# LikPak™

**ECONOTRON SOFTWARE, INC.**

**version 1.0**

Jon Breslaw

June, 2007

Support:

Econotron Software
447 Grosvenor Ave.
Westmount, P.Q. Canada
H3Y-2S5

Tel: (514) 939-3092
Fax: (514) 938-4994
Eml: support@econotron.com
Web: http://www.econotron.com

# Contents

# 1 Introduction

In econometrics, one of the most common tasks is the estimation of the parameters of a model, either by maximum likelihood or by least squares. For least squares, the functional form is straight forward - the equation being estimated. For maximum likelihood, there are two considerations - the parameterization of the model, and the specification of the likelihood function. The concept behind LIKPAK is simple - provide a set of likelihood procedures that are commonly used in econometrics, and show, by example, how a model can be parameterized.

LIKPAK is designed to be used as a template - that is, you select the example that is relevant to your problem, and use that example as a starting point. There are over 50 likelihood functions in the LIKPAK package, corresponding to the set of likelihoods currently used in economics. Each example is backed up with documentation describing typical parameterizations. In addition, since there are a number of different optimization tools available, these examples are repeated for each optimization tool. Currently, LIKPAK provides complete examples for `Maxlik` and `CMLMT`, as well as a number of examples for `MaxlikMT`, `QNewton`, `QNewtonMT`, `SQPSolve` and `SQPSolveMT`. These optimization packages can be categorized into two types - those that use structures (`CMLMT, MaxlikMT, QNewtonMT, SQPSolveMT`) and those that do not use structures (`Maxlik, QNewton, SQPSolve`). There are some 40 example files in each of the `Maxlik` and `CMLMT` folders, and these provide templates for each type of package. For each of the other packages, the example file is commented to show the necessary programming changes.

LIKPAK is designed to simplify the coding required for a maximum likelihood estimation. For example, for statistical functions, truncation and censoring are both supported. For least squares problems, LIKPAK provides the NLS command for single and multiple non-linear equation systems. For languages requiring structures, LIKPAK provides a set of (optional) PV and DS commands that simplify the use of the PV and DS structures, as well as facilitating passing options. And additional utilities are provided for data generation, filtering, Gibbs sampling, and constraining parameters.

# 2 Installation and Configuration

The installation routines decompress and copy files and libraries from the compressed file to your GAUSS folder. Before starting however, read this chapter.

As for all software, please observe the normal rules for copyright material.

It is assumed that you have GAUSS already installed in a directory \gauss, and that it is working properly. In addition, you should have set the defaults for your machine for GAUSS in gauss.cfg,

## 2.1 Installing LikPak for Windows

LIKPAK runs as a 32 bit application – Windows 98, NT 4, ME, 2000, XP and Vista are supported. GAUSS for Windows 4.0 and higher are supported.

LIKPAK for Windows is distributed as a zip file, typically likpak_vsn.zip. Unzip the file in a temporary folder, and then execute the file setup.exe. This will install LIKPAK in the GAUSS folder

## 2.2 Installing LikPak for UNIX and MAC

LIKPAK for UNIX is written entirely in GAUSS , and thus is machine independent. It can run either in terminal or X-window mode. GAUSS for UNIX 4.0 or higher is required.

LIKPAK for UNIX is distributed as a tar file, typically likpak_vsn.tar.gz. From the UNIX prompt:

1. Copy the `.tar.gz` or `.zip` file to /tmp.
2. If the file has a `.tar.gz` extension, unzip it with gunzip:

   gunzip *likpak_vsn123.tar.gz*
3. change directory to your GAUSS or GAUSS Engine directory:

   cd /user/local/gauss
4. Extract the file:

   tar xvf /tmp/likpak_vsn123.tar
   *or*
   unzip /tmp/likpak_vsn123.zip

## 2.3 Running LikPak

LIKPAK consists of some 50 likelihoods, with examples for each one. If your optimization package uses PV and DS structures, look at the examples under likpak\examples\cmlmt; if not, look at the examples under likpak\examples\maxlik. The examples in these two folders demonstrate each of the likelihood functions. Simply execute one of these files by running it from GAUSS , for example:

run likpak\examples\maxlik\tobit.e

If you are using an optimization package other than CMLMT or MAXLIK, refer to the example file shown in the appropriate folder. The example file is commented to show the necessary programming changes. Thus, if you are using SQPSolve, and wish to run a Tobit example, copy tobit.e from the maxlik folder to the sqpsolve folder, and then refer to the example file (expon.e for the comments on what changes are necessary.

# Reference Summary 3

The command reference is arranged in four sections:

| | |
|---|---|
| LikPak Reference | — LikPak processes |
| LikPak Utility Reference | — LikPak utilities |
| DS Utilities Reference | — DS structure utilities |
| PV Utilities Reference | — PV structure utilities |

The commands are arranged alphabetically. For easy reference, a summary of commands arranged by type is given below.

## 3.1 LikPak Processes

**AR Processes**

| | |
|---|---|
| ARFIMA | — ARFIMA process |
| ARIMA | — ARIMA process |
| ARMA | — ARMA process |
| VARMA | — Vector autoregressive moving average process |

**Count Processes**

| | |
|---|---|
| NEGBIN | — Negative binomial process |
| POISSON | — Poisson process |

**Discrete Processes**

| | |
|---|---|
| DBDC | — Double-bounded dichotomous choice process |
| FMNP | — Feasible multinomial probit |
| MNL | — Multinomial logit process |
| MNP | — Multinomial probit process |
| LOGIT | — Binomial logit process |
| ORDLGT | — Ordered logit process |
| ORDPRBT | — Ordered probit process |
| PROBIT | — Binomial multivariate probit process |

**GARCH Processes**

| | |
|---|---|
| AGARCH | — Asymmetric GARCH process |
| ARCH | — ARCH process |
| EGARCH | — Exponential GARCH process |
| FIGARCH | — Fractionally integrated GARCH process |
| GARCH | — GARCH process |
| IGARCH | — Integrated GARCH process |
| MGARCH | — Multivariate GARCH process |
| PGARCH | — Power GARCH process |
| TGARCH | — Truncated GARCH process |

**Statistical Processes**

| | |
|---|---|
| BETA | — Beta process |
| CAUCHY | — Cauchy process |
| CHISQ | — Chi Squared process |
| EXPON | — Exponential process |
| F | — F process |
| GAMMA | — Gamma process |
| GUMBELL | — Gumbell process |
| INVGAUSS | — Inverse Gauss process |
| LAPLACE | — Laplace process |
| LEVY | — Levy process |
| LOGISTIC | — Logistic process |
| LOGLOG | — Log logistic process |
| LOGNORM | — Log normal process |
| NORMAL | — Normal process |
| PARETO | — Pareto process |
| PEARSON | — Pearson process |
| SEV | — Smallest extreme value process |
| STUDENTS_T | — Student's T process |
| VONMISES | — Von Mises process |
| WEIBULLL | — Weibull process |

**Other Processes**

| | |
|---|---|
| BOXCOX | — BoxCox process |
| FPF | — Frontier production process |
| KALMAN | — Kalman process |
| MSM | — Markov switching models process |
| MVN | — Multivariate normal process |
| NEURAL | — Neural networks process |
| NLS | — Nonlinear least squares |
| NPE | — Non parametric estimation |
| SV | — Stochastic volatility process |
| TOBIT | — Tobit process |
| WHITTLE | — Local Whittle process |

## 3.2 LikPak Utilities

| | |
|---|---|
| CENSORED | — Censored process |
| DGP | — Data generation process |
| FILTER | — Data filter |
| LIKSET | — Set LikPak defaults |
| LIKSTAT | — Set LikPak stat flag |
| MROOT | — Largest root |
| PDROOT | — PD Test for smallest root |
| QDFN | — Multivariate normal rectangular probabilities |
| RNDTN | — Truncated multivariate normal random numbers |
| TRUNCATED | — Truncated process |
| WAITKEY | — Keyboard prompt |

## 3.3 DS Utilities

| | |
|---|---|
| dsDATA | — Set data source |
| dsDATAGET | — Retrieve data |
| dsOPTIONS | — Set options |
| dsOPTIONGET | — Retrieve options |

## 3.4 PV Utilities

| | |
|---|---|
| pvCLEAR | — Clear parameter |
| pvCONST | — Set parameter as inactive |
| pvGET | — Retrieve parameter |
| pvGETMASK | — Retrieve matrix mask |
| pvPARAM | — Set parameter as active |
| pvSET | — Set parameter |
| pvSETMASK | — Set parameter mask |

# 4 LikPak Reference

# AGARCH

**Purpose** Creates a vector of log likelihoods for an asymmetric GARCH process.

**Format**

*llf* = **agarch** ( *y*, *indx*, *avec*, *bvec*, *gvec* );
*llf* = **agarch_t** ( *y*, *indx*, *avec*, *bvec*, *gvec dvec* );

**Input**

| | | |
|---|---|---|
| | *y* | is an Nx1 vector of the dependent variable. |
| | *indx* | is an Nx1 vector of the structural index. |
| | *avec* | is a vector of parameters for the ARCH process. |
| | *bvec* | is a vector of parameters for the GARCH process. |
| | *gvec* | is a vector of $\gamma$ parameters. |
| | *dvec* | is a distributional parameter ($\nu$). |

**Output**

| | | |
|---|---|---|
| | *llf* | vector of log likelihoods. |

**Remarks** The structural coefficients and the coefficients of the AGARCH process are estimated using constrained maximum likelihood. The model is given by:

$$\begin{aligned} y_t &= f(x_t, \theta) + \epsilon_t \\ \epsilon_t &\sim N(0, h_t) \\ h_t &= \alpha_0 + \sum_{i=1} \alpha_i (|\epsilon_{t-i}| - \gamma_i \epsilon_{t-i})^2 + \sum_{j=1} \beta_j h_{t-j} \end{aligned}$$

The first equation describes the structural part of the model; thus this can be used for linear or non-linear structural models. The second equation specifies the distribution of the residuals, and the third equation specifies the structural form of the conditional variance $h_t$. The $\alpha$ are the vectors of the weights for the lagged asymmetric $\epsilon^2$ terms; this is the ARCH process. The $\beta$ are the weights for the lagged $h$ terms; this is the GARCH process.

avec is a vector of parameters giving the weights for the lagged asymmetric squared residuals. The first element, which is required, gives the constant. gvec is a vector of parameters for the asymmetric process - the order of gvec should be one less than the order of avec. bvec is the vector of parameters for the GARCH process. Note the stationarity conditions described under GARCH.

See the "General Notes" under GARCH. An example is given in garch.e.

**Example**

```
proc agarch_1(b,dta);
local y,indx,avec,bvec,gam;
y       = dta[.,4];
indx    = findx(b,dta);
avec    = b[3:4];
bvec    = b[5];
gam     = b[6];
retp(agarch(y,indx,avec,bvec,gam));
endp;
```

In this example, a linear AGARCH model is estimated, Under CML, parameter restrictions would ensure that the variance remains positive - each of the avec and bvec coefficients is constrained positive, and $b[4] + b[5] < 1$. This is a simplified model, in that only one $\gamma$parameter is specified – an alternative would be to have a separate $\gamma$ for each lag.

**Source** likpak\src\garch.src

**See Also** garch

**References** Ding, Z., R.F. Engle, and C.W.J. Granger. (1993), "A Long Memory Property of Stock Market Returns and a New Model", *Journal of Empirical Finance*, Vol 1 (1), pp 83-106.

# ARCH

**Purpose** Creates a vector of log likelihoods for an ARCH process.

**Format**

*llf* = **arch** ( *y*, *indx*, *avec* );
*llf* = **arch_t** ( *y*, *indx*, *avec*, *dvec* );

**Input**

| | | |
|---|---|---|
| *y* | is an Nx1 vector of the dependent variable.. |
| *indx* | is an Nx1 vector of the structural index. |
| *avec* | is a vector of parameters for the ARCH process. |
| *dvec* | is a distributional parameter ($\nu$). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The structural coefficients and the coefficients of the ARCH process are estimated using constrained maximum likelihood. ARCH residuals exhibit non-constant variance that are conditional on past variances. The ARCH model is given by:

$$\begin{aligned} y_t &= f(x_t, \theta) + \epsilon_t \\ \epsilon_t &\sim N(0, h_t) \\ h_t &= \alpha_0 + \sum_{i=1} \alpha_i \epsilon_{t-i}^2 \end{aligned}$$

The first equation describes the structural part of the model; thus this can be used for linear or non-linear structural models. The second equation specifies the distribution of the residuals, and the third equation specifies the structural form of the conditional variance $h_t$. The $\alpha$ are the vectors of the weights for the lagged $\epsilon^2$ terms; this is the ARCH process.

avec is a vector of positive parameters giving the weights for the lagged squared residuals. The first element, which is required, gives the constant. If only a single parameter is specified, the model is standard OLS. Note the stationarity conditions described under GARCH.

See the "General Notes" under GARCH. An example is given in garch.e.

**Example**

```
proc arch_1(b,dta);
local y,indx,avec;
y       = dta[.,2];
indx    = b[1]+b[2]*dta[.,1];
avec    = b[3:5];
retp(arch(y,indx,avec));
endp;
```

In this example, a linear ARCH model is estimated, Under CML, parameter restrictions would ensure that the variance remains positive.

**Source** likpak\src\garch.src

**See Also** garch

**References** Engle, R.F. (1982), "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of the U.K. Inflation", *Econometrica*, Vol. 50, pp. 987-1007.

**Purpose** Creates a vector of log likelihoods for a fractional autoregressive integrated moving average process.

**Format** *llf* = **arfima** ( *y*, *d*, *phi*, *theta*, *cnst* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the time series. |
| *d* | is a scalar, degree of differencing. |
| *phi* | is the *P*x1 AR coefficient vector, or scalar zero. |
| *theta* | is the *Q*x1 MA coefficient vector, or scalar zero. |
| *cnst* | is a scalar constant indicator. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Autoregressive Fractionally Integrated Moving Average (ARFIMA) process permits the estimation of long memory models. The ARFIMA $(p, d, q)$ process is given by:

$$\phi(L)(1-L)^d y_t = \theta(L)\epsilon_t$$

where:

$$\begin{aligned}\phi(L) &= 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p \\ \theta(L) &= 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q\end{aligned}$$

and where $L$ is the backward shift operator, and $d$ is the fractional degree of differencing.

The coefficients of the ARFIMA process are estimated using `ML`. When $d$ is an integer, this becomes the ARIMA model. $y$ should be detrended, and have zero mean.

The constant indicator specifies whether a constant is to be included - 0, no constant, 1, constant. A constant should normally be specified for non-differenced series with non-zero mean, unless the constant is explicitly specified as a parameter.

Both stationary and invertibility conditions need to be satisfied. LIKPAK provides a routine called MROOT, which returns the value of the largest root, which must have a modulus less than unity. In addition, besides the normal AR and MA requirements for stationarity and invertibility, requirements on $d$ include $d > -1$ for invertibility, and $-.5 < d < .5$ for stationarity. Consequently, constrained optimization is usually required.

See the examples given in arma.e.

**Example**

```
proc  arfima_1(b,dta);
local  y, d, intd, phi, theta, cnst;
y       = dta[.,2];
d       = b[1];
phi     = b[2];
theta   = b[3];
cnst    = 1;
retp(arfima(y,d,phi,theta,cnst));
endp;
```

This example demonstrates how a (1,d,1) ARFIMA model is estimated. d is the fractional dimension, and there is is a single AR coefficient (phi1) and a single MA coefficient (theta1). A constant is included.

**Source** likpak\src\arma.src

**See Also** arima, arma, mroot, varma

**References**

Box, G.E.P., Jenkins, G.M., and Reinsel, G. C. (1994). *Time Series Analysis, Forecasting and Control*, San Francisco: Holden-Day.

Doornik, J.A. and Ooms, M. (1999). "A package for estimating, forecasting and simulating ARFIMA models: Arfima package 1.0 for Ox", *Discussion paper*, Nuffield College, Oxford.

Sowell, F. (1992). "Maximum likelihood estimation of stationary univariate fractionally integrated time series models", *Journal of Econometrics*, Vol. 53, pp. 165-188.

**Purpose** Creates a vector of log likelihoods for an autoregressive integrated moving average process.

**Format** *llf* = **arima** ( *y*, *d*, *phi*, *theta*, *cnst* );

**Input**

| | | |
|---|---|---|
| *y* | is an Nx1 vector of the time series. |
| *d* | is a scalar, degree of differencing. |
| *phi* | is the *P*x1 AR coefficient vector, or scalar zero. |
| *theta* | is the *Q*x1 MA coefficient vector, or scalar zero. |
| *cnst* | is a scalar constant indicator. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The ARIMA $(p, d, q)$ process is given by:

$$\phi(L)(1 - L)^d y_t = \theta(L)\epsilon_t$$

where:

$$\begin{aligned} \phi(L) &= 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p \\ \theta(L) &= 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q \end{aligned}$$

and where $L$ is the backward shift operator, and $d$ is a non-negative integer.

The coefficients of the ARIMA process are estimated using ML. $d$ is an integer constant. When $d = 0$, this becomes the ARMA model. $y$ should be detrended, and have zero mean.

The constant indicator specifies whether a constant is to be included - 0, no constant, 1, constant. A constant should normally be specified for non-differenced series with non-zero mean, unless the constant is explicitly specified as a parameter.

Both stationary and invertibility conditions need to be satisfied. LIKPAK provides a routine called MROOT, which returns the value of the largest root, which must

have a modulus less than unity. Consequently, constrained optimization is usually required.

See the examples given in arma.e.

**Example**

```
proc  arima_1(b,dta);
local  y,  d, phi, theta, cnst;
y       = dta[.,1];
phi     = b[1];
theta   = b[2];
cnst    = 1;
d       = 1;
retp(arima(y,d,phi,theta,cnst));
endp;
```

This example demonstrates how a (1,d,1) ARIMA model is estimated. There is is a single AR coefficient (phi1) and a single MA coefficient (theta1). This model is estimated with one degree of differencing (d = 1), and with a constant. See the CML example for constraints.

**Source** likpak\src\arma.src

**See Also** arfima, arma, mroot, varma

**References** Box, G.E.P., Jenkins, G.M., and Reinsel, G. C. (1994). *Time Series Analysis, Forecasting and Control*, San Francisco: Holden-Day.

## ARMA

**Purpose** Creates a vector of log likelihoods for an autoregressive moving average process.

**Format** *llf* = **arma** ( *y*, *phi*, *theta*, *cnst* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the time series. |
| *phi* | is the *P*x1 AR coefficient vector, or scalar zero. |
| *theta* | is the *Q*x1 MA coefficient vector, or scalar zero. |
| *cnst* | is a scalar constant indicator. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The ARMA $(p, q)$ process is given by:

$$\phi(L)y_t = \theta(L)\epsilon_t$$

where:

$$\begin{aligned} \phi(L) &= 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p \\ \theta(L) &= 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q \end{aligned}$$

and where $L$ is the backward shift operator.

The coefficients of the ARMA process are estimated using ML. When there is no MA component, this becomes the AR model. *y* should be detrended, and have zero mean.

The constant indicator specifies whether a constant is to be included - 0, no constant, 1, constant. A constant should normally be specified for non-differenced series with non-zero mean, unless the constant is explicitly specified as a parameter.

Both stationary and invertibility conditions need to be satisfied. LIKPAK provides a routine called MROOT, which returns the value of the largest root, which must have a modulus less than unity. Consequently, constrained optimization is usually required.

See the examples given in arma.e.

**Example**

```
proc  arma_1(b,dta);
local  y,  intd, phi, theta, cnst;
y       = dta[.,1];
phi     = b[1];
theta   = b[2];
cnst    = 1;
retp(arma(y,phi,theta,cnst));
endp;
```

This example demonstrates how a (1,1) ARMA model is estimated. There is is a single AR coefficient (phi1) and a single MA coefficient (theta1). This model is estimated with a constant. See the CML example for constraints.

**Source** likpak\src\arma.src

**See Also** arfima, arima, mroot, varma

**References** Hamilton, J.D. (1994), *Time Series Analysis*, Ch. 11.

## Beta

**Purpose** Creates a vector of log likelihoods for a beta process.

**Format** *llf* = **beta_llf** ( *y*, *shape*1, *shape*2 );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable ($0 \leq y \leq 1$). |
| *shape1* | is the first positive shape parameter (scalar or Nx1 vector). |
| *shape2* | is the second positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The beta distribution is continuous in the interval [01]. The model can be used to estimate duration data. Commonly, the expected value of shape ($E(s_i)$) is parameterized as:

$$E(s_i) = \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, shape is forced positive. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **beta_pdf** ( *y*, *shape*1, *shape*2 );
*p* = **beta_cdf** ( *y*, *shape*1, *shape*2 );
*y* = **beta_cdfi** ( *p*, *shape*1, *shape*2 );

See the examples given in beta.e.

**Example**

```
proc  beta_1(b,dta);
local y, shape1, shape2, cen;
y      =  dta[.,1];
shape1 =  exp(dta[.,2:3]*b[1:2]);
```

```
shape2 =  exp(b[3]);
retp(beta_llf(y,shape1,shape2));
endp;
```

This example demonstrates how a `beta` model is estimated. The first shape parameter is parameterized with a linear index, while the second shape parameter is estimated as a parameter. The exponential function is used in each case to force both shape coefficients to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

## Boxcox

**Purpose** Creates a vector of log likelihoods for a BoxCox proces.

**Format** *llf* = **boxcox_llf** ( *y*, *indx*, *lamda* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *indx* | is an Nx1 vector of the index of the independent variables. |
| *lamda* | is the Nx1 vector or scalar BoxCox parameter. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The BoxCox transformation is given by:

$$boxcox(y, \lambda) = \frac{y^{\lambda}-1}{\lambda}$$

Thus this transformation is linear if $\lambda = 1$, and logarithmic if $\lambda = 0$. To allow for all values of $\lambda$, $y$ should be positive.

In the BoxCox model, the dependent variable is subject to the BoxCox transformation. The RHS of the model can be a linear or non linear function of the exogenous varaibles, and these variables can be subject to a BoxCox transformation, with the same or differing values of $\lambda$. See the example for details.

Note that the GAUSS function boxcox returns the BoxCox transformation, while boxcox_llf returns the log likelihood.

See the examples given in boxcox.e.

**Example**

```
proc  boxcox_1(b,dta);
local y,  indx, lamda;
y       =  dta[.,1];
indx    =  dta[.,2:4]*b[1:3];
lamda   =  b[4];
retp(boxcox_llf(y,indx,lamda));
endp;
```

This example demonstrates how a boxcox model is estimated. The structural

form of the RHS is given in `indx`, and a BoxCox transform is applied to the dependent variable, `y`, with value `lamda`.

**Source** likpak\src\econ.src

## Cauchy

**Purpose** Creates a vector of log likelihoods for a Cauchy process.

**Format** *llf* = **cauchy_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable ($0 \leq y \leq 1$). |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Cauchy distribution is unimodal and symmetric, with much heavier tails than the normal distribution. It is also known as the Lorentz distribution. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **cauchy_pdf** ( *y*, *loc*, *scale* );
*p* = **cauchy_cdf** ( *y*, *loc*, *scale* );
*y* = **cauchy_cdfi** ( *p*, *loc*, *scale* );

See the examples given in cauchy.e.

**Example**

```
proc  cauchy_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
indx   = dta[.,2:4]*b[1:3];
scale  = exp(b[4]);
```

```
retp(cauchy_llf(y,indx,scale));
endp;
```

This example demonstrates how a Cauchy model is estimated. The location parameter is parameterized with a linear index, while the scale parameter is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

## Censored

**Purpose** Creates the censored probability density function for the specified process.

**Format** *llf* = **censored** ( *pdf*, *cdf*, *cenvec* );

**Input**

| | |
|---|---|
| *pdf* | is an Nx1 vector of the pdf for the specified process. |
| *cdf* | is an Nx1 vector of the cdf for the specified process) . |
| *shape1* | is the first positive shape parameter (scalar or Nx1 vector). |
| *cenvec* | is an Nx1 vector of the censor variable, or scalar zero |

**Output** *llf* vector of pdf for the censored process.

**Remarks** Censoring occurs if units are removed prior to failure, or are still operating at the conclusion of the test (right censored). *cenvec* is the censor variable, each element of which takes a value of unity if the unit was censored, else zero. For such observations, the pdf is evaluated as the survival rate ($1 - cdf$).

Maximum likelihood estimation using data, some of which is subject to censoring, can be carried out using the log of the censored pdf.

**Example**

```
proc  expon_1(b,dta);
local y, scale, cen, pdf, cdf, llf;
y     = dta[.,1];
scale = exp(dta[.,2:4]*beta[1:3]);
cen   = dta[.,5];
pdf   = expon_pdf(y, scale);
cdf   = expon_cdf(y, scale);
llf   = ln(censored(pdf, cdf, cen));
retp(llf);
endp;
```

This examples shows the proc `expon_1` that would be called from maxlik. `y` is the dependent variable, and `scale` is parameterized as shown - by taking the exponent, `scale` is ensured positive. `cen` is the indicator vector of censoring. The log likelihood is the logarithm of the censored pdf of the exponential distribution.

**See Also** truncated

**Purpose** Creates a vector of log likelihoods for a chi-squared process.

**Format** *llf* = **chisq_llf** ( *y*, *shape* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable ($0 \leq y \leq 1$). |
| *shape* | is the positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The chi-squared distribution is a special case of the gamma distribution. It is frequently used for statistical inference. The shape parameter is the degrees of freedom - in this case, shape is parameterized, so we do not impose that shape is an integer . Commonly, the expected value of shape ($E(s_i)$) is parameterized as:

$$E(s_i) = \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, shape is forced positive. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **chisq_pdf** ( *y*, *shape* );
*p* = **chisq_cdf** ( *y*, *shape* );
*y* = **chisq_cdfi** ( *p*, *shape* );

See the examples given in chisq.e.

**Example**

```
proc  chisq_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
indx   = exp(dta[.,2:3]*b[1:2]);
```

```
retp(chisq_llf(y,indx));
endp;
```

This example demonstrates how a Chisq model is estimated. The shape parameter is parameterized with a linear index; the exponential function is used to force the shape coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a double-bounded dichotomous choice process.

**Format** *llf* = **dbdc** ( *r*, *indx*, *sigma*, *t* );

**Input**

| | |
|---|---|
| *r* | is the Nx2 response variable to the first and second questions. |
| *indx* | is the vector of fitted value for willingness to pay. |
| *sigma* | is the parameter for the standard error. |
| *t* | is the willingness to pay. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The DBDC coefficients are estimated using maximum likelihood; thus this can be used for linear or non-linear models. Given the unobserved latent willingness to pay (wtp) variable $y^*$, and the observed categorical response variable $r_1$ and $r_2$ to a first and second question, then the DBDC model is given by:

$$
\begin{aligned}
y^* &= f(x,\beta) + \epsilon \\
r_1 &= 1 \quad \text{if} \quad y^* > t_0 \\
r_2 &= 1 \quad \text{if} \quad r_1 = 1 \text{ and } y^* > t^u \\
&\qquad \text{or} \quad r_1 = 0 \text{ and } y^* > t^l
\end{aligned}
$$

The wtp vector, $t = \{t_0\ t^u\ t^l\}$, is a three element price vector. $r = \{r_1\ r_2\}$ is an $N$x2 response matrix. The first vector, $r_1$ is a response (yes/no) to whether the wtp exceeds $t_0$, while the second vector, $r_2$ is a response to whether the wtp exceeds $t^u$ or $t_l$, the choice depending on whether the wtp was greater or less than $t_0$. *xb* is the predicted value ($f(x,\widehat{\beta})$) of the latent variable – the structural equation for the latent variable can be linear or non-linear. $\epsilon$ is assumed distributed N(0,$\sigma^2$).

While *t* is usually fixed for all respondents, there are some circumstances where this is not the case. In such a circumstance, *t* can be specified as an $N$x2 matrix, where the first column is the price offered for the first question, and the second column is the price offered for the second question.

An example is given in dbdc.prg.

**Example**

```
proc dbdc_1(beta,dta);
local  r, xb, sigma, t;
r       =  dta[.,1:2];
xb      =  beta[1] + beta[2]*dta[.,3];
sigma   =  beta[3];
t       = { 5, 7, 3 };                // wtp amounts
retp( dbdc(r,xb,sigma,t));
endp;
```

In this example, the structural form for the latent variable is shown in (`xb`). The first question poses a bid value of 5, while the second question poses a value of 7 if the first question was a yes, else poses a value of 3. The first and second columns of `dta` are the categorical responses to each question.

**Source** likpak\src\discrete.src

**References**

Hanemann, W. M., J. Loomis, and B.J. Kanninen (1991), "Statistical Efficiency of Double-bounded Dichotomous Choice Contingent Valuation", *American Journal of Agricultural Economics*, Vol 73, pp.1255-63.

**Purpose** Creates a vector of log likelihoods for an EGARCH process.

**Format** *llf* = **agarch** ( *y*, *indx*, *bvec*, *gvec*, *pvec*, *mvec* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *indx* | is an Nx1 vector of the structural index. |
| *bvec* | is a $p + 1$ vector of parameters for the GARCH process. |
| *gvec* | is a $q$ vector of of parameters for lagged error process. |
| *pvec* | is a 3 element vector ($\theta_0$, $\gamma_0$, $\nu$) |
| *mvec* | is a 2 element vector of EGARCH–M parameters ($m_0$, $m_1$) |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The structural coefficients and the coefficients of Nelson's exponential GARCH or EGARCH process are estimated using maximum likelihood; The generalized form for the EGARCH $(p, q)$ process is:

$$\begin{aligned} y_t &= f(x_t, \beta) + \epsilon_t \\ \epsilon_t &\sim \sqrt{h_t} v_t \\ \ln h_t &= \beta_0 + \sum_{i=1}^{p} \beta_i \ln h_{t-i} + \sum_{j=1}^{q} \gamma_j (\theta_0 v_{t-j} + \gamma_0 [|v_{t-j}| - \mathrm{E}|v_t|]) \end{aligned}$$

The first equation describes the structural part of the model; thus this can be used for linear or non-linear structural models. This structural component should be first estimated using NLS to determine reasonable starting values. The residuals from this process are used by the EGARCH process.

The second equation specifies the distribution of the residuals – $v_t$ is assumed generalized error distributed (GED); this includes the normal as a special case, along with many other distributions with both fatter and thinner tails. When he tail parameter $\nu = 2$, $v_t$ is standard normal; for $\nu < 2$, $v_t$ has thicker tails than the normal, while for $\nu > 2$, $v_t$ has thinner tails than the normal.

The third equation specifies the structural form of the log of the conditional variance $\ln h$. The $\beta$ are the weights for the lagged $\ln h$ terms; this is the GARCH

process in $\ln h$. The second argument to the EGARCH command (¡bvec¿) contain the *p+1* elements of $\beta$, including the constant, $\beta_0$.

The $\gamma$ are the weights for the lagged disturbance $v$ terms; the third argument to the EGARCH command (gvec) are the $q$ elements of $\gamma$. Three additional parameters are specified in ¡pvec¿; these are $\theta_0$, $\gamma_0$, and the tail parameter $\nu$. It is typically the value of $\theta_0$ that permits the asymmetric volatility that is captured in the EGARCH model. Note that for the ¡q¿=1 case, there is an identification problem - this is solved by specifying $\gamma_1$ as a constant equal to unity.

The final argument, mvec, relates to the EGARCH–M process. The conditional variance can be introduced into the structural equation by adjusting the residual:

$$\epsilon_t \rightarrow \epsilon_t - m_0 h_t^{m_1}$$

The two parameters, $m_0$ and $m_1$ , are specified in mvec. A scalar zero is acceptable for no conditional variance in the structural equation.

The EGARCH model is very sensitive to starting values, and can easily blow up. Use OLS and GARCH to get sensible starting values.

See the "General Notes" under GARCH. An example is given in garch.e.

**Example**

```
proc egarch_1(b,dta);
local y,indx,bvec, gvec,pvec,mvec;
y       = dta[.,6];
indx    = findx(b,dta);
bvec    = b[3:4];
gvec    = 1;
pvec    = b[5]|b[6]|2;
mvec    = 0|1;
retp(egarch(y,indx,bvec,gvec,pvec,mvec));
endp;
```

In this example, a linear EGARCH(1,1) model is estimated, with a disturbance assumed distributed normal. The form of the conditional variance for this speci-

fication is:

$$\ln h_t = \beta_0 + \beta_i 1 \ln h_{t-1} + \theta_0 \frac{\epsilon_{t-1}}{\sqrt{h_{t-1}}} + \gamma_0 \left[ \frac{|\epsilon_{t-1}|}{\sqrt{h_{t-1}}} - \sqrt{2/\pi} \right]$$

Initial structural coefficients are derived using NLS. The residuals are specified Note parameter restrictions should be applied to ensure that the variance remains positive - `b[4]` in particular should be constrained. `nu`, the last element of`pvec` ($\theta_0$, $\gamma_0$, $\nu$) has been set to 2, implying that the disturbance is normal, and since it is a (1,1) process, `gvec` is set to unity. “”””””

**Source** likpak\src\garch.src

**See Also** garch

**References**

Engle, R.F., and V.K. Ng (1993), “Measuring and Testing the Impact of News on Volatility”, *Journal of Finance*, Vol. 48(5), pp. 1749-1778.

Nelson, D.B. (1993), “Conditional Heteroskedasticity in Asset Returns: A New Approach”, *Econometrica*, Vol. 59(2), 1993, pp. 347-370.

## Expon

**Purpose** Creates a vector of log likelihoods for an exponential process.

**Format** *llf* = **expon_llf** ( *y*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable ($0 \leq y \leq 1$). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The exponential model can be used to estimate duration data. Commonly, the expected value of scale ($E(s_i)$) is parameterized as:

$$E(s_i) = \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, scale is forced positive. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **expon_pdf** ( *y*, *scale* );
*p* = **expon_cdf** ( *y*, *scale* );
*y* = **expon_cdfi** ( *p*, *scale* );

See the examples given in expon.e.

**Example**

```
proc  expon_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
scale  = exp(dta[.,2:4]*b[1:3]);
retp(expon_llf(y,scale));
endp;
```

This example demonstrates how an exponential model is estimated. The scale parameter is parameterized with a linear index, and is forced positive using the `exp` function.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for an F process.

**Format** *llf* = **f_llf** ( *y*, *shape*1, *shape*2 );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable ($0 \leq y \leq 1$). |
| *shape1* | is the first positive shape parameter (scalar or Nx1 vector). |
| *shape2* | is the second positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The F-distribution arises frequently as the null distribution of a test statistic, such as in the analysis of variance, or in likelihood-ratio tests. Commonly, the expected value of shape ($E(s_i)$) is parameterized as:

$$E(s_i) = \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, shape is forced positive. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **f_pdf** ( *y*, *shape*1, *shape*2 );
*p* = **f_cdf** ( *y*, *shape*1, *shape*2 );
*y* = **f_cdfi** ( *p*, *shape*1, *shape*2 );

See the examples given in f.e.

**Example**

```
proc  f_1(b,dta);
local y, shape1, shape2;
y      = dta[.,1];
```

```
shape1 =  exp(dta[.,2:3]*b[1:2]);
shape2 = exp(b[3]);
retp(f_llf(y,shape1,shape2));
endp;
```

This example demonstrates how an F model is estimated. The first shape parameter is parameterized with a linear index, while the second shape parameter is estimated as a parameter. The exponential function is used to force the shape values to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

# FIGARCH

**Purpose** Creates a vector of log likelihoods for a fractionally integrated GARCH process.

**Format**

*llf* = **figarch** ( *y*, *indx*, *avec*, *bvec*, *gvec* );
*llf* = **figarch_t** ( *y*, *indx*, *avec*, *bvec*, *gvec dvec* );

**Input**

| | | |
|---|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *indx* | is an Nx1 vector of the structural index. |
| *avec* | is a vector of parameters for the ARCH process. |
| *bvec* | is a vector of parameters for the GARCH process. |
| *gvec* | is a dimension parameter. |
| *dvec* | is a distributional parameter ($\nu$). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The structural coefficients and the coefficients of the FIGARCH process are estimated using maximum likelihood. The FIGARCH model is given by:

$$\begin{aligned} y_t &= f(x_t, \theta) + \epsilon_t \\ \epsilon_t &\sim N(0, h_t) \\ h_t &= \alpha_0 + \Theta(L)\epsilon_t^2 + \sum_{j=1} \beta_j h_{t-j} \end{aligned}$$

where:

$$\begin{aligned} A(L) &= \alpha_1 L + \alpha_2 L^2 + ... + \alpha_q L^q \\ B(L) &= \beta_1 L + \beta_2 L^2 + ... + \beta_p L^p \\ \Phi(L) &= [1 - A(L) - B(L)](1 - L)^{-1} \\ \Theta(L) &= 1 - B(L) - \Phi(L)(1 - L)^d \end{aligned}$$

The first equation describes the structural part of the model; thus this can be used for linear or non-linear structural models. The second equation specifies the distribution of the residuals, and the third equation specifies the structural form of the conditional variance $h_t$. The $\beta$ are the weights for the lagged $h$ terms; this is the GARCH process. Both the $\alpha$ and the $\beta$ terms enter as weights for the lagged squared residual.

The first element of avec, which is required, gives the constant. gvec¿ is the dimension parameter ($d$) for the FIGARCH process; normally this parameter should lie between zero and unity. A value close to zero implies a long memory process, while a value close to unity implies a very short memory.

Note the stationarity conditions described under GARCH. In addition, the elements of $\Theta$ should be positive to ensure non-negative conditional variance. This can usually be ensured by requiring that $d + \alpha_1 > 1$

See the "General Notes" under GARCH. An example is given in garch.e.

**Example**

```
proc figarch_1(b,dta);
local y,indx,avec,bvec,gvec;
y       = dta[.,3];
indx    = findx(b,dta);
avec    = b[3:5];
bvec    = b[6];
gvec    = b[7];
retp(figarch(y,indx,avec,bvec,gvec));
endp;
```

In this example, a linear FIGARCH model is estimated. The structural equation is specified in findx. See the CML example for the the parameter restrictions to ensure that the variance remains positive.

**Source** likpak\src\garch.src

**See Also** garch

**References**

Baillie, R.T, T. Bollerslev, and H.O. Mikkelsen. (1996), "Fractionally integrated generalized autoregressive conditional heteroskedasticity", *Journal of Econometrics*, Vol 74, pp 3-30.

**Purpose** Creates a vector of log likelihoods for a multinomial probit process, without parameterization of the covariance matrix.

**Format** *llf* = **fmnp** ( *ycat*, *vmat* );

**Input**

| | |
|---|---|
| *ycat* | is an Nx1 vector or NxK matrix of the alternative chosen. |
| *vmat* | is the NxK matix of utility values for each alternative. |

**Global Input**

| | |
|---|---|
| *cmnpup* | scalar, update _vcmat every _cmnpup iterations (Default = 2) |
| *cmnpck* | scalar, carry out check for improving llf after vcmak: 0 - false, 1 - true. (Default = 1). |
| *mnpsc* | scalar, scaling option for MNP. |
| *mnpint* | scalar, integration algorithm MNP. |

**Output** *llf* vector of log likelihoods.

**Remarks** The structural coefficients and the coefficients of the FMNP process are estimated using maximum likelihood.

`ycat` is an *N*x1 vector in which is specified the alternative chosen for each observation. If ranked data is available, `ycat` is an *N*x*K* vector in which is specified the ranking for each alternative for each observation. Each utility is specified in a separate equation, and since utility differences are evaluated, the first utility is set to zero as a reference; `vmat` is the matrix formed by the concatenation of these utilities. The utilities can be functions of individual characteristics, (multinomial probit), choice characteristics (conditional probit), or a combination, and can be linear or non-linear.

The standard MNP procedure evaluates the probability of selecting the alternative specified in `ycat`. For each observation, the mean value (utility) associated with each alternative is stored in `ymat` The Random Utility Model assumes that the utilities are distributed with the specified mean, and an additive disturbance that is correlated across alternatives. In the MNP formulation, the distribution of these errors is multivariately normal, with a covariance matrix $\Sigma$. For a $K$ alternative model, there are $K^* = .5K(K-1)$ possible two choice combinations. Under FMNP, this covariance matrix is simulated based on the current structural parameter values and knowledge of the alternative actually chosen, or the ranking of

alternatives in the RMNP case. Integration of the multivariate density function is undertaken by QDFN. Exact estimation is the default, and is acceptably rapid for low $K$, or for the factor analytic case. For large $K$, simulation methods using the GHK algorithm are utilized. The QDFN globals must be set before a MNP estimation.

Since the utilities are estimated as differences, a reference is needed; usually this is achieved by setting the first utility equal to zero. Similarly, the scaling of the parameters is determined by the covariance matrix – in the FMNP estimation, the norm of the $K^*$ covariance elements is set to unity.

Note that FMNP requires initialization, and thus the command `Likset` must be called before each estimation. An example is given in mnp.e.

**Example**

```
proc  mnp_1(beta,dta);
local ycat, n, v1,v2, v3, xmat,sig;
ycat = dta[.,1];
x1   = dta[.,2];
x2   = dta[.,3];
x3   = dta[.,4];
n = rows(dta);
v1 = zeros(n,1);
v2 =  beta[1] + beta[2]*x1 + beta[3]*x2;
v3 =  beta[4] + beta[4]*x1 + beta[3]*x3;
xmat = v1~v2~v3;
retp(fmnp(ycat,xmat));
endp;
```

In this example, a linear mixed FMNP model is estimated. `x1` is an individual characteristic, while `x2` and `x3` are choice based characteristics. `ycat` should take values of 1, 2, or 3, depending on which alternative was selected for each observation.

**Source** likpak\src\fmnp.src

**See Also** mnl, mnp

**References**

Breslaw, J.A. (2002). "Multinomial Probit Estimation without Nuisance Parameters", *Econometrics Journal*, Vol. 5(2), pp. 417-434.

**Purpose** Creates a vector of log likelihoods for a frontier production function process.

**Format** *llf* = **fpf** ( *y*, *indx*, *sigma*, *lam* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *indx* | is the Nx1 vector of the index of the independent variables. |
| *sigma* | is the standard error of the residual. |
| *lam* | is $\sigma_\mu/\sigma_\nu$, the ratio of the standard errors. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The frontier production function coefficients are estimated using maximum likelihood. Given a production function, $f(x,\beta)$, the model is given by:

$$\begin{aligned} y &= f(x,\beta) + \epsilon \\ \epsilon &= \nu - \mu \end{aligned}$$

Thus the residuals from the production function, $\epsilon$, consist of two components, $\nu$ and $\mu$, where $\nu$ is $N(0,\sigma_\nu^2)$, and $\mu \geq 0$. The model can be estimated by determining two parameters, $s$, the standard error of $\epsilon$, and *lam*, the ratio of $\sigma_\mu$ to $\sigma_\nu$.

OLS can be used to get starting values of the structural coefficients and the standard error.

An example is given in fpf.e.

**Example**

```
proc fpf_1(b,dta);
local  y, xb, s, lam;
y       =  dta[.,1];
xb      =  dta[.,2:4]*beta[1:3];
s       =  b[4];
lam     =  b[5];
retp(fpf(y,xb,s,lam));
```

```
endp;
```

In this example, the coefficients of a linear equation is estimated for a FPF model.

**Source** likpak\src\econ.src

**References** Madalla, G.S (1983), *Limited-Dependent and Qualitative Variables in Econometrics*, Cambridge University Press, pp. 194-196.

**Purpose** Creates a vector of log likelihoods for a gamma process.

**Format** *llf* = **gamma_llf** ( *y*, *scale*, *shape* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable ($0 \leq y \leq 1$). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |
| *shape* | is the positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The gamma model can be used to estimate duration data. Commonly, the expected value of scale ($E(s_i)$) is parameterized as:

$$E(s_i) = \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, scale is forced positive. Similarly, to ensure that shape is positive, the shape parameter is estimated as an exponential function. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **gamma_pdf** ( *y*, *scale*, *shape* );
*p* = **gamma_cdf** ( *y*, *scale*, *shape* );
*y* = **gamma_cdfi** ( *p*, *scale*, *shape* );

See the examples given in gamma.e.

**Example**

```
proc  gamma_1(b,dta);
local y, scale, shape;
y      = dta[.,1];
scale  = exp(dta[.,2:4]*b[1:3]);
```

```
shape  = exp(b[4]);
retp(gamma_llf(y,scale,shape));
endp;
```

This example demonstrates how a gamma model is estimated. The scale parameter is parameterized with a linear index, while the shape parameter is estimated as a parameter. The exponential function is used to force both scale and shape to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**References**

King, G., J. Alt, N. Burns and M. Laver (1990). "A Unified Model of Cabinet Duration in Parliamentary Democracies," *American Journal of Political Science*, Vol. 34(3) pp. 846-871.

**Purpose** Creates a vector of log likelihoods for a GARCH process.

**Format**

*llf* = **garch** ( *y*, *indx*, *avec*, *bvec* );
*llf* = **garch_t** ( *y*, *indx*, *avec*, *bvec*, *dvec* );

**Input**

| | | |
|---|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *indx* | is an Nx1 vector of the structural index. |
| *avec* | is a vector of parameters for the ARCH process. |
| *bvec* | is a vector of parameters for the GARCH process. |
| *dvec* | is a distributional parameter ($\nu$). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Global Output**

| | |
|---|---|
| *_ht* | conditional variance. |

**Remarks** The structural coefficients and the coefficients of the GARCH process are estimated using maximum likelihood. The GARCH model is given by:

$$\begin{aligned} y_t &= f(x_t, \theta) + \epsilon_t \\ \epsilon_t &\sim N(0, h_t) \\ h_t &= \alpha_0 + \sum_{i=1} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1} \beta_j h_{t-j} \end{aligned}$$

The first equation describes the structural part of the model; thus this can be used for linear or non-linear structural models. The second equation specifies the distribution of the residuals, and the third equation specifies the structural form of the conditional variance $h_t$. The $\alpha$ are the vectors of the weights for the lagged $\epsilon^2$ terms; this is the ARCH process. The $\beta$ are the weights for the lagged $h$ terms; this is the GARCH process. Thus if $\alpha$ is just $\alpha_0$, and $\beta$ is zero, we have OLS; if $\alpha$ is a vector, and $\beta$ is zero, we have standard ARCH; otherwise we have some type of GARCH.

## General Notes

**Models** The following models are supported:

| | |
|---|---|
| ARCH | Single equation ARCH model. |
| GARCH | Single equation generalized ARCH model. |
| AGARCH | Single equation asymmetric GARCH model. |
| EGARCH | Single equation exponential GARCH model. |
| FIGARCH | Single equation fractionally integrated GARCH model. |
| IGARCH | Single equation integrated GARCH model. |
| MGARCH | Multiple equation multivariate GARCH model. |
| PGARCH | Single equation power GARCH model. |
| TGARCH | Single equation truncated GARCH model (GJR). |

**Formula Structure** For all these procedures, the residual is typically:

```
u = y - findx(b,dat);
```

If a moving average process is required, the residual would be (for an MA1 process):

```
e = recserar(u,u[1],theta);
```

Finally, the likelihood is given in the third formula

```
llf = garch(e,0, a1|a2, b1);
```

**Conditional variance** The conditional variance, $h_t$, is available as a global output called _ht - see the Likstat command. The structural formula for such a process would be given by:

```
u = y - s0 - s1*x1 - s2*x2 - thi*sqrt(_ht);
```

**Residual distribution** For the single equation models, the residuals are assumed distributed normal, with the exception of EGARCH, in which they are assumed to have a generalized error distribution (GED). The Student-t distribution can also be specified by calling GARCH_T, etc. An additional distribution parameter ($\nu$) is required.

**Parameter Constraints** GARCH processes normally require parameter constraints to ensure stationarity and nonnegativity of the conditional variances.

$$\begin{aligned} \alpha_0 &> 0 \\ \alpha_i &\geq 0 \\ \beta_i &\geq 0 \\ \sum_{i=1} \alpha_i + \sum_{j=1} \beta_j &< 1 \end{aligned}$$

These conditions can easily be imposed using CML.

An example is given in garch.e.

**Example**

```
proc garch_1(b,dta);
local y,indx,avec,bvec;
y       = dta[.,3];
indx    = findx(b,dta);
avec    = b[3:4];
bvec    = b[5];
retp(garch(y,indx,avec,bvec));
endp
```

In this example, a linear GARCH model is estimated, with the structural equation specified in `findx`. The parameter constraints discussed above should be implemented using CML.

**Source** likpak\src\garch.src

**References**

Engle, R.F. (1982), "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of the U.K. Inflation", *Econometrica*, Vol. 50, pp. 987-1007.

Bollerslev, T. (1986), "Generalized Autoregressive Conditional Heteroscedasticity", *Journal of Econometrics*, Vol. 31, pp. 307-327.

Gouieroux, C. (1997), *ARCH Models and Financial Applications*, Springer-Verlag, New York.

## Gumbel

**Purpose** Creates a vector of log likelihoods for a Gumbel process.

**Format** *llf* = **gumbel_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Gumbel distribution is equivalent to the Type 1 Largest Extreme Value distribution . The Gumbel model can be used to estimate duration data. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The Smallest extreme value is generated by reversing the sign of y.

The following functions are also supported:

*p* = **gumbel_pdf** ( *y*, *loc*, *scale* );
*p* = **gumbel_cdf** ( *y*, *loc*, *scale* );
*y* = **gumbel_cdfi** ( *p*, *loc*, *scale* );

See the examples given in gumbel.e.

**Example**

```
proc  gumbel_1(b,dta);
local y, scale, loc;
y      = dta[.,1];
```

```
loc    = dta[.,2:4]*b[1:3];
scale  = exp(b[4]);
retp(gumbel_llf(y,loc,scale));
endp;
```

This example demonstrates how a Gumbel model is estimated. The location parameter is parameterized with a linear index, while the scale parameter is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

## Invgauss

**Purpose** Creates a vector of log likelihoods for an inverse Gaussian process.

**Format** *llf* = **invgauss_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the positive location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The inverse Gaussian approaches the normal distribution when scale is large, and can be used to estimate duration data. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, scale is forced positive. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **invgauss_pdf** ( *y*, *loc*, *scale* );
*p* = **invgauss_cdf** ( *y*, *loc*, *scale* );
*y* = **invgauss_cdfi** ( *p*, *loc*, *scale* );

See the examples given in invgauss.e.

**Example**

```
proc  invgauss_1(b,dta);
local y, scale, loc;
y      = dta[.,1];
loc    = exp(dta[.,2:3]*b[1:2]);
```

```
scale  = exp(b[3]);
retp(invgauss_llf(y,loc,scale));
endp;
```

This example demonstrates how a `invgauss` model is estimated. The location parameter is parameterized with a linear index, while the scale parameter is estimated as a parameter. The exponential function is used to force both location and scale to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a non-linear least squares process.

**Format** *llf* = **kalman** ( *y*, *x*, *tmat*, *qmat*, *ctmat*, *hmat*, *csmat*, *b*0, *p*0 );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *x* | is an NxK matrix of the RHS variables |
| *tmat* | `btrans` matrix, or scalar zero. |
| *qmat* | `vtrans` matrix, or scalar zero. |
| *ctmat* | `ctrans` vector, or scalar zero. |
| *hmat* | `vmeas` scalar, or zero; |
| *csmat* | `cmeas` scalar, or zero. |
| *b0* | prior coefficients, or scalar zero. |
| *p0* | prior covariance matrix, or scalar zero. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Kalman filter model allows for the vector of coefficients $\beta$ in the classical linear model to randomly change over time. The model is specified in two parts - the measurement equation and the transition equation. The structural equation:

$$y_t = \alpha + X_t\beta_t + \epsilon_t \qquad \epsilon_t \sim N(0, H)$$

is standard, except that the $k$x1 vector $\beta$ changes over time. Note that it is assumed that the model is homoscedastic. The transition equation specifies the time path of $\beta$:

$$\beta_t = \gamma + T\beta_{t-1} + \mu_t \qquad \mu_t \sim N(0, \sigma^2 Q)$$

Thus once an initial value of $\beta_0$ is chosen, then the solution for $\beta$ at each time period (the state vector) will depend only on the matrix T, and the stochastic vector $\mu$. Conditional on the dependent variable $y$ and the independent variables $X$, the model can be evaluated once the system matrices are specified. These are:

$\alpha$ a scalar constant in the measurement equation.

| | |
|---|---|
| $H$ | the variance of the residuals in the measurement equation (assumed homoscedastic). |
| $\gamma$ | a $k$x1 vector of constants in the transition equation. |
| T | a $k$x$k$ matrix of transition coefficients. |
| Q | a $k$x$k$ symmetric matrix of the variance of the residuals in the transition matrix (up to a scalar factor). |
| $\beta_0$ | a $k$x1 vector of coefficients at time zero in the structural equation. |
| $\Omega_0$ | a $k$x$k$ symmetric matrix of the variance of $\beta_0$ (up to a scalar factor) . |

The program control options are given in the arguments to the `kalman` command:

*BTRANS* = matrix of transition coefficients (T). In the default, this is a $k$x$k$ identity matrix.

*VTRANS* = covariance matrix (Q) of the residuals in the transition equation. $\sigma^2$ is factored out of this matrix. In the default this is a $k$x$k$ identity matrix.

*CTRANS* = constant vector ($\gamma$) for the transition equation. In the default this is a $k$x1 vector of zeros.

*VMEAS* = constant, homoscedastic variance term (H) of the residuals in the measurement equation. Default is the identity matrix.

*CMEAS* = constant scalar ($\alpha$) for the measurement equation. In the default this is zero.

*B0* = vector of prior coefficients $\beta_0$ for the structural equation. `P0` must also be specified. In the default, $\beta_0$ is estimated from the first $k$ observations in the sample.

*P0* = covariance matrix ($\Omega_0$) of $\beta_0$. $\sigma^2$ is factored out of this matrix. `B0` must also be specified. In the default, this is set to $(X_k'X_k)^{-1}$ where $X_k$ is the matrix of RHS variables for the first $k$ observations of the sample.

The regression results reported by KALMAN are based on the residuals created at each time interval. The likelihood function is from Harvey (1981). The coefficients reported are the values of the evolving state vector at the last observation. Thus, if there are $n$ observations, then the coefficient values are $\beta_0$ after having been transformed by the transition equation $n$ times. If the default matrices are used, but Q is set to the null matrix, the Kalman process and recursive estimation are identical.

Note that Kalman requires initialization, and thus the command likset must be called before each estimation. An example is given in kalman.e.

**Example**

```
proc kalman_1 (b,dta);
local  y, x,  tmat, qmat, ctmat, hmat, csmat, b0, p0;

y       =  dta[.,1];
x       =  dta[.,2:4];
tmat    =  diagrv(eye(3),beta[1:3]);  // btrans
qmat    =  0;                         // vtrans
ctmat   =  0;                         // ctrans
hmat    =  0;                         // vmeas
csmat   =  0;                         // cmeas
b0      =  0;                         // b0
p0      =  0;                         // p0
retp(kalman(y,x,tmat, qmat, ctmat, ctmat, hmat, b0, p0));
endp;
```

In this example, the parameters of the a diagonal transition matrix (btrans) are estimated using the Kalman filter. All the other options take their default values.

**Source** likpak\src\kalman.src

**References**

Kalman, R. (1960), "A New Approach to Linear Filtering and Prediction Problems", *Journal of Basic Engineering, Transactions ASME, Series D*, Vol. 82, pp. 35-45.

Harvey, A,C. (1981), *Time Series Models*, Philip Allen, London.

Judge, G.G., *et. al.* (1985), *The Theory and Practice of Econometrics*, 2nd edition, Wiley, New York.

**Purpose** Creates a vector of log likelihoods for a Laplace process.

**Format** *llf* = **laplace_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Laplace distribution is also known as the double exponential distribution. The Laplace model can be used to estimate duration data. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **laplace_pdf** ( *y*, *loc*, *scale* );
*p* = **laplace_cdf** ( *y*, *loc*, *scale* );
*y* = **laplace_cdfi** ( *p*, *loc*, *scale* );

See the examples given in laplace.e.

**Example**

```
proc  laplace_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
indx   = dta[.,2:4]*b[1:3];
scale  = exp(b[4]);
```

```
    retp(laplace_llf(y,indx,scale));
    endp;
```

This example demonstrates how a `Laplace` model is estimated. The location parameter is parameterized with a linear index, while the scale parameter is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a Levy process.

**Format** *llf* = **levy_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is a positive Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Levy distribution is stable and has an analytic probability density function. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **levy_pdf** ( *y*, *loc*, *scale* );
*p* = **levy_cdf** ( *y*, *loc*, *scale* );
*y* = **levy_cdfi** ( *p*, *loc*, *scale* );

See the examples given in levy.e.

**Example**

```
proc  levy_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
indx   = dta[.,2:4]*b[1:3];
scale  = exp(b[4]);
retp(levy_llf(y,indx,scale));
```

```
endp;
```

This example demonstrates how a `Levy` model is estimated. The location parameter is parameterized with a linear index, while the scale parameter is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a logistic process.

**Format** *llf* = **logistic_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The logistic distribution has longer tails than the normal distribution, and was originally used for modeling population growth. It can be used to estimate duration data. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **logistic_pdf** ( *y*, *loc*, *scale* );
*p* = **logistic_cdf** ( *y*, *loc*, *scale* );
*y* = **logistic_cdfi** ( *p*, *loc*, *scale* );

See the examples given in logistic.e.

**Example**

```
proc  logistic_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
indx   = dta[.,2:4]*b[1:3];
```

```
scale  = exp(b[4]);
retp(logistic_llf(y,indx,scale));
endp;
```

This example demonstrates how a `logistic` model is estimated. The location parameter is parameterized with a linear index, while the scale parameter is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a binomial logit model.

**Format** *llf* = **logit** ( *y*, *indx* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the alternative chosen (0 or 1). |
| *indx* | is the Nx1 vector of the index (utility). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The probability of success ($y = 1$) in the linear binomial logit model is given by:

$$\Pr(y = 1) = \frac{\exp(-x\beta)}{(1 + \exp(-x\beta)}$$

More generally,

$$\Pr(y = 1) = \frac{\exp(-\text{ indx})}{(1 + \exp(-\text{indx})}$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

An example is given in logit.e.

**Example**

```
proc logit_1(b,dta);
local  y, indx;
y       =  dta[.,1];
indx    =  dta[.,2:4]*beta[1:3];
retp(logit(y,indx));
endp;
```

In this example, the coefficients of a linear index is estimated using a binomial logit model.

**Source** likpak\src\discrete.src

**See Also** mnl, probit

**Purpose** Creates a vector of log likelihoods for a log logistic process.

**Format** *llf* = **loglog_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The log logistic distribution can be used to estimate duration data. It is also known as the Fisk distribution. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **loglog_pdf** ( *y*, *loc*, *scale* );
*p* = **loglog_cdf** ( *y*, *loc*, *scale* );
*y* = **loglog_cdfi** ( *p*, *loc*, *scale* );

See the examples given in loglog.e.

**Example**

```
proc  loglog_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
indx   = dta[.,2:4]*b[1:3];
scale  = exp(b[4]);
```

```
    retp(loglog_llf(y,indx,scale));
    endp;
```

This example demonstrates how a `log logistic` model is estimated. The location parameter is parameterized with a linear index, while the scale parameter is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a log normal process.

**Format** *llf* = **lognorm** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is a positive Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** If *y* is log normally distributed, then ln(*y*) is normally distributed. The log normal distribution can be used to estimate duration data. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **lognorm_pdf** ( *y*, *loc*, *scale* );
*p* = **lognorm_cdf** ( *y*, *loc*, *scale* );
*y* = **lognorm_cdfi** ( *p*, *loc*, *scale* );

See the examples given in lognorm.e.

**Example**

```
proc  lognorm_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
indx   = dta[.,2:4]*b[1:3];
scale  = exp(b[4]);
```

```
retp(lognorm_llf(y,indx,scale));
endp;
```

This example demonstrates how a `log normal` model is estimated. The location parameter is parameterized with a linear index, while the scale parameter is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a multivariate GARCH process.

**Format** *llf* = **mgarch** ( *y*, *indx*, *x*, *c*0, *c*1, *amat*, *bmat* );

**Input**

| | |
|---|---|
| *y* | is an NxG matrix of the dependent variables. |
| *indx* | is an NxG matrix of the structural indices. |
| *x* | is a matrix of weakly exogenous variables. |
| *c0* | is a matrix of parameters for constants. |
| *c1* | is a matrix of parameters for *x*. |
| *amat* | is a matrix of parameters for the ARCH process. |
| *bmat* | is a matrix of parameters for the GARCH process. |
| *gvec* | is a vector of $\gamma$ parameters. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The structural coefficients and the coefficients of the MGARCH process are estimated using maximum likelihood. The Multivariate GARCH model is given by:

$$h_t = C_0 + C_1\tilde{x}_t + \sum_{i=1}^{q} A_i\eta_{t-i} + \sum_{i=1}^{p} B_i h_{t-i}$$

where

$$\begin{aligned} y_{jt} &= f_j(x_t,\beta_j) + \epsilon_{jt} \qquad j = 1,\ldots,G \\ \epsilon_t &\sim N(0,H_t) \\ h_t &= \text{vech}(H_t) \\ \tilde{x}_t &= \text{vech}(x_t x_t') \\ \eta_t &= \text{vech}(\epsilon_t\epsilon_t') \end{aligned}$$

In general, there are $G$ non-linear equations, with a residual vector $\epsilon_j$ for each equation. Based on these residuals, a conditional variance term, $H_t$ is estimated.

Two methods are available - the VEC formulation, and the BEKK formulation; $H_t$ under BEKK should stay positive definite, while this is not necessarily the case under VEC.

The conditional variance $h_t$ consists of four sets of terms; these are a constant ($C_0$), the parameters for the weakly exogenous variables ($C_1$), the ARCH process ($A_i$), and the GARCH process ($G_i$). MGARCH-M can also be carried out, since the conditional variance, $h_t$, is available, and stored in each iteration under the global _HT. The order of each of these depends on the process:

Given $G$ equations, there are $S = .5G(G + 1)$ elements to be estimated for each $H_t$. In addition, if there are J vectors $x$ of weakly exogenous variables, there are $M = .5J(J + 1)$ product pairs, and thus $M$ coefficients to be estimated.

Parameter Dimensions

| | VEC | | BEKK | |
|---|---|---|---|---|
| | Rows | Columns | Rows | Columns |
| $\epsilon$ | N | G | N | G |
| $x$ | N | J | N | J |
| $C_0$ | S | 1 | G | G |
| $C_1$ | S | M | G | M |
| $A_i$ | S | S | G | G |
| $G_i$ | S | S | G | G |

Note that for $C_0$ under BEKK, the $G$x$G$ matrix is upper triangular. Each additional ARCH or GARCH term requires an additional $A_i$ or $G_i$ respectively - see the example below.

The dependent variables and the structural indices are specified in the first two arguments of MGARCH. The estimation is rapid for a two equation system, where the GARCH term(s) are diagonal, since the process can be vectorized. For three or more equations, the conditional variance is derived recursively, which takes considerably longer.

The conditional variance (consisting of $S$ time series) for the MGARCH processes is retrieved The conditional variance (consisting of $S$ time series) for the

MGARCH process is stored in each iteration under the global _HT. (This needs to be uncommented in the GAUSS source code).

See the "General Notes" under GARCH. An example is given in mgarch.e.

**Example**

```
proc mgarch_1(b,dta);
local y, x1, x2, indx1, indx2, indx,
      c0, c1, x, amat, gmat;
y     =  dta[.,1:2];
x1    =  dta[.,3];
x2    =  dta[.,4];
indx1 =  b[1] + b[2]*x1;
indx2 =  b[3] + b[4]*x2;
indx  =  indx1~indx2;
x     =  0;
c0    =  abs(b[5]|0|b[6]);
c1    =  0;
amat  =  abs(b[7]~0~b[8]|0~0~0|0~0~b[11];
gmat  =  abs(diagrv(eye(3),b[9]|0|b[10]));
retp(mgarch(y,indx,x,c0,c1,amat,gmat));
endp;
```

In this example, a system of equations is estimated with residual variance specified as simultaneous GARCH. Although not shown, it makes sense to model each equation separately to get initial starting values. The model here is a GARCH(2,1) VEC process, without exogenous influences.

**Source** likpak\src\garch.src

**See Also** garch

**References** Engle, R.F., and K.F. Kroner (1995), "Multivariate Simultaneous Generalized ARCH", *Econometric Theory*, Vol. 11(1) pp. 122-150.

**Purpose** Creates a vector of log likelihoods for a multinomial logit model.

**Format** *llf* = **mnl** ( *y*, *vmat* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the alternative chosen. |
| *vmat* | is the NxK) matix of utility values for each alternative. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The multinomial logit model is based on the probability function

$$P_j = \frac{exp(U_j - U_k)}{\sum_j exp(U_j - U_k)}$$

where $P_j$ is the probability of selecting alternative $j$, $U_j$ is the utility associated with choice $j$, and $U_k$ is the maximum utility over the possible choices. `y` is a vector in which is specified the alternative chosen for each observation. Each utility is specified in a separate equation, and since utility differences are evaluated, the first utility is set to zero as a reference. `vmat` is the matrix formed by the concatenation of these utilities. The utilities can be functions of individual characteristics, (multinomial logit), choice characteristics (conditional logit), or a combination, and can be linear or non-linear.

An example is given in mnl.e.

**Example**

```
proc  mnl_1(beta,dta);
local ycat, n, v1,v2, v3, xmat;
ycat = dta[.,1];
x1   = dta[.,2];
x2   = dta[.,3];
x3   = dta[.,4];
n = rows(dta);
v1 = zeros(n,1);
v2 =  beta[1] + beta[2]*x1 + beta[3]*x2;
v3 =  beta[4] + beta[4]*x1 + beta[3]*x3;
xmat = v1~v2~v3;
```

```
retp(mnl(ycat,xmat));
endp;
```

In this example, a linear mixed MNL model is estimated. `x1` is a individual characteristic, while `x2` and `x3` are choice based characteristics. `ycat` should take values of 1, 2, or 3, depending on which alternative was selected for each observation.

**Source** likpak\src\discrete.src

**See Also** mnp

**References**

McFadden, D. (1976), " Conditional Logit Analysis of Qualitative Choice Behavior" in P. Zarembka, ed. *Frontiers in Econometrics*, Academic Press, New York.

**Purpose** Creates a vector of log likelihoods for a multinomial probit process.

**Format** *llf* = **mnl** ( *ycat*, *vmat*, *vcmat* );

**Input**

*ycat* is an Nx1 vector of the alternative chosen.

*vmat* is the NxK) matix of utility values for each alternative.

*vcmat*

1. .5$K(K-1)$x1 vector of unique elements in the differenced covariance matrix.
2. $K$x$K$ symmetric, positive definite covariance matrix of the $K$-variate normal density function.
3. $K$x$K$ Cholesky factor of the $K$x$K$ covariance matrix of the $K$-variate normal density function.
4. $K$x($R$+1) matrix for the factor analytic case, where covariance matrix has $R$ factors.

**Global Input**

*_mnpsc* scalar, scaling option for vcmat, such that ||*vcmat*|| = 1. This helps precision somewhat. (Default = 1),

*_mnpint* scalar, integration algorithm:1 - analytical, 2 - simulation. (Default = 1).

MNP uses QDFN to evaluate the multivariate normal integral, and thus the following globals are used - see QDFN for documentation.

*_qdfrep* scalar, the number of replications.
*_qdfrlz* scalar, the number of realizations.
*_qdford* scalar, the order of the integration

**Output** *llf* vector of log likelihoods.

**Remarks** The structural coefficients and the coefficients of the MNP process are estimated using maximum likelihood.

`ycat` is a vector in which is specified the alternative chosen for each observation. Each utility is specified in a separate equation, and since utility differences are evaluated, the first utility is set to zero as a reference; `vmat` is the matrix formed by the concatenation of these utilities. The utilities can be functions of individual

characteristics, (multinomial probit), choice characteristics (conditional probit), or a combination, and can be linear or non-linear.

The MNP procedure evaluates the probability of selecting the alternative specified in ycat. For each observation, the mean value (utility) associated with each alternative is stored in vmat. The Random Utility Model assumes that the utilities are distributed with the specified mean, and an additive disturbance that is correlated across alternatives. In the MNP formulation, the distribution of these errors is multivariately normal, with a covariance matrix $\Sigma$. For a $K$ alternative model, there are $K^* = .5K(K-1)$ possible two choice combinations, and it can be shown that, after allowing for scaling, there can be no more than $K^* - 1$ free parameters in the covariance matrix. This covariance matrix, vcmat, can be entered in the following formats:

1. As a $K^*$x1 vector of parameters, with one held as a constant. No other restrictions are necessary.
2. As a $K$x$K$ positive definite matrix, with $K^* - 1$ free parameters. Rank conditions must be satisfied.
3. As a $K$x$K$ Cholesky decomposition of a PD matrix, stored as an upper triangular matrix, with $K^* - 1$ free parameters. Rank conditions must be satisfied.
4. As a $K$x($R$+1) matrix of factors for the factor analytic case. $\Sigma = D + BB'$. The first column is a vector of variances in the diagonal matrix $D$, and the remaining columns are the $R$ factor loadings. See QDFN.SRC for a full description. Again there should be only $K^* - 1$ free parameters.

Integration of the multivariate density function is undertaken by QDFN. Exact estimation is the default, and is acceptably rapid for low $K$, or for the factor analytic case. For large $K$, simulation methods using the GHK algorithm are utilized. The QDFN globals must be set before a MNP estimation.

Since the utilities are estimated as differences, a reference is needed; usually this is achieved by setting the first utility equal to zero. Note also that identification is fragile in the MNP model without exclusion restrictions (Keane, 1992). The identification problem does not occur if the covariance matrix is specified, or if some explanatory variables do not occur in some of the utilities.

Note that MNP requires initialization, and thus the command `likset` must be called before each estimation. An example is given in mnp.e.

**Example**

```
proc  mnp_1(beta,dta);
local ycat, n, v1,v2, v3, xmat,sig;
ycat = dta[.,1];
x1   = dta[.,2];
x2   = dta[.,3];
x3   = dta[.,4];
n = rows(dta);
v1 = zeros(n,1);
v2 =  beta[1] + beta[2]*x1 + beta[3]*x2;
v3 =  beta[4] + beta[4]*x1 + beta[3]*x3;
xmat = v1~v2~v3;
sig = 1|beta[5]|beta[6];
retp(mnp(ycat,xmat,sig));
endp;
```

In this example, a linear mixed MNP model is estimated. `x1` is a individual characteristic, while `x2` and `x3` are choice based characteristics. `ycat` should take values of 1, 2, or 3, depending on which alternative was selected for each observation. Since $K$ is 3, $K^*$-1 is 2, and hence only two covariance parameters are free. In this example, the three parameters of the differenced covariance matrix ($K^*$) are specified, and one held constant for scaling.

**Source** likpak\src\mnp.src

**See Also** mnl, probit

**References**

Greene, W.H. (1993), *Econometric Analysis*, 2nd ed. Macmillan, New York.

Hajivassiliou, V.A., D. McFadden, and P. Ruud. (1992), "Simulation of Multivariate Normal Orthant Probabilities: Methods and Programs", Cowles Foundation Discussion Paper No. 1021, Yale University, Conn.

Hausman, J.A., and D.A. Wise. (1978). “Conditional Probit Models for Qualitative Choice: Discrete Decisions recognizing Interdependence and Heterogeneous Preferences”, *Econometrica*, Vol. 47, pp. 403-426.

Keane, M.P. (1992), “A Note on Identification in the Multinomial Probit Model”. *Journal of Business & Economic Statistics*, Vol. 10 (2), pp.193-200.

Maddala, G.S. (1983), *Limited-dependent and Qualitative Variables in Econometrics*, Cambridge University Press, Cambridge.

**Purpose** Creates a vector of log likelihoods for a Markov switching model.

**Format** *llf* = **msm** ( *y*, *index*, *sigma*, *prob*, *phi* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of dependent variable. |
| *index* | is an NxS matrix of structural indices. |
| *sigma* | is an Sx1 vector of standard deviation. |
| *prob* | is an S-1xS matrix of transition probabilities. |
| *phi* | is a Kx1 vector of AR parameters. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Global Output**

| | |
|---|---|
| *_mspm* | Markov transition probabilities. |
| *_msepv* | ergodic probability for full state vector. |
| *_mseps* | ergodic probability for primitive states. |
| *_msfp* | filtered probabilities. |
| *_mssp* | smoothed probabilities. |

**Remarks** The MSM coefficients are estimated using maximum likelihood; thus this can be used for linear or non-linear models. MSM allows a given variable to follow different time series processes over different subsamples. The choice of subsample is determined by a Markov process.

Assuming `S` states, the index matrix `index` will be an `NxS` matrix. The indices can be derived from a linear or non-linear structural equation. The residual is calculated as `y - index`. The standard deviation for each residual is parameterized in `sigma` - thus `sigma` will be an `Sx1` vector, or a scalar to restrict the same residual standard deviation across states.

The Markov transition matrix is specified as an `(S-1)xS` matrix of parameters - the last rows is determined residually so the probabilities sum to unity. To ensure positivity, the actual transition probabilities are derived from the square and norm of `prob`. The actual transition probabilities are available in the global `_mspm`.

Autoregressive terms are permitted - the order of the autoregressive structure is specified in `phi`. A value of zero implies no AR structure.

Global outputs can be retrieved using the Likstat command.

Note that MSM requires initialization, and thus the command `likset` must be called before each estimation.An example is given in msm.e.

**Example**

```
proc msm_1 (b,dta);
local  y, indx, sig, prob, phi;
y    =  dta[.,1];
indx = findx(b,dta);
sig  = b[4:5];                    // sig1|sig2
prob = b[6]~b[7];                 //  p11~p12
phi  = b[8:9];                    // phi1|phi2
retp(msm(y,indx,sig,prob,phi));
endp;


proc findx(b,dta);
local delgnp, c, indx1, indx2;
delgnp = dta[.,2];
c = ones(rows(delgnp),1);
indx1 = b[1]*c;
indx2 = (c~delgnp)*b[2:3];
retp(indx1~indx2);
endp;
```

In this example, a two state model is estimated with an AR(2) structure. The first regime is simply a constant (`indx1`), while the second regime (`indx2`) uses `delgnp` as a predictor of y.

**Source** likpak\src\msm.src

**References**

Hamilton, J. D. (1994), *Time Series Analysis*, Princeton University press, pp.685-689.

# MVN

**Purpose** Creates a vector of concentrated log likelihoods for a multivariate normal process.

**Format** *llf* = **mvn** ( *x* );

**Input** *x* is an NxK matrix of observations.

**Output** *llf* vector of log likelihoods.

**Remarks** MVN uses the concentrated likelihood, and so there is no estimate of scale. *x* is assumed distributed multivariate normal - typically *x* will be a residual (single equation) or matrix of residuals (system of equations). Thus

$$x_i = y_i - f(x_i, \beta)$$

The coefficients, $\beta$, are estimated using maximum likelihood; thus this can be used for linear or non-linear models. For a linear model, the estimates are equivalent to OLS.

MVN is useful, since it allows any non-linear least square problem to be cast as a maximum likelihood problem.

See the examples given in normal.e.

**Example**

```
proc  mvn_1(b,dta);
local y, loc, resid;
y      = dta[.,1];
loc    =  dta[.,2:4]*b[1:3];
resid  = y - loc;
retp(mvn(resid));
endp;
```

This example demonstrates how a least squares model is estimated. The structural model can be linear or non-linear, and it can be a single equation, or a system of equations. In this case, a single linear equation is estimated - the results will be the same as OLS.

**Source** likpak\src\mvn.src

**See Also** nls, normal

## Negbin

**Purpose** Creates a vector of log likelihoods for a negative binomial process.

**Format** *llf* = **negbin_llf** ( *y*, *v1*, *v2* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of integer dependent variable. |
| *v1* | is the first positive parameter (scalar or Nx1 vector). |
| *v2* | is the probability parameter ($0 \leq y \leq 1$) (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The negative binomial model is a generalization of the Poisson model, The dependent variable, `y` is a non-negative integer specifying the number of events. The first parameter *v1* is the number of failures before the *v1*th success, while the second parameter *v2* is the probability of success.

Gary King has suggested a parameterization where the conditional mean $\lambda_i$, given by:

$$\lambda_i = \exp(\beta' x_i).$$

In addition, allowing for cross-section heterogeneity, the conditional variance (the dispersion model) is given by:

$$\sigma_i^2 = \lambda_i(1 + \exp(\gamma' z_i)).$$

The following functions are also supported:

*p* = **negbin_pdf** ( *y*, *v1*, *v2* );
*p* = **negbin_cdf** ( *y*, *v1*, *v2* );
*y* = **negbin_cdfi** ( *p*, *v1*, *v2* );

See the examples given in negbin.e.

**Example**

```
proc negbin_1(b,dta);
local  y, ix1,ix2, v1,v2;
y     = dta[.,1];
```

```
ix1   = exp(b[1] + dta[.,3:5]*b[2:4]);
ix2   = exp(b[5]);
v1    = ix1./ix2;
v2    = 1./(1+ix2);
retp( negbin_llf(y,v1,v2));
endp;
```

This example demonstrates how a `negbin` model is estimated. The first parameter is the number of failures before the `v1`th success - in this case the ratio of an index scaled by a parameter, while the second parameter, `v2` is a probability. Nother the exponential function is used to force both arguments to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\count.src

**See Also** poisson

**References** King, G.(1989), "Variance Specification in Event Count Models: From Restrictive Assumptions to a Generalized Estimator", *American Journal of Political Science,*, Vol. 33(3), pp. 762-784.

**Purpose** Returns the fitted values of an artificial neural networks process.

**Format** *llf* = **neural** ( *y*, *x*, *amat*, *bmat*, *model* );

**Input**

| | |
|---|---|
| *y* | is an NxK matrix of the output variables. |
| *x* | is an NxK matrix of the input variables. |
| *amat* | is the parameter matrix of the hidden weights. |
| *bmat* | is the parameter matrix of the output weights. |
| *model* | is a list of parameter options. |

**Output**

| | |
|---|---|
| *z* | vector of log likelihoods. |

**Remarks** `Neural` is used to estimate the hidden and output weights of a neural networks process, using maximum likelihood.

An econometric formulation of a feed forward (ie. non-recursive) single hidden layer ANN is:

$$y_h = F\left(\beta_{h0} + \sum_{j=1}^{q} G(\widetilde{x}'\gamma_j)\beta_{hj}\right) \qquad h = 1, \ldots, g$$

where $y_h$ is a $g$x1 vector of endogenous variables, $\widetilde{x} = (1, x_1, \ldots, x_k)'$ is a $k$x1 vector of explanatory variables, $\gamma_j = (\gamma_{j0}, \gamma_{j1}, \ldots, \gamma_{jk})'$ is a $k+1$x1 vector of hidden weights, $q$ is the number of hidden units, $G$ is the transformation applied in the hidden layer, $\beta_h = (\beta_{h0}, \beta_{h1}, \ldots, \beta_{hq})'$ is a $q+1$x1 vector of output weights, and $F$ is the transformation applied in the output layer. (Observation subscripts are excluded for clarity). Commonly, $G(.)$ is sigmoid:

$$G(\widetilde{x}'\gamma) = 1/(1 + e^{-\widetilde{x}'\gamma})$$

though any mapping on the $\{0, 1\}$ space will do. If $y$ is continuous, $F(.)$ should be linear, ie. $F(z) = z$, while if $y$ is a limited dependent variable, $F(.)$ should also map to the $\{0, 1\}$ space. In econometric terms, this is a system of $g$ non-linear equations, with some common coefficients ($\gamma$) across equations.

*y* is the output variable, either an NxG matrix for the continuous case (CM), or an Nx1 vector of alternatives for the probability case (PM).

*x* is the NxK matrix of inputs

*amat* (K+1)xQ matrix of hidden weights (Q is number of hidden layers), or scalar zero.

*bmat* (Q+1)xG matrix of output weights, or (Q+K+1)xG matrix of augmented output weights.

*model* 6 element vector for model control, or scalar 0 (continuous model), or scalar 1 (prob model)

1. **Model type** 0 - continuous model(CM) [default], 1 - probability model (PM).
2. **Hidden layer** Specifies the transfer function carried out in the hidden layer,chosen from the following: 1 Arctan, 2 Gaussian, 3 Halfsine, 4 Linear, 5 Sigmoid, 6 Stepfn, 7 Tanh, 8 OLS (output layer only), 9 Cdfn. The default is sigmoid.
3. **Output layer** Specifies the transfer function carried out in the output layer. These are defined above. The default is `SIGMOID` for the PM, and `LINEAR` for the CM.
4. **Output scaling** Specifies the type of output scaling that is carried out. 1 Density - Output values sum to unity, 2 Null - Output values unadjusted, 3 Maximum - Index of output with max probability 4 Filter - Output value filtered (CM only). Null is the default under CM, and Density is the default under PM.
5. **Augmentation type** Specifies the type of model estimated. 1 Transfer - Transfer function only, 2 Augment - Linear + Transfer function. The default is the feedforward model consisting of a single hidden and a single output layer, with a transfer function for each. Under the augment option, the hidden layer consists of the sum of the hidden transfer function and a linear function of the inputs - consequently $\beta$ will have $k$ extra elements.
6. **Print summary** specifies whether a description of the `ANN` options actually used should be printed out. This is useful for debugging. 1 Print summary (default), 2 no print.

Under CM, when the output transfer function is linear, it is possible to express the output weights ($\beta$) in a closed form. This results in a significant reduction in the number of parameters that need to be estimated. This occurs when the output transfer function is specified as `OLS`.

Artificial neural networks may often have difficulty converging. In addition, initial values are important, and must be chosen in the context of the selected transfer functions. Start off with a small number of hidden units, and work up. Note that it is often possible to use random hidden weights, and to let the output weights do most of the work.

Note that `neural` requires initialization, and thus the command `likset` must be called before each estimation. An example is given in neural.e.

**Example**

```
proc neural_1(b,dta);
local nopt, y, x, amat, bmat, model, trans, scale, desc;
y       = dta[.,1:2];
x       = dta[.,3:5];
amat    = reshape(beta[1:8],4,2);
bmat    = reshape(beta[9:14],3,2);
model   = 0;
retp(neural(y,x, amat, bmat, model));
endp;

proc neural_2(b,dta);
local nopt, y, x, amat, bmat, model, trans, scale, desc;
y       = dta[.,1];
x       = dta[.,2:3];
amat    = reshape(beta[1:12],3,4);
bmat    = 0
model   = 0;
retp(neural(y,x, amat, bmat, model));
endp;

proc neural_3(b,dta);
local nopt, y, x, amat, bmat, model, trans, scale, desc;
p       = dta[.,1];
x       = dta[.,2:3];
amat    = beta[1:3];
bmat    = reshape(b[4:11],2,4);
model   = 1;
```

```
retp(neural(y,x, amat, bmat, model));
endp;
```

The first example shows a neural estimation of a continuous variable, three (k) input, two (g) output model, with 2 (q) units in the hidden layer. amat is a 4x2 ((k+1)xq) matrix of hidden weights, and bmat is a 3x2 ((q+1)xg) matrix of output weights. The default for a continuous model generates a sigmoid transformation at the hidden level, and no transformation nor scaling at the output level.

The second example shows ta neural estimation of a continuous variable, two (k) input, one (g) output model, with 4 (q) units in the hidden layer. amat is a 3x4 ((k+1)xq) matrix of hidden weights. The output weights are not specified since bmat is scalar zero, resulting in an OLS estimation of the output weights.

The third example shows a probability model estimation for the categorical variable case. There are two (k) inputs, one (q) units in the hidden layer, and four (g) outputs. p is a categorical variable, taking values of one through four, corresponding to the alternative selected. amat is a 3x1 ((k+1)xq) matrix of hidden weights, and bmat is a 2x4 ((q+1)xg) matrix of output weights. The default for the probability model generates a sigmoid transformation at the hidden and output level, and a scaling such that the sum of the outputs equals unity.

**Source** likpak\src\neural.src

**References**

Kuan, C.M., and H. White (1994), "Artificial Neural Networks: An Econometric Perspective", *Econometric Reviews*, Vol. 13 (1), pp. 1-91.

Webb, A.R., and D. Lowe (1988), "A hybrid optimisation strategy for adaptive feed-forward layer networks", RSRE Memorandum 4193, Royal Signals and Radar Establishment, Malvern, UK.

# NLS

**Purpose** Creates a vector of log likelihoods for a non-linear least squares process.

**Format** *llf* = **nls** ( *y*, *indx* );

**Input**

| | |
|---|---|
| *y* | is an NxK matrix of the dependent variables. |
| *indx* | is an NxK matrix of the RHS index. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The structural coefficients are estimated using maximum likelihood, under the assumption that the residuals are distributed normal. This ensures a minimum sum of squares or residuals, and thus permits the estimation of an equation or system of equations using maximum likelihood.

The NLS command can be used for a single equation (see cobbdouglass.e) or for a system of equations (see nls.e). In each case, the equation can be linear or non-linear.

**Example**

```
proc nls_12(b,dta);
local y, indx, xb1, xb2, ix;
y      = dta[.,1:2];
xb1 =  dta[.,3:5]*b[1:3];
ix = { 3,4,6,7 };
xb2 =  dta[.,ix]*b[1:4];
indx   = xb1~xb2;
retp(nls(y,indx));
endp;
```

This example shows how a system of linear equations is estimated using maximum likelihood.

**Source** likpak\src\mvn.src

**Purpose** Creates a vector of log likelihoods for a normal process.

**Format** *llf* = **normal_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale (std dev) parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The normal distribution is also known as the Gaussian distribution. Many physical phenomena can be approximated using the normal model by assuming that the total effect is the sum of independent additive effects. Similarly, the sampling distribution of the sample mean is approximately normal, even if the population distribution is unknown. It is the most widely used family of distributions, and many statistical tests are based on the assumption of normality.

The scale parameter - the standard deviation - must be positive. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models. For a linear model, the estimates are equivalent to OLS.

The following functions are also supported:

*p* = **normal_pdf** ( *y*, *loc*, *scale* );
*p* = **normal_cdf** ( *y*, *loc*, *scale* );
*y* = **normal_cdfi** ( *p*, *loc*, *scale* );

See the examples given in normal.e.

**Example**

```
proc  normal_1(b,dta);
local y, loc, scale, cen;
y      = dta[.,1];
loc    = dta[.,2:4]*b[1:3];
scale  = exp(b[4]);
retp(normal_llf(y,indx,scale));
endp;
```

This example demonstrates how a `normal` model is estimated. The location parameter is parameterized with a linear index, while the scale parameter (the standard deviation) is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**See Also** `MVN`

**Purpose** Creates a vector of conditional means or density based on a nonparametric or semiparametric estimation.

**Format** *llf* = **npe** ( *y*, *x*, *h*, *oplist* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the response variable. |
| *x* | is the kernel index, or independent variables. |
| *h* | is the window width. |
| *oplist* | is a 3x1 vector of program options. |

**Global Input**

| | |
|---|---|
| *_debug* | scalar, display options summary: 0 - false, 1 - true. (Default = 0) |
| *_kernel* | string, pointer to user supplied kernel name. |
| *_period* | vector, number of points in Fourier transform. (Default = 0) |
| *_weight* | scalar, weights used for smeared estimate [.25 .5 .25] |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Global Output**

| | |
|---|---|
| *_nperslt* | vector or matrix of forecasts. |

**Remarks** NPE (Non-Parametric Estimation) returns the likelihood for a general nonparametric or semiparametric procedure. NPE estimates a univariate Gaussian kernel if **x** is a vector, or a multiplicative Gaussian kernel if **x** is a matrix. Thus the metric that is used to weight the observations in the kernel is the Euclidean distance. The Gaussian kernel for the *m*–variate index is given by:

$$K_h(x_i - x^*) = \frac{1}{(2\pi)^{.5m}} \prod_{j=1}^{m} e^{-.5\left(\frac{x_j^* - x_{ji}}{h}\right)^2}$$

where $h$ is the window width, $x_i$ is the index for observation i, and $x^*$ is the reference index. Prior to being used in the kernel, the index is scaled to have unit variance, such that a single window width can be used. In the default, if ¡x¿ is a vector, the convolution of the data with the kernel will be undertaken using a fast Fourier transform (FFT); this is much faster than using direct calculation.

The window width, ¡h¿, is crucial in kernel estimations, since it determines the amount of smoothing undertaken. If $h$ is set to zero, $h$ is set automatically at:

$$h = \left(\frac{4}{n(m+2)}\right)^{1/(4+m)}$$

where $n$ is the number of observations, and $m$ is the number of columns in $x$. This value for ¡h¿ is optimal for density estimation under certain conditions. In general, it is better to estimate $h$ using cross-validation (see below).

NPE can be used to allow for an estimation of parameters, and/or window width. In the nonparametric case, the only parameter involved is the window width $h$, and this can be estimated using cross validation. Similarly, the parameters of the index can be estimated with known (or default) $h$ in a semiparametric context (without cross-validation), or both the index coefficients and the window width can be simultaneously using cross validation.

The maximum likelihood CV, which is used for all modes except conditional mean, is given by:

$$\text{MLCV}(h) = n^{-1} \sum_{i=1}^{n} \log \hat{f}_{-i}(x_i)$$

where the *leave one out* density estimate $\hat{f}_{-i}(x_i)$ is constructed from all the data points except $x_i$ :

$$\hat{f}_{-i}(x_i) = (n-1)^{-1} h^{-1} \sum_{j \neq i} K_h(x_i - x_j)$$

The window width, $h$ is then chosen to maximize MLCV.

Least squares cross-validation (LSCV) is used for estimating the Nadaraya-Watson conditional mean:

$$\hat{m}(x_i) = \frac{\sum_{j=1}^{n} K_h(x_i - x_j) y_j}{\sum_{j=1}^{n} K_h(x_i - x_j)}$$

The LSCV is specified as:

$$\text{LSCV}(h) = \sum_{i=1}^{n} (y_i - \hat{m}_{-i}(x_i))^2$$

where the *leave one out* conditional mean $\hat{m}_{-i}(x_i)$ is defined as:

$$\hat{m}_{-i}(x_i) = \frac{\sum_{j \neq i} K_h(x_i - x_j) y_j}{\sum_{j \neq i} K_h(x_i - x_j)}$$

Again, the window width is chosen to minimize LSCV. In both cases, if a semi-parametric approach, such as projection pursuit, is being used, then the parameters of the index ($x$) can be estimated concurrently with $h$. The sum of squares is converted to a likelihood using a multivariate normal transformation.

The program control options are specified in the 3 element vector `oplist`. The options available are:

*Cross-validate* 0 - no cross validation, 1 - cross validation.
Specifies whether the estimation process is to use cross validation (*leave-one-out*) or not. Cross-validation is required if $h$ is a parameter to be estimated.

*Method* 0 - fourier, 1 - direct.
Specifies whether the convolution is to be estimated using the fast Fourier transform, or direct estimation. If the index has one column, the default is Fourier. The number of points used in the FFT is given by `_periods`; the default is $2^m : 2^m > n$ where $n$ is the sample size. If the index has more than one column, the direct estimation process will used.

*Mode* 0 - conditional mean, 1 - density, 2 - discrete, 3 - smeared, 4 - frequency.
Specifies the type of estimation mode. Conditional mean returns an estimate of the Nadaraya-Watson conditional mean of the response variable $y$. Density ignores $y$ and returns the density $\hat{f}(x_i)$ for each point $i$. Frequency does the same, but without normalization, so that the totals do not sum to unity. Discrete takes a categorical variable $y : y \in \{1 \ldots m\}$, and returns the nonparametric probability of being in the observed category class for each observation. Smeared does the same as discrete, but assumes that the categories are ordered; the probability returned is the "smeared" probability over the neighboring categories. The default weighting is .25, .5, .25 centered on the observed category; the user can alter the weighting by specifying the elements in `_weight` - there must be an odd number of elements.

NPE uses the Gaussian kernel by default. The user can specify an alternative kernel by specifying a pointer in `_kernel` - for example,

```
_kernel = &kernproc;
```

where kernproc is a pointer to a procedure written by the user, and which takes the same arguments as proc gskernel. Only the direct estimation method will be utilized in this context.

In the default, the estimates are derived for all observations in the data set. If y contains missing values, then these observations are not used in the kernel. However, global outputs, which can be retrieved using the Likstat command, will be available for all values of x, which is useful for forecasting.

Note that NPE requires initialization, and thus the command likset must be called before each estimation. A number of examples of NPE estimation are given in npe.e and npe2.e. See npe.e for examples of using NPE for density and smoothing estimation.

**Example**

```
1. proc npe_1(b,dta);
   local y,x,h,oplist;
   y      = dta[.,1];
   x      = dta[.,3:4];
   h      = b[1];
   oplist = { 1, 1, 0 } ;  // cross validate, direct, cm
   retp(npe(y,x,h,oplist));
   endp;


2. proc npe_2(b,dta);
   local y,x,indx, a,h,oplist;
   y      = dta[.,1];
   x      = dta[.,2:4];     // includes const
   a      = b[1:3];
   h      = b[4];
   indx   = x*a;
   oplist = { 1, 0, 0 } ;  // cross validate, fourier, cm
      retp(npe(y,indx,h,oplist));
   endp;
```

```
3.  proc spe_1(b,dta);
    local y,x,indx,beta,h,oplist;
    y      = dta[.,1];
    x      = dta[.,2:4];
    beta   = b[1:3];
    h      = 0;
    indx   = x*beta;
    oplist = { 0, 0, 2 } ;  // nocv, fourier, discrete
    retp(npe(y,indx,h,oplist));
    endp;
```

The first example shows how the window width is estimated using a conditional mean and cross-validation. `x` is a multiple index, and so must be estimated directly - which is slower than using Fourier on an index. In the second example, a semi-parametric estimation occurs with a window width estimated concurrently (cross-validation), using the Fourier method. Only `b[3]` is an active parameter - the constant cannot be estimated in a semi-parametric context, and one normalization `b[2]` is required. The third example is a discrete estimation of the parameters of the kernel index. `y` is a discrete variable. The window width is not estimated - the default value is used - and so cross validation is not required. `b[1]` is not estimate as a normalization is required.

**Source** likpak\src\npe.src

**References**

Härdle W. (1990), *Applied Nonparametric Regression*, Cambridge University Press, New York.

Klein, R.W., and R.H. Spady (1993), "An Efficient Semiparametric Estimator of the Binary Response Model". *Econometrica*, Vol. 61 (2), pp. 387-421.

Nadaraya, E.A. (1964), "On estimating regression", *Theory Prob. Appl.*, Vol.10, pp. 186-190.

Silverman, B.W. (1982), "Algorithm AS 176. Kernel density estimation using the fast Fourier transform", *Applied Statistics*, Vol. 31, pp. 93-97.

Silverman, B.W. (1990), *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, London.

Watson, G.S. (1964), “Smooth regression analysis”, *Sankhyā, Series A*, Vol. 26, pp. 359-372.

**Purpose** Creates a vector of log likelihoods for an ordered logit model.

**Format** *llf* = **ordlgt** ( *y*, *indx* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the alternative chosen. |
| *indx* | is the NxK+1) matix of utility values for each alternative. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** For a $K$ choice model, `y` takes the value of the alternative chosen, ie. between 1 and $K$. `indx` is an $N$x$K+1$ matrix of utilities for each alternative for each observation. Thus the ordered logit model is identical to the logit model, except that the choice of constant in the utility is determined by the alternative chosen.The structural and threshold coefficients are estimated using maximum likelihood; that this can be used for linear or non-liner models.

An example is given in ordered.e.

**Example**

```
proc ordlgt_1(b,dta);
local  y, indx, shape,  alow, ahigh, aval, bval;
y     = dta[.,1];
alow  = -30;
ahigh = 30;
aval  = (alow|beta[1:4]|ahigh)';
bval  = beta[5:7];
indx  = aval - dta[.,2:4]*bval;
retp(ordlgt(y,indx));
endp;
```

In this example, there are five alternatives - hence `y` takes integer values in the range 1 to 5. For the ordered model with 5 choices, there are 6 delineation points, with the lowest being $-\infty$ and the highest being $+\infty$. Thus there are 4 threshold coefficients to be estimated, along with 3 structural coefficients (`bval`).

**Source** likpak\src\discrete.src

**See Also** ordprbt

**Purpose** Creates a vector of log likelihoods for an ordered probit model.

**Format** *llf* = **ordprbt** ( *y*, *indx* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the alternative chosen. |
| *indx* | is the NxK+1) matix of utility values for each alternative. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** For a $K$ choice model, `y` takes the value of the alternative chosen, ie. between 1 and $K$. `indx` is an $N$x$K$ + 1 matrix of utilities for each alternative for each observation. Thus the ordered probit model is identical to the probit model, except that the choice of constant in the utility is determined by the alternative chosen. The structural and threshold coefficients are estimated using maximum likelihood; that this can be used for linear or non-liner models.

An example is given in ordered.e.

**Example**

```
proc ordprbt_1(b,dta);
local  y, indx, shape,  alow, ahigh, aval, bval;
y     = dta[.,1];
alow  = -30;
ahigh = 30;
aval  = (alow|beta[1:4]|ahigh)';
bval  = beta[5:7];
indx  = aval - dta[.,2:4]*bval;
retp(ordprbt(y,indx));
endp;
```

In this example, there are five alternatives - hence `y` takes integer values in the range 1 to 5. For the ordered model with 5 choices, there are 6 delineation points, with the lowest being $-\infty$ and the highest being $+\infty$. Thus there are 4 threshold coefficients to be estimated, along with 3 structural coefficients (`bval`).

**Source** likpak\src\discrete.src

**See Also** ordlgt

**Purpose** Creates a vector of log likelihoods for a Pareto process.

**Format** *llf* = **pareto_llf** ( *y*, *loc*, *shape* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is an Nx1 location parameter (scalar or Nx1 vector) |
| *shape* | is the positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Pareto distribution was historically used to describe the allocation of wealth among individuals. The Pareto distribution has two parameters - location and shape. However, the maximum likelihood estimate of the location parameter is simply the minimum of *y*. Thus only shape is estimated. Typically, the expected value of shape ($E(s_i)$) is parameterized as:

$$E(s_i) = \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, shape is forced positive. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The maximum likelihood estimate of the location parameter is the minimum value of `y`, and thus `loc` is ignored in the `pareto_llf` call.

The following functions are also supported:

*p* = **pareto_pdf** ( *y*, *loc*, *shape* );
*p* = **pareto_cdf** ( *y*, *loc*, *shape* );
*y* = **pareto_cdfi** ( *p*, *loc*, *shape* );

See the examples given in pareto.e.

**Example**

```
proc  pareto_1(b,dta);
local y, loc, shape;
y      = dta[.,1];
shape  = dta[.,2:3]*b;
loc    = 0; // dummy
retp(pareto_llf(y,loc,shape));
endp;
```

This example demonstrates how a `Pareto` model is estimated. The shape parameter is parameterized with a linear index. The location parameter is not estimated using `pareto_llf`, since it is the minimum of `y`.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a Type III Pearson process.

**Format** *llf* = **pearson_llf** ( *y*, *loc*, *scale*, *shape* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |
| *shape* | is the positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Pearson family is a generalization of the normal distribution, and are used in modeling financial markets. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **pearson_pdf** ( *y*, *loc*, *scale*, *shape* );
*p* = **pearson_cdf** ( *y*, *loc*, *scale*, *shape* );
*y* = **pearson_cdfi** ( *p*, *loc*, *scale*, *shape* );

See the examples given in pearson.e.

**Example**

```
proc  pearson_1(b,dta);
local y, scale, loc;
y      = dta[.,1];
loc    = dta[.,2:3]*beta[1:2];
```

```
scale  = b[3];
shape  = b[4];
retp(pearson_llf(y,loc,scale,shape));
endp;
```

This example demonstrates how a `Pearson` model is estimated. The location parameter is parameterized with a linear index, while the scale and shape values are estimated as parameters.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a power GARCH process.

**Format**

*llf* = **pgarch** ( *y*, *indx*, *avec*, *bvec*, *gvec* );
*llf* = **pgarch_t** ( *y*, *indx*, *avec*, *bvec*, *gvec dvec* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *indx* | is an Nx1 vector of the structural index. |
| *avec* | is a vector of parameters for the ARCH process. |
| *bvec* | is a vector of parameters for the GARCH process. |
| *gvec* | is a 2 element parameter vector ($\gamma$ and $\delta$). |
| *dvec* | is a distributional parameter ($\nu$). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The structural coefficients and the coefficients of the PGARCH process are estimated using maximum likelihood. The PGARCH model is given by:

$$\begin{aligned} y_t &= f(x_t, \theta) + \epsilon_t \\ \epsilon_t &\sim N(0, h_t) \\ h_t &= \alpha_0 + \sum_{i=1} \alpha_i (|\epsilon_{t-i}| - \gamma \epsilon_{t-i})^\delta + \sum_{j=1} \beta_j h_{t-j} \end{aligned}$$

The first equation describes the structural part of the model; thus this can be used for linear or non-linear structural models. The second equation specifies the distribution of the residuals, and the third equation specifies the structural form of the conditional variance $h_t$. The $\alpha$ are the vectors of the weights for the lagged asymmetric $\epsilon^2$ terms; this is the ARCH process. The $\beta$ are the weights for the lagged $h$ terms; this is the GARCH process.

avec is a vector of parameters giving the weights for the lagged asymmetric squared residuals. The first element, which is required, gives the constant. bvec is the vector of parameters for the GARCH process. gvec is a two element vector of parameters for the asymmetric process consisting of $\gamma$ and $\delta$. Note the stationarity conditions described under GARCH.

See the "General Notes" under GARCH. An example is given in garch.e.

**Example**

```
proc pgarch_1(b,dta);
local y,indx,avec,bvec,gam;
y       = dta[.,4];
indx    = findx(b,dta);
avec    = b[3:4];
bvec    = b[5];
gam     = b[6];
retp(pgarch(y,indx,avec,bvec,gam));
endp;
```

In this example, a linear PGARCH model is estimated,with the structural equation specified in `findx`. Under CML, parameter restrictions would ensure that the variance remains positive.

**Source** likpak\src\garch.src

**See Also** garch

**References**

Ding, Z., R.F. Engle, and C.W.J. Granger. (1993), "A Long Memory Property of Stock Market Returns and a New Model", *Journal of Empirical Finance*, Vol 1 (1), pp 83-106.

**Purpose** Creates a vector of log likelihoods for a Poisson process.

**Format** *llf* = **poisson_llf** ( *y*, *v1* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of integer dependent variable. |
| *v1* | is the positive mean parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Poisson discrete probability distribution expresses the probability of a number of events occurring in a fixed period of time. The Poisson model, with mean parameter $\lambda$, is:

$$Prob(Y_i = y_i) = \frac{e^{-\lambda_i}\lambda_i^{y_i}}{y_i!}, \qquad y_i = 0, 1, 2, ...$$

where typically the expected value of $\lambda_i$ is parameterized as:

$$E(\lambda_i) = \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, $\lambda$ is forced positive. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **poisson_pdf** ( *y*, *v1* );
*p* = **poisson_cdf** ( *y*, *v1* );
*y* = **poisson_cdfi** ( *p*, *v1* );

See the examples given in poisson.e.

**Example**

```
proc poisson_1(b,dta);
local y, indx;
y    =  dta[.,1];
indx =  exp(b[1] + dta[.,3:5]*b[2:4]);
retp(poisson_llf(y,indx));
endp;
```

This example demonstrates how a `Poisson` model is estimated. $\lambda$ is parameterized with a linear index, and is forced positve using the exponential function.

**Source** likpak\src\count.src

**See Also** negbin

**Purpose** Creates a vector of log likelihoods for a multivariate binomial probit model.

**Format** *llf* = **probit** ( *y*, *indx*, *rvec* );

**Input**

| | |
|---|---|
| *y* | is an NxK vector of the alternative chosen (0 or 1) for each equation. |
| *indx* | is the NxK vector of the index (utility) for each equation. |
| *rvec* | is the vector of correlation coefficients |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The probability of success ($y = 1$) in the linear univariate binomial probit model is given by:

$$\Pr(y = 1) = \Phi(-x\beta)$$

where $\Phi$ is the cdf of the standard normal distribution. More generally,

$$\Pr(y = 1) = \Phi(-\text{indx})$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

For the univariate case, $K = 1$, and `rvec` is zero. For the bivariate case, $K = 2$, and `rvec` is the correlation coefficient. For the trivariate case, $K = 3$, and `rvec` consists of the three element correlation coefficient vector $r_{12}, r_{13}, r_{23}$.

An example is given in probit.e.

**Example**

```
proc probit_12(b,dta);
local y,i ndx1, indx2, xb, rho;
y      =  dta[.,1:2];
indx1  =  b[1] + dta[.,4:5]*b[2:3];
indx2  =  b[4] + dta[.,6:7]*b[5:6];
```

```
xb     =  indx1~indx2;
rho    =  beta[7];
retp(probit(y,xb,rho));
endp;
```

In this example, the coefficients of two linear indices are estimated using a bivariate binomial probit model.

**Source** likpak\src\discrete.src

**See Also** mnl, logit, mnp

**Purpose** Creates a vector of log likelihoods for a smallest extreme value process.

**Format** *llf* = **sev_llf** ( *y*, *loc*, *scale* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The smallest extreme value model can be used to estimate duration data. Commonly, the expected value of location ($E(loc_i)$) is parameterized as:

$$E(loc_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **sev_pdf** ( *y*, *loc*, *scale* );
*p* = **sev_cdf** ( *y*, *loc*, *scale* );
*y* = **sev_cdfi** ( *p*, *loc*, *scale* );

See the examples given in sev.e.

**Example**

```
proc  sev_1(b,dta);
local y, indx, scale, cen;
y      = dta[.,1];
indx   = dta[.,2:4]*b[1:3];
scale  = exp(b[4]);
retp(sev_llf(y,indx,scale));
```

```
endp;
```

This example demonstrates how a SEV model is estimated. The location parameter is parameterized with a linear index, while scale is estimated as a parameter. The exponential function is used to force the scale coefficient to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a Student's t process.

**Format** *llf* = **students_t_llf** ( *y*, *shape* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *shape* | is the positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** Also called the t-distribution. It is frequently used for statistical inference. The shape parameter is the degrees of freedom - in this case, shape is parameterized, so we do not impose that shape is an integer. Typically, the expected value of shape ($E(s_i)$) is parameterized as:

$$E(s_i) = 1 + \exp(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

By using the exponential function, shape is forced positive. The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **students_t_pdf** ( *y*, *shape* );
*p* = **students_t_cdf** ( *y*, *shape* );
*y* = **students_t_cdfi** ( *p*, *shape* );

See the examples given in students_t.e.

**Example**

```
proc  students_t_1(b,dta);
local y, shape;
y      = dta[.,1];
shape  = 1+exp(dta[.,2:3]*b[1:2]);
```

```
retp(students_t_llf(y,shape));
endp;
```

This example demonstrates how a Students_t model is estimated. Shape is parameterized as shown, which forces shape to be positive and greater than unity.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a stochastic volatility process.

**Format** *llf* = **sc** ( *y*, *indx*, *gvec* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *indx* | is an Nx1 vector of the structural index. |
| *gvec* | is a 3x1 vector of parameters for the SV process. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Global Output**

| | |
|---|---|
| *_ht* | vector of the log conditional variance. |

The coefficients of the SV process are estimated using quasi maximum likelihood, based on the Kalman filter algorithm. The SV model is given by:

$$\begin{aligned} y_t &= \sqrt{(h_t)}\epsilon_t \\ \epsilon_t &\sim N(0,1) \\ \log h_t &= \gamma_0 + \gamma_1 \log h_{t-1} + \sigma_v \upsilon_t \\ \upsilon_t &\sim N(0,1) \end{aligned}$$

The first equation describes the structure of the model. Typically, $y_t$ would be the residuals from a previously estimated model. The second and fourth equations specify the distribution of the residuals, while the third equation specifies the structural form of the conditional variance $h_t$. `gvec` consists of the three parameters in this equation: $\gamma_0$, $\gamma_1$, and $\sigma_v^2$.

The first equation can be transformed into:

$$\log y_t^2 = -1.27 + \log h_t + \eta_t$$

where $E(\eta_t) = 0$ and $V(\eta_t) = .5\pi^2$. This is the measurement equation, while the $\log h_t$ equation is the transition equation. When $\eta_t$ is approximated by a normal

distribution, we have a standard dynamic linear model, that can be estimated using the Kalman Filter algorithm.

y should be detrended and have zero mean. It is usually not a good idea to estimate structural parameters concurrently with the SV process, since there are significant identification issues, and hence index will normally be zero.

Note that $\sigma_v^2 > 0$ and stationarity requires that $|\gamma_1| < 1$. Consequently, constrained ML is usually required.

The log conditional variance, $h_t$, is available as a global output called _ht - see the Likstat command.

Note that SV requires initialization, and thus the command Likset must be called before each estimation. An example of SV is given in sv.e.

**Example**

```
proc sv_1 (b,dta);
local  resid, indx, gpar;
resid  = dta[.,1];
indx   = 0;
gpar   = b[1:3];
retp(sv(y,indx,gpar));
endp;
```

In this example, the first column of dta are the residuals from a previous regression. A stochastic volatility model is estimated using these residuals.

**Source** likpak\src\kalman.src

**See Also** kalman

**References**

Harvey, A.C., E. Ruiz, and N. Shephard. (1994), "Multivariate Stochastic Variance Models", *Review Econ. Studies*, Vol 61, pp 247-264.

Mills, T. (1999), *The Econometric Modelling of Financial Time Series*, 2nd ed. Cambridge University Press.

**Purpose** Creates a vector of log likelihoods for a truncated GARCH process (GJR).

**Format**

*llf* = **tgarch** ( *y*, *indx*, *avec*, *bvec*, *gvec* );
*llf* = **tgarch_t** ( *y*, *indx*, *avec*, *bvec*, *gvec dvec* );

**Input**

| | | |
|---|---|---|
| | *y* | is an Nx1 vector of the dependent variable. |
| | *indx* | is an Nx1 vector of the structural index. |
| | *avec* | is a vector of parameters for the ARCH process. |
| | *bvec* | is a vector of parameters for the GARCH process. |
| | *gvec* | is a vector of $\gamma$ parameters. |
| | *dvec* | is a distributional parameter ($\nu$). |

**Output**

| | | |
|---|---|---|
| | *llf* | vector of log likelihoods. |

**Remarks** The structural coefficients and the coefficients of the TGARCH process are estimated using maximum likelihood. The TGARCH or GJR model is given by:

$$\begin{aligned} y_t &= f(x_t, \theta) + \epsilon_t \\ \epsilon_t &\sim N(0, h_t) \\ \lambda_{it} &= \gamma_i \tilde{} \text{ if and only if } \epsilon_{t-i} < 0 \\ h_t &= \alpha_0 + \sum_{i=1}(\alpha_i + \lambda_{it})\epsilon_{t-i}^2 + \sum_{j=1}\beta_j h_{t-j} \end{aligned}$$

The first equation describes the structural part of the model; thus this can be used for linear or non-linear structural models. The second equation specifies the distribution of the residuals, and the third equation specifies the structural form of the conditional variance $h_t$. The $\alpha$ are the vectors of the weights for the lagged $\epsilon^2$ terms; this is the ARCH process. The $\beta$ are the weights for the lagged $h$ terms; this is the GARCH process.

avec is a vector of parameters giving the weights for the lagged asymmetric squared residuals. The first element, which is required, gives the constant. bvec is the vector of parameters for the GARCH process. gvec is a vector of parameters for the asymmetric process - the order of gvec should be one less than the order of avec.

Note the stationarity conditions described under GARCH.

See the "General Notes" under GARCH. An example is given in garch.e.

**Example**

```
proc tgarch_1(b,dta);
local y,indx,avec,bvec,gam;
y       = dta[.,4];
indx    = findx(b,dta);
avec    = b[3:4];
bvec    = b[5];
gam     = b[6];
retp(garch(y,indx,avec,bvec,gam));
endp;
```

In this example, a linear TGARCH model is estimated,with the structural equation specified in `findx`. Under CML, parameter restrictions would ensure that the variance remains positive.

**Source** likpak\src\garch.src

**See Also** garch

**References**

Glosten, L.R., R. Jagannathan, and D.E. Runkle (1993). "On the Relation between Expected Value and Volatility of the Nominal Excess Returns on Stocks", *Journal of Finance*, Vol 48, pp. 1779-1801.

**Purpose** Creates a vector of log likelihoods for a Tobit process.

**Format** *llf* = **tobit** ( *y*, *indx*, *sigmasq*);

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the censored variable. |
| *indx* | is the Nx1 vector of the index of the independent variables. |
| *sigmasq* | is the parameter for the residual variance. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Tobit coefficients are estimated using maximum likelihood; thus this can be used for linear or non-linear models. Given the unobserved latent variable $y^*$, and the observed variable $y$, then the Tobit model is given by:

$$\begin{aligned} y^* &= f(x,\beta) + \epsilon \\ y &= 0 \qquad \text{if } y^* \leq 0 \\ y &= y^* \qquad \text{if } y^* > 0 \end{aligned}$$

The dependent variable is treated as zero if $y$ takes non positive values. Models with upper or lower truncation points at values different from zero can be estimated by an appropriate transformation of the dependent variable, or by customizing the likelihood function.

An example is given in tobit.e.

**Example**

```
proc tobit_1(b,dta);
local y, indx, sigmasq;
y       = dta[.,1];
indx    = b[1] + dta[.,3:4]*b[2:3];
sigmasq = b[4];
retp(tobit(y,indx,sigmasq));
endp;
```

In this example, a standard linear Tobit model with truncation below zero is estimated. The RHS index is stipulated in `indx`.

**Source** likpak\src\econ.src

**References** Tobin, J. (1958), "Estimation of Relationships for Limited Dependent Variables", *Econometrica*, Vol. 26, pp. 24-36.

**Purpose** Creates the truncated probability density function for the specified process.

**Format** *llf* = **truncated** ( *pdf*, *cdfl*, *cdfr* );

**Input**

| | |
|---|---|
| *pdf* | is an Nx1 vector of the pdf for the specified process. |
| *cdfl* | is a scalar or Nx1 vector of the cdf of the left hand truncation for the specified process. |
| *cdfr* | is a scalar or Nx1 vector of the cdf of the right hand truncation for the specified process. |

**Output**

| | |
|---|---|
| *llf* | vector of pdf for the truncated process. |

**Remarks** Truncation occurs if the support for the specified function is restricted to the region between the left hand and right hand truncation points. The pdf of a truncated distribution is:

$$f(x|TL \leq x < TR) = \frac{f(x)}{(1 - F(TL))F(TR)}$$

wher $TL, TR$ are the left and right truncation points, and $F$ is the cumulative distribution function.

Maximum likelihood estimation can be carried out using the log of the truncated pdf.

**Example**

```
proc  expon_1(b,dta);
local y, scale, tl,tr, pdf, cdfl, cdfr, llf;
y     = dta[.,1];
scale = exp(dta[.,2:4]*beta[1:3]);
tl   = dta[.,5];
tr   = dta[.,6];
pdf   = expon_pdf(y, scale);
cdfl   = expon_cdf(tl, scale);
cdfr   = expon_cdf(tr, scale);
llf   = ln(truncated(pdf, cdfl,cdfr));
retp(llf);
endp;
```

This examples shows the proc `expon_1` that would be called from maxlik. `y` is the dependent variable, and `scale` is parameterized as shown - by taking the exponent, `scale` is ensured positive. `tl, tr` are vectors of left and right truncation points. (They could have been scalars). The log likelihood is the logarithm of the truncated pdf of the exponential distribution.

**See Also** censored

**Purpose** Creates a vector of log likelihoods for a vector autoregressive moving average process.

**Format** *llf* = **varma** ( *y*, *phi*, *theta*, *cnst* );

**Input**

| | |
|---|---|
| *y* | is an NxK vector of the time series. |
| *phi* | is the $P * K$xK AR coefficient vector, or scalar zero. |
| *theta* | is the $Q * K$xK MA coefficient vector, or scalar zero. |
| *cnst* | is a scalar constant indicator. |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The coefficients of the VARMA process are estimated using maximum likelihood. When there is no MA component, this becomes the VAR model. When only a single time series is specified, this becomes the ARMA model.

The constant indicator specifies whether a constant is to be included - 0, no constant, 1, constant. A constant should normally be specified for non-differenced series with non-zero mean, unless the constant is explicitly specified as a parameter.

Both stationary and invertibility conditions need to be satisfied. LIKPAK provides a routine called MROOT, which returns the value of the largest root, which must have a modulus less than unity. Consequently, constrained optimization is usually required.

See the examples given in varma.e.

**Example**

```
proc varma_1(b,dta);
local y, phi, theta, cnst;
y       =  dta[.,1:2];
phi     =  reshape(b[1:4],2,2);
theta   =  reshape(b[5,8],2,2);
cnst    =  1;
retp(varma(y,phi,theta,cnst));
endp;
```

This example demonstrates how a two variable VARMA model is estimated. P and Q are both set to unity, and thus $\phi$ and $\theta$ are both 2x2 matrices.. This model is estimated with a constant. See the CML example for constraints.

**Source** likpak\src\varma.src

**See Also** arfima, arima, arma, mroot

**References** Hamilton, J.D. (1994), *Time Series Analysis*, Ch. 11.

**Purpose** Creates a vector of log likelihoods for a Von Mises process.

**Format** *llf* = **vonmises_llf** ( *y*, *loc*, *shape* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable ($0 \leq y - loc \leq 2\pi$). |
| *loc* | is the location parameter (scalar or Nx1 vector) ($0 \leq loc \leq 2\pi$). |
| *shape* | is the positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** *y* is a circular random variate, ($-\pi \leq y - loc \leq \pi$). The von Mises distribution can be regarded as the circular analogue of the normal distribution on the line. Typically, the expected value of loc ($E(loc_i)$) is parameterized as:

$$E(loc_i) = 2\pi \text{ sigmoid}(indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

and:

$$\text{sigmoid}(x) = 1./(1 + \exp(-x))$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models.

The following functions are also supported:

*p* = **vonmises_pdf** ( *y*, *loc*, *shape* );
*p* = **vonmises_cdf** ( *y*, *loc*, *shape* );
*y* = **vonmises_cdfi** ( *p*, *loc*, *shape* );

See the examples given in vonmises.e.

**Example**

```
proc  vonmises_1(b,dta);
```

```
    local y, shape, loc;
    y      = dta[.,1];
    loc    = 2*pi*sigmoid(dta[.,2:3]*b[1:2])
    shape  = exp(b[3];
    retp(vonmises_llf(y,loc,shape));
    endp;
fn sigmoid(x) = 1./(1+exp(-x));
```

This example demonstrates how a `von Mises` model is estimated. The location parameter is parameterized with a linear index, and transformed into the range $0, 2\pi$ using the sigmoid function. Shape is estimated as a parameter, forced positive using the exponential function; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a Weibull process.

**Format** *llf* = **weibull_llf** ( *y*, *loc*, *scale*, *shape* );

**Input**

| | |
|---|---|
| *y* | is an Nx1 vector of the dependent variable. |
| *loc* | is the location parameter (scalar or Nx1 vector). |
| *scale* | is the positive scale parameter (scalar or Nx1 vector). |
| *shape* | is the positive shape parameter (scalar or Nx1 vector). |

**Output**

| | |
|---|---|
| *llf* | vector of log likelihoods. |

**Remarks** The Weibull distribution is often used in survival analysis due to its flexibility - it can be used instead of the normal or exponential function Commonly, the expected value of scale ($E(scale_i)$) is parameterized as:

$$E(scale_i) = (indx_i).$$

where the index is a function of explanatory variables, $x_i$:

$$indx_i = f(x_i, \beta)$$

The coefficients, $\beta$, of the index are estimated using maximum likelihood; thus this can be used for linear or non-linear models. The scale parameter only affects the intercept, and is usually set to unity.

The following functions are also supported:

*p* = **weibull_pdf** ( *y*, *loc*, *scale*, *shape* );
*p* = **weibull_cdf** ( *y*, *loc*, *scale*, *shape* );
*y* = **weibull_cdfi** ( *p*, *loc*, *scale*, *shape* );

See the examples given in weibull.e.

**Example**

```
proc  weibull_1(b,dta);
local y, scale, shape,loc;
y      = dta[.,1];
```

```
loc    = 0;
scale  = exp(dta[.,2:4]*b[1:3]);
shape  = exp(beta[4]);
retp(weibull_llf(y,loc,scale, shape));
endp;
```

This example demonstrates how a `Weibull` model is estimated. Location is set to zero. Scale is parameterized with a linear index, while shape is estimated as a parameter. The exponential function is used to force the both scale and shape to be positive; this could be done using parameter constraints with CML.

**Source** likpak\src\statistical.src

**Purpose** Creates a vector of log likelihoods for a local Whittle process.

**Format** *llf* = **whittle** ( *y*, *d*, *vopt* );

| | |
|---|---|
| *y* | is an Nx1 vector of the time series. |
| *d* | is the scalar memory parameter. |
| *vopt* | is a 3x1 vector of Whittle options, or scalar zero. |

**Output** *llf* vector of log likelihoods.

**Remarks** The local Whittle estimator is a semi-parametric estimator of the degree of differencing in a fractionally integrated process, based on the periodogram.

The fractionally integrated process is given by:

$$(1 - L)^d y_t = \epsilon_t \mathbf{1}\{t \geq 1\},\ t = 0, \pm 1, \ldots$$

where $L$ is the backward shift operator, **1{.}** is the indicator function and $d$ is the fractional degree of differencing. $d$ is estimated using maximum likelihood.

The local Whittle estimator involves the summation of the frequencies up to $2\pi m/n$ where $m$ is the truncation value. The default value is $n^{0.6}$

The program control options are specified in the 3 element vector `vopt`. The options available are:

*Whittle method* Specifies the type of model:
- 1 Local Whittle (default)
- 2 Exact Local Whittle
- 3 Feasible Exact Local Whittle

*Truncation length* Default = $n^{0.6}$.

*Padding* Specifies whether padding will occur for the Fourier transform. Given the sample size, the fast Fourier transform will always be used if padding is not required. Otherwise, if no padding is specified, the slower discrete Fourier transform will be used.
- 1 No padding
- 2 Padding (default)

An example is given in whittle.e.

**Example**

```
proc whittle_1(b, dta);
local y, d, vopt;
y       = dta[.,1];
d       = b[1];
vopt    = 2|80|0;
retp(whittle(y,d,vopt));
endp;
```

In this example, the memory parameter d for the time series y is estimated using an exact local Whittle estimator based on a truncation value of 80.

**Source** likpak\src\whittle.src

**References**

Robinson, P.M. (1995), "Gaussian semiparametric estimation of long range dependences", *Annals of Statistics*, Vol. 23, pp. 1630-1661.

Shimotsu, K and P.C.B Phillips (2005), "Exact Local Whittle Estimation of Fractional Integration", *Annals of Statistics*, Vol. 33, pp. 1890-1933.

# LikPak Utilities Reference 5

## DGP

**Purpose** To create a data vector or matrix of a particular type of stochastic process.

**Format** *y* = **dgp** ( *vstruct* );

**Input** *vstruct* is a DGPS structure,

**Output** *y* data vector or matrix.

**Remarks** DGP provides a method for creating a data vector or matrix of a particular type of process. The only input argument required is a data structure (`vstruct`), which is a structure of type DGPS. For each type of data generating process, only those elements of`vstruct` that are relevant need be specified.

A brief description of each process follows. In each case a structure of the form `struct DGPS vs;` is assumed:

| | | |
|---|---|---|
| `arch` | vs.index | structural component |
| | vs.arch | ARCH parameter vector |
| | vs.process | string: `arch` |
| `arch_t` | vs.index | structural component |
| | vs.arch | ARCH parameter vector |
| | vs.df | degrees of freedom for t distribution |
| | vs.process | string: `arch_t` |
| `arfima` | vs.ar | autoregressive parameter vector |
| | vs.ma | moving average parameter vector |
| | vs.diff | fractional differencing parameter |
| | vs.stderr | residual standard error |
| | vs.constant | process constant |
| | vs.process | string: `arfima` |

| | | |
|---|---|---|
| arima | vs.ar | autoregressive parameter vector |
| | vs.ma | moving average parameter vector |
| | vs.diff | integer differencing parameter |
| | vs.stderr | residual standard error |
| | vs.constant | process constant |
| | vs.process | string: arima |
| arma | vs.ar | autoregressive parameter vector |
| | vs.ma | moving average parameter vector |
| | vs.stderr | residual standard error |
| | vs.constant | process constant |
| | vs.process | string: arma |
| brownian | vs.diff | fractional parameter (Hurst) |
| | vs.process | string: brownian |
| garch | vs.index | structural component |
| | vs.arch | ARCH parameter vector |
| | vs.garch | GARCH parameter vector |
| | vs.process | string: garch |
| garch_t | vs.index | structural component |
| | vs.arch | ARCH parameter vector |
| | vs.garch | GARCH parameter vector |
| | vs.df | degrees of freedom for t distribution |
| | vs.process | string: garch_t |
| gaussian | vs.variance | variance |
| | vs.process | string: gaussian |
| linear | vs.index | structural component variable list |
| | vs.variance | residual variance |
| | vs.process | string: linear |
| linear_t | vs.index | structural component variable list |
| | vs.variance | residual variance |
| | vs.df | degrees of freedom for t distribution |
| | vs.process | string: linear_t |

| | | |
|---|---|---|
| logit | vs.index | structural utility variable list |
| | vs.prob | % data in alternative #1 (binomial only) |
| | vs.variance | disturbance variance vector or matrix |
| | vs.process | string: logit |
| poisson | vs.index | structural component ($\ln \lambda$) |
| | vs.process | string: poisson |
| probit | vs.index | structural utility variable list |
| | vs.prob | % data in alternative #1 (binomial only) |
| | vs.variance | disturbance variance matrix |
| | vs.process | string: probit |
| tobit | vs.index | structural component |
| | vs.stderr | residual standard error |
| | vs.process | string: tobit |
| wiener | vs.process | string: wiener |

**General Notes**

BROWNIAN If diff, the Hurst parameter, is not specified, a standard Brownian process is generated. Otherwise, a fractional Brownian process is generated for $0 < \text{diff} < 1$.

GARCH ARCH and GARCH processes can generate the conditional variance as a GAUSS global stored under the name _ht.

LINEAR For the linear model, a single equation is implied if a single index variable is specified. For the normally distributed error, either vs.stderr or vs.variance must be specified, while for the t_distribution, vs.df is required.

A multivariate normal or t distribution is implied if there is more than a single index variable - each variable represents the structural form for that equation. Both distributions require a residual variance specified in vs.variance, while for the t distribution, vs.df is required. This DGP returns an endogenous variable for each equation, and so the GENR statement cannot be used in this particular case. The results are returned in the variable list specified in vs.vlist.

QR For the logit and probit processes, binomial logit and probit

is implied if a single index variable is specified. vs.prob is the mean probability of alternative 1. The logit and probit models assume the disturbances are distributed with a Weibull or normal distribution respectively, which requires scaling by specifying either vs.scale (logit), vs.stderr or vs.variance. This DGP returns a categorical vector with elements of 0 and 1.

A multinomial logit or probit distribution is implied if there is more than a single index variable - each variable represents the structural utility associated with that alternative. For MNL, only the diagonal elements of vs.variance are used. This DGP returns a categorical vector with elements {1..*k*}, where *k* is the number of alternatives.

An example is given in arma.e.

**Example**

```
#include likpak/src/likpak.sdf

1.   struct DGPS gs;
     gs.nobs = 100;
     gs.arch = .4 |.15 ;
     gs.garch = .3;
     gs.index = xb;
     gs.process = "garch";
     y = dgp(gs);

2.   struct DGPS qrs;
     xb = 4 +5*x1-3*x2;
     qrs.nobs = rows(xb);
     qrs.index = xb;
     qrs.prob = .4;
     qrs.stderr = 1;
     qrs.process = "probit";
     y = dgp(qrs);

3.   struct DGPS qrls;
     x0 = 0*c;
     x1 = 2-4*z1;
```

```
x2 = 3+5*ln(z3);
qrls.nobs = rows(x2);
qrls.index = x0 x1 x2;
qrls.scale = .25;
qrls.process = "logit";
ycat = dgp(qrls);

4.  let vmat[2,2] = .5 .2 .2 .8;
    struct DGPS ls;
    ls.nobs = 100;
    ls.index = xb1~xb2;
    ls.variance = vmat;
    ls.process = "linear";
    y =  dgp(ls);
```

The first example demonstrates the creation of a vector y consisting of a structural component ($10+2*x$) and a residual with a garch distribution, with the parameters for the arch process specified in `gs.arch` (the first element is the constant), and the parameters for the garch process specified in `gs.garch`. The second example shows how a binomial process is specified using DGP; 40% of the generated data will fall in category 1. The third example demonstrates a multinomial logit DGP, with 3 alternatives. Example 4 shows how a linear system is generated with correlated error structure. The structural components (the RHS) are in `xb1` and `xb2`, and the matrix of endogenous variables - `y` is created.

**Source** likpak\src\dgpx.src

**Purpose** Filters data using a variety of filters.

**Format** $y$ = **filter** ($ftype$, $x$, $p1$, $p2$, $y0$ );

| | | |
|---|---|---|
| | *ftype* | string, the name of the filter. |
| | *x* | *N*x*K* matrix, data to be filtered. |
| | *p1* | *P*x*K* matrix or scalar, first parameter for the specified filter. |
| | *p2* | *Q*x*K* matrix or scalar, second parameter for the specified filter. |
| | *y0* | *P*x*K* matrix or scalar, initial values. |
| **Output** | *y* | *N*x*K* matrix of filtered data. |

**Remarks** The available filters are:

ARMA `y = filter(<ärma;x,phi,theta,y0>);`

$$(1 - \phi(L))y \quad = \quad (1 + \theta(L))x$$

This is an autoregressive moving average filter. `phi` is the AR component, and `theta` is the MA component. The first elements of `y` are pre-specified with `y0`, which should have the same order as `phi`.

DEDIFF `y = filter(<ädediff;x,d,0,0>);`

This is an inverse difference filter, where `d` is the integer degree of differencing.

DETREND `y = filter(<ädetrend;x,0,0,0>);`

This is a detrending filter - `y` has zero mean, and is detrended.

DIFF `y = filter(<ädiff;x,d,0,y0>);`

$$y \quad = \quad (1 - L)^d x$$

This is a difference filter, where `d` is the degree of differencing - both integer and fractional differencing are supported. The first elements of `y` are pre-specified with `y0`, which should be of order `ceil(d)`.

hp `y = filter(<d̈etrend;x,w,0,0>);`
This is the Hodrick-Prescott filter, where `w` is a parameter that controls the trade off between fit and smoothness. `w = 0` implies that `y` has the same trend component as the original series. Suggested values for `w` are 100 for annual data, 1600 for quarterly, and 14400 for monthly data.

LINEAR `y = filter(l̈inear;x,a,b,y0);`

$$\begin{aligned} y_n &= b_1 * x_n + b_2 * x_{n-1} + \cdots + b_{nb+1} * x_{n-nb} \\ &\quad -a_1 * y_{n-1} - \cdots - a_{na} * y_{n-na} \end{aligned}$$

This is a one dimensional recursive digital filter. The data in vector `x` is filtered by vectors `a` and `b` to create `y`. The linear filter is an IIR (infinite impulse response) or recursive filter. Initial conditions are specified in `y0`.

STANDARD `y = filter(<s̈tandard;x,0,0,0>);`
This filter creates the standardized series `y` with zero mean and unit variance.

**Example**

```
library likpak;
```

1.
```
let a = .5 .3 .1;
let b = .2 ;
let y0 = 0 0 0;
y = filter("linear", x, a, b, y0);
```

2.
```
d = 1;
xd = filter("diff', x, d, 0, x[1]);
x2 = filter("dediff", xd, d, 0, 0);
```

Example 1 shows a linear filter - in effect an AR(3) process, but with the current and one period lag value of `x` being part of the process. Example 2 shows a first order differencing, followed by its inverse.

**Source** likpak\src\filterx.src

**Purpose** Resets LIKPAK global variables to default values.

**Format** **likset;**

**Input** none.

**Output** none.

**Remarks** While most of the maximum likelihood procedures used in LIKPAKdo not require initialization, there are a few that do require initialization, and for these procedures, LIKSET is required.

**Source** likpak\src\likutil.src

## Likstat

**Purpose** Sets a flag so that global variables are created in a likelihood call.

**Format** **likstat;**

**Input** none.

**Output** none.

**Remarks** A number of statistics are typically available, or can be easily calculated, during a likelihood call - for example, the GARCHconditional variance. It is inefficient to do this during an estimation run. However, once the parameters have been estimated, the likelihood procedure can be rerun after calling LIKSTAT to create these statistics as globals.

**Example**

```
1. {x1,f,g,h,ret} = maxprt(maxlik(dta,0,&garch_1,x0));

   // get  stats
   likstat;
   call garch_1(x1,dta);
   " conditional variance " _ht;

2. out1 = CMLmtprt(CMLmt(&garch_1,p0,d0,c0));

   // get  stats
   likstat;
   call garch_1(out1.par,d0,0);
   " conditional variance " _ht;
```

In the first example, a garch estimation is carried out using a procedure `garch_1`. After the procedure, the estimated parameters are stored in `x1`. The next call to `garch_1` after setting `likstat` creates the conditional variance as a global. Example 2 shows the same example for CMLMT.

**Source** likpak\src\likutil.src

**Purpose** Returns the modulus of the largest root of a vector or matrix.

**Format** $z$ = **mroot** (*phi* );

**Input** *phi* vector or matrix.

**Output** $z$ scalar modulus.

**Remarks** The stationarity conditions for a single equation AR process, with AR coefficients, $\phi$, the roots of the characteristic equation:

$$C(z) = 1 - \phi_1 z - phi_2 z^2 - \cdots - phi_p z^p = 0$$

require a modulus greater than 1, or "lie outside the unit circle". For a VAR or VARMA process, a similar set of conditions hold.

For an AR(1) process, this is equivalent to the requirement that $|\phi_1| < 1$. MROOT provides an equivalent test, which can be used in both a single equation and multi equation contexts. This test returns a largest root (z) which is less than unity if the process is stationary.

Similarly, a process with MA coefficients $\theta$ is invertible if the largest root of $\theta$ has a modulus less than unity.

An example is given in varma.e.

**Example**

```
struct cmlmtControl c0;
c0 = cmlmtcontrolcreate;
c0.IneqProc = &ineqp;  // roots less than unity

proc ineqp(struct PV p, struct DS d);
   local phi, theta, c;
   phi = pvUnpack(p,"phi");
   theta = pvUnpack(p,"theta");
   c = zeros(2,1);
   c[1] = 1-mroot(phi);
```

```
    c[2] = 1-mroot(theta);
    retp(c);
endp;
```

The ARMA process is estimated using CML where the constraints impose both stationarity and invertibility on the AR and MA coefficients respectively.

**Source** likpak\src\toolsx.src

**Purpose** Returns the smallest root for a set of correlation coefficients.

**Format** *z* = **pdroot** (*rho* );

**Input**

| | | |
|---|---|---|
| *rho* | correlation coefficient vector. |

**Output**

| | | |
|---|---|---|
| *z* | scalar, smallest root. |

**Remarks** PDROOT creates the correlation matrix from the correlation coefficients rho; this matrix is positive definite if all its characteristic roots are positive. PDROOT returns the smallest root. Thus if this root is greater than zero, the correlation matrix is positive definite.

An example is given in probit.e.

**Example**

```
struct cmlmtControl c0;
c0 = cmlmtcontrolcreate;
c0.IneqProc = &ineqp;          // correlation matrix PD

proc ineqp(struct PV p, struct DS d);
  local rho, c;
  rho    =  pvUnpack(p,"rho");
  c      =  pdroot(rho)-.0001;
  retp(c);
endp
```

This example (from the estimation of a trivariate probit model), restricts the correlation coefficients to lie in the prescribed range.

likpak\src\toolsx.src

**Purpose** Integrates the K-variate normal density function over a range of upper and lower bounds.

**Format** *y* = **qdfn** ( *xh*, *xl*, *omega* );

**Input**

| | |
|---|---|
| *xh* | *K*x1 or *K*x*N* matrix, the upper limits of the *K*-variate normal density function. |
| *xl* | *K*x1 or *K*x*N* matrix, the lower limits of the *K*-variate normal density function. |
| *omega* | *K*x*K* symmetric, positive definite covariance matrix of the *K*-variate normal density function for the exact or simulation case. *K*x(*R* + 1) for the factor analytic case, where the covariance matrix has *R* factors. |

**Global Input**

| | |
|---|---|
| *_qdfmth* | global scalar, the choice of method:<br>*_qdfmth* = 0. The normal density function is evaluated using internal GAUSS functions if $K \leq 3$.<br>*_qdfmth* = 1. The probability is evaluated using a smooth recursive simulator. *K* is unrestricted.<br>*_qdfmth* = 2. The probability is evaluated using a factor analytic method providing the covariance matrix has three or less factors. *K* is unrestricted. |
| *_qdfrep* | global scalar, the number of replications. (20) |
| *_qdfrlz* | global scalar, the number of realizations. (1) |
| *_qdford* | global scalar, the order of the integration 2, 3, 4, 6, 8, 12, 16, 20, 24, 32, 40. (16) |

**Output**

| | |
|---|---|
| *y* | *N*x1 vector of the estimated integrals evaluated between the limits given by `xh` and `xl`. |

**Remarks** This procedure returns the probability of an *K*-dimensional rectangle where the probability density function is *K*-dimensional normal. It can evaluate the probability for a single set of upper and lower bounds, or can evaluate the probability for *N* points, providing the covariance matrix is constant.

The evaluation in the case when *K* is three or less can be carried out using Gauss functions. For *K* greater than 3, two methods are employed. The first is a factor analytic. This is only feasible if the covariance matrix has a limited factor

structure. That is, the covariance matrix ($\Omega$) can be written as:

$$\Omega = D + BB'$$

where $D$ is a $K$x$K$ diagonal matrix, and $B$ is a $K$x$R$ matrix, where $R$ is the number of factors. $R$ cannot be greater than 3. For example, for the single factor case, $B$ will be a $K$x1 vector. The factor analytic method is exact for those cases where the covariance matrix satisfies the relationship shown above, providing "_qdford" is set sufficiently high – the default is 16, but higher values may be necessary for the $R = 3$ case.

The second method that can be used for large $K$ is the method of simulation. Until recently, these methods have converged too slowly to be of sufficient interest. Recent work has resulted in a number of simulators being proposed that are consistent, and converge very quickly. The most promising is the smooth recursive simulator proposed by Geweke, Hajivassiliou and Keane (GHK). It is far slower than factor analytic for the same degree of accuracy, but can be used for any positive definite covariance matrix. The accuracy can be increased by increasing either *_qdfrep*, the number of replications, and/or *_qdfrlz*, the number of realizations.

**Example**

```
library likpak ;
let xh[3,2] = 1  1 2  0  3  1 ;
let xl[3,2] = 0 -2 0 -5  0 -2 ;
let b[3,2] = .3 .1 .5 -.3 .7 .6;
let d = 1 1.5 2 ;
omega = eye(3).*d + b*b';
vmat = d~b ;
_qdfmth = 0;
z0 = qdfn(xh,xl,omega)
_qdfmth = 1;
z1 = qdfn(xh,xl,omega)
_qdfmth = 2;
z2 = qdfn(xh,xl,vmat)
z = z0~z1~z2; z;
```

```
          0 -2            1   1
xl =      0 -5      xh =  2   0
          0 -2            3   1

          0.3  0.1        1.0
 b =      0.5 -0.3    d =  1.5
          0.7  0.6        2.0

               1.10  0.12  0.27             1.0  0.3  0.1
 omega =       0.12  1.84  0.17      vmat = 1.5  0.5 -0.3
               0.27  0.17  2.85             2.0  0.7  0.6

z =       0.072139   0.071460   0.072139
          0.251917   0.252609   0.251917
```

This integrates the two factor 3-variate normal density function over the specified range for two observations.

**Source** likpak\src\qdfn.src

**Purpose** Creates a matrix of (pseudo) random variables distributed truncated multivariate normal.

**Format** *y* = **rndtn** ( *xh*, *xl*, *mu*, *omega* );

**Input**

| | |
|---|---|
| *xh* | *K*x1 or *K*x*N* matrix, the upper limits of the *K*-variate normal density function. |
| *xl* | *K*x1 or *K*x*N* matrix, the lower limits of the *K*-variate normal density function. |
| *mu* | *K*x1 or *K*x*N* matrix, means of the *K*-variate normal density function. |
| *omega* | *K*x*K* symmetric, positive definite covariance matrix of the K-variate normal density function. |

**Global Input**

| | |
|---|---|
| *_rtnrep* | global scalar, the number of Gibbs replications (default = 20). |
| *_rtnpnt* | global scalar, 1 - print iteration number (default = 0). |

**Output**

| | |
|---|---|
| *y* | 1x*K* vector or *N*x*K* matrix of random numbers derived from the multivariate normal density function between the limits given by `xh` and `xl`. |

**Remarks** The truncated multivariate normal number generator is based upon the uniform number generator, and uses the same seed. The methodology uses the Gibbs Sampler, which is based on a Markov chain that utilizes univariate truncated normal densities to construct conditional variates, and has the truncated multivariate normal as its limiting distribution.

**Example**

```
library likpak ;
let xh = 2 1;
let xl = 0 -1;
let omega[2,2] = 1 .8 .8 1;
let mu[2,5] = 3  3  3  0  0
              0  0  0  0  0;
y = rndtn(xh,xl,mu,omega);
```

```
xh =  2        xl =  0    mu =     3  3  3  0  0
      1              -1             0  0  0  0  0


omega =  1.0   0.8
         0.8   1.0


y =    1.7296491       0.02386495
       1.9555895       0.14874314
       1.9291931      -0.42081598
       0.79121303      0.56267756
       0.34698671     -0.51624094
```

This simulates the bivariate truncated normal density function over the specified delineation range for five observations with specified means.

**Source** likpak\src\rndtn.src

**Purpose** Pauses program with message until a key is pressed.

**Format** **waitkey**(1);

**Input** none.

**Output** none.

**Remarks** Pauses for any key, with error message, and ability to stop execution.

**Source** likpak\src\likutil.src

# 6 DS Utilities Reference

## dsData

**Purpose** Stores data in a structure of type DS.

**Format** *d*0 = **dsdata** ( *d*0, *dta* );

**Input**

| | |
|---|---|
| *d0* | an instance of a structure of type DS. |
| *dta* | string or matrix, data name. |

**Output**

| | |
|---|---|
| *d0* | an instance of a structure of type DS. |

**Remarks** This command places a copy of the data into a DS structure. dta can be a matrix, or a string corresponding to a filename.

**Example**

```
struct DS p0;
d0 = dsCreate;
d0 = dsData(d0,dta);
```

In this example, a data matrix, dta is created in a DS structure.

**Source** likpak\src\likpak.src

**See Also** dsdataget

**Purpose** Retrieves data from a structure of type DS.

**Format** *d*0 = **dsdata** ( *d*0, *indx* );

**Input**

| | |
|---|---|
| *d0* | an instance of a structure of type DS. |
| *indx* | scalar or vector, data columns. |

**Output**

| | |
|---|---|
| *dta* | data matrix. |

**Remarks** This command places retrieves data elements from a DS structure. indx can be a scalar or vector.

**Example**

```
struct DS p0;
d0 = dsCreate;
d0 = dsData(d0,dta);
     .
y = dsDataGet(d0,1);
xindx = { 2, 3, 6 };
x = dsDataGet(d0,xindx);
```

In this example, a vector y is created from the first column of the data stored in a DS structure, and a matrix x from columns 2,3, and 6 of the same data.

**Source** likpak\src\likpak.src

**See Also** dsdata

## dsOptions

**Purpose** Stores options in a structure of type DS.

**Format** *d*0 = **dsoptions** ( *d*0, *options* );

**Input**

| | |
|---|---|
| *d0* | an instance of a structure of type DS. |
| *options* | options vector. |

**Output**

| | |
|---|---|
| *d0* | an instance of a structure of type DS. |

**Remarks** This command places a copy of an options vector into a DS structure.

**Example**

```
struct DS p0;
d0 = dsCreate;
oplist = { 2, 3, 5 };
d0 = dsData(d0,oplist);
```

In this example, an options vector, `oplist` is created in a DS structure.

**Source** likpak\src\likpak.src

**See Also** dsoptionget

**Purpose** Retrieve options from a structure of type DS.

**Format** *options* = **dsoptionget** ( *d*0, *indx* );

**Input**

| | |
|---|---|
| *d0* | an instance of a structure of type DS. |
| *indx* | scalar or vector, option elements. |

**Output**

| | |
|---|---|
| *options* | option values. |

**Remarks** This command retrieves options from a DS structure.

**Example**

```
struct DS p0;
d0 = dsCreate;
oplist = { 1, 2, 5.5 };
d0 = dsOptions(d0, oplist);
     .
scale = dsOptionGet(d0,1);
```

In this example, an options element, `scale` is retrieved as the first option in a DS structure.

**Source** likpak\src\likpak.src

**See Also** dsoptions

# 7 PV Utilities Reference

## pvClear

**Purpose** Clears matrices stored in a structure of type PV.

**Format** *p*0 = **pvclear** ( *p*0, *nm* );

**Input**

| | | |
|---|---|---|
| *p0* | an instance of a structure of type PV. |
| *nm* | string, matrix name. |

**Output**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |

**Remarks** This command deletes the specified matrix from the PV structure.

**Example**

```
struct PV p0;
p0 = pvCreate;
p0 = pvPack(p0,alpha,"alpha");
p0 = pvPack(p0,beta,"beta");
   .
   .
p0 = pvClear(p0,"alpha");
```

In this example, a PV structure is created with two matrices - alpha and beta. The alpha matrix is subsequently deleted.

**Source** likpak\src\pvutils.src

**Purpose** Sets the mask for a specified matrix to zeros (not active)in a structure of type PV.

**Format** *p*0 = **pvconst** ( *p*0, *nm* );

**Input**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |
| *nm* | string, matrix name. |

**Output**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |

**Remarks** This command specifies that the specified matrix in the PV structure is treated as a constant. This is carried out by setting the mask for that matrix to zeros. This is often useful when one wants to hold some parameters constant.

**Example**

```
struct PV p0;
p0 = pvCreate;
p0 = pvPack(p0,alpha,"alpha");
p0 = pvPack(p0,beta,"beta");
   .
   .
p0 = pvConst(p0,"alpha");
```

In this example, a PV structure is created with two matrices - alpha and beta. The alpha matrix is subsequently set as a constant.

**Source** likpak\src\pvutils.src

**See Also** pvparam

## pvGet

**Purpose** Gets the specified matrix from a structure of type PV.

**Format** *x* = **pvget** ( *p*0, *nm* );

**Input**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |
| *nm* | string, matrix name. |

**Output**

| | |
|---|---|
| *x* | MxN matrix. |

**Remarks** This command retrieves the specified matrix from the PV structure. It is the equivalent to the pvUnpack statement.

**Example**

```
struct PV p0;
p0   = pvCreate;
p0   = pvSet(p0,beta,"beta");
     .
b    = pvGet(p0,"beta");
```

This example shows how the current value of a matrix in a PV structure can be retrieved.

**Source** likpak\src\pvutils.src

**See Also** pvset

**Purpose** Gets the mask for a specified matrix from a structure of type PV.

**Format** *mask* = **pvgetmask** ( *p*0, *nm* );

**Input**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |
| *nm* | string, matrix name. |

**Output**

| | |
|---|---|
| *mask* | MxN matrix, mask matrix of zeros and ones. |

**Remarks** This command retrieves the mask for the specified matrix in the PV structure. The 1's in the mask matrix indicate an element stored in the packed matrix is a parameter, while a zero implies that the element is held as a constant.

**Example**

```
struct PV p0;
p0 = pvCreate;
beta = { 2 3, 4 5 };
p0   = pvPackm(p0,beta,"beta");
p0   = pvSetmask(p0,"beta",0~1|0~1);
mask = pvGetMask(p0,"beta");
```

This example shows how the current mask for a matrix in a PV structure can be retrieved.

**Source** likpak\src\pvutils.src

**See Also** pvsetmask

## pvParam

**Purpose** Sets the mask for a specified matrix to ones (active) in a structure of type PV.

**Format** *p*0 = **pvparam** ( *p*0, *nm* );

**Input**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |
| *nm* | string, matrix name. |

**Output**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |

**Remarks** This command specifies that the specified matrix in the PV structure is treated as parameters. This is carried out by setting the mask for that matrix to zeros. This is often useful when one wants to hold some parameters constant, and then subsequently make them parameters again.

**Example**

```
struct PV p0;
p0 = pvCreate;
p0 = pvPack(p0,alpha,"alpha");
p0 = pvPack(p0,beta,"beta");
p0 = pvConst(p0,"alpha");
  .
p0 = pvParam(p0,"alpha");
```

In this example, a PV structure is created with two matrices - alpha and beta. The alpha matrix is initially set as a constant, and subsequently set as a parameter.

**Source** likpak\src\pvutils.src

**See Also** pvconst

**Purpose** Sets the matrix and mask in a structure of type PV.

**Format** *p*0 = **pvset** ( *p*0, *nm*, *x*, *mask* );

**Input**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |
| *nm* | string, matrix name. |
| *x* | MxN matrix. |
| *mask* | MxN matrix, mask matrix of zeros and ones. |

**Output**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |

**Remarks** This command sets the matrix and mask in a PV structure. The 1's in the mask matrix indicate an element stored in the packed matrix is a parameter, while a zero implies that the element is held as a constant. The mask must either be the same order as x, or scalar zero (matrix elements are all constant) or scalar unity (matrix elements are all parameters). This command can be used both for new matrices, or to replace existing matrices of the same name.

**Example**

```
struct PV p0;
p0 = pvCreate;
beta = { 2 3, 4 5 };
mask = { 0 1, 0 1 };
p0 = pvSet(p0,alpha,"alpha",1);
p0 = pvSet(p0,beta,"beta",mask);
```

In this example, two matrices are stored within a PV structure. alpha is stored as parameters (mask is set to unity), while beta is set so that the first column of is treated as constants, while the second column is treated as parameters.

**Source** likpak\src\pvutils.src

**See Also** pvGet

## pvSetMask

**Purpose** Sets the mask for a specified matrix in a structure of type PV.

**Format** *p*0 = **pvsetmask** ( *p*0, *nm*, *mask* );

**Input**

| | |
|---|---|
| *p0* | an instance of a structure of type PV. |
| *nm* | string, matrix name. |
| *mask* | MxN matrix, mask matrix of zeros and ones. |

**Output** *p0* an instance of a structure of type PV.

**Remarks** This command sets the mask for the specified matrix in the PV structure. The 1's in the mask matrix indicate an element stored in the packed matrix is a parameter, while a zero implies that the element is held as a constant.

**Example**

```
struct PV p0;
p0 = pvCreate;
beta = { 2 3, 4 5 };
p0 = pvPack(p0,beta,"beta");
p0 = pvSetmask(p0,"beta",0~1|0~1);
```

In this example, a PV structure is created with the 2x2 matrix beta. The mask is set so that the first column of betais held as constants, while the second column is treated as parameters.

**Source** likpak\src\pvutils.src

**See Also** pvgetmask

# Index 8

# Index