T-p-1} \Delta u_{ij} \right)$$

and

$$v_{is}^l = u_{is}^l - u_{i1}^l - (T-p-1)\overline{\Delta u_i}$$

where $\overline{\Delta u_i}$ is the mean of Δu_{is} across s.

From these, both the variance and test statistic are constructed

$$\sigma_i^2 = \frac{1}{T-p-2} \sum_{s=1}^{T-p-1} (\Delta u_{is} - \overline{\Delta u_i}) u_{is}$$

and

$$\lambda = \frac{\Sigma_{i=1}^{N} \Sigma_{t=p+2}^{T} y_{it}^{l} \cdot \frac{\Delta y_{it}}{\sigma_i^2}}{\sqrt{\Sigma_{i=1}^{N} \Sigma_{t=p+2}^{T} \left(y_{it}^{l}\right)^2 / \sigma_i^2}}$$

Example

For this example we will use a matrix of fifteen autoregressive time series created using the **simarmamt** data generating function (included in the **TSMT** library). All series will include a constant, but no trend:

```
b = 0.75;
p = 1;
q = 0;
const = 1;
tr = 0;
n = 1000;
k = 15;
std = 1;
seed = 10191;
yt = simarmamt(b,p,q,const,tr,n,k,std,seed);
```

This results in a panel of fifteen AR(1) time series with 1000 observations each, constants equal to 1, and a common autoregressive coefficient equal to 0.75. It also has a constant, which must be indicated as an input into the **breitung** function

```
trend = 0;
constant = 1;
```

In addition, the function inputs include an indicator whether to demean the data across panels and a specification of the lag selection.

```
demean = 0;
lags = 3;
```

Finally, call the **breitung** function to test for unit roots and print the output.

```
bstat = breitung(yt,trend,constant,demean,lags);
print "The Breitung test statstic is:";;
print bstat;
```

This will produce the following output:

```
The Breitung test statstic is:     -4.89
```

References

1. Breitung, J. 2000. "The local power of some unit root tests for panel data." In *Advances in Econometrics*, **Volume 15: Nonstationary Panels, Panel Cointegration, and Dynamic Panels**, ed. B. H. Baltagi, 161–178. Amsterdam: JAI Press.
2. Breitung, J., and S. Das. 2005. "Panel unit root tests under cross-sectional dependence." *Statistica Neerlandica* 59: 414–433.

7.2 References

1. Judge, George C., R. Carter Hill, William E. Griffiths, Helmut Lutkepohl, and Tsoung-Chao Lee. 1988. **Introduction to the Theory and Practice of Econometrics. Second Edition**, New York: Wiley.

2. Hsiao, Cheng. 1986. **Analysis of Panel Data.** Cambridge: Cambridge University Press.

8 ARIMA

The **TSMT** module includes procedures for the computation of estimates and forecasts for the autoregressive integrated moving average model. The model may include fixed regressors such as linear or quadratic time trends, or other explanatory variables which are predetermined. Forecasts are computed using the estimated parameters and errors.

8.1 ARIMA Models

This program will compute estimates of the parameters and standard errors for a time series model with ARMA errors. If the model contains only autoregressive parameters, then **arimamt** gives the same estimates as **autoregmt**. **arimamt** reports standard errors, parameters estimates, model selection criteria, roots of the parameters, the Ljung-Box portmanteau statistic and the covariance and correlation matrices.

The model estimated is of the general form:

$$\phi(L)\left[(1-L)^d y_t - x_t\beta\right] = \theta(L)\epsilon_t$$

where

$$
\begin{aligned}
L^j y_t &= y_{t-j} \\
\phi(L) &= 1 - \phi_2 L^2 - \ldots - \phi_p L^p \\
\theta(L) &= 1 - \theta_1 L^2 - \ldots - \theta_q L^q
\end{aligned}
$$

where it is assumed that e_t is a white noise error term, distributed as $N\left(0, \sigma^2\right)$. Such models are referred to as ARIMA(*p, d, q*), where *p* is the autoregressive order, *d* is the difference order and *q* is the moving average order.

The parameters to be estimated are thus: $\phi(\mathrm{P} \times 1), \theta(\mathrm{Q} \times 1), \beta(\mathrm{M} \times 1)$ and $\sigma^2(1 \times 1)$.

The **arimamt** procedure computes starting values or allows the user to specify starting values. User specified starting values are useful when the user wants to determine whether the parameter estimates computed by **arimamt** correspond to the global maximum of the log likelihood function and not just a local maximum. Finally, the **tsforecast** procedure computes forecasts for the series *h* steps ahead using the estimated parameters and errors returned by the **arimamt** procedure.

8.2 References

1. Granger, C.W.J. and Newbold, Paul. 1986. **Forecasting Economic Time Series. Second Edition**, San Diego: Academic Press.

2. Ansely, Craig F. 1979. "An Algorithm for the Exact Likelihood of a Mixed Autoregressive-Moving Average Process," *Biometrika* 66:59-65.

9 Autoregression

The **TSMT** module includes procedures for the computation of estimates for the autoregression model with autoregressive errors of any specified order and the computation of autocorrelations and autocovariances.

9.1 Autoregression Models

This program will compute estimates of the parameters and standard errors for a regression model with autoregressive errors. Thus, it can be used for models for which the Cochrane-Orcutt or similar procedure can be used. It is also similar to the SAS autoreg procedure except that this routine will compute the maximum likelihood estimates based upon the exact likelihood function.

The model estimated is of the general form:

$$y_t = x_t\beta + u_t$$
$$u_t - \phi_1 u_{t-1} - \ldots - \phi_p u_{t-p} = e_t$$

where it is assumed that e_t is a white noise error term, distributed as $N\left(0, \sigma^2\right)$.

The parameters to be estimated are thus: $\beta(\mathrm{K} \times 1)$, $\phi(\mathrm{L} \times 1)$ and σ^2 (a scalar). The order of the process is L.

In addition, this program will estimate the autocovariances and autocorrelations of the error term *u*. It produces initial estimates of these based upon the residuals of an OLS regression. Then it computes the maximum likelihood estimates of these based upon the maximum likelihood estimates of the other parameters.

9.2 References

1. Judge, George C., R. Carter Hill, William E. Griffiths, Helmut Lutkepohl, and Tsoung-Chao Lee. 1988. **Introduction to the Theory and Practice of Econometrics. Second Edition**, New York: Wiley.

10 Command Reference

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

acfmt

Purpose

Computes sample autocorrelations for a univariate time series.

Library

tsmt

Format

a = **acfmt**(*x*,*l*,*d*);

Input

x	Nx1 vector. The mean is subtracted automatically.

l	scalar, the maximum lags to compute.
d	scalar, the difference order.

Output

a	l×1 vector, sample autocorrelations.

Remarks

This function is similar to **autocor**; however, **acfmt** allows the users to compute the autocorrelations for the differenced data.

Source

tsutilmt.src

adjrsq

Purpose

Finds the adjusted R-Squared statistic following the estimation of a linear regression model. It requires both the original data and the residuals from the estimate as inputs.

Library

tsmt

Format

{*r_sq, adj_rsq*} = **adjrsq**(*yt,res,num_vars*);

Input

yt	T×M numerical matrix of panel series data.
res	T×M numerical matrix of estimation residuals.
num_vars	scalar, number of estimated coefficients (including constant) in the original regression.

Output

r_sq	M×1 matrix of standard R-Squared statistics.
adj_rsq	M×1 matrix of adjusted R-Squared statistics.

Remarks

The adjusted R-Squared provides a diagnostic tool for evaluating goodness-of-fit following linear regressions. The statistic adjusts the standard R-Squared to account for the addition of variables in the model. Because the addition of variables will always increase the R-Squared the adjusted R-Squared generally provides a truer measure the explanatory power of the model. The adjusted R-Squared is given by

$$R_{adj} = 1 - \frac{MSE}{MST}$$

where

$$MSE = \frac{1}{n-p-1}\sum_{i=1}^{n} e_i^2$$

and,

with p = number of estimated coefficients, including a constant.

Example

This example utilizes a simple multivariate linear model. To begin consider randomly generating an 5 x 500 matrix of independent data (X), and a 1 x 500 vector of dependent data (Y):

```
struct  olsmtControl oc0;
struct  olsmtOut oOut;
oc0 = olsmtControlCreate();
oc0.res = 1;
oOut = olsmt(oc0,0,y,x);
```

Finally, call the **adjrsq** function:

```
res = oOut.resid;
num_var = cols(x) + oc0.con;
{ r, adj_r }= adjrsq(y,res,num_var);
```

In calling this function we input the residuals from the **olsmt** output structure, *oOut.resid*. Because the **olsmt** function automatically includes a constant, by default the number of variables estimated in this regression is equal to the columns of the dependent variable plus one. The code above uses a more general method for setting the number of estimated variables using the **olsmt** control structure. The scalar oc0.con is an element of the olsmt control structure used to specify whether to include a constant. When no constant is included, *oc0.con* is set to 0, and when a constant is included, *oc0.con* is set to one.

Running all blocks of code above produces the following output:

```
Valid cases:       100    Dependent variable:         Y
Missing cases:       0    Deletion method:         None
Total SS:        7.658    Degrees of freedom:        94
R-squared:       0.046    Rbar-squared:          -0.005
Residual SS:     7.307    Std error of est:       0.279
F(5,94):         0.901    Probability of F:       0.484
Durbin-Watson:   2.083
Standard         Prob    Standardized Cor with
Var   Estimate  Error  t-value >|t|  Estimate Dep Var
-----------------------------------------------------
CONST  0.5232  0.0286  18.3244   0.000   ---        ---
X1     0.0399  0.0335   1.1907   0.237   0.1212   0.1069
X2    -0.0373  0.0285  -1.3075   0.194  -0.1323  -0.1300
X3    -0.0202  0.0303  -0.6647   0.508  -0.0671  -0.0727
X4    -0.0241  0.0275  -0.8764   0.383  -0.0885  -0.0948
X5     0.0138  0.0264   0.5241   0.601   0.0532   0.0434
The standard R squared is
     0.0457298
```

```
The adjusted R squared is
    -0.0158360
```

This example is more relevant when it utilizes the simple multivariate linear model generated by the following data generation process:

$$y_t = 0.4 + 4.75x_1 + 0.9x_2 + 3.2x_3 - 2.1x_4 - 2.9x_4 + \varepsilon_t$$

To begin consider randomly generating a 5x500 matrix of independent data (Xt) and the appropriate 1x500 vector of dependent data (Yt):

```
rndseed 89102;
xt = rndn(100,5);
yt = 0.4 + 4.75*xt[.,1] + 0.9*xt[.,2] + 3.2*xt[.,3]
    - 2.1*xt[.,4] - 2.9*xt[.,5] + rndn(100,1);

struct olsmtControl oc0_2;
struct olsmtOut oOut_2;
oc0_2 = olsmtControlCreate();
oc0_2.res = 1;
oOut_2 = olsmt(oc0_2,0,yt,xt);
res = oOut_2.resid;
num_var = cols(xt) + oc0_2.con;
{ r, adj_r }= adjRsq(yt,res,num_var);
```

This produces the following output:

```
Valid cases:         100      Dependent variable:        Y
Missing cases:         0      Deletion method:        None
Total SS:       3461.187      Degrees of freedom:       94
```

```
R-squared:          0.972      Rbar-squared:          0.971
Residual SS:       95.912      Std error of est:      1.010
F(5,94):          659.640      Probability of F:      0.000
Durbin-Watson:      2.093
Standard           Prob    Standardized Cor with
Var  Estimate Error    t-value  >|t|  Estimate  Dep Var
-------------------------------------------------------------
CONST  0.2996  0.1032    2.9036   0.005    ---        ---
X1     4.7128  0.1076   43.7811   0.000    0.7671   0.6528
X2     0.9561  0.1058    9.0379   0.000    0.1600   0.3434
X3     3.3507  0.1178   28.4434   0.000    0.5081   0.3188
X4    -2.0465  0.1078  -18.9913   0.000   -0.3302  -0.2412
X5    -2.8348  0.1055  -26.8741   0.000   -0.4814  -0.3634

The standard R squared is 0.972289

The adjusted R squared is 0.970502
```

aggData

Purpose

Estimates a p'th order threshold autoregression and tests the hypothesis of a linear autoregression, using the statistics described in "Inference when a nuisance parameter is not identified under the null hypothesis." (Hansen, 1996).

Library

tsmt

Format

```
x_new = aggData(xt,st_freq,end_freq,method);
```

Input

xt	Nxk matrix, data.
st_freq	string, starting frequency of data: "M" for monthly or "Q" for quarterly.
end_freq	string, ending frequency of data: "M" for monthly or "Q" for quarterly.
method	string, method of aggregation, "B" for beginning of period, "E" for end of period, "AVE" for moving average.

Output

x_new	matrix, aggregated data.

Source

aggregatedata.src

arimamt

Purpose

Estimates coefficients of a univariate time series model with autoregressive-moving average errors. Model may include fixed regressors.

Library

tsmt

Format

amo = **arimamt**(*amc*,*y*);

Input

amc	An instance of an *arimamtControl* structure. The following members of *amc* are referenced within this routine:	
	amc.ar	scalar, the autoregressive order. Default = 0.
	amc.const	scalar, if 1, a constant is estimated, 0 otherwise. N×1 matrix, fixed regressors. The number of rows in the fixed regressor matrix must equal the number of rows for *y* after

differencing. Default = 1.

amc.diff — scalar, the order of differencing. Default = 0.

amc.itol — 3×1 vector, controls the convergence criterion.

- **[1]** Maximum number of iterations. Default = 100.
- **[2]** Minimum percentage change in the sum of squared errors. Default = 1e-8.
- **[3]** Minimum percentage change in the parameter values. Default = 1e-6.

amc.ma — scalar, the moving average order. Default = 0.

amc.output — scalar, controls printing of output .

- **0** Nothing will be printed by **arimamt**.
- **1** Final results are printed.
- **2** Final results, iterations results, residual

		autocorrelations, Box-Ljung statistic, and covariance and correlation matrices are printed.
		Default = 1.
	amc.ranktol	scalar, the tolerance used in determining if any of the singular values are effectively zero when computing the rank of a matrix. Default = 1e-13.
	amc.start	scalar 0, then **arimamt** computes starting values K×1 vector, starting values. Default = 0.
	amc.varn	1×(M+1) vector of parameter names. This is used for models with fixed regressors. The first element contains the name of the independent variable; the second through M^{th} elements contain the variable names for the fixed regressors. If *amc.varn* = 0, the fixed regressors labeled as $X_0, X_1, \ldots, X_M$. Default = 0.
y	N×1 vector, data.	

Output

amo An instance of an *arimamtOut* structure containing the following members:

amo.aic	scalar, value of the Akaike information criterion.
amo.b	K×1 vector, estimated model coefficients.
amo.e	N×1 vector, residual from fitted model.
amo.ll	scalar, the value of the log likelihood function.
amo.sbc	scalar, value of the Schwartz Bayesian criterion.
amo.vcb	K×K matrix, the covariance matrix of estimated model coefficients.

Remarks

There are other members of the *arimamtControl* structure which are used by **arimamt**'s likelihood function but need not be set by the user. These are *amc.b*, *amc.y*, *amc.n*, *amc.e*, *amc.k*, *amc.m*, *amc.inter*.

arimamt forces the autoregressive coefficients to be invertible (in other words, the autoregressive roots have modulus greater than one). The moving average roots will have modulus one or greater. If a moving average root is one, **arimamt** reports a missing value for the moving average coefficient's standard deviation, t-statistic and p-value. This is because these values are meaningless when one of the moving average roots is equal to one. A moving average root equal to one suggests that the data may have been over-differenced.

Source

arimamt.src

arimamtControlCreate

Purpose

Sets the members of an instance of an *arimamtControl* structure to default values.

Library

tsmt

Format

amc = **arimamtControlCreate**();

Input

None

Output

amc	An instance of an *arimamtControl* structure with its members set to default values.

Remarks

Putting this instruction at the top of all programs that invoke **arimamt** is generally good practice. This will prevent control variables from being inappropriately defined when a program is run either several times or after another program that also calls **arimamt**.

Source

arimamt.src

autocormt

Purpose

Computes specified autocorrelations for each column of a matrix.

Library

tsmt

Format

```
a = autocormt(x,f,l);
```

Input

x	N×K matrix. Autocorrelations will be computed for each column separately. *x* is assumed to have 0 mean.
f	scalar, in range [0, **rows**(*x*) − 1], denoting the first autocorrelation to compute.
l	scalar, in range [0, **rows**(*x*) − 1], denoting the last autocorrelation to compute. It must be that $f \leq l$; if $l = 0$ and $f = 0$, then *l* is set to **rows**(*x*) − 1 and all autocorrelations from *f* to *l* are computed. If $l = 0$ and $f < 0$, then only the 0^{th} order autocorrelation is computed (this equals *x*′*x*).

Output

a	G×K matrix, where $G = l - f + 1$, containing the autocorrelations of order $f, f+1, \ldots, l$ for each of the columns of *x*. If the variance of any variable is 0, missings will be returned for that variable.

Remarks

The 0^{th} autocorrelation will always be 1.

The data are assumed to have 0 mean. Thus, use:

```
x = x - meanc(x)';
```

prior to the use of this function if the mean is not 0.

Source

autoregmt.src

autocovmt

Purpose

Computes specified autocovariances for each column of a matrix.

Library

tsmt

Format

a = **autocovmt**(*x*,*f*,*l*);

Input

x	N×K matrix. Autocovariances will be computed for each column separately. *x* is assumed to have 0 mean.
f	scalar, in range [0, **rows**$(x) - 1$], denoting the first autocovariance to compute.
l	scalar, in range [0, **rows**$(x) - 1$], denoting the last autocovariance to compute. It must be that $f \leq l$; if $l = 0$ and $f = 0$, then *l* is set to **rows**$(x) - 1$ and all autocovariances are computed. If $l = 0$ and $f < 0$, then only the 0^{th} order autocovariance is computed (this equals $x'x$).

Output

a	G×K matrix, where $G = l - f + 1$, containing the autocovariances of order $f, f + 1, \ldots, l$ for each of the columns of *x*.

Remarks

The 0^{th} autocovariance is just the variance of the variable. The divisor for each autocovariance is the number of elements involved in its computation.

Thus, the p^{th} order cross product is divided by $N - P$, where $N =$ **rows**(x),

to obtain the p^{th} order autocovariance.

The data are assumed to have 0 mean. Thus, use:

```
x = x - meanc(x)';
```

prior to the use of this function if the mean is not 0.

Source

autoregmt.src

automtControlCreate

Purpose

Sets the members of an instance of an *automtControl* structure to default values.

Library

tsmt

Format

arc = **automtControlCreate()**;

Input

None

Output

arc	An instance of an *automtControl* structure with its members set to default values.

Remarks

Putting this instruction at the top of all programs that invoke **autoreg** or **autoregmt** is generally good practice. This will prevent control variables from being inappropriately defined when a program is run either several times or after another program that also calls one of these procedures.

Source

autoregmt.src

autoregmt

Purpose

Estimates coefficients of a regression model with autoregressive errors of any specified order.

Library

tsmt

Format

```
aro = autoregmt(arc,d0,lagvars,order);
```

Input

arc	An instance of an *automtControl* structure. The following members of *arc* are referenced within this routine:	
	arc.const	scalar. If 1, constant will be used in model. Default = 1.
	arc.header	string, specifies the format for the output header. *arc.header* can contain zero or more of the following characters:
		t title is to be printed.
		l lines are to bracket the title.
		d a date and time is to be printed.
		v version number of program is to be printed.
		f file name of program is

	to be printed Example: `arc.header = "tld";` If *arc.header* = "", no header is printed. Default = "tldvf".
arc.init	scalar. If 1, only initial estimates will be computed. Default = 0.
arc.iter	scalar. If 0, iteration information will not be printed. If 1, iteration information will be printed (*arc.output* must be nonzero). Default = 0.
arc.maxvec	scalar, the maximum number of elements allowed in any one matrix. Default = 20000.
arc.output	scalar, if nonzero, results are printed to screen. Default = 1.
arc.title	string, a title to be printed at the top of the output header (see *arc.header*). By default, no title is printed (*arc.title*="").
arc.tol	scalar, convergence tolerance.

Default = 1e-5.

d0 1×1 or 2×1 instance of a *DS* data structure. If there are independent variables, d0 is 2×1. For the first instance in d0,

d0[1].datamatrix	if time series is stored in a matrix in memory, N×1 matrix, time series.
d0[1].dname	string, if time series is stored in a **GAUSS** dataset, name of dataset.
d0[1].vnames	string array, if time series is stored in a **GAUSS** dataset, column name of time series in the **GAUSS** dataset.

If there are independent variables in the model, then the second instance in *d0* includes:

d0[2].data	matrix, if independent variables are stored in a matrix in memory, N×K matrix, time series.
d0[2].dname	string, if the independent variables are stored in a **GAUSS** dataset, name of dataset.
d0[2].vnames	string array, if the independent variables are stored in a **GAUSS** dataset, column names of the independent variables in the

	GAUSS dataset.
lagvars	K×1 vector, the number of periods to lag the variables in *indvars*. If there are no lagged variables, set to scalar 0. The variables in *indvars* will be lagged the number of periods indicated in the corresponding entries in *lagvars*.
order	scalar, order of the autoregressive process; must be greater than 0 and less than the number of observations.

Output

aro An instance of an autoOut structure containing the following members:

aro.acor	(L+1)×1 vector, autocorrelations.
aro.acov	(L+1)×1 vector, autocovariances.
aro.chisq	scalar, -2 * log-likelihood.
aro.coefs	K×1 vector, estimated regression coefficients.
aro.phi	L×1 vector, lag coefficients.
aro.rsq	scalar, explained variance.

aro.sigsq	scalar, variance of white noise error.
aro.tobs	scalar, number of observations.
aro.vcb	K×K matrix, covariance matrix of estimated regression coefficients.
aro.vcphi	L×L matrix, covariance matrix of *phi*.
aro.vsig	scalar, variance of `aro.sigsq` (variance of the variance of white noise error).

Remarks

This program will handle only datasets that fit in memory.

All autoregressive parameters are estimated up to the specified lag. You cannot estimate only the first and fourth lags, for instance.

The algorithm will fail if the model is not stationary at the estimated parameters. Thus, in that sense it automatically tests for stationarity.

Source

autoregmt.src

breitung

Purpose

Panel series unit root testing. The z-statistics constructed from the mean t-statistic has an asymptotic standardized normal distribution and tests the null hypothesis that all series are I(1) against the alternative that all series are I(0)

Library

tsmt

Format

bstat = **breitung**(*y*,*trend*,*constant*,*demean*,*lags*);

Input

y	T×M matrix, data, M > 5.
trend	scalar, 0 = no trend, 1 = trend.
constant	if nonzero, constant included in model.
demean	scalar, 0 to specify no demeaning, or 1 to subtract cross-sectional means.
lags	scalar, number of lags.

Output

bstat	test statistic.

Remarks

The Breitung panel series unit root test utilizes the sample mean of the t-statistics across all individual series within a panel of time series variables. However, the procedure pre-adjusts data to address biased estimation.

It is assumed that the autoregressive parameter is constant across all panels. This allows the use of the standard t-statistic but requires that the panels be strongly balanced.

The procedure performs an individual ADF test on each series n then forms the sample mean of the t-statistics. The z-statistic constructed from the mean t-statistic has an asymptotic standardized normal distribution and tests the null hypothesis that all series are I(1) against the alternative that all series are I(0).

chowfcst

Purpose

Tests and dates likely structural breaks using out-of-sample forecasts.

Library

tsmt

Format

{ *cs*,*prob* } = **chowfcst**(*yt*,*res*,*window*);

Input

yt	Tx1 numerical vector of panel series data.
xt	TxK numerical matrix of estimation regressors.
window	scalar, a positive integer specifying a fixed window size of K<*window*<T. Coefficient estimates from this window will be used to forecast observations for the last T-*window* observations.

Output

cs	scalar, the Chow out-of-sample forecast F-stat.
prob	scalar, probability.

Remarks

The Chow (1960) test uses out-of-sample forecasts, given a coefficient estimates from a subset of the data sample. Under the null hypothesis of constant coefficients, the out-of-sample forecasts are expected to be unbiased, equivalently the forecast errors are expected to have zero mean.

Reference

Chow, G.C. (1960). Tests of equality between sets of coefficients in two linear regressions, *Econometrica, 52*, 211-22.

Source

chowfcst.src

cusum

Purpose

The Brown, Durbin, and Evans (1975) CUSUM test considers the empirical fluctuation process of the cumulative sums of standardized residuals. Under the null hypothesis of constant coefficients the residuals should have zero mean. Hence, significant deviation from zero at time, t, indicates possible structural change at time t. The test is valid for dynamic models.

Library

tsmt

Format

{ *cs*, *cst2* } = **cusum**(*yt*,*xt*,*gr*,*minwin*);

Input

yt	Tx1 numerical vector of panel series data.

xt	TxK numerical matrix of estimation regressors.
gr	scalar, 1 to graph the vector of CUSUM test statistics, including boundaries or 0 for no graph.
minwin	scalar, minimum regression window size.

Output

cst	Tx1 vector containing CUSUM test statistics.
cst2	Tx1 vector containing CUSUM sq test statistics.

Remarks

The Brown, Durbin, and Evans (1975) CUSUM test considers the empirical fluctuation process of the cumulative sums of standardized residuals. Under the null hypothesis of constant coefficients the residuals should have zero mean. Hence, significant deviation from zero at time, *t*, indicates possible structural change at time *t*. The test is valid for dynamic models.

Reference

Brown, R.L., Durbin, J., and Evans, J.M. (1975). Techniques for testing the constancy of regression relationships over time, *Journal of Royal Statistical Society, Series B, 35*, 149-192. .

Source

cusum.src

dfgls

Purpose

Test for unit root in univariate time series.

Library

tsmt

Format

{ *t_stat*,*z_crit* } = **dfgls**(*y*,*max_lags*,*trend*);

Input

y	N×1 vector, data.
max_lags	maximum number of lags to be tested.
trend	scalar, 0 = no trend, 1 = trend.

Output

tstat	t-statistic.

zcrit	Elliot, Rothenberg and Stock (1996) critical values for the GLS detrended unit root test at the 1%, 2.5%, 5%, and 10% significance level.

ecmmt

Purpose

Calculates and returns parameter estimates for an error correction model.

Library

tsmt

Format

vmo = **ecmmt**(*vmc*,*d0*);

Input

vmc An instance of a *varmamtControl* structure. The following members of vmc are referenced within this routine:

vmc.ar	scalar, order of AR process. Default = 0.
vmc.rho	scalar, number of cointegrating relations. Set to -1 to have **GAUSS** estimate this value.

Default = 0.

vmc.header string, specifies the format for the output header. *vmc.header* can contain zero or more of the following characters:

t	title is to be printed.
l	lines are to bracket the title.
d	a date and time is to be printed.
v	version number of program is to be printed.
f	file name being analyzed is to be printed.

Example:

```
vmc.header = "tld";
```

If *vmc.header* = "", no header is printed. Default = "tldvf".

vmc.IndEquations K×L matrix of zeros and ones. Used to set zero restrictions on the *x* variables to be

	estimated. Used only if the number of equations, *vmc.L*, is greater than one. Elements set to one indicate the coefficients to be estimated. If *vmc.L* = 1, all coefficients will be estimated. If *vmc.L* > 1 and *vmc.IndEquations* is set to a missing value (the default), all coefficients will be estimated.
vmc.lags	scalar, number of lags over which ACF and Diagnostics are calculated. Default = 12.
vmc.start	Instance of a *PV* structure containing starting values. See Section 3.7.2 for an example.
vmc.nodet	scalar. Set *vmc.nodet* = 1 to suppress the constant term from the fitted regression and include it in the co-integrating regression; otherwise, set *vmc.nodet* = 0. Default = 0.
vmc.nwtrunc	scalar, the number of autocorrelations to use in calculating the Newey-West correction. If *vmc.nwtrunc* = 0, **GAUSS** will use a truncation lag given by Newey and West, *vmc.nwtrunc* $= 4(T / 100)^{2/9}$.
vmc.ctl	An instance of an *sqpsolvemtControl* structure.

	vmc.ctl.covType	scalar, if 2, QML standard errors are computed, if 0, none; otherwise Wald-type.
	vmc.ctl.printIters	scalar, iteration information printed every *swc.ctl.printIters*-th iteration.
	See documentation for *sqpsolvemtControl* for further information regarding members of this structure.	
vmc.olsqtol	scalar, the tolerance used in determining if diagonal elements are approaching zero in **olsqrmt**. Default = 1e-14.	
vmc.output	scalar, if nonzero, results are printed to screen. Default = 1.	
vmc.row	scalar. Specifies how many rows of the dataset are to be read per iteration of the read loop. By default, the number of rows to be read is calculated by **ecmmt**.	
vmc.scale	scalar or an L×1 vector, scales for the time series. If scalar, all series are multiplied by the	

value. If an L×1 vector, each series is multiplied by the corresponding element of *vmc.scale*. Default = 4/standard deviation (found to be best by experimentation).

vmc.SetConstraints scalar, set to a nonzero value to impose stationarity and invertibility by constraining roots of the AR and MA characteristic equations to be outside the unit circle. Set to zero to estimate an unconstrained model. Default = 1.

vmc.title string, a title to be printed at the top of the output header (see *vmc.header*). By default, no title is printed (*vmc.title* = "").

d0 1×1 or 2×1 instance of a *DS* data structure. If there are independent variables, *d0* is 2×1. For the first instance in *d0*,

d0[1].datamatrix if time series is stored in a matrix in memory, N×1 matrix, time series.

d0[1].dname string, if time series is stored in a **GAUSS** dataset, name of dataset.

d0[1].vnames string array, if time series is stored in a **GAUSS** dataset, column name of time series in the **GAUSS** dataset.

If there are independent variables in the model, then the second

instance in *d0* includes:

d0[2].data	matrix, if independent variables are stored in a matrix in memory, N×K matrix, time series.
d0[2].dname	string, if the independent variables are stored in a **GAUSS** dataset, name of dataset.
d0[2].vnames	string array, if the independent variables are stored in a **GAUSS** dataset, column name of the independent variables in the **GAUSS** dataset.

Output

vmo An instance of a *varmamtOut* structure containing the following members:

vmo.aa	L×r matrix of coefficients, such that $aa * bb = \Pi$ (see remarks below).
vmo.acfm	L×(p*L) matrix, the autocorrelaton function. The first *L* columns are the lag *l* ACF; the last *L* columns are the lag *p* ACF.
vmo.aic	L×1 vector, the Akaike Information Criterion.
vmo.arroots	p×1 vector of AR roots, possibly complex.

vmo.bb	r×L matrix, eigenvectors spanning the cointegrating space of dimension *r*.
vmo.bic	L×1 vector, the Schwarz Bayesian Information Criterion.
vmo.covpar	Q×Q matrix of estimated parameters where Q is the number of estimated parameters. The parameters are in the row-major order: Π, $AR(1)$ to $AR(p)$, *beta* (if x variables were present in the estimation), and the constants.
vmo.fct	L×1 vector, the likelihood value.
vmo.lagr	An instance of an *sqpsolvemtLagrange* structure containing the following members:

vmo.lagr.lineq	linear equality constraints.
vmo.lagr.nlineq	nonlinear equality constraints.
vmo.lagr.linineq	linear inequality constraints.
vmo.lagr.nlinineq	nonlinear inequality constraints.
vmo.lagr.bounds	bounds.

When an inequality or bounds constraint is active, its associated Lagrangean is nonzero. The linear Lagrangeans precede the nonlinear Lagrangeans in the covariance matrices.

vmo.lrs L×1 vector, the likelihood ratio statistic.

vmo.maroots q×1 vector of MA roots, possibly complex.

vmo.pacfm L×p*L) matrix, the partial autocorrelation function, computed only if a univariate model is estimated. The first *L* columns are the lag *1* ACF; the last *L* columns are the lag *p* ACF.

vmo.par An instance of a *PV* structure containing the parameter estimates, which can be retrieved using **pvUnpack**. For example,

```
struct varmamtOut vout;
vout = varmaxmt(vmc,y,0);
ph = pvUnpack(vout.par, "zeta");
th = pvUnpack(vout.par, "pi");
vc = pvUnpack(vout.par, "vc");
```

The complete set of parameter matrices and arrays that can be unpacked depending on the model is:

phi L×p×p array, autoregression

		coefficients.
	theta	L×q×q array, moving average coefficients.
	vc	L×L residual covariance matrix.
	beta	L×K regression coefficient matrix.
	beta0	L×1 constant vector.
	zeta	L×p×ar array of ecm coefficients.
	pi	L×L matrix. *Note that 'pi' is a reserved word in **GAUSS**. Users will need to assign this to a different variable name.*
vmo.portman	*vmc.lags*-(p+q)×3 matrix of portmanteau statistics for the multivariate model and Ljung-Box statistics for the univariate model. The time period is in column one, the *Qs* (portmanteau) statistic in column two and the p-value in column three.	

vmo.residuals	T×L matrix, residuals.	
vmo.retcode	2×1 vector, return code. First element:	
	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.
	Second element:	
	0	covariance matrix of parameters failed.
	1	ML covariance matrix.

	2	QML covariance matrix.
	3	Cross-Product covariance matrix.
vmo.ss	L×2 matrix, the sum of squares for Y in column one and the sum of squared error in column two.	
vmo.va	r×1 vector, eigenvalues.	

Remarks

Errors are assumed to be distributed N(0,Q).

Source

varmamt.src

garch

Purpose

Estimates parameters of univariate time series.

Library

tsmt

Format

```
out1 = garch(f0,d0);
```

Input

f0	*garchControl* structure.	
	f0.p	scalar, order of the GARCH parameters.
	f0.q	scalar, order of the ARCH parameters.
	f0.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 - skew generalized t.
	f0.asymmetry	scalar, if nonzero assymetry terms are added.
	f0.inmean	scalar, GARCH-in-mean, square root of conditional variance is included in the mean equation.
	f0.unitRoot	scalar, if nonzero IGARCH model,

		parameters contain a unit root.
	f0.cvConstraintsType	scalar, type of enforcement of nonnegative conditional variances, 0 - direct constraints, 1 - Nelson & Cao constraints
	f0.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
d0	1×1 or 2×1 *DS* structure, data.	
	d0[1].endoVector	N×1 vector, time series..
	d0[1].exoMatrix	N×k matrix, independent variables (optional).

Output

out1	*garchEstimation* structure.	
	aic	scalar, Akiake criterion.
	bic	scalar, Bayesian information criterion.

lrs	scalar, likelihood ratio statistic.
numbObs	scalar, number of observations.
df	scalar, degrees of freedom.
par	instance of *PV* structure containing parameter estimates.
retcode	scalar, return code.
	1 normal convergence.
	1 normal convergence.
	2 forced exit.
	3 function calculation failed.
	4 gradient calculation failed.
	5 Hessian claculation failed.
	6 line search failed.
	7 error with constraints.
	8 function complex.
moment	K×K matrix, moment matrix of parameter estimates.
climits	K×2 matrix, confidence limits.

garchm

Purpose

Estimates parameters of GARCH-in-mean model, i.e., where the square root of the conditional variance is added to the mean equation.

Library

tsmt

Format

out1 = **garchm**(*f0*,*d0*);

Input

f0	*garchControl* structure.	
	f0.p	scalar, order of the GARCH parameters.
	f0.q	scalar, order of the ARCH parameters.
	f0.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 - skew generalized t.

	f0.asymmetry	scalar, if nonzero assymetry terms are added.
	f0.inmean	scalar, GARCH-in-mean, square root of conditional variance is included in the mean equation.
	f0.unitRoot	scalar, if nonzero IGARCH model, marameters contain a unit root.
	f0.cvConstraintsType	scalar, type of enforcement of nonnegative conditional variances, 0 - direct constraints, 1 - Nelson & Cao constraints.
	f0.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
d0	1×1 or 2×1 *DS* structure, data.	
	d0[1].endoVector	N×1 vector, time series.
	d0[1].exoMatrix	N×k matrix, independent

variables (optional).

Output

out1 *garchEstimation* structure.

aic	scalar, Akiake criterion.	
bic	scalar, Bayesian information criterion.	
lrs	scalar, likelihood ratio statistic.	
numbObs	scalar, number of observations.	
df	scalar, degrees of freedom.	
par	instance of *PV* structure containing parameter estimates.	
retcode	scalar, return code.	
	1	normal convergence.
	1	normal convergence.
	2	forced exit.
	3	function calculation failed.
	4	gradient calculation failed.

	5	Hessian claculation failed.
	6	line search failed.
	7	error with constraints.
	8	function complex.
moment	K×K matrix, moment matrix of parameter estimates.	
climits	K×2 matrix, confidence limits.	

getlrv

Purpose

Estimate long-run variance following user-selected kernel.

Library

tsmt

Format

{ *LRV*, *bw* } = **getlrv**(*y*, *kernel*, *lag_method*, *model*);

Input

y	N×1 vector, data.
kernel	string, "Bartlet," "Parzen," "Quad."
lag_method	string, "LLC," "NW."
model	scalar, -1 = no constant nor trend; 0 = constant; 1 = constant and trend.

Output

LRV	long run variance.
bw	selected bandwidth.

gjrgarch

Purpose

Estimates parameters of GJR GARCH model, i.e., one with asymmetry parameters (Glosten, L.R., Jagannathan, R., and Runkle, D.E., 1993).

Library

tsmt

Format

```
out1 = GJRGarch(f0,d0);
```

Input

f0	*garchControl* structure.	
	f0.p	scalar, order of the GARCH parameters.
	f0.q	scalar, order of the ARCH parameters.
	f0.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 - skew generalized t.
	f0.inmean	scalar, GARCH-in-mean, square root of conditional variance is included in the mean equation.
	f0.unitRoot	scalar, if nonzero IGARCH model, marameters contain a unit root.
	f0.cvConstraintsType	scalar, type of

		enforcement of nonnegative conditional variances, 0 - direct constraints, 1 - Nelson & Cao constraints.
	f0.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
d0	1×1 or 2×1 *DS* structure, data.	
	d0[1].endoVector	N×1 vector, time series.
	d0[1].exoMatrix	N×k matrix, independent variables (optional).

Output

out1	*garchEstimation* structure.	
	aic	scalar, Akiake criterion.
	bic	scalar, Bayesian information criterion.
	lrs	scalar, likelihood ratio statistic.
	numbObs	scalar, number of observations.

df	scalar, degrees of freedom.
par	instance of *PV* structure containing parameter estimates.
retcode	scalar, return code.
	1 normal convergence.
	1 normal convergence.
	2 forced exit.
	3 function calculation failed.
	4 gradient calculation failed.
	5 Hessian claculation failed.
	6 line search failed.
	7 error with constraints.
	8 function complex.
moment	K×K matrix, moment matrix of parameter estimates.
climits	K×2 matrix, confidence limits.

hansen

Purpose

Test for stability of all parameters using a cumulative sums of weighted full sample residuals. The test employs the locally best invariant tests in a Lagrange multiplier format.

Library

tsmt

Format

{ *ny*,*prob* } = **hansen**(*yt*,*xt*);

Input

yt	Tx1 numerical vector of panel series data.
xt	TxK numerical matrix of estimation regressors.

Output

ny	scalar, the Hansen-Nyblom test statistic.
crit	vector, 1%, 2.5%, 5%, 7.5%, 10%, and 20% critical values

Reference

1. Nyblom, J. (1989). Testing for the constancy of parameters over time, *Journal of American Statistical Association, 84(405)*, 223-230.
2. Hansen, B.E. (1992). Testing for parameter instability in linear models, *Journal of Policy Modeling*, 14(4): 517-533.

Source

hansen.src

igarch

Purpose

Estimates integrated GARCH model, i.e., a model containing a unit root.

Library

tsmt

Format

out1 = **igarch**(*f0*,*d0*);

Input

f0	*garchControl* structure.

f0.p	scalar, order of the GARCH parameters.
f0.q	scalar, order of the ARCH parameters.
f0.density	scalar, density of error term, 0 - Normal, 1 - Student's t, 3 - skew generalized t.
f0.asymmetry	scalar, if nonzero assymetry terms are added.
f0.inmean	scalar, GARCH-in-mean, square root of conditional variance is included in the mean equation.
f0.stConstraintsType	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.
	f0.cvConstraintsType	scalar, type of enforcement of nonnegative conditional variances, 0 - direct constraints, 1 - Nelson & Cao constraints.
	f0.covType	scalar, type of covariance matrix of parameters, 1 - ML, 2 - QML, 3 - none.
d0	1×1 or 2×1 *DS* structure, data.	
	d0[1].endoVector	N×1 vector, time series..
	d0[1].exoMatrix	N×k matrix, independent variables (optional).

Output

out1	*garchEstimation* structure.	
	aic	scalar, Akiake criterion.
	bic	scalar, Bayesian information criterion.
	lrs	scalar, likelihood ratio statistic.

numbObs	scalar, number of observations.
df	scalar, degrees of freedom.
par	instance of *PV* structure containing parameter estimates.
retcode	scalar, return code.
	1 normal convergence.
	1 normal convergence.
	2 forced exit.
	3 function calculation failed.
	4 gradient calculation failed.
	5 Hessian claculation failed.
	6 line search failed.
	7 error with constraints.
	8 function complex.
moment	K×K matrix, moment matrix of parameter estimates.
climits	K×2 matrix, confidence limits.

ips

Purpose

Panel series unit root testing. This test uses the sample mean of the t-statistics across all individual series within a panel of time series variables.

Library

tsmt

Format

{ *zstat*, *pcrit* } = **ips**(*y*,*trend*,*demean*,*lags*, *print_out*);

Input

y	N×k matrix, data, k >= 5.
trend	scalar, 0 =no trend, 1 = trend.
demean	if nonzero, means removed.
lags	scalar or k×1 vector, lags.
print_out	if nonzero, intermediate quantities printed to screen.

Output

zstat	test statistic.
pcrit	critical values for unit root testing at the 1%, 2.5%, 5%, and 10% significance level.

kpss

Purpose

Test for stationarity using a Lagrange Multiplier score statistic.

Library

tsmt

Format

{ *tstat*, *crit* } = **kpss**(*y*, *max_lags*, *trend*, *qsk*, *auto*, *print_out*);

Input

y	N×1 vector, data.
max_lags	if max_lags <= 0, maximum lag set using Schwert criterion; if -1, Schwert criterion = 12; if 0, Schwert

	criterion = 4; else if max_lags > 0, maximum lag = max_lags.
trend	scalar, 0 no trend, 1 trend.
qsk	if nonzero, quadratic spectral kernel is used.
auto	if nonzero, automatic *max_lags* computed.
print_out	if nonzero, intermediate quantities printed to the screen.

Output

tstat	test statistic for each lag.
crit	critical values for stationary test at the 1%, 2.5%, 5%, and 10% significance level.

llc

Purpose

Panel series unit root testing. The Levin-Lin-Chu panel series unit root test assumes a homogenous autoregressive parameter and independently distributed error terms across all series.

Library

tsmt

Format

tstat = **llc**(*y*,*trend*,*constant*,*demean*,*lags*,*kernel*,
lag_meth,*print_out*);

Input

y	N×k matrix, data, k > 5.
trend	scalar, 0 - no trend, 1 - trend.
constant	if nonzero, constant included in model.
demean	if nonzero, means removed.
lags	scalar, number of lags.
kernel	string, "Bartlet," "Parzen," "Quad."
lag_meth	string, "LLC," "NW."
print_out	if nonzero, intermediate quantities printed to the screen.

Output

tstat	test statistic.

lsdvmt

Purpose

Estimates the parameters of the least squares dummy variable model with bias correction.

Library

tsmt

Format

out = **lsdvmt**(*lsc*,*d0*,*series_len*,*num_lags*);

Input

lsc	An instance of a *lsdvmtControl* structure. The following members of *lsc* are referenced within this routine:	
	lsc.Constrain	scalar, if nonzero constraints will be applied to the autoregression coefficients. Default = 1.
	lsc.scale	scalar, if nonzero, data are scaled.
	lsc.output	scalar, determines the output to be printed.
	lsc.title	string, title to be printed at top of

header.

d0 1×1 or 2×1 instance of a *DS* data structure. If there are independent variables, *d0* is 2×1. For the first instance in *d0*:

d0[1].datamatrix	if time series is stored in a matrix in memory, N×1 matrix, time series.
d0[1].dname	string, if time series is stored in a **GAUSS** dataset, name of dataset.
d0[1].vnames	string array, if time series is stored in a **GAUSS** dataset, column name of time series in the **GAUSS** dataset.

If there are independent variables in the model, then the second instance in *d0* includes:

d0[2].data	matrix, if independent variables are stored in a matrix in memory, N×K matrix, time series.
d0[2].dname	string, if the independent variables are stored in a **GAUSS** dataset, name of dataset.
d0[2].vnames	string array, if the independent variables are stored in a **GAUSS** dataset, column name of the independent variables in the

	GAUSS dataset.
series_len	scalar, length of time series, that is, the number of time periods in the data.
num_lags	scalar, number of lagged dependent variables on the right-hand side.

Output

out	An instance of an *lsdvOut* structure. The following members of *out* are referenced within this routine:

AutoCoefficients	J×1 vector, uncorrected autoregression coefficients.
RegCoefficients	K×1 vector, uncorrected regression coefficients.
AutoCoefficientsCorr	J×1 vector, bias corrected autoregression coefficients.
RegCoefficientsCorr	K×1 vector, bias corrected regression coefficients.
AutoStderrs	J×1 vector, autoregression coefficient standard errors.

RegStderrs	K×1 vector, regression coefficient standard errors.
CovPar	K×K matrix, covariance matrix of parameters
SSresidual	scalar, residual sums of squares.
SStotal	scalar, total sums of squares.
SSexplained	scalar, explained sums of squares.
SSpooledResidual	scalar, pooled residual sums of squares.
biasCorr	K+J×1 vector, bias corrections.
Lagrange	J×2 matrix, Lagrangeans for constraints.
numCases	scalar, number of cases.
numMissing	scalar, number of observations with missing data.
numDF	scalar, number of degrees of freedom.

numObservations	scalar, number of observations.
numParameters	scalar, number of parameters.
numPeriods	scalar, number of periods in time series.

Remarks

The data must be contained in a **GAUSS** dataset cross-sectional unit by cross-sectional unit, with one variable containing an index for the units. From each cross-sectional unit all observations must be grouped together. For example, for the first cross-sectional unit there may be 10 rows in the dataset, for the second cross-sectional unit there may be another 10 rows, and so on. Each row in the dataset contains measurements on the endogenous and exogenous variables measured for each observation along with the index identifying the cross-sectional unit.

The index variable must be a series of integers. While all observations for each cross-sectional unit must be grouped together, they do not have to be sorted according to the index.

Example

The following example is taken from the program `lsdvmt.e`, located in the `examples` subdirectory. The program uses the sample data in `jdatamt.dat`.

```
library tsmt;

struct lsdvmtControl lsc;
struct lsdvmtOut out;
lsc = lsdvmtControlCreate();

z = loadd("lsdvmt");
out = lsdvmt(tsc,z[.,1],z[.,2:4],5,1);
```

Source

lsdvmt.src

lsdvmtControlCreate

Purpose

Sets the members of an instance of an *lsdvmtControl* structure to default values.

Library

tsmt

Format

lsc = **lsdvmtControlCreate();**

Input

None

Output

lsc	An instance of an *lsdvmtControl* structure with its members set to default values.

Remarks

Putting this instruction at the top of all programs that invoke **lsdvmt** is generally good practice. This will prevent control variables from being inappropriately defined when a program is run either several times or after another program that also calls **lsdvmt**.

Source

lsdvmt.src

macfmt

Purpose

Finds an autocorrelation function matrix for multiple dependent variables.

Library

tsmt

Format

x = **macfmt**(*y*,*lag*);

Input

y	T×L matrix of data.
lag	scalar, the lag for which an autocorrelation matrix is desired. Specify 0 to obtain the initial correlation.

Output

x	L×L matrix of autocorrelations, *res* and *res*(-lag).

Source

varmamt.src

mosum

Purpose

Tests and dates likely structural breaks using a moving window sums of recursive residuals. Estimation procedure is user specified and dynamic models are permitted.

Library

tsmt

Format

ms = **mosum**(*yt*,*xt*,*h*,*gr*,*level*);

Input

yt	Tx1 numerical vector of panel series data.
xt	TxK numerical matrix of estimation regressors.
h	scalar, a 0<*h*<1, bandwidth parameter for determining moving window size..
gr	scalar, any number greater than zero will graph the vector of MOSUM test statistics, including boundaries.
level	scalar, confidence level for creating statistic boundaries.

Output

ms	vector containing T-K MOSUM test statistics.

Remarks

Consider the a linear model such that

$$y_t = x_t'\beta + u_t,$$
$$u_t \sim \left(0, \sigma^2\right),$$
$$t = 1, \ldots, n$$

The MOSUM test utilizes the standardized one-step ahead recursive estimation residuals. The estimation residual are given by

$$w_t = \frac{y_t - \widehat{B}''_{t-1} x_y}{\sqrt{f_t}}$$

where

$$f_t = \widehat{\sigma}^2 \left[1 + x_t' \left(x_t' x_t \right)^{-1} x_t \right]$$

The resulting MOSUM test statistic utilizes the cumulative sum of these residuals such that

$$MOSUM_t = \sum_{i=k+\left\lfloor N_\eta t \right\rfloor + 1}^{k+\left\lfloor N_\eta t \right\rfloor + \lfloor \eta h \rfloor} \frac{\widehat{w}_j}{\widehat{\sigma}_w}$$

where

$$N_\eta = \frac{n - \lfloor \eta h \rfloor}{(1-h)}$$

and

$$\hat{\sigma}_w^2 = \frac{1}{n-k}\sum_{t=1}^{n}(w_t - \overline{w})^2$$

numCombReplace

Purpose

Calculates the number of possible permutations of n number of items chosen k times, with replacement.

Library

tsmt

Format

y = **numCombReplace**(*n*,*k*);

Input

n	number of items.
k	number of times items are chosen.

Output

y	number of possible combinations allowing for

replacement.

Example

To find the number of permutations of 10 items, chosen 3 times, allowing for replacement we call the **GAUSS** procedure **numCombReplace**

```
numCombReplace(10,3);
```

The resulting output reads

```
        220.0000
```

Source

permutate.src

numPerm

Purpose

Calculates the number of possible permutations of n number of items chosen k times, without replacement.

Library

tsmt

Format

num = **numPerm**(*n*,*k*);

Input

n	number of items.
k	number of times items are chosen.

Output

num	number of possible permutations without replacement.

Example

To find the number of permutations of 10 items, chosen 3 times, withut replacement we call the **GAUSS** procedure **numPerm**

```
numperm(10,3);
```

The resulting output reads

```
        720.0000
```

Source

permutate.src

numPermReplace

Purpose

Calculates the number of possible permutations of n number of items chosen k times with replacement.

Library

tsmt

Format

num = **numPermReplace**(*n*,*k*);

Input

n	number of items.
k	number of times items are chosen.

Output

num	number of possible permutations allowing for replacement.

Example

To find the number of permutations of 10 items, chosen 3 times, allowing for

replacement we call the **GAUSS** procedure **numPermReplace**

```
numPermReplace(10,3);
```

The resulting output reads

```
        1000.0000
```

Source

permutate.src

nwmt

Purpose

Finds the Newey-West covariance matrix.

Library

tsmt

Format

x = **nwmt**(*covb*,*resid*,*nwtrunc*);

Input

covb	Q×Q matrix, covariance matrix for the AR parameters.

resid	T×L matrix of residuals.
nwtrunc	scalar, the number of autocorrelations to use in calculating the Newey-West correction (q in the Remarks section below). If **nwtrunc** = 0, **GAUSS** will use a truncation lag given by Newey and West, $q = 4(T/100)^{2/9}$.

Output

x	Q×Q matrix, Newey-West adjusted covariances.

Remarks

The Newey-West correction is used to account for the effect of heteroskedasticity and residual serial correlation on estimated parameter standard errors. The adjusted parameter covariance matrix is $(X'X)^{-1}\Omega(X'X)^{-1}$ where.

$$\Omega = \sum_{t=1}^{T} \varepsilon_t^2 x_t x'_t + \sum_{j=1}^{q} \left[1 - \frac{j}{q+1}\right] \sum_{t=j+1}^{T} (x_t \varepsilon_t \varepsilon_{t-j} x'_{t-j} + x_{t-j} \varepsilon_{t-j} \varepsilon_t x'_t) \quad (11)$$

Source

varmamt.src

nyblom

Purpose

Tests for stability of all parameters using a cumulative sums of weighted full sample residuals. The test employs the locally best invariant tests in a Lagrange multiplier format.

Library

tsmt

Format

{ *ny*,*crit* } = **nyblom**(*yt*,*xt*);

Input

yt	Tx1 numerical vector of panel series data.
xt	TxK numerical matrix of estimation regressors.

Output

ny	scalar, the Nyblom test statistic.
crit	vector, 1%, 2.5%, 5%, 7.5%, 10%, and 20% critical values.

Reference

1. Nyblom, J. (1989). Testing for the constancy of parameters over time, *Journal of American Statistical Association*, 84(405), 223-230.
2. Hansen, B.E. (1992). Testing for parameter instability in linear models, *Journal of Policy Modeling*, 14(4): 517-533.

Source

hansen.src

pacmt

Purpose

Computes partial autocorrelations for a univariate time series.

Library

tsmt

Format

a = **pacfmt**(*y*, *l*, *d*);

Input

y	N×1 vector, data.
l	scalar, number of partial correlations to compute.

d	scalar, order of differencing.

Output

a	1×1 vector, partial autocorrelations.

Source

tsutilmt.src

paramConfigmt

Purpose

Returns parameter estimates from **ecmmt** and **varmaxmt**.

Library

tsmt

Format

coeffs = **paramConfigmt**(*p*,*q*,*coeffs*,*se*,*x*,*ecmflag*);

Input

p	scalar, order of the AR process.

q	scalar, order of the MA process.
coeffs	L*(p+q+K+1)×1 vector of coefficient estimates in the order AR, MA, x, constant.
se	L*(p+q+K+1)×1 vector of standard error estimates in the order AR, MA, x, constant.
x	T×K matrix of explanatory variables.
ecmflag	scalar, equals one if an ECM model was estimated, zero otherwise.

Output

coeffs	Compact matrix created using **vput**. Read it using **vread**. It contains:
	pi — L×L matrix, the impact matrix. Only returned if an ECM model was estimated. *Note that* **pi** *is a reserved word in* ***GAUSS****. Users will need to assign this to a different variable name.*
	pi_se — L×L matrix of impact coefficient standard errors. Only returned if an ECM model was estimated.
	phi — p*(L×L) matrix of AR coefficient estimates stacked in the order

	$AR(1), \ldots, AR(p)$.
phi_se	p*(L×L) matrix of AR standard errors stacked in the order $AR(1), \ldots, AR(p)$.
theta	q*(L×L) matrix of MA coefficient estimates stacked in the order $MA(1), \ldots, MA(q)$. Only returned if a **varmaxmt** model was estimated.
theta_se	q*(L×L) matrix of MA standard errors stacked in the order $MA(1), \ldots, MA(q)$. Only returned if a **varmaxmt** model was estimated.
beta	L×K matrix of *x* coefficient estimates. Only returned if a **varmaxmt** model was estimated.
beta_se	L×K matrix of *x* coefficient standard errors. Only returned if a **varmaxmt** model was estimated.

Source

varmamt.src

permReplace

Purpose

Lists all possible permutations with replacement for n number of items, chosen k times.

Library

tsmt

Format

y = **permReplace**(*n*,*k*);

Input

n	number of items.
k	number of times items are chosen.

Output

y	a matrix listing all possible permutations.

Example

To list all permutations with replacement for 3 items, chosen 2 times, we call

the **GAUSS** procedure **permReplace**

```
permReplace(3,2);
```

The resulting output reads

```
      1.0000000       1.0000000
      1.0000000       2.0000000
      1.0000000       3.0000000
      2.0000000       1.0000000
      2.0000000       2.0000000
      2.0000000       3.0000000
      3.0000000       1.0000000
      3.0000000       2.0000000
      3.0000000       3.0000000
```

Source

permReplace.src

permutate

Purpose

Lists all possible permutations without replacement for n number of items, chosen k times.

Library

tsmt

Format

y = **permutate**(*n*, *k*);

Input

n	number of items.
k	number of times items are chosen.

Output

y	a matrix listing all possible permutations.

Source

permutate.src

rolling

Purpose

Performs rolling OLS regressions for a provided vector of dependent data and matrix of independent regressors.

Library

tsmt

Format

{ *coef*,*res* } = **rolling**(*yt*,*xt*,*window*,*add*,*graph*);

Input

yt	Tx1 numerical vector of panel series data.
xt	TxK numerical matrix of estimation regressors.
window	scalar, a positive integer specifying a fixed window size of K<*window*<T. A window size of less than zero results in an expanding window.
add	scalar, specifying the initial observation for the forward expanding *window*. Negative values indicate a backward window expansion, beginning with the last *add* number of observations. The *add* input is valid only if a negative *window* size is provided.
graph	scalar, any number greater than zero will graph the rolling values of all coefficient estimates, including the constant.

Output

coef	matrix, rolling coefficient estimates.
res	matrix, one-step ahead rolling residuals.

Remarks

The **GAUSS** **rolling** procedure performs rolling OLS regressions for a provided vector of dependent data and matrix of independent regressors.

Reference

Zivot, E., and Wang, J. (2002). **Modeling Financial Time Series with S-PLUS.** Springer-Verlag, New York.

Source

rolling.src

sbControlCreate

Purpose

Sets the members of a declared structural break control structure to default values.

Library

tsmt

Format

sbc0 = **sbControlCreate**();

Input

None

Output

sbc0	An instance of a *sbControlCreate* structure with all members set to default values.

Source

sbcontrolcreate.src

sbreak

Purpose

Estimates the p'th order threshold autoregression model.

Library

tsmt

Format

sbout = **sbreak**(*yt*,*xt*,*zt*,*sbc0*);

Input

yt	Nx1 vector, data.
xt	autoregressive order of the TAR model.
zt	scalar or vector, lags (below p) to omit from autoregression [0 implies an AR(p)].
sbc0	sbControl structure containing the following elements:
omit	scalar or vector, lags (below p) to omit from autoregression [0 implies an AR(p)].
lowerQuantile	scalar, the lower quantile for trimming data.
upperQuantile	scalar, the upper quantile for trimming data.
printOutput	scalar, 0 or 1, 1 indicates screen output.
graph	scalar, 0 or 1, 1 indicates screen output.
dstart	scalar, starting date in DT scalar format.
freq	frequency of the data per year. Valid options include:.

1	yearly.
4	quarterly.
12	yearly.

Output

sbOutput	sbOutput structure containing the following elements:
breakDate	matrix, MxM of date breaks estimated for possible number of breaks less than or equal to m.
breakSSR	MxM, vector of ssr associated with all number of breaks less than or equal to m.

References

Bai, J and Perron, P. (2003) Computation and analysis of multiple structural change models, *Journal of Applied Econometrics, 18(1)*, 1-22.

Source

sb.src

selectLags

Purpose

Select lags based on method of statistical inference.

Library

tsmt

Format

{ *opt*,*crit_mat* } = **selectLags**(*y*,*max_lag*,*method*);

Input

y	N×1 vector, data.
max_lag	scalar, maximum lags.
method	string, "AIC," "BIC," "HQC," "LRS," "FPE."

Output

opt	scalar, optimum lags.
crit_mat	vector, critical values for each lag.

simarmamt

Purpose

Simulate ARMA time series process.

Library

tsmt

Format

y = **simarmamt**(*b*,*p*,*q*,*const*,*trend*,*n*,*k*,*std*,*seed*);

Input

b	K×1 vector, coefficient values for theoretical ARMA process.
p	scalar, the autoregressive order.
q	scalar, the moving average order.
const	scalar, value of the constant term N×M matrix, fixed regressor matrix.
trend	scalar, trend parameter or 0 for no trend.
n	scalar, the number of observations to generate.
k	scalar, the number of replications to generate.

std	scalar, the standard deviation of the error process.
seed	scalar, the value of the seed. If *seed* = 0, then **rndn** is used, otherwise **rndns** is used.

Output

y	N×K matrix, simulated ARMA process. Each column represents an independent realization of a univariate time series.

Remarks

simarmamt only simulates times series which are generated by normally distributed errors.

If your simulation is large or if your available memory is limited, make several calls to **simarmamt** during a simulation. Keep in mind that there is some overhead computing the starting values with the desired multivariate distribution.

If the process you are simulating lies on or near a boundary, try generating a longer time series, then trim the beginning observations. In general, **simarmamt** should give reasonable results since the starting values are normalized to have required multivariate normal distribution.

Source

simarmamt.src

stackData

Purpose

Stacks columns of panel series into a single stacked vector of data.

Library

tsmt

Format

y_st = **stackData**(*yt*);

Input

yt	TxK matrix of cross sectional data.

Output

y_st	(T*K)x1 Matrix of stacked cross sectional data, i.e. yt [.,1] | yt[.,2] | yt[.,3] | ... .| yt[.,K].

Source

stackdata.src

standardizeData

Purpose

Demeans and standardizes data matrix with standard deviation.

Library

tsmt

Format

y_std = **standardizeData**(*yt, demean*);

Input

yt	Tx1 vector of time series data.
demean	scalar, indicator variable to demean data [1 to demean data].

Output

y_std	Tx1 numerical vector of standardized time series data.

Source

standardizedata.src

starTest

Purpose

Estimates a p'th order threshold autoregression and tests the hypothesis of a linear autoregression, using the statistics described in "Inference when a nuisance parameter is not identified under the null hypothesis." (Hansen, 1996).

Library

tsmt

Format

{ *s3*,*p3* } = **starTest**(*yt*,*p*,*omit*);

Input

yt	Nx1 vector, data.
p	autoregressive order of the TAR model.
omit	scalar or vector, lags (below p) to omit from autoregression [0 implies an AR(p)].

Output

s3	scalar, value of the LM test statistic.

p3	scalar, p-value of s3.

References

1. Hansen, B.E. (1996). Inference when a nuisance parameter is not identified under the null hypothesis, *Econometrica*, 64(2), 413-430.
2. Franses, P.H. and Dijk, D. (2000) **Non-linear Time Series Models in Empirical Finance**. Cambridge University Press, New York.

Source

startest.src

svarmaxmt

Purpose

Computes exact maximum likelihood parameter estimates for a VARMAX model with seasonal parameters.

Library

tsmt

Format

vmo = **svarmaxmt**(*vmc*,*d0*);

Input

vmc An instance of a *varmamtControl* structure. The following members of *vmc* are referenced within this routine:

vmc.adforder	scalar, number of AR lags in the ADF test statistic. Default = 1.
vmc.ar	scalar, number of AR matrices to be estimated. Default = 0.
vmc.ma	scalar, number of MA matrices to be estimated. Default = 0.
vmc.diff	scalar, order of differencing to achieve stationarity. Default = 0.
vmc.sar	scalar, number of seasonal AR matrices to be estimated. Default = 0.
vmc.sma	scalar, number of seasonal MA matrices to be estimated. Default = 0.
vmc.sdiff	scalar, order of seasonal differencing to achieve stationarity. Default = 0.
vmc.s	scalar, seasonal period. Default = 0.
vmc.ctl	instance of *sqpsolvemtControl* structure.
	vmc.ctl.covType scalar, if 2, QML standard errors are

		computed, if 0, none; otherwise Wald-type.
	vmc.ctl.printIters	scalar, iteration information printed every *swc.ctl.printIters*-th iteration.
	See documentation for *sqpsolvemtControl* for further information regarding members of this structure.	
vmc.critl	scalar, the significance levels defining p-values. Default = .95.	
vmc.header	string, specifies the format for the output header. *vmc.header* can contain zero or more of the following characters:	
	t	title is to be printed.
	l	lines are to bracket the title.
	d	a date and time is to be printed.
	v	version number of program is to be

		printed.
	f	file name being analyzed is to be printed

Example:

```
vmc.header = "tld";
```

If *vmc.header* = "", no header is printed. Default = "tldvf".

vmc.IndEquations	K×L matrix of zeros and ones. Used to set zero restrictions on the *x* variables to be estimated. Only used if the number of equations, *vmc.L* is greater than one. Elements set to indicate the coefficients to be estimated. If *vmc.L* = 1, all coefficients will be estimated. If *vmc.L* > 1 and *vmc.IndEquations* is set to a missing value (the default), all coefficients will be estimated.
vmc.lags	scalar, number of lags over which ACF and Diagnostics are calculated. Default = 12.
vmc.nodet	scalar. Set *vmc.nodet* = 1 to suppress the constant term from the fitted regression and include it in the co-integrating regression;

	otherwise, set *vmc.nodet* = 0. Default = 0.
vmc.nwtrunc	scalar, the number of autocorrelations to use in calculating the Newey-West correction. If *vmc.nwtrunc* = 0, **GAUSS** will use a truncation lag given by Newey and West, *vmc.nwtrunc* $= 4(T / 100)^{2/9}$.
vmc.output	scalar. Set to 0 to suppress all printing from **varmaxmt**. Set *vmc.output* > 0 to print results. Default = 1.
vmc.scale	scalar or an L×1 vector, scales for the time series. If scalar, all series are multiplied by the value. If an L×1 vector, each series is multiplied by the corresponding element of *vmc.scale*. Default = 4/standard deviation (found to be best by experimentation).
vmc.SetConstraints	scalar, set to a nonzero value to impose stationarity and invertibility by constraining roots of the AR and MA characteristic equations to be outside the unit circle. Set to zero to estimate an unconstrained model. Default = 1.
vmc.Start	Instance of a *PV* structure containing starting values. See Section 3.7.2 for discussion of setting starting values. By default, **varmaxmt**

calculates starting values.

vmc.title string, a title to be printed at the top of the output header (see *vmc.header*). By default, no title is printed (*vmc.title* = "").

d0 1×1 or 2×1 instance of a *DS* data structure. If there are independent variables, d0 is 2×1. For the first instance in d0:

d0[1].datamatrix if time series is stored in a matrix in memory, N×1 matrix, time series.

d0[1].dname string, if time series is stored in a **GAUSS** dataset, name of dataset.

d0[1].vnames string array, if time series is stored in a **GAUSS** dataset, column name of time series in the **GAUSS** dataset.

If there are independent variables in the model, then the second instance in *d0* includes:

d0[2].data matrix, if independent variables are stored in a matrix in memory, N×K matrix, time series.

d0[2].name string, if the independent variables are stored in a **GAUSS** dataset, name of dataset.

d0[2].vnames string array, if the independent variables are stored in a **GAUSS** dataset, column names of the independent variables in the **GAUSS** dataset.

Output

vmo An instance of a varmamtOut structure containing the following members:

vmo.acfm	L×(p*L) matrix, the autocorrelation function. The first *L* columns are the lag *1* ACF; the last *L* columns are the lag *p* ACF.
vmo.aic	L×1 vector, the Akaike Information Criterion.
vmo.arroots	p×1 vector of AR roots, possibly complex.
vmo.bic	L×1 vector, the Schwarz Bayesian Information Criterion.
vmo.covpar	Q×Q matrix of estimated parameters. The parameters are in the row-major order: *AR(1)* to *AR(p)*, *MA(1)* to *MA(q)*, *beta* (if *x* variables were present in the estimation), and the constants.
vmo.fct	L×1 vector, the likelihood value.
vmo.lagr	An instance of an *sqpsolvemtLagrange* structure containing the following members:

vmo.lagr.lineq	linear equality constraints.
vmo.lagr.nlineq	nonlinear equality

		constraints.
	vmo.lagr.linineq	linear inequality constraints.
	vmo.lagr.nlinineq	nonlinear inequality constraints.
	vmo.lagr.bounds	bounds.
	When an inequality or bounds constraint is active, its associated Lagrangean is nonzero. The linear Lagrangeans precede the nonlinear Lagrangeans in the covariance matrices.	
vmo.lrs	L×1 vector, the Likelihood Ratio Statistic.	
vmo.maroots	q×1 vector of MA roots, possibly complex.	
vmo.pacfm	L×(p*L) matrix, the partial autocorrelation function, computed only if a univariate model is estimated. The first *L* columns are the lag *1* ACF; the last *L* columns are the lag *p* ACF.	
vmo.par	An instance of a *PV* structure containing the parameter estimates, which can be retrieved using **pvUnpack**. For example,	

```
struct  varmamtOut vout;
vout = varmaxmt(vmc,y,0);
```

```
ph = pvUnpack(vout.par,"phi");
th = pvUnpack(vout.par,"theta");
vc = pvUnpack(vout.par,"vc");
```

The complete set of parameter matrices and arrays that can be unpacked depending on the model is:

phi	L×p×p array, autoregression coefficients.
theta	L×q×q array, moving average coefficients.
vc	L×L residual covariance matrix.
beta	L×K regression coefficient matrix.
beta0	L×1 constant vector.
zeta	L×p×ar array of ECM coefficients.
pi	L×L matrix. *Note that* **pi** *is a reserved word in* ***GAUSS****. Users will*

		need to assign this to a different variable name.
vmo.portman	*vmc.lags*-(p+ q)×3 matrix of portmanteau statistics for the multivariate model and Ljung-Box statistics for the univariate model. The time period is in column one, the *Qs* (portmanteau) statistic in column two and the p-value in column three.	
vmo.residuals	T×L matrix, residuals.	
vmo.retcode	2×1 vector, return code. First element:	
	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.
	Second element:	
	0	covariance matrix of parameters failed.
	1	ML covariance matrix.
	2	QML covariance matrix.
	3	Cross-Product covariance matrix.
vmo.ss	L×2 matrix, the sum of squares for Y in column one and the sum of squared error in column 2.	

Remarks

The SVARMAX model includes parameters for describing seasonal effects in a VARMAX model. The SARIMA model is a special case that can also be estimated with **`svarmaxmt`**.

Using widely used terminology, the SVARMAX model can be described as SVARMAX(p,d,q,P,D,Q,s) where

p	vmo.ar	autoregression order.
d	vmo.diff	differencing parameter.
q	vmo.ma	moving average order.
P	vmo.sar	seasonal autoregressive order.
D	vmo.sdiff	seasonal differencing parameter.
Q	vmo.sma	seasonal moving average order.
s	vmo.s	seasonal order.

The model represented by

$$(1-L)^d(1-L^s)^D\phi(L)\Phi(L^s)Y_t = \theta(L)\Theta(L^s)\epsilon_t$$

where

$$\begin{aligned}
\phi(L) &= 1-\phi_1 L-\phi_2 L^2 \ldots \phi_p L^p \\
\Phi(L^s) &= 1-\Phi_1 L^s-\Phi_2 L^{2s} \ldots \Phi_P L^{Ps} \\
\theta(L) &= 1+\theta_1 L+\theta_2 L^2 \ldots \theta_q L^q \\
\Theta(L) &= 1+\Theta_1 L^s+\Theta_2 L^{2s} \ldots \Theta_Q L^{Qs}
\end{aligned}$$

Errors are assumed to be distributed N(0,Q). The estimation procedure assumes that all series are stationary. Setting `vmc.SetConstraints` to a

nonzero value enforces stationarity, by constraining the roots of the characteristic equations

$$1 - \phi_1 z - \phi_2 z^s \ldots \phi_p z^p$$

and

$$1 - \Phi_1 z^s - \Phi_2 z^{2s} \ldots \Phi_P z^{Ps}$$

are constrained to be outside the unit circle.

If any estimated parameters in the coefficient matrices are on a constraint boundary, the Lagrangeans associated with these parameters will be nonzero. These Lagrangeans are stored in *vmo.lagr*. Standard errors are generally not available for parameters on constraint boundaries.

Source

svarmamt.src

switchmt

Purpose

Estimates the parameters of the Markov switching regression model.

Library

tsmt

Format

```
swo = switchmt(swc,d,num_states,num_lags);
```

Input

swc An instance of a *switchmtControl* structure. The following members of *swc* are referenced within this routine:

swc.constVariance	scalar, if nonzero, error variances are constant across states, otherwise if zero, not.
swc.relevantStates	scalar, if nonzero, lagged states are relevant for time series variable, otherwise if zero, only the current state is relevant.
swc.aBayes	scalar, if nonzero, "a" parameter controlling the Bayesian prior as described in James D. Hamilton, 1991, "A quasi-Bayesian approach to estimating parameters for mixtures of Normal distributions," *Journal of Business and Economic Statistics*, 9:27-39.
swc.bBayes	scalar, if nonzero, "b" parameter controlling the Bayesian prior.
swc.cBayes	scalar, if nonzero, "c" parameter controlling the Bayesian prior.
swc.userTransEqp	scalar, pointer to user-provided function for

	setting equality constraints on transition probability matrix.	
swc.start	instance of a *PV* structure containing start values.	
	beta0	1, *num_states* by 1 vector, constants.
	beta	2, *num_states* by K, coefficients on K independent variables if any.
	phi	3, *num_lags* by 1 vector, autoregression coefficients.
	sigma	4, scalar or *num_states* by 1 vector, error variances. If *swc.constVariance* is zero, it is a scalar, otherwise it is a vector.
	p	5, *num_states* by *num_states* matrix, transition probabilities.

For example:

```
swc.start = pvPacki
     (swc.start,3|-3,"beta0",
1);
swc.start = pvPacki
     (swc.start,.1|.01,"Phi",
3);
swc.start = pvPacki
     (swc.start,1,"Sigma",4);
swc.start = pvPacki
     (swc.start,(.8~.1)|
(.2~.9),"P",
     5);
```

swc.ctl instance of an *sqpsolvemtControl* structure.

swc.ctl.covType scalar, if 2, QML standard errors are computed, if 0, none; otherwise Wald-type.

swc.ctl.printIters scalar, iteration information printed every *swc.ctl.printIters*-th iteration.

See documentation for

sqpsolvemtControl for further information regarding members of this structure.

swc.header

string, specifies the format for the output header. *swc.header* can contain zero or more of the following characters:

t	title is to be printed.
l	lines are to bracket the title.
d	a date and time is to be printed.
v	version number of program is to be printed.
f	file name being analyzed is to be printed.

Example:

```
swc.header = "tld";
```

If *swc.header* = "", no header is printed. Default = "tldvf".

	swc.output	scalar, if nonzero, results are printed to screen. Default = 1 .
d	1×1 or 2×1 instances of a *DS* data structure. If the time series and independent variables, if any are in a dataset, if there are no independent variables	
	d.dname	string containing name of **GAUSS** dataset.
	d.vnames	string containing name of time series.
	and if there are independent variables	
	d[1].dname	string containing name of **GAUSS** dataset.
	d[1].vnames	string containing name of time series.
	d[2].vnames	K×1 string array containing names of independent variables.
	Note: create vector of *DS* instances using reshape: d = **reshape**(*d,2,1*).	
	If the time series and independent variables, if any, are in matrices stored in memory, if there are no independent variables	
	d.dataMatrix	N×1 vector, time series.
	d.vnames	string containing name of time series, optional.
	and if there are independent variables	

d[1].dataMatrix N×1 vector, time series.

d[1].vnames string containing name of time series, optional.

d[2].dataMatrix N×K matrix, independent variables.

d[2].vnames K×1 string array containing names, optional of independent variables, optional.

Note: create vector of *DS* instances using reshape:
d = **reshape**(*d,2,1*).

num_states scalar, number of states.

num_lags scalar, number of lagged values of the dependent variable.

Output

out An instance of a *switchmtOut* structure containing the following members:

out.par instance of a *PV* structure containing the estimates:

beta0 1, *num_states*×1 vector, constants.

	beta	2, *num_states*×K, coefficients on K independent variables if any.
	phi	3, *num_lags*×1 vector, autoregression coefficients.
	sigma	4, scalar or *num_states*×1 vector, error variances. If *vmc.constVariance* is zero, it is a scalar, otherwise it is a vector.
	p	5, *num_states*×*num_states* matrix, probabilities.

For example,

```
consts = pvUnpack(out.par,
     "beta0");
```

or

```
consts = pvUnpack(out.par,1);
```

out.covPar	M×M matrix, covariance matrix of parameters.

out.logl	scalar, log-likelihood at maximum.	
out.retcode	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.lagr	instance of *sqpsolvemtLagrange* structure.	
	out.lagr.lineq	M×1 vector, Lagrangeans of linear equality constraints.
	out.lagr.nlineq	N×1 vector,

	Lagrangeans of nonlinear equality constraints.
`out.lagr.linineq`	P×1 vector, Lagrangeans of linear inequality constraints.
`out.lagr.nlinineq`	Q×1 vector, Lagrangeans of nonlinear inequality constraints.
`out.lagr.bounds`	K×2 matrix, Lagrangeans of bounds.

Whenever a constraint is active, its associated Lagrangean will be nonzero. For any constraint that is inactive throughout the iterations as well as at convergence, the corresponding Lagrangean matrix will be set to a scalar missing value.

Example

This example reproduces the results for the French exchange rate in "Long Swings in the Exchange Rate: Are They in the Data and Do Markets Know It?" by Charles Engel and James D. Hamilton, *American Economic Review*, Sept. 1990.

```
load y0[58,3] = exdata.txt;

struct  DS d0;
d0.dataMatrix = y0[.,1];

struct switchmtControl c0;
c0 = switchmtControlCreate();

c0.constVariance = 0;
c0.output = 1;
c0.aBayes = .2;
c0.bBayes = 1;
c0.cBayes = .1;

 /*
** The log-likelihood is somewhat flat and thus
** the problem requires a good starting point.
*/

b0 = { 3.3, -2.7 };
sig = { 10, 37 };
p = { .8 .2, .2 .8 };

struct PV st0;
st0 = pvPacki(pvCreate(),b0,"beta0",1);
st0 = pvPacki(st0,sig,"sigma",4);
st0 = pvPacki(st0,p,"p",5);

c0.start = st0;

struct switchmtOut out0;
```

```
out0 = switchmt(c0,d0,2,0);
```

Source

switchmt.src

TARControlCreate

Purpose

Sets the members of a declared structural break control structure to default values.

Library

tsmt

Format

tar0 = **TARControlCreate**();

Input

None

Output

tar0	An instance of a *TARControlCreate* structure with all members set to default values.

Source

tarcontrolcreate.src

tarTest

Purpose

Estimates the p'th order threshold autoregression model.

Library

tsmt

Format

TARout = **tarTest**(*yt*,*p*,struct *TARControl tar0*);

Input

yt	Nx1 vector, data.	
p	autoregressive order of the TAR model.	
tar0	TARControl structure containing the following elements:	
	p	scalar, Autoregressive order of the STAR model.

omit	scalar, Nx1 vector number of lags (below p) to omit from the matrix.
lowerQuantile	scalar, the lower quantile.
upperQuantile	scalar, the upper quantile.
rep	scalar, the number of simulation replications.
printOutput	scalar, 0 or 1, 1 prints output to the screen.
graph	scalar, 0 or 1, 1 turns on plotting.
dstart	scalar, start date of the time series in DT scalar format as used by **plotTS**.
freq	scalar, Data frequency, 12 for monthly, 4 for quarterly or 1 for annual.

Output

TAROut	TAROutstructure containing the following return elements:
tests	vector of test statistics (in order): SupLM, ExpLM,

	AveLM, SupLMs, ExpLMs, AveLMs.
pvalues	vector, estimated asymptotic p-values or test statistics.
coefficients	matrix, first column contains estimated coefficients and second column contains standard errors.
regimeErrorVariance	vector, 2x1, error variance for Regime 1 and Regime 2, respectively.
thresholdLag	scalar, threshold variable lag.
thresholdValue	scalar, threshold estimate.
errorVariance	scalar, threshold model error variance.

References

1. Hansen, B.E. (1996). Inference when a nuisance parameter is nost identified under the null hypothesis, *Econometrica*, 64(2), 413-430.
2. Franses, P.H. and Dijk, D. (2000) **Non-linear Time Series Models in Empirical Finance**. Cambridge University Press, New York.

Source

tartest.src

tautocovmt

Purpose

Computes the theoretical autocovariances given the coefficient values from an ARMA(*p*,*q*) process.

Library

tsmt

Format

g = **tautocovmt**(*b*,*p*,*q*);

Input

b	Kx1 vector, parameter coefficients.
p	scalar, the autoregressive order.
q	scalar, the moving average order.

Output

g	[Max(p,q)+1]×1 vector, theoretical autocovariances.

Remarks

The theoretical autocorrelations are found by dividing *g* by *g*[1].

Source

tautocovmt.src

tscsmt

Purpose

Estimates the parameters of the pooled time-series cross-section regression model.

Library

tsmt

Format

tso = **tscsmt**(*tsc*,*d0*,*grp*);

Input

tsc	An instance of a *tscsmtControl* structure. The following members of *tsc* are referenced within this routine:
	tsc.header string, specifies the format for the output header. *tsc.header* can contain zero or more of the following characters:

	t	title is to be printed.
	l	lines are to bracket the title.
	d	a date and time is to be printed.
	v	version number of program is to be printed
	f	file name being analyzed is to be printed

Example:

```
tsc.header = "tld";
```

If *tsc.header* = "", no header is printed. Default = "tldvf".

tsc.ise	scalar. If 1, the individual-specific effects are not printed. Default = 0.
tsc.output	scalar, if nonzero, results are printed to screen. Default = 1.
tsc.meth	scalar. Possible values are:

	0	Uses the fixed effects estimates of the individual-specific effects to estimate the variance components of the random effects model. Use this option if there are a different number of observations for each cross-sectional unit. The chi-squared test for the individual error components equal to 0 may not be correct if there are a different number of observations for each individual.
	1	Uses regression on group means to estimate variance components.
	Default = 0.	
tsc.mnsfn	string, the name of a file in which	

to save the group means of the dataset. By default, *tsc.mnsfn* = "", so the means are not saved.

tsc.model scalar, controls the type of models to be estimated. Possible values are:

0	All models are estimated.
1	The random effects (error components model) is not estimated.

tsc.row scalar. Specifies how many rows of the dataset are to be read per iteration of the read loop. By default, the number of rows to be read is calculated by **tscsmt**.

tsc.rowfac scalar, "row factor." If **tscsmt** fails due to insufficient memory while attempting to read a **GAUSS** dataset, *tsc.rowfac* may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** dataset. For example, setting

```
tsc.rowfac = 0.8;
```

causes **GAUSS** to read in 80% of the rows of the **GAUSS** dataset that were read when the failure due to insufficient memory occurred. *tsc.rowfac* has an effect only when *tsc.row* = 0. Default = 1.

tsc.stnd — scalar. If 1, print standardized estimates of regression parameters. Default = 1.

tsc.title — string, a title to be printed at the top of the output header (see *tsc.header*). By default, no title is printed (*tsc.title* = "").

d0 — 1×1 or 2×1 instance of a *DS* data structure. If there are independent variables, d0 is 2×1. For the first instance in d0:

d0[1].datamatrix — if time series is stored in a matrix in memory, N×1 matrix, time series.

d0[1].dname — string, if time series is stored in a **GAUSS** dataset, name of dataset.

d0[1].vnames — string array, if time series is stored in a **GAUSS** dataset, column name of time series in the

GAUSS dataset.

If there are independent variables in the model, then the second instance in *d0* includes:

d0[2].data	matrix, if independent variables are stored in a matrix in memory, N×K matrix, time series.
d0[2].dname	string, if the independent variables are stored in a **GAUSS** dataset, name of dataset.
d0[2].vnames	string array, if the independent variables are stored in a **GAUSS** dataset, column names of the independent variables in the **GAUSS** dataset.

Output

tso An instance of a tscsmtOut structure containing the following members:

tso.bdv	K×1 vector, regression coefficients from the dummy effects model (excluding individual-variables regression model).

tso.vcdv	K×K matrix, variance-covariance matrix of the dummy variables regression model.
tso.mdv	(K+1)×(K+1) matrix, moment matrix of the transformed variables (including a constant) from the dummy variables regression model.
tso.bec	K×1 vector, regression coefficients from the random effects regression model.
tso.vcec	K×K matrix, variance-covariance matrix of the random effects regression model..
tso.mec	(K+1)×(K+1) matrix, moment matrix of the transformed variables (including a constant) from the random effects regression model.

Remarks

The data must be contained in a **GAUSS** dataset cross-sectional unit by cross-sectional unit, with one variable containing an index for the units. From each cross-sectional unit all observations must be grouped together. For example, for the first cross-sectional unit there may be 10 rows in the dataset, for the second cross-sectional unit there may be another 10 rows, and so on. Each row

in the dataset contains measurements on the endogenous and exogenous variables measured for each observation along with the index identifying the cross-sectional unit.

The index variable must be a series of integers. While all observations for each cross-sectional unit must be grouped together, they do not have to be sorted according to the index.

Example

The following example is taken from the program `tscsmt.e`, located in the `examples` subdirectory. The program uses the sample data in `jdatamt.dat`.

```
library tsmt;

struct tscsmtControl tsc;
struct tscsmtOut tso;
tsc = tscsmtControlCreate();

lhs = "x2";
exog = "x3";
inname = "jdatamt";

output file = jdatamt.out reset;
grp = "x1";
tsc.meth = 1;
tso = tscsmt(tsc,inname,lhs,exog,grp);
output off;
```

Source

tscsmt.src

tscsmtControlCreate

Purpose

Sets the members of an instance of a *tscsmtControl* structure to default values.

Library

tsmt

Format

tsc = **tscsmtControlCreate**();

Input

None

Output

tsc	An instance of a *tscsmtControl* structure with its members set to default values.

Remarks

Putting this instruction at the top of all programs that invoke **tscsmt** is generally good practice. This will prevent control variables from being inappropriately defined when a program is run either several times or after another program that also calls **tscsmt**.

Source

tscsmt.src

tsforecastmt

Purpose

Estimates forecasts using estimation results obtained from **arimamt**.

Library

tsmt

Format

f = **tsforecastmt**(*amc*,*b*,*y*,*p*,*d*,*q*,*const*,*e*,*h*);

Input

amc	An instance of an *arimamtControl* structure. The following members of *amc* are referenced within this

	routine:
	amc.critl scalar, confidence level to compute for forecast confidence bounds. Default = 0.95.
	amc.output scalar, controls printing of output .
	0 Nothing is printed.
	1 Forecasts, confidence bounds and forecast standard errors are printed.
b	K×1 vector, estimated coefficients
y	N×1 vector, data.
p	scalar, the autoregressive order.
d	scalar, the order of differencing
q	scalar, the moving average order.
const	scalar, if 1, a constant is estimated, 0 otherwise.
e	N×1 vector, residuals reported by **arimamt** program.
h	scalar, the number of step-ahead forecasts to computer.

Output

f	h×3 matrix,	
	[.,1]	Lower forecast confidence bounds.
	[.,2]	Forecasts.
	[.,3]	Upper forecast confidence bounds.

Remarks

Data must be transformed before being sent to **tsForecast**.

tsForecast does not compute forecasts for models with fixed regressors.

Source

forecastmt.src

varmamtControlCreate

Purpose

Sets the members of an instance of a *varmamtControl* structure to default values.

Library

tsmt

Format

vmc = **varmamtControlCreate**();

Input

None

Output

amc	An instance of a *varmamtControl* structure with its members set to default values.

Remarks

Putting this instruction at the top of all programs that invoke **varmaxmt** or **ecmmt** procedures is generally good practice. This will prevent control variables from being inappropriately defined when a program is run either several times or after another program that also calls **varmaxmt** or **ecmmt**procedures.

Source

varmamt.src

varmaxmt

Purpose

Computes exact maximum likelihood parameter estimates for a VARMAX model.

Library

tsmt

Format

vmo = **varmaxmt**(*vmc*,*d0*);

Input

vmc An instance of a *varmamtControl* structure. The following members of *vmc* are referenced within this routine:

vmc.adforder	scalar, number of AR lags in the ADF test statistic. Default = 2.
vmc.ar	scalar, order of AR process. Default = 0.
vmc.critl	scalar, the significance levels defining p-values. Default = .95.
vmc.diff	scalar, the order of differencing to achieve stationarity. Default = 0.

vmc.ctl	instance of an *sqpsolvemtControl* structure.
	vmc.ctl.covType scalar, if 2, QML standard errors are computed, if 0, none; otherwise Wald-type.
	vmc.ctl.printIters scalar, iteration information printed every *swc.ctl.printIters*-th iteration.
	See documentation for *sqpsolvemtControl* for further information regarding members of this structure.
vmc.header	string, specifies the format for the output header. *vmc.header* can contain zero or more of the following characters:
	t title is to be printed.
	l lines are to bracket the title.
	d a date and time is to be printed.

	v	version number of program is to be printed.
	f	file name being analyzed is to be printed

Example:

```
vmc.header = "tld";
```

If *vmc.header* = "", no header is printed. Default = "tldvf".

vmc.IndEquations K×L matrix of zeros and ones. Used to set zero restrictions on the variables to be estimated. Only used if the number of equations, *vmc.L* is greater than one. Elements set to indicate the coefficients to be estimated. If *vmc.L* = 1, all coefficients will be estimated. If *vmc.L* > 1 and *vmc.IndEquations* is set to a missing value (the default), all coefficients will be estimated.

vmc.lags scalar, number of lags over which ACF and Diagnostics are calculated. Default = 12.

vmc.ma scalar, number of MA matrices to be estimated.

	Default = 0.
vmc.nodet	scalar. Set *vmc.nodet* = 1 to suppress the constant term from the fitted regression and include it in the co-integrating regression; otherwise, set *vmc.nodet* = 0. Default = 0.
vmc.nwtrunc	scalar, the number of autocorrelations to use in calculating the Newey-West correction. If *vmc.nwtrunc* = 0, GAUSS will use a truncation lag given by Newey and West, *vmc.nwtrunc* $= 4(T / 100)^{2/9}$.
vmc.output	scalar. Set to 0 to suppress all printing from **varmaxmt**. Set *vmc.output* > 0 to print results. Default = 1.
vmc.scale	scalar or an L×1 vector, scales for the time series. If scalar, all series are multiplied by the value. If an L×1 vector, each series is multiplied by the corresponding element of vmc.scale. Default = 4/standard deviation (found to be best by experimentation).
vmc.SetConstraints	scalar, set to a nonzero value to impose stationarity and invertibility by constraining roots of the AR and MA characteristic equations to be outside the unit circle. Set to zero to estimate an unconstrained model.

Default = 1.

vmc.Start Instance of a *PV* structure containing starting values. See Section 3.7.2 for discussion of setting starting values. By default, **varmaxmt** calculates starting values.

vmc.title string, a title to be printed at the top of the output header (see *vmc.header*). By default, no title is printed (*vmc.title* = "").

d0 1×1 or 2×1 instance of a *DS* data structure. If there are independent variables, d0 is 2×1. For the first instance in d0,

d0[1].datamatrix if time series is stored in a matrix in memory, N×1 matrix, time series.

d0[1].dname string, if time series is stored in a **GAUSS** dataset, name of dataset.

d0[1].vnames string array, if time series is stored in a **GAUSS** dataset, column name of time series in the **GAUSS** dataset.

If there are independent variables in the model, then the second instance in *d0* includes:

d0[2].data matrix, if independent variables are stored in a matrix in memory, N×K matrix, time series.

d0[2].dname string, if the independent variables are stored

	in a **GAUSS** dataset, name of dataset.
d0[2].vnames	string array, if the independent variables are stored in a **GAUSS** dataset, column names of the independent variables in the **GAUSS** dataset.

Output

vmo An instance of a varmamtOut structure containing the following members:

vmo.acfm	L×(p*L) matrix, the autocorrelation function. The first *L* columns are the lag *1* ACF; the last *L* columns are the lag *p* ACF.
vmo.aic	L×1 vector, the Akaike Information Criterion.
vmo.arroots	p×1 vector of AR roots, possibly complex.
vmo.bic	L×1 vector, the Schwarz Bayesian Information Criterion.
vmo.covpar	Q×Q matrix of estimated parameters. The parameters are in the row-major order: AR(1) to AR(p), MA(1) to MA(q), *beta* (if *x* variables were present in the

estimation), and the constants.

vmo.fct L×1 vector, the likelihood value.

vmo.lagr An instance of an *sqpsolvemtLagrange* structure containing the following members:

vmo.lagr.lineq	linear equality constraints.
vmo.lagr.nlineq	nonlinear equality constraints.
vmo.lagr.linineq	linear inequality constraints.
vmo.lagr.nlininеq	nonlinear inequality constraints.
vmo.lagr.bounds	bounds. When an inequality or bounds constraint is active, its associated Lagrangean is nonzero. The linear

Lagrangeans precede the nonlinear Lagrangeans in the covariance matrices.

vmo.lrs L×1 vector, the Likelihood Ratio Statistic.

vmo.maroots q×1 vector of MA roots, possibly complex.

vmo.pacfm L×(p*L) matrix, the partial autocorrelation function, computed only if a univariate model is estimated. The first *L* columns are the lag *1* ACF; the last *L* columns are the lag *p* ACF.

vmo.par An instance of a *PV* structure containing the parameter estimates, which can be retrieved using **pvUnpack**. For example,

```
struct varmamtOut vout;
vout = varmaxmt(vmc,y,0);
ph = pvUnpack(vout.par,
    "phi");
th = pvUnpack(vout.par,
    "theta");
vc = pvUnpack(vout.par,
    "vc");
```

The complete set of parameter matrices and arrays that can be unpacked depending on the model is:

phi	L×p×p array, autoregression coefficients.
theta	L×q×q array, moving average coefficients.
vc	L×L residual covariance matrix.
beta	L×K regression coefficient matrix.
beta0	L×1 constant vector.
zeta	L×p×ar array of ecm coefficients.
pi	L×L matrix. *Note that pi is a reserved word in **GAUSS**. Users*

		will need to assign this to a different variable name.
vmo.portman	*vmc.lags*-(p+ q)×3 matrix of portmanteau statistics for the multivariate model and Ljung-Box statistics for the univariate model. The time period is in column one, the *Qs* (portmanteau) statistic in column two and the p-value in column three.	
vmo.residuals	T×L matrix, residuals.	
vmo.retcode	2×1 vector, return code. First element:	
	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.
	Second element:	
	0	covariance matrix of parameters failed.
	1	ML covariance matrix.
	2	QML covariance matrix.
	3	Cross-Product covariance matrix.
vmo.ss	L×2 matrix, the sum of squares for Y in column one and the sum of squared error in column 2.	

Remarks

Errors are assumed to be distributed N(0,Q). The estimation procedure assumes that all series are stationary. Setting *vmc.SetConstraints* to a nonzero value enforces stationarity, by constraining the roots of the characteristic equation

$$1 - \Phi_1 z - \Phi_2 z^2 - \ldots - \Phi_p z^p$$

to be outside the unit circle (where $\Phi_i, i = 1, \ldots, p$ are the AR coefficient matrices).

If any estimated parameters in the coefficient matrices are on a constraint boundary, the Lagrangeans associated with these parameters will be nonzero. These Lagrangeans are stored in *vmo.lagr*. Standard errors are generally not available for parameters on constraint boundaries.

Source

varmamt.src

vmadfmt

Purpose

Compute the Augmented Dickey-Fuller statistic, allowing for deterministic polynomial time trends of an arbitrary order.

Library

tsmt

Format

```
{ alpha,tstat,adf_t_crit } = vmadfmt(x,p,l);
```

Input

x	Time series variable.
p	Order of the time-polynomial to include in the ADF regression. Set *p* = -1 for no deterministic part.
l	Number of lagged changes of *x* to include in the fitted regression.

Output

alpha	Estimate of the autoregressive parameter
tstat	ADF t-statistic.
adf_t_crit	6×1 vector of critical values for the adf-t-statistic: 1%, 5%, 10%, 90%, 95%, 99%.

Source

varmamt.src

vmc_sja

Purpose

Returns critical values for the Johansen Maximum Eigenvalue statistic. Computed using 8000 iterations and 500 observations.

Library

tsmt

Format

c_values = **vmc_sja**(*n*,*p*);

Input

n	scalar, number of variables in the system.
p	scalar, order of the time-polynomial to include in the fitted regression.

Output

c_values	6×1 vector of critical values.

Source

varmamt.src

vmc_sjt

Purpose

Returns critical values for the Johansen Trace statistic.

Library

tsmt

Format

c_values = **vmc_sjt**(*n*,*p*);

Input

n	scalar, number of variables in the system.
p	scalar, order of the time-polynomial to include in the fitted regression.

Output

c_values	6×1 vector of critical values.

Source

varmamt.src

vmcadfmt

Purpose

Compute the Augmented Dickey-Fuller statistic applied to the residuals of a cointegrating regression, allowing for deterministic polynomial time trends of an arbitrary order.

Library

tsmt

Format

{ *alpha*,*tstat*,*vmrztcrit* } = **vmcadfmt**(*y*,*x*,*p*,*l*);

Input

y	Dependent variable.
x	Explanatory variables.
p	Order of the time-polynomial to include in the cointegrating regression. Set *p* = -1 for no deterministic part.
l	Number of lagged changes of the residuals to include in the fitted regression.

Output

alpha	Estimate of the autoregressive parameter.
tstat	ADF t-statistic.
vmrztcrit	6×1 vector of critical values for the adf-t-statistic: 1%, 5%, 10%, 90%, 95%, 99%.

Source

varmamt.src

vmdetrendmt

Purpose

Returns residuals from a regression of data on a time trend polynomial.

Library

tsmt

Format

res = **vmdetrendmt**(*y*,*p*);

Input

y	T×L matrix of data.
p	scalar. If $p = -1$ returns the data. Use $p = 0$ for demeaning, $p = 1$ for regression against a constant term and trend, $p > 1$ for a higher order polynomial time trend.

Output

res	T×L matrix of residuals.

Source

varmamt.src

vmdiffmt

Purpose

Differences matrices.

Library

tsmt

Format

```
y = vmdiffmt(x, d);
```

Input

x	T×X matrix.
d	scalar, the number of periods over which differencing occurs.

Output

y	(T-d)×K matrix, the differenced data.

Source

vmutilsmt.src

vmforecastmt

Purpose

Calculates forecasts from a VARMAX model.

Library

tsmt

Format

```
f = vmforecastmt(vmc,vmo,y,x,t);
```

Input

vmc	An instance of a *varmamtControl* structure. The following members of vmc are referenced within this routine: *vmc.ar* scalar, AR order. *vmc.ma* scalar, MA order.
vmo	An instance of a *varmamtOut* structure returned by a call to a VARMAX or ECM procedure.
y	T×L matrix, the variables to be forecast.
x	t×K matrix of *x* covering only the forecast horizon, in the order $T+1, \ldots, T+t$ or the scalar zero if there are no *x* variables.
t	scalar, the number of periods to forecast.

Output

f	t×(L+1) matrix. Column one contains the period forecast. The remaining columns contain the forecast values.

Remarks

The **varmaxmt** and **ecmmt** procedures estimate centered models and do not return intercepts. However, **vmforecastmt** allows intercepts, so that it might be used with the results of other estimation procedures.

Source

varmamt.src

vmppmt

Purpose

Returns Phillips-Perron unit root test statistics and critical values.

Library

tsmt

Format

{ *ppb*,*ppt*,*pptcrit* } = **vmppmt**(*y*,*p*,*nwtrunc*);

Input

y	Tx1 vector, a time series.
p	scalar, order of the time-polynomial to include in the regression. Set *p* = -1 for no deterministic part, *p* = 0

	for a constant term, and $p = 1$ for a constant with trend.
nwtrunc	scalar, the number of autocorrelations to use in calculating the Newey-West correction (q in the Remarks section below). If *nwtrunc* = 0, **GAUSS** will use a truncation lag given by Newey and West, $q = 4(T / 100)^{2/9}$

Output

ppb	scalar, estimate of the autoregressive parameter, the p coefficient below.
ppt	scalar, the adjusted t-statistic for testing: $H_0 : \rho = 1$.
pptcrit	6×1 vector of critical values, vector of critical values for the adjusted t-statistic, in the order 1%, 5%, 10%, 90%, 95%, 99%.

Remarks

Phillips (1987) and Phillips and Perron (1988) test for unit roots by adjusting the OLS estimate of an *AR(1)* coefficient for serial correlation in the OLS residuals. Three specifications are considered, an *AR(1)* model without a drift, an *AR(1)* with a drift, and an *AR(1)* model with a drift and linear trend:

$$
\begin{aligned}
Y_t &= \rho Y_{t-1} + \varepsilon_t \\
Y_t &= \alpha + \rho Y_{t-1} + \varepsilon_t \\
Y_t &= \alpha + \delta t + \rho Y_{t-1} + \varepsilon_t
\end{aligned}
$$

The unit root null hypothesis is $H_0 : (\rho - 1) = 0$

Hamilton (1994, pp. 506-511) tests this hypothesis using two statistics that are analogs of the Phillips and Perron (1988) Z_α and Z_t statistics. Hamilton's statistics are based on OLS estimation of the above equations. They allow an identical formula for each statistic to be used for all three cases.

The **vmppmt** procedure returns the Z_t statistic as calculated by Hamilton and critical values. Suppose any of the equations are estimated by OLS, returning $\hat{\rho}_T$ and $\hat{\sigma}_{\hat{\rho}_T}$ (the OLS estimates of ρ an the standard error of $\hat{\rho}_T$ respectively), $t_T = \left(\rho - 1\right) / \hat{\sigma}_{\hat{\rho}_T}$ (the usual OLS t statistic for testing H_0), $\hat{\varepsilon}_t$ (the OLS residuals), and s_T (the estimated standard error of the regression).

Hamilton's Z_t statistic is:

$$
Z_t = \left(\hat{\gamma}_0 / \hat{\lambda}^2 \right)^{\frac{1}{2}} t_T - \left\{ \frac{1}{2} \left(\hat{\lambda}^2 - \hat{\gamma}_0 \right) / \hat{\lambda} \right\} \left\{ T \left(\hat{\sigma}_{\hat{\rho}_T} / s_T \right) \right\}
$$

$\hat{\lambda}^2$ is an estimate of the asymptotic variance of the sample mean of ε_t. In the **vmppmt** procedure $\hat{\lambda}^2$ is estimated using the Newey-West (1987) estimator:

$$\hat{\lambda}^2 = \hat{\gamma}_0 + 2\sum_{j=1}^{q}\left[1 - j/\left(q+1\right)\right] - \hat{\gamma}_j$$

where $\hat{\gamma}_j = T^{-1}\sum_{t=j+1}^{T}\hat{\varepsilon}_t\hat{\varepsilon}_{t-j}$ are the sample autocovariances of ε_t.

The **nwtrunc** argument sets the number of autocorrelations to use in calculating the Newey-West correction (q in the above equation). If **nwtrunc** = 0, **GAUSS** will use a truncation lag given by Newey and West, $q = 4(T/100)^{2/9}$.

Under the null hypothesis, the Z_t statistics has the same asymptotic distribution as a Dickey-Fuller statistic.

References

1. Hamilton, James D., (1994). **Time Series Analysis**, Princeton University Press.

2. Newey, W.K. and West, K.D., (1987), "A Simple Positive Semi-Definite Heteroskedasticity and Autocorrelation-Consistent Covariance Matrix," *Econometrica*, 55, 703-708.

Source

varmamt.src

vmrztcritmt

Purpose

Returns τ critical values for the Augmented Dickey-Fuller statistic, derived from the residuals of a cointegrating regression. Depends on p, the AR order in the fitted regression, the number of observations, and the number of explanatory variables.

Library

tsmt

Format

c_values = **vmrztcritmt**(*nobs*,*n*,*p*);

Input

nobs	scalar, number of observations in the series.
n	scalar, column dimension of *x*.
p	scalar, order of the time-polynomial in the null hypothesis.

Output

c_values	6×1 vector of critical values.

Source

varmamt.src

vmsjmt

Purpose

Computes Johansen's (1988) ML Trace and Maximum Eigenvalue statistics.

Library

tsmt

Format

{ *ev*,*evec*,*lr1*,*lr2* } = **vmsjmt**(*x*,*p*,*k*,*nodet*);

Input

x	T×L matrix.
p	scalar, order of the time-polynomial in the fitted regression. Set p = -1 for no deterministic part, p = 0 for a constant term, and p = 1 for a constant with trend.
k	scalar, number of lagged difference terms to use when computing the estimator.

nodet	scalar. Set *nodet* = 1 to suppress the constant term from the fitted regression and include it in the co-integrating regression; otherwise, set *nodet* = 0.

Output

ev	L×1 vector of eigenvalues.
evec	L×L matrix of eigenvectors. The first *r* columns are the unnormalized cointegrating vectors.
lr1	L×1 vector of Johansen's likelihood ratio Trace statistics for the null hypotheses of H0; at most *r* cointegrating vectors versus H1: not H0, $r = 0, \ldots, L - 1$.
lr2	L×1 vector of Johansen's Maximum Eigenvalue Statistics for the null hypotheses of H0: *r* cointegrating vectors versus H1: *r*+1 cointegrating vectors, $r = 0, \ldots, L - 1$.

Source

varmamt.src

vmztcritmt

Purpose

Returns τ critical values for the Augmented Dickey-Fuller statistic, depending on the nuber of observations and p, the AR order in the fitted regression. Computed using 10000 iterations.

Library

tsmt

Format

c_values = **vmztcritmt**(*nobs*,*p*);

Input

nobs	scalar, number of observations in the series.
p	scalar, order of the time-polynomial in the null hypothesis.

Output

c_values	6×1 vector of critical values in order 1%, 5%, 10%, 90%, 95%, 99%.

Source

varmamt.src

zandrews

Purpose

The Zivot and Andrews (1992) unit root test uses a t-test statistic for testing the null hypothesis of stationarity. The procedure tests the null hypothesis of zero innovation variance in the residual against the alternative of non-zero residual innovation variance.

Library

tsmt

Format

{ *t_test*, *break_pt* } = **zandrews**(*yt*, *max_lags*, *trim_end*, *break_type*, *which_output*);

Input

yt	T×1 vector of time series data.
max_lags	scalar, specifies the maximum lag order to be used in calculating the test statistic. A good default is to calculate *max_lags* as T^0.25.

trim_end	scalar, fraction of data range to skip at either end. A good default is 0.15. Range is 0 to 0.25.
break_type	scalar, -1 for intercept break, 0 for trend break, or 1 for a break in both.
which_output	scalar, 0 for no output, 1 to print statistics or 2 to print statistics and display of graph of unit-root test statistics across different break points.

Output

t_test	scalar, reports Zivot-Andrews test statistic.
break_pt	scalar, observation where structural break is most likely to occur.

Index