Maximum Likelihood Estimation

Documentation Version: January 15, 2001

Contents

Chapter 1

Installation

1.1 UNIX

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

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

   `tar xvf` *device_name*

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

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

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

1.1.1 Solaris 2.x Volume Management

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

```
volcheck
```

to signal the system to mount the floppy.

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

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

To turn on volume management, become the superuser and type

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

1.2 DOS

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

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

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

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

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

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

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

   DEVICE=C:\DOS\ANSI.SYS

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

1.3 Differences Between the UNIX and DOS Versions

- In the DOS version, when the global **__output** = 2, information may be written to the screen using commands requiring the ANSI.SYS screen driver. These are not available in the current UNIX version, and therefore setting **__output** = 2 may have the same effect as setting **__output** = 1.

- If the functions can be controlled during execution by entering keystrokes from the keyboard, it may be necessary to press *Enter* after the keystroke in the UNIX version.

- On the Intel math coprocessors used by the DOS machines, intermediate calculations have 80-bit precision, while on the current UNIX machines, all calculations are in 64-bit precision. For this reason, **GAUSS** programs executed under UNIX may produce slightly different results, due to differences in roundoff, from those executed under DOS.

Chapter 2

Maximum Likelihood Estimation

MaxLik

written by

Ronald Schoenberg

This module contains a set of procedures for the solution of the constrained maximum likelihood problem

2.1 Getting Started

GAUSS 3.2.8+ is required to use these routines.

2.1.1 README Files

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

2.1.2 Setup

In order to use the procedures in the *MAXIMUM LIKELIHOOD* Module, the **MAXLIK** library must be active. This is done by including `maxlik` in the **library** statement at the top of your program or command file:

```
library maxlik,pgraph;
```

This enables **GAUSS** to find the *MAXIMUM LIKELIHOOD* procedures. If you plan to make any right hand references to the global variables (described in the *REFERENCE* section), you also need the statement:

```
#include maxlik.ext;
```

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

```
maxset;
```

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

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

_ml_ver 3×1 matrix, the first element contains the major version number of the *MAXIMUM LIKELIHOOD* Module, the second element the minor version number, and the third element the revision number.

If you call for technical support, you may be asked for the version number of your copy of this module.

2.1.3 Converting Older MAXLIK Command Files

The **MAXLIK** module includes a utility for processing command files to change global names of previos versions of **MAXLIK** (i.e., **MAXLIK 3.x** through **MAXLIK 3.1.4**) to the global names of **MAXLIK 4.x** and vice versa. This utility is a standalone executable program that is called outside of **GAUSS**. The format is,

chgvar *control_file target_directory file...*

The *control_file* is an ASCII file containing a list of the symbols to change in the first column and the new symbol names in the second column. The **MAXLIK** module comes with two control_files:

ml3toml4	**MAXLIK 3.x** to **MAXLIK 4.x**
ml4toml3	**MAXLIK 4.x** to **MAXLIK 3.x**

chgvar processes each file and writes a new file with the same name in the target directory.

A common use for **chgvar** is translating a command file that had been used before with **MAXLIK 3.x** to one that can be run with **MAXLIK 4.x**. For example:

```
mkdir new
chgvar ml3toml4 new max*.cmd
```

This would convert every file matching `max*.cmd` in the current directory and create a new file with the same name in the new directory.

The reverse translation is also possible. However, there are some global names in **MAXLIK 4.x** that don't have a corresponding global in **MAXLIK 3.x**, and in these cases no translation occurs.

Further editing of the file may be necessary after processing by **chgvar**.

You may edit the control files or create your own. They are ASCII files with each line containing a pair of names, the first column being the old name, and the second column the new name.

2.2 The Log-likelihood Function

MAXIMUM LIKELIHOOD is a set of procedures for the estimation of the parameters of models via the maximum likelihood method with general constraints on the parameters, along with an additional set of procedures for statistical inference.

MAXIMUM LIKELIHOOD solves the general maximum likelihood problem

$$L = \sum_{i=1}^{N} \log P(Y_i; \theta)^{w_i}$$

where N is the number of observations, $P(Y_i, \theta)$ is the probability of Y_i given θ, a vector of parameters, and w_i is the weight of the i-th observation.

The *MAXIMUM LIKELIHOOD* procedure **MAXLIK** finds values for the parameters in θ such that L is maximized. In fact **MAXLIK** minimizes $-L$. It is important to note, however, that the user must specify the log-probability to be *maximized*. **MAXLIK** transforms the function into the form to be minimized.

MAXLIK has been designed to make the specification of the function and the handling of the data convenient. The user supplies a procedure that computes $\log P(Y_i; \theta)$, i.e., the log-likelihood, given the parameters in θ, for either an individual observation or set of observations (i.e., it must return either the log-likelihood for an individual observation or a vector of log-likelihoods for a matrix of observations; see discussion of the global variable **__row** below). **MAXLIK** uses this procedure to construct the function to be minimized.

2.3 Algorithm

MAXIMUM LIKELIHOOD finds values for the parameters using an iterative method. In this method the parameters are updated in a series of iterations beginning with a starting values that you provide. Let θ_t be the current parameter values. Then the succeeding values are

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

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

Direction

Define

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

The direction, δ is the solution to

$$\Sigma(\theta_t)\delta = \Psi(\theta_t)$$

This solution requires that Σ be positive definite.

Line Search

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

2.3.1 Derivatives

The minimization requires the calculation of a Hessian, Σ, and the gradient, Ψ. **MAXLIK** computes these numerically if procedures to compute them are not supplied.

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

2.3.2 The Secant Algorithms

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

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

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

In addition, **MAXLIK** includes a scoring method, BHHH (Berndt, Hall, Hall, and Hausman). This method computes the gradient of the likelihood by observation, i.e., the Jacobian, and estimates Σ as the cross-product of this Jacobian.

Secant Methods (BFGS and DFP)

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

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

2.3.3 Berndt, Hall, Hall, and Hausman's (BHHH) Method

BHHH is a method proposed by Berndt, Hall, Hall and Hausman (1974) for the maximization of log-likelihood functions. It is a *scoring* method that uses the cross-product of the matrix of first derivatives to estimate the Hessian matrix.

This calculation can be time-consuming, especially for large data sets, since a gradient matrix exactly the same size as the data set must be computed. For that reason BHHH cannot be considered a preferred choice for an optimization algorithm.

2.3.4 Polak-Ribiere-type Conjugate Gradient (PRCG)

The conjugate gradient method is an improvement on the steepest descent method without the increase in memory and computational requirements of the secant methods. Only the gradient is stored, and the calculation of the new direction is different:

$$d_{t+1} = -g_{t+1} + \beta_t d_t$$

where t indicates t-th iteration, d is the direction, g is the gradient. The conjugate gradient method used in **MAXLIK** is a variation called the Polak-Ribiere method where

$$\beta_t = \frac{(g_{t+1} - g_t)' g_{t+1}}{g_t' g_t}$$

The Newton and secant methods require the storage on the order of the Hessian in memory, i.e., $8k^2$ bytes of memory, where k is the number of parameters. For a very large problem this can be prohibitive. For example, 200 parameters will require 3.2 megabytes of memory, and this doesn't count the copies of the Hessian that may be generated by the program. For large problems, then, the PRCG and STEEP methods may be the only alternative. As described above, STEEP can be very inefficient in the region of the minimum, and therefore the PRCG is the method of choice in these cases.

2.3.5 Line Search Methods

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

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

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

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

Given θ and d, this is a function of a single variable ρ. Line search methods attempt to find a value for ρ that decreases m. STEPBT is a polynomial fitting method, BRENT and HALF are iterative search methods. A fourth method called ONE forces a step length of 1. The default line search method is STEPBT. If this, or any selected method, fails, then BRENT is tried. If BRENT fails, then HALF is tried. If all of the line search methods fail, then a random search is tried (provided **_max_RandRadius** is greater than zero).

STEPBT

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

BRENT

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

HALF

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

BHHHStep

This is a variation on the golden search method. A sequence of step lengths are computed, interpolating or extrapolating using a golden ratio, and the method exits when the function decreases by an amount determined by **_max_Interp**.

2.3.6 Random Search

If the line search fails, i.e., no ρ is found such that $m(\theta_t + \rho\delta) < m(\theta_t)$, then a search is attempted for a random direction that decreases the function. The radius of the random search is fixed by the global variable, **_max_RandRadius** (default = .01), times a measure of the magnitude of the gradient. **MAXLIK** makes **_max_MaxTry** attempts to find a direction that decreases the function, and if all of them fail, the direction with the smallest value for m is selected.

The function should never increase, but this assumes a well-defined problem. In practice, many functions are not so well-defined, and it often is the case that convergence is more likely achieved by a direction that puts the function somewhere else on the hyper-surface even if it is at a higher point on the surface. Another reason for permitting an increase in the function here is that halting the minimization altogether is only alternative if it is not at the minimum, and so one might as well retreat to another starting point. If the function repeatedly increases, then you would do well to consider improving either the specification of the problem or the starting point.

2.3.7 Weighted Maximum Likelihood

Weights are specified by setting the **GAUSS** global, **__weight** to a weighting vector, or by assigning it the name of a column in the **GAUSS** data set being used in the estimation. Thus if a data matrix is being analyzed, **__weight** must be assigned to a vector.

MAXLIK assumes that the weights sum to the number of observations, i.e, that the weights are frequencies. This will be an issue only with statistical inference. Otherwise, any multiple of the weights will produce the same results.

2.3.8 Active and Inactive Parameters

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

This feature may also be used for model testing. **_max_NumObs** times the difference between the function values (the second return argument in the call to **MAXLIK**) is chi-squared distributed with degrees of freedom equal to the number of fixed parameters in **_max_Active**.

2.3.9 Example

This example estimates coefficients for a tobit model:

```
library maxlik;
#include maxlik.ext;
maxset;

proc lpr(x,z);
    local t,s,m,u;
    s = x[4];
    if s <= 1e-4;
      retp(error(0));
    endif;
    m = z[.,2:4]*x[1:3,.];
    u = z[.,1] ./= 0;
    t = z[.,1]-m;
    retp(u.*(-(t.*t)./(2*s)-.5*ln(2*s*pi)) + (1-u).*(ln(cdfnc(m/sqrt(s)))));
endp;

x0 = { 1, 1, 1, 1 };
__title = "tobit example";

{x,f,g,cov,ret} = maxlik("tobit",0,&lpr,x0);
call maxprt(x,f,g,cov,ret);
```

The output is:

```
===========================================================================
                             tobit example
===========================================================================
 MAXLIK Version 4.0.8                                  8/31/95   1:11 pm
===========================================================================
                            Data Set:  tobit
---------------------------------------------------------------------------

return code =    0
normal convergence

Mean log-likelihood        -1.13291
Number of cases     100

Covariance matrix of the parameters computed by the following method:
Inverse of computed Hessian

Parameters    Estimates     Std. err.  Est./s.e.  Prob.    Gradient
------------------------------------------------------------------
P01              0.0104        0.0845    0.123   0.4510     -0.0000
```

MaxLik

```
P02            -0.2081       0.0946   -2.200   0.0139     -0.0000
P03            -0.0998       0.0801   -1.245   0.1065     -0.0000
P04             0.6522       0.0999    6.531   0.0000     -0.0000

Correlation matrix of the parameters
   1.000   0.035   0.155  -0.090
   0.035   1.000  -0.204   0.000
   0.155  -0.204   1.000  -0.030
  -0.090   0.000  -0.030   1.000

Number of iterations    17
Minutes to convergence     0.03200
```

2.4 Managing Optimization

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

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

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

2.4.1 Scaling

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

2.4.2 Condition

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

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

2.4.3 Starting Point

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

2.4.4 Diagnosis

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

When you have selected **_max_Diagnostic** $=$ 2 or 3, **MAXLIK** inserts the matrices into **_max_Diagnostic** using the **vput** command. The matrices are extracted using the **vread** command. For example,

```
_max_Diagnostic = 2;
call MAXPrt(maxlik("tobit",0,&lpr,x0));
h = vread(_max_Diagnostic,"hessian");
d = vread(_max_Diagnostic,"direct");
```

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

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

When nested calls to **MAXLIK** are made, i.e., when the procedure for computing the log-likelihood itself calls its own version of **MAXLIK**, **_max_Diagnostic** returns the matrices of the outer call to **MAXLIK** only.

2.5 Gradients

2.5.1 Analytical Gradient

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

This procedure has two input arguments, a $K \times 1$ vector of parameters and an $N_i \times L$ submatrix of the input data set. The number of rows of the data set passed in the argument to the call of this procedure may be less than the total number of observations when the data are stored in a **GAUSS** data set and there was not enough space to store the data set in RAM in its entirety. In that case subsets of the data set are passed to the procedure in sequence. The gradient procedure must be written to return a gradient (or more accurately, a "Jacobian") with as many rows as the input submatrix of the data set. Thus the gradient procedure returns an $N_i \times K$ matrix of gradients of the N_i observations with respect to the K parameters. The **MAXLIK** global, **_max_GradProc** is then set to the pointer to that procedure. For example,

```
library maxlik;
#include maxlik.ext;
maxset;

proc lpsn(b,z);   /*  Function - Poisson Regression */
   local m;
   m = z[.,2:4]*b;
   retp(z[.,1].*m-exp(m));
endp;

proc lgd(b,z);    /*  Gradient */
   retp((z[.,1]-exp(z[.,2:4]*b)).*z[.,2:4]);
```

```
endp;

x0 = { .5, .5, .5 };
_max_GradProc = &lgd;
_max_GradCheckTol = 1e-3;

{ x,f0,g,h,retcode } = MAXLIK("psn",0,&lpsn,x0);
call MAXPrt(x,f0,g,h,retcode);
```


In practice, unfortunately, much of the time spent on writing the gradient procedure is devoted to debugging. To help in this debugging process, **MAXLIK** can be instructed to compute the numerical gradient along with your prospective analytical gradient for comparison purposes. In the example above this is accomplished by setting **_max_GradCheckTol** to 1e-3.

2.5.2 User-Supplied Numerical Gradient

You may substitute your own numerical gradient procedure for the one used by **MAXLIK** by default. This is done by setting the **MAXLIK** global, **_max_UserGrad** to a pointer to the procedure.

MAXLIK includes some numerical gradient functions in `gradient.src` which can be invoked using this global. One of these procedures, **gradre**, computes numerical gradients using the Richardson Extrapolation method. To use this method set

```
_max_UserNumGrad = &gradre;
```

2.5.3 Analytical Hessian

You may provide a procedure for computing the Hessian, $\Sigma(\theta) = \partial^2 L/\partial\theta\partial\theta'$. This procedure has two arguments, the $K \times 1$ vector of parameters, an $N_i \times L$ submatrix of the input data set (where N_i may be less than N), and returns a $K \times K$ symmetric matrix of second derivatives of the objection function with respect to the parameters.

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

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

```
    library maxlik;
    #include maxlik.ext;

    proc lnlk(b,z);
        local dev,s2;
        dev = z[.,1] - b[1] * exp(-b[2]*z[.,2]);
        s2 = dev'dev/rows(dev);
        retp(-0.5*(dev.*dev/s2 + ln(2*pi*s2)));
    endp;

    proc grdlk(b,z);
        local d,s2,dev,r;
        d = exp(-b[2]*z[.,2]);
        dev = z[.,1] - b[1]*d;
        s2 = dev'dev/rows(dev);
        r = dev.*d/s2;
/*      retp(r~(-b[1]*z[.,2].*r));        correct gradient */
        retp(r~(z[.,2].*r));          /* incorrect gradient */
    endp;

    proc hslk(b,z);
        local d,s2,dev,r, hss;
        d = exp(-b[2]*z[.,2]);
        dev = z[.,1] - b[1]*d;
        s2 = dev'dev/rows(dev);
        if s2 <= 0;
            retp(error(0));
        endif;
        r = z[.,2].*d.*(b[1].*d - dev)/s2;
        hss = -d.*d/s2~r~-b[1].*z[.,2].*r;
        retp(xpnd(sumc(hss)));
    endp;

    maxset;
    _max_HessProc = &hslk;
    _max_GradProc = &grdlk;
    _max_GradCheckTol = 1e-3;

    startv = { 2, 1 };

    { x,f0,g,cov,retcode } = MAXLIK("nlls",0,&lnlk,startv);
    call MAXPrt(x,f0,g,cov,retcode);
```

The gradient is incorrectly computed, and **MAXLIK** responds with an error message. It is clear that the error is in the calculation of the gradient for the second parameter.

```
analytical and numerical gradients differ
      numerical       analytical
    -0.015387035     -0.015387035
     0.031765317     -0.015882659
```

```
========================================================================
              analytical Hessian and analytical gradient
========================================================================
 MAXLIK Version 4.0.0                                 2/08/95  10:10 am
========================================================================
                           Data Set:  nlls
------------------------------------------------------------------------

return code =    7
function cannot be evaluated at initial parameter values

Mean log-likelihood         1.12119
Number of cases     150

The covariance of the parameters failed to invert

Parameters     Estimates      Gradient
----------------------------------------------
P01           2.000000     -0.015387
P02           1.000000     -0.015883

Number of iterations    .
Minutes to convergence           .
```

2.5.4 User-Supplied Numerical Hessian

You may substitute your own numerical Hessian procedure for the one used by **MAXLIK** by default. This done by setting the **MAXLIK** global, **_max_UserHess** to a pointer to the procedure. This procedure has three input arguments, a pointer to the log-likelihood function, a $K \times 1$ vector of parameters, and an $N_i \times K$ matrix containing the data. It must return a K × K matrix which is the estimated Hessian evaluated at the parameter vector.

2.6 Inference

MAXLIK includes four classes of methods for analyzing the distributions of the estimated parameters:

- Wald
- Profile likelihood
- Bootstrap

MaxLik

- Bayesian

The Wald type statistical inference is the most commonly used method which relies on a quadratic approximation to the log-likelihood surface, and uses an estimate of the covariance matrix of the parameters for computing standard errors and confidence limits. **MAXLIK** provides three methods for estimating the covariance matrix, the inverse of the Hessian, the inverse of the cross-products of the first derivatives, and the quasi-maximum likelihood (or QML) estimate which is computed from both the Hessian and the cross-product of the first derivatives.

The bootstrap and Bayesian methods both produce simulated "data" sets of the parameters from which kernel density plots, histograms, surface plots, and confidence limits may be computed.

The profile likelihood method computes confidence limits directly from the log-likelihood surface. Profile likelihood confidence limits are to be prefered to Wald confidence limits when the quadratic approximation is poor which is likely to be the case in particular for nonlinear models. The profile likelihood inference package includes a procedure for computing confidence limits as well as likelihood profile traces and profile t traces used for evaluating the shape of the log-likelhood surface.

2.6.1 Wald Inference

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

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

where

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

Estimates of A and B are

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

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

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

When **_max_CovPar** $= 1$, $\hat{A}^{-1}$, the inverse of the Hessian, is returned as the covariance matrix of the parameters.

When **_max_CovPar** $= 2$, **MAXLIK** returns $\hat{B}^{-1}$, the cross-product of the first derivatives computed by observation (i.e., the "Jacobian" of the log-likelihood) as the covariance matrix of the parameters.

When **_max_CovPar** is set to 3, **MAXLIK** returns $\hat{A}^{-1}\hat{B}\hat{A}^{-1}$, the QML covariance matrices of the parameters.

When the QML method has been selected, the covariance matrices computed from the Hessian and the cross-product of first derivatives will both be returned in the global variables, **_max_HessCov** and **_max_XprodCov**, respectively. A rough measure of the misspecification in the model may be gauged from the extent to which the covariance matrices computed from the Hessian and the cross-product of first derivatives diverge. A method for computing a statistic to measure this divergence (thereby providing a test for misspecification) has been developed by White (1981,1982).

The QML covariance matrix is expensive to compute since it requires the calculation of both the matrix of second derivatives and the first derivatives by case. The expense will usually be worth it, however, because this matrix will always generate the correct standard errors (unless there is a misspecification in the model that renders the parameter estimates inconsistent in which case no method will produce correct standard errors). To determine whether either the Hessian or the cross-product covariance matrix of parameters are sufficiently correct by themselves it would be necessary to compute them both anyway.

When Computing the Covariance Matrix of the Parameters Fails

The computation of the covariance matrix of the parameters may fail if there is not enough information in the data to identify the model parameters, or if the model specification includes parameters that cannot be identified for any set of data. In these cases there may be some utility in a collinearity analysis of the matrix used in the computation of the covariance matrix of the parameters. This matrix is stored in the global variable **_max_FinalHess** before the inversion attempt. If the inversion fails (of the Hessian if **_max_CovPar** $= 1$, or of the cross-product of the first derivatives if **_max_CovPar** $= 2$), **MAXLIK** will return a missing code for the covariance matrix and the user can then retrieve the matrix stored in **_max_FinalHess** for a collinearity analysis. Linear dependencies in this matrix will indicate which parameters are not identified and an analysis of these linear dependencies may suggest tactics for respecifying the model.

2.6.2 Profile Likelihood Inference

Wald confidence limits for parameters assume the appropriateness of the quadratic approximation to the log-likelihood surface. For some models, in particular nonlinear models, this approximation may not be satisfactory. In this case, the profile likelihood confidence limit would be prefered.

The profile likelihood confidence region is defined as the set of points (Cook and Wiesberg, 1990, Meeker and Escobar, 1995):

$$\{\theta \mid \sqrt{2(L(\hat{\theta}) - L(\theta))} \geq \chi^2_{(1-\alpha;k)}\}$$

where

$$L(\theta) = \sum_{i=1}^{N} \log P(Y_i; \theta)$$

and K is the length of θ.

For individual parameters this method is implemented in **MAXLIK** in the following way: define

$$G(\phi) = \min(Logl(\theta) \mid \eta_i'\theta = \phi) \tag{2.1}$$

where η_i is a conformable vector of zeros with a one in position i.

Then the lower profile likelihood confidence limit at the $1 - \alpha$ interval are the values of ϕ such that

$$G(\phi) = \chi^2_{(1-\alpha;k)}.$$

and the upper limit is found by redefining Equation 2.1 as a maximum.

Example

This examples illustrates and compares Wald confidence limits and profile likelihood confidence limits:

```
library maxlik;
#include maxlik.ext;
maxset;

proc lpr(x,z);
    local t,s,m,u;
    s = x[4];
    if s <= 1e-4;
      retp(error(0));
    endif;
    m = z[.,2:4]*x[1:3,.];
    u = z[.,1] ./= 0;
```

```
    t = z[.,1]-m;
    retp(u.*(-(t.*t)./(2*s)-.5*ln(2*s*pi)) +
         (1-u).*(ln(cdfnc(m/sqrt(s))))
);
endp;

x0 = { 1, 1, 1, 1 };

{x,f,g,cov,ret} = maxlik("tobit",0,&lpr,x0);

__title = "Wald Confidence Limits";
cl1 = maxtlimits(x,cov);
call maxclprt(x,f,g,cl1,ret);

__title = "Profile Likelihood Confidence Limits";
cl2 = maxpflclimits(x,f,"tobit",0,&lpr);
call maxclprt(x,f,g,cl2,ret);
```

The output is:

```
===========================================================================
                         Wald Confidence Limits
===========================================================================
 MAXLIK Version 4.0.8                                   8/31/95   1:16 pm
===========================================================================
                            Data Set:  tobit
---------------------------------------------------------------------------


return code =    0
normal convergence

Mean log-likelihood        -1.13291
Number of cases     100


                             0.95 confidence limits
Parameters    Estimates     Lower Limit   Upper Limit   Gradient
-------------------------------------------------------------------
P01              0.0104       -0.1573        0.1781     -0.0000
P02             -0.2081       -0.3958       -0.0203     -0.0000
P03             -0.0998       -0.2588        0.0593     -0.0000
P04              0.6522        0.4540        0.8505     -0.0000

Number of iterations    17
Minutes to convergence     0.03200
```

```
==========================================================================
                  Profile Likelihood Confidence Limits
==========================================================================
 MAXLIK Version 4.0.8                                  8/31/95   1:16 pm
==========================================================================
                            Data Set:  tobit
--------------------------------------------------------------------------


return code =    0
normal convergence

Mean log-likelihood       -1.13291
Number of cases     100


                             0.95 confidence limits
Parameters    Estimates     Lower Limit   Upper Limit   Gradient
------------------------------------------------------------------
P01              0.0104       -0.1560        0.1720     -0.0000
P02             -0.2081       -0.3918       -0.0245     -0.0000
P03             -0.0998       -0.2562        0.0549     -0.0000
P04              0.6522        0.4928        0.8885     -0.0000

Number of iterations    17
Minutes to convergence     0.03200
```

In this example, the model is conditionally linear and we see that the Wald and profile likelihood limits are quite similar.

2.6.3 Profile Trace Plots

MAXProfile generates profile t plots as well as plots of the likelihood profile traces for all of the parameters in the model in pairs. The profile t plots are used to assess the nonlinearity of the distributions of the individual parameters, and the likelihood profile traces are used to assess the bivariate distributions. The input and output arguments to **MAXProfile** are identical to those of **MAXLIK**. But in addition to providing the maximum likelihood estimates and covariance matrix of the parameters, a series of plots are printed to the screen using **GAUSS**' Publication Quality Graphics. A screen is printed for each possible pair of parameters. There are three plots, a profile t plot for each parameter, and a third plot containing the likelihood profile traces for the two parameters.

The discussion in this section is based on Bates and Watts (1988), pages 205-216, which is recommended reading for the interpretation and use of profile t plots and likelihood profile traces.

The Profile t Plot

Define

$$\tilde{\theta_k} = (\tilde{\theta}_1, \tilde{\theta}_2, ..., \tilde{\theta}_{k-1}, \theta_k, \tilde{\theta}_{k+1}, ..., \tilde{\theta}_K)$$

This is the vector of maximum likelihood estimates *conditional* on θ_k, i.e., where θ_k is fixed to some value. Further define the profile t function

$$\tau(\theta_k) = sign(\theta_k - \hat{\theta}_k)(N - K)\sqrt{2\left[L(\tilde{\theta}_k) - L(\hat{\theta}_k)\right]}$$

For each parameter in the model, τ is computed over a range of values for θ_k. These plots provide exact likelihood intervals for the parameters, and reveal how nonlinear the estimation is. For a linear model, τ is a straight line through the origin with unit slope. For nonlinear models, the amount of curvature is diagnostic of the nonlinearity of the estimation. High curvature suggests that the usual statistical inference using the t-statistic is hazardous.

The Likelihood Profile Trace

The likelihood profile traces provide information about the bivariate likelihood surfaces. For nonlinear models the profile traces are curved, showing how the parameter estimates affect each other and how the projection of the likelihood contours onto the (θ_k, θ_ℓ) plane might look. For the (θ_k, θ_ℓ) plot, two lines are plotted, $L(\tilde{\theta}_k)$ against θ_k and $L(\tilde{\theta}_\ell)$ against θ_ℓ.

If the likelihood surface contours are long and thin, indicating the parameters to be collinear, the profile traces are close together. If the contours are fat, indicating the parameters to be more uncorrelated, the profile traces tend to be perpendicular. And if the contours are nearly elliptical, the profile traces are straight. The surface contours for a linear model would be elliptical and thus the profile traces would be straight and perpendicular to each other. Significant departures of the profile traces from straight, perpendicular lines, therefore, indicate difficulties with the usual statistical inference.

To generate profile t plots and likelihood profile traces from the example in Section 2.3.9, it is necessary only to change the call to **MAXLIK** to a call to **MAXProfile**:

```
call MAXPrt(MAXProfile("tobit",0,&lpr,x0));
```

MAXProfile produces the same output as **MAXLIK** which can be printed out using a call to **MAXPRT**.

For each pair of parameters a plot is generated containing an xy plot of the likelihood profile traces of the two parameters, and two profile t plots, one for each parameter.

2.6.4 Bootstrap

The bootstrap method is used to generate empirical distributions of the parameters, thus avoiding the difficulties with the usual methods of statistical inference described above.

MAXBoot

Rather than randomly sample with replacement from the data set, **MAXBoot** performs **_max_NumSample** weighted maximum likelihood estimations where the weights are Poisson pseudo-random numbers with expected value equal to the the number of observations. **_max_NumSample** is set by the **MAXBoot** global variable. The default is 50 re-samplings. Efron and Tibshirani (1993:52) suggest that 100 is satisfactory, 50 is often enough to give a good estimate, and rarely are more than 200 needed.

The mean and covariance matrix of the bootstrapped parameters is returned by **MAXBoot**. In addition **MAXBoot** writes the bootstrapped parameter estimates to a **GAUSS** data set for use with **MAXHist**, which produces histograms and surface plots, **MAXDensity**, which produces kernel density plots, and **MAXBlimits**, which produces confidence limits based on the bootstrapped coefficients. The data set name can be specified by the user in the global **_max_BootFname**. However, if not specified, **MAXBoot** selects a temporary filename.

MAXDensity

MAXDensity is a procedure for computing kernel type density plots. The global, **_max_Kernel** permits you to select from a variety of kernels, normal, Epanechnikov, biweight, triangular, rectangular, and truncated normal. For each selected parameter, a plot is generated of a smoothed density. The smoothing coefficients may be specified using the global, **_max_Smoothing**, or **MAXDensity** will compute them.

MAXHist

MAXHist is a procedure for visually displaying the results of the bootstrapping in univariate histograms and bivariate surface plots for selected parameters. The univariate discrete distributions of the parameters used for the histograms are returned by **MAXHist** in a matrix.

Example

To bootstrap the example in Section 2.3.9, the only necessary alteration is the change the call to **MAXLIK** to a call to **MAXBoot**:

```
_max_BootFname = "bootdata";

call MAXPrt(maxlikboot("tobit",0,&lpr,x0));

call MAXDensity("bootdata",0);
call MAXHist("bootdata",0);
```

2.6.5 Bayesian Inference

The **MAXLIK** proc **MAXBayes** generates a simulated posterior of the parameters of a maximum likelihood estimation using the weighted likelihood bootstrap method described in Newton and Raftery (1994). In this method, a weighted bootstrap is conducted using weighted Dirichlet random variates for weights. After generating the weighted bootstrapped parameters, "Importance" weights are computed:

$$r(\hat{\theta}) = \pi(\hat{\theta})e^{L(\hat{\theta})}/\hat{g}(\hat{\theta})$$

where $\pi(\hat{\theta})$ is the prior distribution of the parameters, and $\hat{g}(\hat{\theta})$ is a normal kernel density estimate of the of the parameters using Terrell's (1990) method of maximum smoothing. The SIR algorithm, described in Rubin (1988), is applyed to the bootstrapped parameters using these importance weights.

The Dirichlet variates are weighted to generate over-dispersion in order to make sure they have coverage with respect to the posterior distribution. This weight is stored in the **MAXLIK** global, **_max_BayesAlpha**, and is set to 1.4 by default. See Newton and Raftery (1994) for a discussion of this weight.

Example

This example computes ordinary maximum likelihood estimates, and then calls **MAXBayes** which generates a simulated posterior. The call to **MAXDensity** produces kernel density plots and returns the data used in the plots. This information is used to determine the modes of the simulated posterior distributions and **MAXPrt** prints that information to output.

```
library maxlik,pgraph;
#include maxlik.ext;
#include pgraph.ext;
graphset;
maxset;

proc lpr(x,z);
    local t,s,m,u;
```

```
    s = x[4];
    if s <= 1e-4;
      retp(error(0));
    endif;
    m = z[.,2:4]*x[1:3,.];
    u = z[.,1] ./= 0;
    t = z[.,1]-m;
    retp(u.*(-(t.*t)./(2*s)-.5*ln(2*s*pi)) + (1-u).*(ln(cdfnc(m/sqrt(s)))));
endp;

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

__title = "Maximum Likelihood Estimates";
{x0,f,g,cov,ret} = maxlik("tobit",0,&lpr,start);
call maxprt(x0,f,g,cov,ret);

_max_BootFname = "bayes";
_max_NumSample = 500;

{x1,f,g,cov,ret} = maxBayes("tobit",0,&lpr,x0);

{ px,py,smth } = maxDensity("bayes",0);
x_mode = diag(px[maxindc(py),.]);

__title = "modal Bayesian estimates";
call maxprt(x_mode,f,g,cov,ret);
```

```
===============================================================================
                         Maximum Likelihood Estimates
===============================================================================
 MAXLIK Version 4.0.12                                      9/01/95  11:18 am
===============================================================================
                               Data Set:  tobit
-------------------------------------------------------------------------------


return code =    0
normal convergence

Mean log-likelihood        -1.13291
Number of cases     100

Covariance matrix of the parameters computed by the following method:
Inverse of computed Hessian

Parameters    Estimates     Std. err.  Est./s.e.  Prob.    Gradient
------------------------------------------------------------------
P01              0.0104        0.0873    0.119   0.4525      0.0000
```

```
P02            -0.2081       0.0946   -2.200   0.0139      0.0000
P03            -0.0998       0.0800   -1.247   0.1062      0.0000
P04             0.6522       0.0999    6.531   0.0000      0.0000

Correlation matrix of the parameters
   1.000   0.030   0.151  -0.092
   0.030   1.000  -0.205   0.000
   0.151  -0.205   1.000  -0.029
  -0.092   0.000  -0.029   1.000

Number of iterations    17
Minutes to convergence     0.01462



===============================================================================
                           modal Bayesian estimates
===============================================================================
 MAXLIK Version 4.0.12                                     9/01/95  11:20 am
===============================================================================
                               Data Set:  tobit
-------------------------------------------------------------------------------


return code =    0
normal convergence

Mean log-likelihood      -0.0117326
Number of cases     100

Covariance matrix of the parameters computed by the following method:
Bayesian covariance matrix

Parameters    Estimates     Std. err.  Est./s.e.  Prob.    Gradient
------------------------------------------------------------------
P01             0.1729       0.1661    1.041   0.1488      0.0000
P02            -0.2054       0.1930   -1.065   0.1435      0.0000
P03            -0.1425       0.1735   -0.821   0.2057      0.0000
P04             0.6598       0.2329    2.833   0.0023      0.0000

Correlation matrix of the parameters
   1.000  -0.165   0.362   0.465
  -0.165   1.000  -0.473   0.026
   0.362  -0.473   1.000   0.322
   0.465   0.026   0.322   1.000

Number of iterations    7
Minutes to convergence     0.00354
```

2.7 Run-Time Switches

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

Alt-G	Toggle **_max_GradMethod**
Alt-V	Revise **_max_GradTol**
Alt-O	Toggle **__output**
Alt-M	Maximum Tries
Alt-I	Compute Hessian
Alt-E	Edit Parameter Vector
Alt-C	Force Exit
Alt-A	Change Algorithm
Alt-J	Change Line Search Method
Alt-H	Help Table

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

Alt-1	Steepest Descent (STEEP)
Alt-2	Broyden-Fletcher-Goldfarb-Shanno (BFGS)
Alt-3	Davidon-Fletcher-Powell (DFP)
Alt-4	Newton-Raphson (NEWTON) or (NR)
Alt-5	Berndt, Hall, Hall & Hausman (BHHH)
Alt-6	Polak-Ribiere Conjugate Gradient (PRCG)

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

Shift-1	no search (1.0 or 1 or ONE)
Shift-2	cubic or quadratic method (STEPBT)
Shift-3	step halving method (HALF)
Shift-4	Brent's method (BRENT)
Shift-5	BHHH step method (BHHHSTEP)

2.8 Calling MAXLIK Recursively

The procedure that computes the log-likelihood may itself call **MAXLIK**. This version of **MAXLIK** nested inside the procedure is actually a separate copy of **MAXLIK** with its own set of globals and must have its own log-likelihood function (or otherwise you would have infinite recursion).

When calling **MAXLIK** recursively, the following considerations apply:

- Variable selection (as opposed to case selection) can be done on any level by means of the second argument in the call to each copy of **MAXLIK**.
- Data sets can be opened by nested copies of **MAXLIK**. If a nested copy of **MAXLIK** is going to use the data set opened by the outer copy of **MAXLIK**, then pass a null string in the first argument in the call. If it is going to analyze a different data set from the outer copy, then pass it the data set name in a string. You may also load and store a data set in memory in the command file and pass it in the first argument in the nested call to **MAXLIK**.
- Before the call to the nested copy of **MAXLIK**, the global variables should be reset by calling **MAXCLR**. You must not use **MAXSET** because that will clear information about the data sets opened and processed in the outer copy. The only differences between **MAXSET** and **MAXCLR** are references to these globals.
- You may also want to disable the keyboard control of the nested copies. This is done by setting the global **_max_Key** = 0 after the call to **MAXCLR** and before the call to the nested **MAXLIK**.

2.9 Using _MAXLIK Directly

When **MAXLIK** is called, it directly references all the necessary globals and passes its 4 arguments and the values of the globals to a function called **_maxlik**. When **_maxlik** returns, **MAXLIK** then sets the output globals to the values returned by **_maxlik** and returns 5 arguments directly to the user. **_maxlik** makes no global references to matrices or strings (except to **_max_eps2** which is set to the cube of machine precision), and all procedures it references have names that begin with an underscore "_".

_maxlik can be used directly in situations where you do not want any of the global matrices and strings in your program. If **MAXLIK**, **MAXPRT**, **MAXSET**, and **MAXCLR** are not referenced, the global matrices and strings in `maxlik.dec` will not be included in your program.

The documentation for **MAXLIK**, the globals it references, and the code itself should be sufficient documentation for using **_maxlik**.

2.10 Error Handling

2.10.1 Return Codes

The fourth argument in the return from **MAXLIK** contains a scalar number that contains information about the status of the iterations upon exiting **MAXLIK**. The

MaxLik

following table describes their meanings:

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

2.10.2 Error Trapping

Setting the global **__output** = 0 turns off all printing to the screen. Error codes, however, still are printed to the screen unless error trapping is also turned on. Setting the trap flag to 4 causes **MAXLIK** to *not* send the messages to the screen:

```
trap 4;
```

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

2.11 References

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

Bates, Douglas M. and Watts, Donald G., 1988. *Nonlinear Regression Analysis and Its Applications.* New York: John Wiley & Sons.

Berndt, E., Hall, B., Hall, R., and Hausman, J. 1974. “Estimation and inference in nonlinear structural models”. *Annals of Economic and Social Measurement* 3:653-665.

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

Cook, R.D. and Weisberg, S., 1990. "Confidence Curves in Nonlinear Regression", *Journal of the American Statistical Association*, 85: 544-551.

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

Efron, Gradley, Robert J. Tibshirani, 1993. *An Introduction to the Bootstrap.* New York: Chapman & Hall.

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

Judge, G.G., R.C. Hill, W.E. Griffiths, H. Lütkepohl and T.C. Lee. 1988. *Introduction to the Theory and Practice of Econometrics.* 2nd Edition. New York:Wiley.

Judge, G.G., W.E. Griffiths, R.C. Hill, H. Lütkepohl and T.C. Lee. 1985. *The Theory and Practice of Econometrics.* 2nd Edition. New York:Wiley.

Meeker, W.Q and Escobar, L.A., 1995. "Teaching about approximate confidence regions based on maximum likelihood estimate", *The American Statistician*, 49: 48-53.

Newton, M.A. and A.E. Raftery, 1994. "Approximate Bayesian inference with the weighted likelihood bootstrap", J.R. Statist. Soc. B, 56: 3-48.

Rubin, D.B., 1988. "Using the SIR algorithm to simulate posterior distributions", in *Bayesian Statistics 3*, J.M. Bernardo, M.H. DeGroot, D.V. Lindley, and A.F.M. Smith (eds.), pp. 395-402.

Terrell, G.R., 1990. "The maximal smoothing principle in density estimation", *Journal of the American Statistical Association*, 85: 470-477.

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

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

Chapter 3

Maximum Likelihood Reference

Purpose

Computes estimates of parameters of a maximum likelihood function.

Library

`maxlik`

Format

{ *x*,*f*,*g*,*cov*,*retcode* } = **MAXLIK(***dataset*,*vars*,**&***fct*,*start***)**

Input

dataset	string containing name of **GAUSS** data set – or – $N \times NV$ matrix, data
vars	$NV \times 1$ character vector, labels of variables selected for analysis – or – $NV \times 1$ numeric vector, indices of variables selected for analysis. If *dataset* is a matrix, *vars* may be a character vector containing either the standard labels created by **MAXLIK** (i.e., either V1, V2,..., or V01, V02,..... See discussion of the global variable **__vpad** below, or the user-provided labels in **__altnam**).
&*fct*	a pointer to a procedure that returns either the log-likelihood for one observation or a vector of log-likelihoods for a matrix of observations (see discussion of the global variable **__row** in global variable section below).
start	$K \times 1$ vector, start values.

Output

x	$K \times 1$ vector, estimated parameters
f	scalar, function at minimum (the mean log-likelihood)
g	$K \times 1$ vector, gradient evaluated at x
h	$K \times K$ matrix, covariance matrix of the parameters (see discussion of the global variable **_max_CovPar** below).
retcode	scalar, return code. If normal convergence is achieved, then $retcode = 0$, otherwise a positive integer is returned indicating the reason for the abnormal termination:

- **0** normal convergence
- **1** forced exit.
- **2** maximum iterations exceeded.
- **3** function calculation failed.
- **4** gradient calculation failed.
- **5** Hessian calculation failed.
- **6** line search failed.
- **7** function cannot be evaluated at initial parameter values.
- **8** error with gradient
- **9** gradient vector transposed
- **10** secant update failed
- **11** maximum time exceeded
- **12** error with weights
- **34** data set could not be opened.
- **99** termination condition unknown.

Globals

The globals variables used by **MAXLIK** can be organized in the following categories according to which aspect of the optimization they affect:

Options **_max_Options**

Descent and Line Search **_max_Algorithm**, **_max_Delta**, **_max_LineSearch**, **_max_Maxtry**, **_max_Extrap**, **_max_Interp**, **_max_RandRadius**, **_max_UserSearch**

Covariance Matrix of Parameters **_max_CovPar**, **_max_XprodCov**, **_max_HessCov**, **_max_FinalHess**

Gradient **_max_GradMethod**, **_max_GradProc**, **_max_UserNumGrad**, **_max_HessProc**, **_max_UserNumHess**, **_max_GradStep**, **_max_GradCheckTol**

Terminations Conditions **_max_GradTol**, **_max_MaxIters**, **_max_MaxTime**

Data **_max_Lag**, **_max_NumObs**, **__weight**, **__row**, **__rowfac**

Parameters **_max_Active**, **_max_ParNames**

Miscellaneous **__title**, **_max_IterData**, **_max_Diagnostic**

The list below contains an alphabetical listing of each global with a complete description.

_max_Active vector, defines fixed/active coefficients. This global allows you to fix a parameter to its starting value. This is useful, for example, when you wish to try different models with different sets of parameters without having to re-edit the function. When it is to be used, it must be a vector of the same length as the starting vector. Set elements of ˙max˙Active to 1 for an active parameter, and to zero for a fixed one.

_max_Algorithm scalar, selects optimization method:

- **1** STEEP - Steepest Descent
- **2** BFGS - Broyden, Fletcher, Goldfarb, Shanno method
- **3** DFP - Davidon, Fletcher, Powell method
- **4** NEWTON - Newton-Raphson method
- **5** BHHH - Berndt, Hall, Hall, Hausman method
- **6** PRCG - Polak-Ribiere Conjugate Gradient

Default = 3

_max_CovPar scalar, type of covariance matrix of parameters

- **0** not computed
- **1** computed from Hessian calculated after the iterations
- **2** Quasi-maximum likelihood (QML) covariance matrix of the parameters

Default = 1;

_max_Delta scalar, floor for eigenvalues of Hessian in the NEWTON algorithm. When nonzero, the eigenvalues of the Hessian are augmented to this value.

_max_Diagnostic scalar.

- **0** nothing is stored or printed
- **1** current estimates, gradient, direction, function value, Hessian, and step length are printed to the screen
- **2** the current quantities are stored in **_max_Diagnostic** using the **vput** command. Use the following strings to extract from **_max_Diagnostic** using **vread**:

function	"function"
estimates	"params"
direction	"direct"
Hessian	"hessian"
gradient	"gradient"
step	"step"

When **_max_Diagnostic** is nonzer, **__output** is forced to 1.

_max_GradTol scalar, convergence tolerance for gradient of estimated coefficients. When this criterion has been satisifed MAXLIK exits the iterations. Default = 1e-5.

_max_Extrap scalar, extrapolation constant in BRENT. Default = 2.

_max_FinalHess $K \times K$ matrix, the Hessian used to compute the covariance matrix of the parameters is stored in **_max_FinalHess**. This is most useful if the inversion of the hessian fails, which is indicated when **MAXLIK** returns a missing value for the covariance matrix of the parameters. An analysis of the Hessian stored in **_max_FinalHess** can then reveal the source of the linear dependency responsible for the singularity.

_max_GradCheckTol scalar. Tolerance for the deviation of numerical and analytical gradients when proc's exist for the computation of analytical gradients or Hessians. If set to zero, the analytical gradients will not be compared to their numerical versions. When adding procedures for computing analytical gradients it is highly recommended that you perform the check. Set **_max_GradCheckTol** to some small value, 1e-3, say when checking. It may have to be set larger if the numerical gradients are poorly computed to make sure that **MAXLIK** doesn't fail when the analytical gradients are being properly computed.

_max_GradMethod scalar, method for computing numerical gradient.

0 central difference

1 forward difference (default)

_max_GradProc scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. For example, the statement:

```
_max_GradProc=&gradproc;
```

tells **MAXLIK** that a gradient procedure exists as well where to find it. The user-provided procedure has two input arguments, an $K \times 1$ vector of parameter values and an N×K matrix of data. The procedure returns a single output argument, an $N \times K$ matrix of gradients of the log-likelihood function with respect to the parameters evaluated at the vector of parameter values.

For example, suppose the log-likelihood function is for a Poisson regression, then the following would be added to the command file:

```
proc lgd(b,z);
    retp((z[.,1]-exp(z[.,2:4]*b)).*z[.,2:4]);
endp;

_max_GradProc = &lgd;
```

Default = 0, i.e., no gradient procedure has been provided.

_max_GradStep scalar, increment size for computing gradient. When the numerical gradient is performing well, set to a larger value (1e-3, say). Default is the cube root of machine precision.

_max_HessCov $K \times K$ matrix. When **_max_CovPar** is set to 3 the information matrix covariance matrix of the parameters, i.e., the inverse of the matrix of second order partial derivatives of the log-likelihood by observations, is returned in **_max_HessCov**.

_max_HessProc scalar, pointer to a procedure that computes the hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. For example, the instruction:

```
_max_HessProc = &hessproc;
```

tells MAXLIK that a procedure has been provided for the computation of the hessian and where to find it. The procedure that is provided by the user must have two input arguments, a $K \times 1$ vector of parameter values and an N×P data matrix. The procedure returns a single output argument, the $K \times K$ symmetric matrix of second order derivatives of the function evaluated at the parameter values.

_max_Interp scalar, interpolation constant in BRENT. Default = .25.

_max_IterData 3x1 vector, contains information about the iterations.

The first element contains the # of iterations, the second element contains the elapsed time in minutes of the iterations, and the third element contains a character variable indicating the type of covariance matrix of the parameters.

_max_Lag scalar, if the function includes lagged values of the variables **_max_Lag** may be set to the number of lags. When **_max_Lag** is set to a nonzero value then **__row** is set to 1 (that is, the function must evaluated one observation at a time), and **MAXLIK** passes a matrix to the user-provided function and gradient procedures. The first row in this matrix is the (i - **_max_Lag**)-th observation and the last row is the i-th observation. The read loop begins with the (**_max_Lag**+1)-th observation. Default = 0.

_max_LineSearch scalar, selects method for conducting line search. The result of the line search is a *step length*, i.e., a number which reduces the function value when multiplied times the direction..

1 step length = 1.

2 cubic or quadratic step length method (STEPBT)

3 step halving (HALF)

4 Brent's step length method (BRENT)

5 BHHH step length method (BHHHSTEP)

Default = 2.

Usually **_max_LineSearch** = 2 is best. If the optimization bogs down, try setting **_max_LineSearch** = 1, 4 or 5. **_max_LineSearch** = 3 generates slower iterations but faster convergence and **_max_LineSearch** = 1 generates faster iterations but slower convergence.

When any of these line search methods fails, **MAXLIK** attempts a random search of radius **_max_RandRadius** times the truncated log to the base 10 of the gradient when **_max_RandRadius** is set to a nonzero value. If **_max_UserSearch** is set to 1, **MAXLIK** enters an interactive line search mode.

_max_MaxIters scalar, maximum number of iterations.

_max_MaxTime scalar, maximum time in iterations in minutes. This global is most useful in bootstrapping. You might want 100 re-samples, but would be happy with anything more than 50 depending on the time it took. Set **_max_NumSample** = 100, and **_max_MaxTime** to maximum time you would be willing to wait for results. Default = 1e+5, about 10 weeks.

_max_MaxTry scalar, maximum number of tries to find step length that produces a descent.

_max_NumObs scalar, number of cases in the data set that was analyzed.

_max_Options character vector, specification of options. This global permits setting various **MAXLIK** options in a single global using identifiers. The following

```
_max_Options = { bfgs stepbt forward screen };
```

sets to the default values, i.e. the descent method to BFGS, the line search method to STEPBT, the numerical gradient method to central differences, and ¨OUTPUT = 2.

The following is a list of the identifiers:

Algorithms STEEP, BFGS, DFP, NEWTON, BHHH, PRCG

Line Search ONE, STEPBT, HALF, BRENT, BHHHSTEP

Covariance Matrix NOCOV, INFO, XPROD, HETCON

Gradient method CENTRAL, FORWARD

Output method NONE, FILE, SCREEN

__output scalar, determines printing of intermediate results. Generally when **__output** is nonzero, i.e., where there some kind of printing during the iterations, the time of the iterations is degraded.

- **0** nothing is written
- **1** serial ASCII output format suitable for disk files or printers
- **2** output is suitable for screen only. ANSI.SYS must be active.
- **≥5** same as **__output = 1** except that information is printed only every **__output**-th iteration.

When **_max_Diagnostic** is nonzero, **__output** is forced to 1.

_max_ParNames $K \times 1$ character vector, parameter labels.

_max_RandRadius scalar, if set to a nonzero value (1e-2, say) and all other line search methods fail then **MAXLIK** attempts **_max_MaxTry** tries to find a random direction within radius determined by **_max_RandRadius** that is a descent. Default = 1e-2.

_max_UserNumGrad scalar, pointer to user provided numerical gradient procedure. The instruction

```
_max_UserNumGrad = &userproc;
```

tells **MAXLIK** that a procedure for computing the numerical gradients exists. The user-provided procedure has three input arguments, a pointer to a function that computes the log-likelihood function, a $K \times 1$ vector of parameter values, and an $K \times P$ matrix of data. The procedure returns a single output argument, an $N \times K$ matrix of gradients of each row of the input data matrix with respect to each parameter.

MAXLIK includes a procedure, **GRADRE**, for computing numerical derivatives using the Richardson Extrapolation method. It is invoked by setting the global to a pointer to this function:

```
_max_UserNumGrad = &gradre;
```

__row scalar, specifies how many rows of the data set are read per iteration of the read loop. See the *REMARKS* Section for a more detailed discussion of how to set up your log-likelihood to handle more than one row of your data set. By default, the number of rows to be read is calculated by **MAXLIK**.

__rowfac scalar, "row factor". If **MAXLIK** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

causes **GAUSS** to read in 80% of the rows of the **GAUSS** data set that were read when **MAXLIK** failed due to insufficient memory.

This global has an affect only when **__row** = 0. Default = 1.

__title string title of run

_max_UserNumHess scalar, pointer to user provided numerical Hessian procedure. The instruction

```
_max_UserHess = &hessproc;
```

tells **MAXLIK** that a procedure for computing the numerical Hessian exists. The user-provided procedure three input arguments, a pointer to a function that computes the log-likelihood function, a $K \times 1$ vector of parameter values, and an N×P matrix of data. The procedure returns a single output argument, a $K \times K$ Hessian matrix of the function with respect to the parameters.

_max_UserSearch scalar, if nonzero and if all other line search methods fail **MAXLIK** enters an interactive mode in which the user can select a line search parameter

__weight vector, frequency of observations. By default all observations have a frequency of 1. zero frequencies are allowed. It is assumed that the elements of **__weight** sum to the number of observations.

_max_XprodCov $K \times K$ matrix. When **_max_CovPar** is set to 3 the cross-product matrix covariance matrix of the parameters, i.e., the inverse of the cross-product of the first derivatives of the log-likelihood computed by observations, is is returned in **_max_XprodCov**.

Remarks

Writing the Log-likelihood Function

The user must provide a procedure for computing the log-likelihood for either one observation, or for a matrix of observations. The procedure must have two input arguments: first, a vector of parameter values, and second, one or more rows of the data matrix. The output argument is the log-likelihood for the observation or observations in the second argument evaluated at the parameter values in the first argument. Suppose that the function procedure has been named *pfct*, the following considerations apply:

The format of the procedure is:

logprob **=** *pfct***(***x***,***y***);**

where

x	column vector of parameters of model
y	one or more rows of the data set (if the data set has been transformed, or if *vars* $\neq 0$, i.e., there is selection, then y is a transformed, selected observation) if **__row** $= n$, then n rows of the data set are read at a time if **__row** $= 0$, the maximum number of rows that fit in memory is computed by **MAXLIK**.

The output from the procedure *pfct* is the log-likelihood for a single observation or a vector of log-likelihoods for a set of observations. If it is not possible to compute the log-likelihood for a set of observations, then either **__row** may be set to 1 to force **MAXLIK** to send one observation at a time to *pfct* or the procedure computing the function may contain a loop. If possible, *pfct* should be written to compute a vector of log-likelihoods for a set of observations because this speeds up the computations significantly. If **_max_Lag** ≥ 1, then **__row** is forced to 1.

Setting **__row**$= 0$ causes **MAXLIK** to send the entire matrix to *pfct* if it is stored entirely in memory, or to compute the maximum number of rows if it is a **GAUSS** data set stored on disk (Note that even if the data starts out in a **GAUSS** data set, **MAXLIK** determines whether the data set will fit in memory, and if it does, then it reads the data set into an array in memory). If you are getting *insufficient memory* messages, then set **__rowfac** to a positive value less than 1.

Supplying an Analytical GRADIENT Procedure

To decrease the time of computation, the user may provide a procedure for the calculation of the gradient of the log-likelihood. The global variable **_max_GradProc** must contain the pointer to this procedure. Suppose the name of this procedure is *gradproc*. Then,

g **=** *gradproc***(***x***,***y***);**

where the input arguments are

x	vector of coefficients
y	one or more rows of data set.

and the output argument is

g row vector of gradients of log-likelihood with respect to coefficients, or a matrix of gradients (i.e., a Jacobian) if the data passed in y is a matrix (unless **_max_Lag** ≥ 1 in which case the data passed in y is a matrix of lagged values but a row vector of gradients is passed back in g).

It is important to note that the gradient is row oriented. Thus if the function that computes the log-likelihood returns a scalar value (**__row** $= 1$), then a row vector of the first derivatives of the log-likelihood with respect to the coefficients must be returned, but if the procedure that computes the log-likelihood returns a column vector, then **_max_GradProc** must return a matrix of first derivatives in which rows are associated with observations and columns with coefficients.

Providing a procedure for the calculation of the first derivatives also has a significant effect on the calculation time of the Hessian. The calculation time for the numerical computation of the Hessian is a quadratic function of the size of the matrix. For large matrices, the calculation time can be very significant. This time can be reduced to a linear function of size if a procedure for the calculation of analytical first derivatives is available. When such a procedure is available, **MAXLIK** automatically uses it to compute the numerical Hessian.

The major problem one encounters when writing procedures to compute gradients and Hessians is in making sure that the gradient is being properly computed. **MAXLIK** checks the gradients and Hessian when **_max_GradCheckTol** is nonzero. **MAXLIK** generates both numerical and analytical gradients, and viewing the discrepancies between them can help in debugging the analytical gradient procedure.

Supplying an Analytical HESSIAN Procedure.

Selection of the NEWTON algorithm becomes feasible if the user supplies a procedure to compute the Hessian. If such a procedure is provided, the global variable **_max_HessProc** must contain a pointer to this procedure. Suppose this procedure is called *hessproc*, the format is

h **=** *hessproc*(x,y)**;**

The input arguments are

x $K \times 1$ vector of coefficients

y one or more rows of data set

and the output argument is

h $K \times K$ matrix of second order partial derivatives evaluated at the coefficients in x.

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

Source

`maxlik.src`

Purpose

Computes a simulated posterior of the parameters of a maximum likelihood function

Library

`maxlik`

Format

{ *x*,*f*,*g*,*cov*,*retcode* } = **MAXBayes(***dataset*,*vars*,**&***fct*,*start***)**

Input

dataset string containing name of **GAUSS** data set
– or –
$N \times NV$ matrix, data

vars $NV \times 1$ character vector, labels of variables selected for analysis
– or –
$NV \times 1$ numeric vector, indices of variables selected for analysis.

If *dataset* is a matrix, *vars* may be a character vector containing either the standard labels created by **MAXBayes** (i.e., either V1, V2,..., or V01, V02,..... See discussion of the global variable **__vpad** below, or the user-provided labels in **__altnam**).

&*fct* a pointer to a procedure that returns either the log-likelihood for one observation or a vector of log-likelihoods for a matrix of observations (see discussion of the global variable **__row** in global variable section below).

start $K \times 1$ vector, start values.

Output

x $K \times 1$ vector, means of simulated posterior

f scalar, mean weighted bootstrap log-likelihood

g $K \times 1$ vector, means gradient of weighted bootstrap

h $K \times K$ matrix, covariance matrix of simulated posterior

retcode scalar, return code. If normal convergence is achieved, then $retcode = 0$, otherwise a positive integer is returned indicating the reason for the abnormal termination:

0 normal convergence

1	forced exit.
2	maximum iterations exceeded.
3	function calculation failed.
4	gradient calculation failed.
5	Hessian calculation failed.
6	line search failed.
7	function cannot be evaluated at initial parameter values.
8	error with gradient
9	gradient vector transposed
10	secant update failed
11	maximum time exceeded
12	error with weights
34	data set could not be opened.
99	termination condition unknown.

Globals

The **MAXLIK** procedure global variables are also applicable.

_max_BayesAlpha scalar, exponent of the Dirichlet random variates used for weights for the weighted bootstrap. See Newton and Raftery, "Approximate Bayesian Inference with the Weighted Likelihood Bootstrap", J.R. Statist. Soc. B (1994), 56:3-48. Default = 1.4.

_max_BootFname string, file name of **GAUSS** data set (do not include .DAT extension) containing bootstrapped parameter estimates. If not specified, **MAXBayes** selects a temporary name.

_max_MaxTime scalar, maximum amount of time spent in re-sampling. Default = 1e5 (about 10 weeks).

_max_NumSample scalar, number of samples to be drawn. Default = 100.

_max_PriorProc scalar, pointer to proc for computing prior. This proc takes the parameter vector as its only argument, and returns a scalar probability. If a proc is not provided, a uniform prior is assumed.

Remarks

MAXBayes generates **_max_NumSample** simulations from the posterior distribution of the parameters using a weighted likelihood bootstrap method. The simulation is put into a **GAUSS** data set. The file name of the data set is either the name found in the global **_max_BootFname**, or a temporary name. If **MAXBayes** selects a file name, it returns that file name in **_max_BootFname**.

The simulated parameters in this data set can be used as input to the **MAXLIK** procedures **MAXHist** and **MAXDensity** for further analysis.

The output from **MAXDensity** can also be used to compute modal estimates of the parameters.

Source

`maxbayes.src`

Purpose

Computes bootstrapped estimates of parameters of a maximum likelihood function.

Library

`maxlik`

Format

{ *x*,*f*,*g*,*cov*,*retcode* } = **MAXBoot(***dataset*,*vars*,**&***fct*,*start***)**

Input

dataset string containing name of **GAUSS** data set
– or –
$N \times NV$ matrix, data

vars $NV \times 1$ character vector, labels of variables selected for analysis
– or –
$NV \times 1$ numeric vector, indices of variables selected for analysis.
If *dataset* is a matrix, *vars* may be a character vector containing either the standard labels created by **MAXBoot** (i.e., either V1, V2,..., or V01, V02,..... See discussion of the global variable **__vpad** below, or the user-provided labels in **__altnam**).

&*fct* a pointer to a procedure that returns either the log-likelihood for one observation or a vector of log-likelihoods for a matrix of observations (see discussion of the global variable **__row** in global variable section below).

start $K \times 1$ vector, start values.

Output

x $K \times 1$ vector, means of re-sampled parameters

f scalar, mean re-sampled function at minimum (the mean log-likelihood)

g $K \times 1$ vector, means of re-sampled gradients evaluated at the estimates

h $K \times K$ matrix, covariance matrix of the re-sampled parameters

retcode scalar, return code. If normal convergence is achieved, then *retcode* $= 0$, otherwise a positive integer is returned indicating the reason for the abnormal termination:

0 normal convergence

1	forced exit.
2	maximum iterations exceeded.
3	function calculation failed.
4	gradient calculation failed.
5	Hessian calculation failed.
6	line search failed.
7	function cannot be evaluated at initial parameter values.
8	error with gradient
9	gradient vector transposed
10	secant update failed
11	maximum time exceeded
12	error with weights
34	data set could not be opened.
99	termination condition unknown.

Globals

The **MAXLIK** procedure global variables are also applicable.

_max_BootFname string, file name of **GAUSS** data set (do not include .DAT extension) containing bootstrapped parameter estimates. If not specified, **MAXBoot** selects a temporary name.

_max_MaxTime scalar, maximum amount of time spent in re-sampling. Default = 1e5 (about 10 weeks).

_max_NumSample scalar, number of samples to be drawn. Default = 100.

Remarks

MAXBoot generates **_max_NumSample** random samples of size **_max_NumObs** from the data set with replacement and calls **MAXLIK**. **MAXBoot** returns the mean vector of the estimates in the first argument and the covariance matrix of the estimates in the third argument.

A **GAUSS** data set is also generated containing the bootstrapped parameter estimates. The file name of the data set is either the name found in the global **_max_BootFname**, or a temporary name. If **MAXBoot** selects a file name, it returns that file name in **_max_BootFname**. The coefficients in this data set may be used as input to the **MAXLIK** procedures **MAXHist** and **MAXDensity** for further analysis.

Source

`maxboot.src`

Purpose

Generates histograms and surface plots from **GAUSS** data sets

Library

`maxlik`

Format

cl = **MAXBlimits(***dataset***)**

Input

dataset	string containing name of **GAUSS** data set – or – N×K matrix, data

Output

cl	$K \times 2$ matrix, lower (first column) and upper (second column) confidence limits of the selected parameters

Globals

_max_Alpha (1-**_max_Alpha**)% confidence limits are computed. The default is .05

_max_Select selection vector for selecting coefficients to be included in profiling, for example

```
_max_Select = { 1, 3, 4 };
```

selects the 1st, 3rd, and 4th parameters for profiling.

Remarks

MAXBlimits sorts each column of the parameter data set and computes (1-**_max_Alpha**)% confidence limits by measuring back **_max_Alpha**/2 times the number of rows from each end of the columns. The confidence limits are the values in those elements. If amount to be measured back from each end of the columns doesn't fall exactly on an element of the column, the confidence limit is interpolated from the bordering elements.

Source

`maxblim.src`

Purpose

Formats and prints the output from a call to **MAXLIK** along with confidence limits

Library

`maxlik`

Format

{ x,f,g,cl,$retcode$ } = **MAXPrt(**x,f,g,cl,$retcode$**);**

Input

x	$K \times 1$ vector, parameter estimates
f	scalar, value of function at minimum
g	$K \times 1$ vector, gradient evaluated at x
cl	$K \times 2$ matrix, lower (first column) and upper (second column) confidence limits
$retcode$	scalar, return code.

Output

The input arguments are returned unchanged.

Globals

__header string. This is used by the printing procedure to display information about the date, time, version of module, etc. The string can contain one or more of the following characters:

"t"	print title (see **__title**)
"l"	bracket title with lines
"d"	print date and time
"v"	print version number of program
"f"	print file name being analyzed

Example:

```
__header = "tld";
```

Default = "tldvf".

__title string, message printed at the top of the screen and printed out by **MAXPrt**. Default = "".

Remarks

Confidence limits computed by **MAXBlimits** or **MAXTlimits** may be passed in the fourth argument in the call to **MAXPrt**:

```
{ b,f,g,cov,ret } = MAXBoot("tobit",0,&lpr,x0);
cl = MAXBLimit(_max_BootFname,0);
call MAXCLPrt(b,f,g,cl,ret);
```

Source

`maxlik.src`

Purpose

Generates histograms and surface plots from **GAUSS** data sets

Library

`maxlik`

Format

{ *px*, *py*, *smth* } = **MAXDensity(***dataset*,*vars***)**

Input

dataset string containing name of **GAUSS** data set
– or –
N×K matrix, data

vars $K \times 1$ character vector, labels of variables selected for analysis
– or –
$K \times 1$ numeric vector, indices of variables selected for analysis.
If *dataset* is a matrix, *vars* may be a character vector containing either the standard labels created by **MAXDensity** (i.e., either V1, V2,..., or V01, V02,..... See discussion of the global variable **__vpad** below, or the user-provided labels in **__altnam**).

Output

px **_max_NumPoints** × K matrix, abscissae of plotted points

py **_max_NumPoints** × K matrix, ordinates of plotted points

smth K × 1 vector, smoothing coefficients

Globals

The **MAXLIK** procedure global variables are also applicable.

_max_Kernel K × 1 character vector, type of kernel:

NORMAL normal kernel

EPAN Epanechnikov kernel

BIWGT biweight kernel

TRIANG triangular kernel

RECTANG rectangular kernel

TNORMAL truncated normal kernel

If **_max_Kernel** is scalar, the kernel is the same for all parameter densities. Default = NORMAL.

_max_NumPoints scalar, number of points to be computed for plots

_max_EndPoints K × 2 matrix, lower (in first column) and upper (in second column) endpoints of density. Default is minimum and maximum, respectively, of the parameter values. If 1 × 2 matrix, endpoints are the same for all parameters.

_max_Smoothing K × 1 vector, smoothing coefficients for each plot. If scalar, smoothing coefficient is the same for each plot. If zero, smoothing coefficient is computed by **MAXDensity**. Default = 0.

_max_Truncate K × 2 matrix, lower (in first column) and upper (in second column) truncation limits for truncated normal kernel. If 1x2 matrix, truncations limits are the same for all plots. Default is minimum and maximum, respectively.

__output If nonzero, K density plots are printed to the screen, otherwise no plots are generated.

Source

`maxdens.src`

Purpose

Generates histograms and surface plots from **GAUSS** data sets

Library

`maxlik`

Format

{ *tab*, *cut* } = **MAXHist(***dataset*,*vars***)**

Input

dataset string containing name of **GAUSS** data set
– or –
N×K matrix, data

vars $K \times 1$ character vector, labels of variables selected for analysis
– or –
$K \times 1$ numeric vector, indices of variables selected for analysis.

If *dataset* is a matrix, *vars* may be a character vector containing either the standard labels created by **MAXHist** (i.e., either V1, V2,..., or V01, V02,..... See discussion of the global variable **__vpad** below, or the user-provided labels in **__altnam**).

Output

tab **_max_NumCat** × K matrix, univariate distributions of bootstrapped parameters

cut **_max_NumCat** × K matrix, cutting points

Globals

The **MAXLIK** procedure global variables are also applicable.

_max_Center $K \times 1$ value of center category in histograms. Default is initial coefficient estimates.

_max_CutPoint **_max_NumCat** × 1 vector, output, cutting points for histograms

_max_Increment $K \times 1$ vector, increments for cutting points of the histograms. Default is 2 * **_max_Width** * std dev / **_max_NumCat**.

_max_NumCat scalar, number of categories in the histograms

_max_Width scalar, width of histograms, default = 2

__output If nonzero, K density plots are printed to the screen, otherwise no plots are generated.

Remarks

If **__output** is nonzero, $K(K-1)/2$ plots are printed to the screen displaying univariate histograms and bivariate surface plots of the bootstrapped parameter distributions in pairs.

The globals, **_max_Center**, **_max_Width**, and **_max_Increment** may be used to establish cutting points (which is stored in **_max_Increment**) for the tables of re-sampled coefficients in *tab* The numbers in **_max_Center** fix the center categories, **_max_Width** is a factor which when multiplied times the standard deviation of the estimate determines the increments between categories. Alternatively, the increments between categories can be fixed directly by supplying them in **_max_Increment**.

Source

`maxhist.src`

Library

`maxlik`

Purpose

Computes profile t plots and likelihood profile traces for maximum likelihood models

Format

{ *x*,*f*,*g*,*cov*,*retcode* } = **MAXProfile(***dataset*,*vars*,**&***fct*,*start***)**

Input

dataset — string containing name of **GAUSS** data set
– or –
$N \times NV$ matrix, data

vars — $NV \times 1$ character vector, labels of variables selected for analysis
– or –
$NV \times 1$ numeric vector, indices of variables selected for analysis.

If *dataset* is a matrix, *vars* may be a character vector containing either the standard labels created by **MAXProfile** (i.e., either V1, V2,..., or V01, V02,..... See discussion of the global variable **__vpad** below, or the user-provided labels in **__altnam**).

&*fct* — a pointer to a procedure that returns either the log-likelihood for one observation or a vector of log-likelihoods for a matrix of observations (see discussion of the global variable **__row** in global variable section below).

start — $K \times 1$ vector, start values.

Output

x — $K \times 1$ vector, means of re-sampled parameters

f — scalar, mean re-sampled function at minimum (the mean log-likelihood)

g — $K \times 1$ vector, means of re-sampled gradients evaluated at the estimates

h — $K \times K$ matrix, covariance matrix of the re-sampled parameters

retcode — scalar, return code. If normal convergence is achieved, then *retcode* $= 0$, otherwise a positive integer is returned indicating the reason for the abnormal termination:

0 normal convergence

1	forced exit.
2	maximum iterations exceeded.
3	function calculation failed.
4	gradient calculation failed.
5	Hessian calculation failed.
6	line search failed.
7	function cannot be evaluated at initial parameter values.
8	error with gradient
9	gradient vector transposed
10	secant update failed
11	maximum time exceeded
12	error with weights
34	data set could not be opened.
99	termination condition unknown.

Globals

The **MAXLIK** procedure global variables are also relevant.

_max_NumCat scalar, number of categories in profile table. Default = 16.

_max_Increment K × 1 vector, increments for cutting points, default is 2 * **_max_Width** * std dev / **_max_NumCat**. If scalar zero, increments are computed by **MAXProfile**.

_max_Center K × 1 vector, value of center category in profile table. Default values are coefficient estimates.

_max_Select selection vector for selecting coefficients to be included in profiling, for example

```
_max_Select = { 1, 3, 4 };
```

selects the 1st, 3rd, and 4th parameters for profiling.

_max_Width scalar, width of profile table in units of the standard deviations of the parameters. Default = 2.

Remarks

For each pair of the selected parameters, three plots are printed to the screen. Two of the are the profile t trace plots that describe the univariate profiles of the parameters, and one of them is the profile likelihood trace describing the joint distribution of the two parameters. Ideally distributed parameters would have univariate profile t traces that are straight lines, and bivariate likelihood profile traces that are two straight lines intersecting at right angles. This ideal is generally not met by nonlinear models, however, large deviations from the ideal indicate serious problems with the usual statistical inference.

Source

`maxprof.src`

Library

`maxlik`

Purpose

Computes profile likelihood confidence limits

Format

cl = MAXPflClimits(*b*,*f*,*dataset*,*vars*,**&***fct***)**

Input

b	$K \times 1$ vector, maximum likelihood estimates
f	scalar, function at minimum (mean log-likelihood)
dataset	string containing name of **GAUSS** data set – or – $N \times NV$ matrix, data
vars	$NV \times 1$ character vector, labels of variables selected for analysis – or – $NV \times 1$ numeric vector, indices of variables selected for analysis. If *dataset* is a matrix, *vars* may be a character vector containing either the standard labels created by **MAXPflClimits** (i.e., either V1, V2,..., or V01, V02,..... See discussion of the global variable **__vpad** below, or the user-provided labels in **__altnam**).
&*fct*	a pointer to a procedure that returns either the log-likelihood for one observation or a vector of log-likelihoods for a matrix of observations (see discussion of the global variable **__row** in global variable section below).

Output

cl	$K \times 2$ vector, upper (first column) and lower (second column) confidence limits for the parameters in *b*

Globals

_max_Alpha (1-**_max_Alpha**)% confidence limits are computed. The default is .05

_max_NumObs scalar, number of observations. Must be set. If the call to **MaxPflClimits** comes after a call to **MAXLIK**, it will be set by **MAXLIK**.

_max_Select selection vector for selecting parameters for analysis. For example,

```
_max_Select = { 1, 3, 4 };
```

selects the 1st, 3rd, and 4th parameters for limits.

Remarks

MAXPflClimits computes profile likelihood confidence limits given a maximum likelihood estimation. b and f should be returns from a call to MAXLIK. This will also properly set up **_max_NumObs** for **MAXPflClimits**.

MAXPflClimits solves for the confidence limits as a parametric likelihood problem. Thus it itself calls **MAXLIK** several times for each confidence limit. The screen output is turned off for these runs. However, the computation can be time consuming, and if you wish to check on its progress, type O, or Alt-O, and revise the **__OUTPUT** global. This will turn on the screen output for that run. The parameter number is printed on the title and this will tell you what parameter it is presently working on.

Source

`maxpflcl.src`

Purpose

Formats and prints the output from a call to **MAXLIK**.

Library

`maxlik`

Format

{ x,f,g,h,*retcode* } = **MAXPrt(**x,f,g,h,*retcode*);

Input

x	$K \times 1$ vector, parameter estimates
f	scalar, value of function at minimum
g	$K \times 1$ vector, gradient evaluated at x
h	$K \times K$ matrix, covariance matrix of parameters
retcode	scalar, return code.

Output

The input arguments are returned unchanged.

Globals

__header string. This is used by the printing procedure to display information about the date, time, version of module, etc. The string can contain one or more of the following characters:

"t"	print title (see **__title**)
"l"	bracket title with lines
"d"	print date and time
"v"	print version number of program
"f"	print file name being analyzed

Example:

```
__header = "tld";
```

Default = "tldvf".

__title string, message printed at the top of the screen and printed out by **MAXPrt**. Default = "".

Remarks

The call to **MAXLIK** can be nested in the call to **MAXPrt**:

```
{ x,f,g,h,retcode } = MAXPrt(MAXLIK(dataset,vars,&fct,start));
```

Source

`maxlik.src`

Purpose

Resets *MAXIMUM LIKELIHOOD* global variables to default values.

Library

`maxlik`

Format

MAXSet;

Input

None

Output

None

Remarks

Putting this instruction at the top of all command files that invoke **MAXLIK** is generally good practice. This prevents globals from being inappropriately defined when a command file is run several times or when a command file is run after another command file has executed that calls **MAXLIK**.

Source

`maxlik.src`

Purpose

computes Wald confidence limits

Library

`maxlik`

Format

cl = **MAXTlimits(**b**,**cov**)**

Input

b	$K \times 1$ vector, parameter estimates
cov	$K \times K$ matrix, covariance matrix of parameter estimates

Output

cl	$K \times 2$ matrix, lower (first column) and upper (second column) confidence limits of the selected parameters

Globals

_max_Alpha (1-**_max_Alpha**)% confidence limits are computed. The default is .05

_max_NumObs scalar, number of observations. Must be set.

_max_Select selection vector for selecting coefficients to be included in profiling, for example

```
_max_Select = { 1, 3, 4 };
```

selects the 1st, 3rd, and 4th parameters for profiling.

Remarks

MAXTlimits returns $b[i] \pm t(\_max\_NumObs - K; \_max\_Alpha/2) \times \sqrt{cov[i,i]}$

The global **_max_NumObs** must be set. If **MAXTlimits** is called immediately after a call to **MAXLIK**, **_max_NumObs** will be set by **MAXLIK**.

Source

`maxlik.src`

Chapter 4

Event Count and Duration Regression

by

Gary King

Department of Government

Harvard University

This module contains procedures for estimating statistical models of event count or duration data.

The programs included in this module implement maximum likelihood estimators for parametric statistical models of events data. Data based on events come in two forms: *event counts* and *durations* between events. Event counts are dependent variables that take on only nonnegative integer values, such as the number of wars in a year, the number of medical consultations in a month, the number of patents per firm, or even the frequency in the cell of a contingency table. Dependent variables that are measured as durations between events measure time and may take on any nonnegative real number; examples include the duration of parliamentary coalitions or time between coups d'etat. Note that the *same* underlying phenomena may be represented as either event counts (e.g., number of wars) or durations (time between wars), and some of the programs included in the **COUNT** module enable you to estimate exactly the same parameters with either form of data.

A variety of statistical models have been proposed to analyze events data, and the programs here provide some that I have developed, along with others I have found particularly useful in my research. I list here the specific programs included in this module, the models each program can estimate, and citations to the work for which I wrote each program. More complete references to the literature on event count and duration models appear at the end of this document.

Poisson	Poisson regression (King, 1988, 1987), truncated Poisson regression (1989d: Section 5), and log-linear and log-proportion models for contingency tables (1989a: Chapter 6).
Negbin	Negative binomial regression (1989b), truncated negative binomial regression (1989d: Section 5), truncated or untruncated variance function models (1989d: Section 5), overdispersed log-linear and log-proportion models for contingency tables (1989a: Chapter 6).
Hurdlep	Hurdle Poisson regression model (1989d: Section 4).
Supreme	Seemingly unrelated Poisson regression model (1989c).
Supreme2	Poisson regression model with unobserved dependent variables (1989d: Section 6).
Expon	Exponential duration model with or without censoring (King, Alt, Burns, and Laver, 1989).
Expgam	Exponential-Gamma duration model with or without censoring (King, Alt, Burns, and Laver, 1989).
Pareto	Pareto duration model with or without censoring (King, Alt, Burns, and Laver, 1989).

4.1 Getting Started

GAUSS 3.1.0+ is required to use these routines.

4.1.1 README Files

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

4.1.2 Setup

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

```
library count,quantal,pgraph;
```

This enables **GAUSS** to find the **COUNT** and required *MAXIMUM LIKELIHOOD* procedures. If you plan to make any right hand references to the global variables (which are described in a later section), you also need the statement:

```
#include count.ext;
```

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

```
countset;
```

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

The version number of each module is stored in a global variable. For the **COUNT** Module, this global is:

_cn_ver $3{\times}1$ matrix, the first element contains the major version number of the **COUNT** Module, the second element the minor version number, and the third element the revision number.

If you call for technical support, you may be asked for the version number of your copy of this module.

4.2 About the COUNT Procedures

The format of the programs included in this module are all very similar:

```
{ b,vc,llik } =   Expon(dataset,dep,ind);
{ b,vc,llik } =  Expgam(dataset,dep,ind);
{ b,vc,llik } =  Pareto(dataset,dep,ind);
{ b,vc,llik } = Poisson(dataset,dep,ind);
{ b,vc,llik } =  Negbin(dataset,dep,ind1,ind2);
{ b,vc,llik } = Hurdlep(dataset,dep,ind1,ind2);
{ b,vc,llik } = Supreme(dataset,dep1,dep2,ind1,ind2);
{ b,vc,llik } = Supreme2(dataset,dep1,dep2,ind1,ind2,ind3);
```

An example program file looks like this:

```
library count;
CountSet;
dep = { wars };
ind = { age, party, unem };
dataset = "sample";

call Poisson(dataset,dep,ind);
```

You may run these lines, or ones like them, from the **GAUSS** editor or interactively in command mode.

4.2.1 Inputs

The variable *dataset* is always the first argument. This may either be a matrix or a string containing the name of a **GAUSS** data set.

The dependent variable (or variables) is specified in each program by naming a symbol or a column number. For example,

```
dep = { durat };
```

or

```
dep = 7;
```

The independent variable vector (or vectors) is also specified in each program with variable names or column numbers. For example,

```
ind = { age, sex, race, height, size, iq };
```

or

```
ind = { 2, 4, 5, 6, 7 };
```

For each procedure, the data set and dependent variables must be specified. However, since constant terms are automatically included as part of independent variable vectors, you may occasionally wish to include no additional independent variables. You may do this easily by setting the relevant vector to zero. For example, *ind* = 0. For another example, you may wish to run the negative binomial regression model with a scalar dispersion parameter rather than a variance function: *ind2* = 0.

4.2.2 Outputs

Printed output is controlled by the global **__output**, described in the section below. This section describes the outputs *b*, *vc*, and *llik* on the left hand side of the expressions above.

b vector, the maximum likelihood estimates for all the parameters. The mean vector comes first; the variance function, other mean vectors, and scalar dispersion parameters, if any, come next.

vc matrix, the variance-covariance matrix evaluated at the maximum. The standard errors are **SQRT(DIAG(***vc***))**. If you choose the global **__CovPar** = 3, *vc* contains heteroskedastic-consistent parameter estimates.. See Section 2.6 for more discussion of options for statistical inference in maximum likelihood models.

llik scalar, the value of the log-likelihood function at the maximum.

4.2.3 Global Control Variables

_cn_Inference scalar character. Determines the type of statistical inference.

BOOT generates bootstrapped estimates and covariance matrix of estimates

MAXLIK generates maximum likelihood estimates

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Censor scalar, allows you to include a variable indicating which observations are censored. This is used by the exponential, exponential-gamma, and Pareto models of duration data. Alternatively, you may set it to a symbol **_cn_Censor** = "varname" if you are using a **GAUSS** data set, or a number (**_cn_Censor** = 11) if the data set is a matrix in memory. The censoring variable should be 0 for censored observations and 1 for others.

By default, no observations are censored.

_cn_Fix scalar, name of index number of logged variable among the regressors with coefficient fixed to 1.0. By default, no logged variables are included.

In some of the programs, you have the option of including the log of a variable and fixing its coefficient to 1.0. To include the variable (the program takes the log), set **_cn_Fix** to a variable name or number (**_cn_Fix** = "totals" or **_cn_Fix** = 12). The default (**_cn_Fix** = 0) includes no additional variable. In most event count data, the observation period is the same length for all i (a year, month, etc.). However, in others, the observation period varies. For example, suppose one observed the number of times a citizen was contacted by a candidate in the interval between two public opinion polls; since polls typically take some time to administer, the observation period would vary over the individuals. In still other situations, the observation period may be the same length but the population of potential events varies. For example, if one observed the number of suicides per state, one would need some way to include information on differing state sizes in the analysis. It turns out that both of these situations can be dealt with in the same way by including an additional variable in the stochastic portion of the model. But (as explained in King, 1989, Section 5.8), this procedure turns out to be mathematically equivalent to including the log of this additional variable in the regression component, and constraining its coefficient to

CLMcount

1.0. There is often little harm in just including the log of this variable and estimating its coefficient with all the others, but several of the programs allow one to make this constraint.

_cn_Dispersion scalar, set this to a value to change the starting value for only the dispersion parameter in the negative binomial (**Negbin**), generalized event count (**Hurdlep**), exponential-gamma (**Expgam**), Pareto (**Pareto**), and seemingly-unrelated Poisson models (**Supreme**, **Supreme2**). By default, a special starting value is not used for the dispersion parameter.

_cn_Precision scalar, the number of digits printed to the right of the decimal point on output. Default = 4.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

- **0** calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.
- **1** uses a vector of user supplied start values stored in the global variable **_cn_StartValue**.
- **2** uses a vector of zeros.
- **3** uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue L×1 vector, start values if **_cn_Start** = 1.

_cn_ZeroTruncate scalar, specifies whether or not the model is a truncated model. For the Poisson and negative binomial models, **_cn_ZeroTruncate** = 0 estimates a truncated-at-zero version of the model. By default, the regular untruncated model is estimated.

__altnam K×1 vector, alternate names for variables when a matrix is passed to a **COUNT** procedure. When a data matrix is passed to a **COUNT** procedure and the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__output scalar, determines printing of intermediate results.

- **0** nothing is written.
- **1** serial ASCII output format suitable for disk files or printers.
- **2** (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set are read per iteration of the read loop. By default, the number of rows to be read is calculated automatically.

__rowfac scalar, "row factor". If a **COUNT** procedure fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

causes **GAUSS** to read in 80% of the rows originally calculated.

This global has an affect only when **__row** = 0.

Default = 1.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

0 Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....

1 Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

4.2.4 Statistical Inference

MAXLIK statistical inference features may be accessed through the **COUNT** global, **_cn_Inference**. **_cn_Inference** has the following settings:

maxlik	maximum likelihood estimates
boot	bootstrapped estimates

That is to generate bootstrapped estimates, set

```
_cn_Inference = "boot";
```

Bootstrapping

In addition to the usual standard errors, you may generate bootstrap standard errors. Setting **_cn_Inference** = BOOT causes **COUNT** to call **MAXBoot**. This generates bootstrapped estimates and covariance matrices of the estimates.

The bootstrapped parameters are also stored in a **GAUSS** data set. The name of the data set can be determined by setting **_max_BootFname** to a file name, or by default it will be set to BOOT# where # is a four digit number incremented from 0001 until a name not in use is found. For further details about the bootstrap, see Section 2.6.4.

The data set thus generated can be used for computing confidence intervals of the coefficients using **MAXBlimits**. Also, density estimates and plots can be generated using **MAXDensity**, and histograms and surface plots of the coefficients can be produced using **MAXHist**. For further details about **MAXDensity**, see Section 2.6.4, and for further details about **MAXHist** see Section 2.6.4.

4.2.5 Problems with Convergence

All the programs use maximum likelihood estimation by numerically maximizing a different likelihood function. As with virtually all nonlinear iterative procedures, convergence works most of the time, but not every time. Problems to be aware of include the following:

1. The explanatory variables in each regression function should not be highly collinear among themselves.
2. The model should have more observations than parameters; indeed, the more observations, the better.
3. Starting values should not be too far from the optimal values.
4. The model specified should fit the data.
5. The Poisson hurdle model must have at least some observations with $y_i = 0$ and should take on at least two other values greater than zero.
6. The truncated models should have no observations with zeros (if inadvertently included, a message appears and the program stops).
7. The models with scalar dispersion parameters and variance functions should have maximum likelihood estimates that are bounded so that, for example, in the negative binomial model $\hat{\sigma}^2 > 1$

If you avoid the potential problems listed in the last paragraph, you should have little problem with convergence. Of course, avoiding these problems with difficult data sets is not always easy nor obvious. In these cases, problems may be indicated by the following situations:

1. iterations sending the parameters off in unreasonable directions or creating very large numbers.
2. the program actually bombing out.
3. a single iteration taking an extraordinarily long time.
4. the program taking more than 40 or 50 iterations with no convergence.

If one of these problems occur, you have several options. First, look over the list in the last paragraph. To verify that the problem does indeed exist, you might try running your data on the Poisson regression model if you have event count data, or the exponential regression model if you have duration data. Both are known to be globally concave and tend to converge very easily. If this model works, but another does not, you probably do have a problem.

In the case of problems, you must consider iteration a participatory process. When is iterating, you can press **Alt-H** to receive a list of options that may be changed during iteration. See *MAXLIK REFERENCE* for a full explanation of each. I find that the following practices tend to work well:

1. If the program has produced many iterations without much progress, try pressing **Alt-I** every few iterations to force the program to calculate the information matrix or switch Newton-Raphson iterations. Either of these may not work if the iterations are not far enough along.
2. The number of zeros to the right of the decimal point on the relative gradients (printed on the screen while the program is iterating) is the approximate precision of your final estimates. If the program is having trouble converging, but the gradients are small enough (i.e., you have sufficient precision for your substantive problem), press **Alt-C** to force the program to converge.
3. If the program bombs out very quickly, changing the starting values are your best bet (with the global **_cn_Start**). The default starting values created with least squares, **_cn_Start** = 0, usually works best. If that does not work, you can also try creating them yourself, by thinking about what the answer is likely to be or by running a simpler model. For example, the exponential-gamma model is sometimes problematic; however, the exponential model often provides good starting values for the effect parameters. Thus if the other methods do not work, you might try the following:

CLMcount

```
library count;
CountSet;
dep = { durat };
ind = { unem, infl, age };
dataset = "datafile";
{ b,vc,llik } = Expon(dataset,dep,ind);
_cn_StartValue = b;
_cn_Start = 1;
call Expgam(dataset,dep,ind);
```

You can also choose one of the other methods of creating starting values by changing the **_cn_Start** global (described above).

4.3 Annotated Bibliography

Allison, Paul. 1984. *Event History Analysis*. Beverly Hills: Sage. [A simple overview of event history methods for duration data.]

Bishop, Yvonne M.M.; Stephen E. Fienberg; and Paul W. Holland. 1975. *Discrete Multivariate Analysis* Cambridge, Mass.: M.I.T. Press. [Models for contingency tables.]

Cameron, A. Colin and Pravin K. Trivedi. 1986. "Econometric Models Based on Count Data: Comparisons and Applications of Some Estimators and Tests," *Journal of Applied Econometrics* 1, 29–53. [Review of the econometric literature on event counts.]

Grogger, Jeffrey T. and Richard T. Carson. 1988. "Models for Counts from Choice Based Samples," Discussion Paper 88-9, Department of Economics, University of California, San Diego. [Truncated event count models.]

Gourieroux, C.; A. Monfort; and A. Trognon. 1984. "Pseudo Maximum Likelihood Methods: Applications to Poisson Models," *Econometrica* 52: 701–720. [A three-stage robust estimation method for count data.]

Hall, Bronwyn H.; Zvi Griliches; and Jerry A. Hausman. 1986. "Patents and R and D: Is there a Lag?" *International Economic Review*. 27, 2 (June): 265–83. [Nice example of a applying a variety of different estimators to single equation count models.]

Hausman, Jerry; Bronwyn H. Hall; and Zvi Griliches. 1984. "Econometrics Models for Count Data with An Application to the Patents-R&D Relationship," *Econometrica*. 52, 4 (July): 909-938. [Count models for time-series cross sectional panels.]

Holden, Robert T. 1987. "Time Series Analysis of a Contagious Process," *Journal of the American Statistical Association*. 82, 400 (December): 1019–1026. [A time series model of count data applied to airline hijack attempts.]

Jorgenson, Dale W. 1961. "Multiple Regression Analysis of a Poisson Process," *Journal of the American Statistical Association* 56,294 (June): 235–45. [The Poisson regression model.]

Kalbfleisch, J.D. and R.L. Prentice. 1980. *The Statistical Analysis of Failure Time Data*. New York: Wiley. [Summary of research on many models of duration data.]

King, Gary. 1989a. *Unifying Political Methodology: The Likelihood Theory of Statistical Inference*. New York: Cambridge University Press. [Introduction to likelihood, maximum likelihood, and a large variety of statistical models as special cases; Chapter 5 is discrete regression models.]

__________. 1989b. "Variance Specification in Event Count Models: From Restrictive Assumptions to a Generalized Estimator," *American Journal of Political Science*, vol. 33, no. 3 (August):762-784. [Poisson-based models with over- and under-dispersion.]

__________. 1989c. "A Seemingly Unrelated Poisson Regression Model," *Sociological Methods and Research*. 17, 3 (February, 1989): 235–255. [A model for simultaneously analyzing a pair of event count variables in a SURM framework.]

__________. 1989d. "Event Count Models for International Relations: Generalizations and Applications," *International Studies Quarterly*, vol. 33, no. 3 (June):123-147. [Hurdle models, truncated models, and models with unobserved dependent variables, all for event count data.]

__________. 1988. "Statistical Models for Political Science Event Counts: Bias in Conventional Procedures and Evidence for The Exponential Poisson Regression Model," *American Journal of Political Science*, 32, 3 (August): 838–863. [Introduction to count models; analytical and Monte Carlo comparisons of LS, logged-LS, and Poisson regression models.]

__________. 1987. "Presidential Appointments to the Supreme Court: Adding Systematic Explanation to Probabilistic Description," *American Politics Quarterly*, 15, 3 (July): 373–386. [An application of the Poisson regression model.]

King, Gary; James Alt; Nancy Burns; Michael Laver. 1990. "A Unified Model of Cabinet Duration in Parliamentary Democracies," American Journal of Political Science, vol. 34, no. 3 (August):846-871. [Exponential model of duration data with censoring.]

McCullagh, P. And J.A. Nelder 1983. *Generalized Linear Models.* London: Chapman and Hall. [A unified approach to specifying and estimating this class of models. Some count and duration models are covered.]

Mullahy, John. 1986. "Specification and Testing of Some Modified Count Data Models," *Journal of Econometrics.* 33: 341–65. [Several hurdle-type models of event count data.]

Tuma, Nancy Brandon and Michael T. Hannan. 1984. *Social Dynamics.* New York: Academic Press.

Chapter 5

Count Reference

■ Purpose

Formats and prints the output from calls to **COUNT** procedures with confidence limits

■ Library

`count`

■ Format

{ *b*,*cl*,*llik* } = **CountCLPrt(***b*,*cl*,*llik***);**

■ Input

b	(K+1)×1 vector, maximum likelihood estimates of the effect parameters stacked on top of the dispersion parameter.
cl	$(K + 1) \times 2$ matrix, confidence limits
llik	scalar, value of the log-likelihood function at the maximum.

■ Output

The input arguments are returned unchanged.

■ Remarks

Confidence limits computed by **MAXBLimits** may be passed in the fourth argument in the call to **CountCLPrt**:

```
_cn_Inference = { boot };
{ b,vc,llik } = Expgam(dataset,dep,ind);
cl = MAXBlimits(_max_BootFname);
call CountCLPrt(b,cl,llik);
```

■ Source

`count.src`

Purpose

Formats and prints the output from calls to **COUNT** procedures.

Library

`count`

Format

{ *b*,*vc*,*llik* } = **CountPrt(***b*,*vc*,*llik***);**

Input

b	(K+1)×1 vector, maximum likelihood estimates of the effect parameters stacked on top of the dispersion parameter.
vc	K+1)×(K+1) matrix, variance-covariance matrix of the estimated parameters
llik	scalar, value of the log-likelihood function at the maximum.

Output

The input arguments are returned unchanged.

Remarks

The call to **COUNT** procedures can be nested in the call to the **CountPrt**:

```
{ b,vc,llik } = countprt(Expgam(dataset,dep,ind));
```

Source

`count.src`

Purpose

Resets **COUNT** global variables to default values.

Library

`count`

Format

CountSet;

Input

None

Output

None

Remarks

Putting this instruction at the top of all command files that invoke **COUNT** procedures is generally good practice. This prevents globals from being inappropriately defined when a command file is run several times or when a command file is run after another command file has executed that calls a **COUNT** procedure.

CountSet calls **Set** which calls **GAUSSET**.

Source

`count.src`

Purpose

Estimates an exponential-gamma regression model, for the analysis of duration data, with maximum likelihood.

Library

`count`

Format

{ *b*,*vc*,*llik* } = **Expgam(***dataset*,*dep*,*ind***);**

Input

dataset string, name of **GAUSS** data set.
– or –
N×K matrix, data

dep string, the name of the dependent variable.
– or –
scalar, the index of the dependent variable.

ind K×1 character vector, names of the independent variables.
– or –
K×1 numeric vector, indices of independent variables.
Set to 0 to include only a constant term.

If *dataset* is a matrix, *dep* or *ind* may be a string or character variable containing either the standard labels created by (V1, V2,..., or V01, V02,...., depending on the value of **__vpad**), or the user-provided labels in **__altnam**.

Output

b (K+1)×1 vector, maximum likelihood estimates of the effect parameters stacked on top of the dispersion parameter.

vc (K+1)×(K+1) matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If you choose the global **__CovPar** = 3, *vc* contains heteroskedastic-consistent parameter estimates.

llik scalar, value of the log-likelihood function at the maximum.

Globals

MAXLIK globals are also relevant

_cn_Inference string, determines the type of statistical inference.

- **boot** generates bootstrapped estimates and covariance matrix of estimates
- **maxlik** generates maximum likelihood estimates (default)

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Censor string, the name of the censor variable from *dataset*.
– or –
scalar, the index of the censor variable from *dataset*.
By default, no censoring is used.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

- **0** calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.
- **1** uses a vector of user supplied start values stored in the global variable **_cn_StartValue**.
- **2** uses a vector of zeros.
- **3** uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue (K+1)×1 vector, start values if **_cn_Start** = 1.

_cn_Precision scalar, number of decimal points to print on output. Default = 4.

__altnam K×1 vector, alternate names for variables when a matrix is passed to **Expgam**. When a data matrix is passed to **Expgam** and when the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__miss scalar, determines how missing data will be handled.

- **0** Missing values will not be checked for, and so the data set must not have any missings. This is the fastest option.
- **1** Listwise deletion. Removes from computation any observation with a missing value on any variable included in the analysis.

Default = 0.

__output scalar, determines printing of intermediate results.

0 nothing is written.

1 serial ASCII output format suitable for disk files or printers.

2 (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set will be read per iteration of the read loop. By default, the number of rows to be read will be calculated automatically.

__rowfac scalar, "row factor". If **Expgam** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

will cause **GAUSS** to read in 80% of the rows originally calculated.

This global has an affect only when **__row** = 0.

Default = 1.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

0 Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....

1 Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

Remarks

Let the n duration observations (nonnegative real numbers) for the dependent variable be denoted as $y_1, \ldots, y_n$. Assume that y_i follows a gamma distribution with expected value μ_i and variance $\mu_i^2\sigma^2$. Let the mean μ_i be an exponential-linear function of a vector of explanatory variables, x_i:

$$E(y_i) \equiv \mu_i = \exp(x_i\beta) \tag{5.1}$$

The program includes a constant term as the first column of x_i and allows one to include any number of explanatory variables. Note that μ_i from a duration model equals $1/\lambda_i$ from an event count model; thus, one need only change the sign of the effect parameters to get estimates of the same parameters from these different kinds of data.

The dispersion σ^2 is parametrized as follows:

$$\sigma_i^2 = \exp(\gamma) \tag{5.2}$$

EXPGAM reports estimates of β and γ.

For an introduction to the exponential gamma regression model see King, Alt, Burns, and Laver (1989) or Kalbfleisch and Prentice (1980).

Example

Exponential-Gamma Regression Model of Duration Data

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind = { unem, poverty, allianc };
{ b,vc,llik } = Expgam(dataset,dep,ind);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

A vector of effect parameters and a scalar dispersion parameter are estimated. The vector includes one element corresponding to each explanatory variable named in *ind* and a constant term. Five parameters are estimated in this example.

Censored Exponential-Gamma Regression Model of Duration Data

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind = { unem, poverty, allianc };
_Censor = { v12 };
{ b,vc,llik } = Expgam(dataset,dep,ind);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

A vector of effect parameters and a scalar dispersion parameter are estimated. The vector includes one element corresponding to each explanatory variable named in *ind* and a constant term. Five parameters are estimated in this example.

Source

`expgam.src`

Purpose

Estimates an exponential regression model or censored exponential regression model with maximum likelihood.

Library

`count`

Format

{ *b*,*vc*,*llik* } = **Expon(***dataset*,*dep*,*ind***);**

Input

dataset	string, name of **GAUSS** data set. – or – N×K matrix, data
dep	string, the name of the dependent variable – or – scalar, the index of the dependent variable
ind	K×1 character vector, names of the independent variables – or – K×1 numeric vector, indices of independent variables Set to 0 to include only a constant term.

If *dataset* is a matrix, *dep* or *ind* may be a string or character variable containing either the standard labels created by (V1, V2,..., or V01, V02,...., depending on the value of **__vpad**), or the user-provided labels in **__altnam**.

Output

b	K×1 vector, maximum likelihood estimates of the effect parameters.
vc	K×K matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If the global **__CovPar** is set to 3, *vc* will contain heteroskedastic-consistent parameter estimates.
llik	scalar, value of the log-likelihood function at the maximum.

Globals

MAXLIK globals are also relevant.

_cn_Inference string, determines the type of statistical inference.

- **boot** generates bootstrapped estimates and covariance matrix of estimates
- **maxlik** generates maximum likelihood estimates

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Censor string, the name of the censor variable from *dataset*
– or –
scalar, the index of the censor variable from *dataset*
By default, no censoring is used.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

- **0** calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.
- **1** will use a vector of user supplied start values stored in the global variable **_cn_StartValue**.
- **2** uses a vector of zeros.
- **3** uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue K×1 vector, start values if **_cn_Start** = 1.

_cn_Precision scalar, number of decimal points to print on output. Default = 4.

__altnam K×1 vector, alternate names for variables when a matrix is passed to **Expon**. When a data matrix is passed to **Expon** and the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__miss scalar, determines how missing data will be handled.

- **0** Missing values will not be checked for, and so the data set must not have any missings. This is the fastest option.
- **1** Listwise deletion. Removes from computation any observation with a missing value on any variable included in the analysis.

Default = 0.

__output scalar, determines printing of intermediate results.

- **0** nothing is written.
- **1** serial ASCII output format suitable for disk files or printers.
- **2** (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set will be read per iteration of the read loop. By default, the number of rows to be read will be calculated automatically.

__rowfac scalar, "row factor". If **EXPON** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

will cause **GAUSS** to read in 80% of the rows originally calculated.

This global only has an affect when **__row** = 0.

Default = 1.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

- **0** Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....
- **1** Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

■ Remarks

Let y_i $(i = 1, \ldots, n)$ take on any non-negative real number representing a duration. Often y_i is only measured as an integer, such as the number of days or months. Even so, if your dependent variable is a measure of time, duration models, and not event count models, are called for. Let y_i be distributed exponentially with mean μ_i. Also let $E(y_i) \equiv \mu_i = \exp(x_i\beta)$. Note that μ_i from a duration model equals $1/\lambda_i$ from an event count model; thus, one need only change the sign of the effect parameters to get estimates of the same parameters from these different kinds of data.

For an introduction to the exponential regression model and the censored exponential regression model see Kalbfleisch and Prentice (1980) and King, Alt, Burns, and Laver (1989).

Example

Exponential Regression Model

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind = { unem, poverty, allianc };
{ b,vc,llik } = Expon(dataset,dep,ind);
output file = count.out on;
call CountPrt(b,vc,llik);
output off;
```

A single vector of effect parameters are estimated. This vector includes one element corresponding to each explanatory variable named in *ind* and a constant term.

Censored Exponential Regression Model

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind = { unem, poverty, allianc };
_cn_Censor = { notseen };
{ b,vc,llik } = Expon(dataset,dep,ind);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

A single vector of effect parameters are estimated. This vector includes one element corresponding to each explanatory variable named in *ind* and a constant term.

Source

`expon.src`

■ Purpose

Estimates a hurdle Poisson regression model, for the analysis of event counts, with maximum likelihood.

■ Library

`count`

■ Format

{ *b*,*vc*,*llik* } = **Hurdlep(***dataset*,*dep*,*ind***);**

■ Input

dataset string, name of **GAUSS** data set.
– or –
N×K matrix, data

dep string, the name of the dependent variable
– or –
scalar, the index of the dependent variable

ind1 K×1 character vector, names of first event independent variables
– or –
K×1 numeric vector, indices of first event independent variables

ind2 K×1 character vector, names of second event independent variables
– or –
K×1 numeric vector, indices of second event independent variables

If *dataset* is a matrix, *dep*, *ind1*, or *ind2* may be a string or character variable containing either the standard labels created by (V1, V2,..., or V01, V02,...., depending on the value of **__vpad**), or the user-provided labels in **__altnam**.

■ Output

b (K+L)×1 vector, maximum likelihood estimates of the effect parameters stacked on top of the dispersion parameter.

vc (K+L)×(K+L) matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If you choose the global **__CovPar** = 3, *vc* will contain heteroskedastic-consistent parameter estimates.

llik scalar, value of the log-likelihood function at the maximum.

Globals

MAXLIK globals are also relevant

_cn_Inference string, determines the type of statistical inference.

boot generates bootstrapped estimates and covariance matrix of estimates

maxlik generates maximum likelihood estimates

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

0 calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.

1 will use a vector of user supplied start values stored in the global variable **_cn_StartValue**.

2 uses a vector of zeros.

3 uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue (K+L)×1 vector, start values if **_cn_Start** = 1.

_cn_Precision scalar, number of decimal points to print on output. Default = 4.

__altnam K×1 vector, alternate names for variables when a matrix is passed to **Hurdlep**. When a data matrix is passed to **Hurdlep** and the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__miss scalar, determines how missing data will be handled.

0 Missing values will not be checked for, and so the data set must not have any missings. This is the fastest option.

1 Listwise deletion. Removes from computation any observation with a missing value on any variable included in the analysis.

Default = 0.

__output scalar, determines printing of intermediate results.

- **0** nothing is written.
- **1** serial ASCII output format suitable for disk files or printers.
- **2** (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set will be read per iteration of the read loop. By default, the number of rows to be read will be calculated automatically.

__rowfac scalar, "row factor". If **Hurdlep** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

will cause **GAUSS** to read in 80% of the rows originally calculated.

This global only has an affect when **__row** = 0.

Default = 1.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

- **0** Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....
- **1** Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

Remarks

Let the n event count observations (nonnegative integers) for the dependent variable be denoted as $y_1, \ldots, y_n$. y_i is then a random dependent variable representing the number of events that have occurred during observation period i. Let λ_{0i} be the rate of the first event occurrence and λ_{+i} be the rate for all additional events after the first. If these are the expected values of two separate Poisson processes, we have the hurdle Poisson regression model. These means are parametrized as usual:

$$\lambda_{0i} = \exp(x_i\beta) \tag{5.3}$$

and

$$\lambda_{+i} = \exp(z_i\gamma) \tag{5.4}$$

where x_i and z_i are (possibly) different vectors of explanatory variables. The program produces estimates of β and γ. If $\beta = \gamma$ and $x = z$, this model reduces to the Poisson.

For an introduction to the Hurdle Poisson regression model see Mullahy (1986) and King (1989d).

Example

Hurdle Poisson Regression Model:

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind1 = { unem, poverty, allianc };
ind2 = { race, sex, age, partyid, x4, v5 };
{ b,vc,llik } =  Hurdlep(dataset,dep,ind1,ind2);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

Two vectors of effect parameters are estimated. Each includes one element corresponding to each explanatory variable plus a constant term (in the example, four parameters appear in the first regression function and seven in the second).

Source

`hurdlep.src`

Purpose

Estimates a negative binomial regression model or truncated-at-zero negative binomial regression model with maximum likelihood.

Library

`count`

Format

{ *b*,*vc*,*llik* } = **Negbin(***dataset*,*dep*,*ind1*,*ind2***);**

Input

dataset string, name of **GAUSS** data set.
– or –
N×K matrix, data

dep string, the name of the dependent variable
– or –
scalar, the index of the dependent variable

ind1 K×1 character vector, names of first event independent variables
– or –
K×1 numeric vector, indices of first event independent variables
Set to 0 to include only a constant term.

ind2 K×1 character vector, names of second event independent variables
– or –
K×1 numeric vector, indices of second event independent variables
Set to 0 for a scalar dispersion parameter.

If *dataset* is a matrix, *dep*, *ind1*, or *ind2* may be a string or character variable containing either the standard labels created by (V1, V2,..., or V01, V02,...., depending on the value of **__vpad**), or the user-provided labels in **__altnam**.

Output

b (K+1)×1 or (K+L)×1 vector, maximum likelihood estimates of the effect parameters stacked on top of either the dispersion parameter or the coefficients of the variance function.

vc (K+1)×(K+1) or (K+L)×(K+L) matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If you choose the global **__CovPar** = 3, *vc* will contain heteroskedastic-consistent parameter estimates.

llik scalar, value of the log-likelihood function at the maximum.

Globals

MAXLIK globals are also relevant.

_cn_Inference string, determines the type of statistical inference.

boot generates bootstrapped estimates and covariance matrix of estimates

maxlik generates maximum likelihood estimates

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Fix scalar, name of index number of logged variable among the regressors with coefficient constrained to 1.0 By default, no logged variables are included.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

0 calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.

1 will use a vector of user supplied start values stored in the global variable **_cn_StartValue**.

2 uses a vector of zeros.

3 uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue (K+1)×1 or (K+L)×1 vector, start values if **_cn_Start** = 1.

_cn_Dispersion scalar, start value for scalar dispersion parameter. Default = 3.

_cn_Precision scalar, number of decimal points to print on output. Default = 4.

_cn_ZeroTruncate scalar, specifies which model is used:

0 truncated-at-zero negative binomial model

1 negative binomial model is used.

__altnam K×1 vector, alternate names for variables when a matrix is passed to **Negbin**. When a data matrix is passed to **Negbin** and the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__miss scalar, determines how missing data will be handled.

- **0** Missing values will not be checked for, and so the data set must not have any missings. This is the fastest option.
- **1** Listwise deletion. Removes from computation any observation with a missing value on any variable included in the analysis.

Default = 0.

__output scalar, determines printing of intermediate results.

- **0** nothing is written.
- **1** serial ASCII output format suitable for disk files or printers.
- **2** (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set will be read per iteration of the read loop. By default, the number of rows to be read will be calculated automatically.

__rowfac scalar, "row factor". If **Negbin** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

will cause **GAUSS** to read in 80% of the rows originally calculated.

This global only has an affect when **__row** = 0.

Default = 1.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

- **0** Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....

1 Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

Remarks

Let y_i be a random dependent variable representing the number of events that have occurred during observation period i ($i = 1, \ldots, n$). Assume that y_i follows a negative binomial distribution with expected value λ_i and variance $\lambda_i \sigma^2$. Let the mean λ_i (the rate of event occurrence, which must be greater than zero) be an exponential-linear function of a vector of explanatory variables, x_i:

$$E(y_i) \equiv \lambda_i = \exp(x_i \beta) \tag{5.5}$$

The program includes a constant term as the first column of x_i and allows one to include any number of explanatory variables.

σ^2 is parametrized as follows:

$$\sigma_i^2 = 1 + \exp(z_i \gamma) \tag{5.6}$$

where $z_i = 1$, if estimating a scalar dispersion parameter, or a vector of explanatory variables, if estimating a variance function. The program calculates estimates of β and γ.

For an introduction to the negative binomial regression model, see Hausman, Hall, and Griliches (1984) and King (1989b); for information on the truncated negative binomial model, see Grogger and Carson (1988), and on the variance function model with or without truncation see King (1989d: Section 5)

Example

Negative Binomial Regression Model

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind1 = { unem, poverty, allianc };
{ b,vc,llik } = Negbin(dataset,dep,ind1,0);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

A single vector of effect parameters and one scalar dispersion parameter are estimated. The vector of effect parameters includes one element corresponding to each explanatory variable and a constant term. In the example, five parameters are estimated.

Negative Binomial Variance Function Regression Model

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep1 = { wars };
ind1 = { unem, poverty, allianc };
ind2 = { partyid, x4 };
{ b,vc,llik } = Negbin(dataset,dep,ind1,ind2);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

Two vectors of effect parameters are estimated, one for the mean *ind1* and one for the variance function *ind2*. Each vector includes a constant term and one element corresponding to each explanatory variable. The example estimates seven parameters.

Truncated-at-zero Negative Binomial Regression Model

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep1 = { wars };
ind1 = { unem, poverty, allianc };
_cn_ZeroTruncate = 0;
{ b,vc,llik } = Negbin(dataset,dep,ind1,0);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

A single vector of effect parameters and one scalar dispersion parameter are estimated. The vector of effect parameters includes one element corresponding to each explanatory variable and a constant term. In the example, five parameters are estimated.

Truncated-at-zero Negative Binomial Variance Function Regression Model

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep1 = { wars };
ind1 = { unem, poverty, allianc };
ind2 = { partyid, x4 };
_cn_ZeroTruncate = 0;
{ b,vc,llik } = Negbin(dataset,dep,ind1,0);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

Two vectors of effect parameters are estimated, one for the mean and one for the variance function. Each vector includes a constant term and one element corresponding to each explanatory variable. In the example, the variables specified in *ind1* pertain to the expected value and *ind2* to the variance. Seven parameters are estimated.

Source

`negbin.src`

Purpose

Estimates a Pareto regression model, for the analysis of duration data, with maximum likelihood.

Library

`count`

Format

{ *b*,*vc*,*llik* } = **Pareto(***dataset*,*dep*,*ind***);**

Input

dataset string, name of **GAUSS** data set.
– or –
N×K matrix, data

dep string, the name of the dependent variable
– or –
scalar, the index of the dependent variable

ind K×1 character vector, names of the independent variables
– or –
K×1 numeric vector, indices of independent variables
Set to 0 to include only a constant term.

If *dataset* is a matrix, *dep* and *ind* may be a string or character variable containing either the standard labels created by (V1, V2,..., or V01, V02,...., depending on the value of **__vpad**), or the user-provided labels in **__altnam**.

Output

b (K+1)×1 vector, maximum likelihood estimates of the effect parameters stacked on top of the dispersion parameter.

vc (K+1)×(K+1) matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If the global **__CovPar** is set to 3, *vc* will contain heteroskedastic-consistent parameter estimates.

llik scalar, value of the log-likelihood function at the maximum.

Globals

MAXLIK globals are also relevant.

_cn_Inference string, determines the type of statistical inference.

boot generates bootstrapped estimates and covariance matrix of estimates

maxlik generates maximum likelihood estimates

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Censor string, the name of the censor variable from *dataset*
– or –
scalar, the index of the censor variable from *dataset*

Each element of censor variable is 0 if censored, or 1 if not.

By default, no censoring is used.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

0 calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.

1 will use a vector of user supplied start values stored in the global variable **_cn_StartValue**.

2 uses a vector of zeros.

3 uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue (K+1)×1 vector, start values if **_cn_Start** = 1.

_cn_Dispersion scalar, start value for scalar dispersion parameter. Default = 3.

_cn_Precision scalar, number of decimal points to print on output. Default = 4.

__altnam K×1 vector, alternate names for variables when a matrix is passed to **Pareto**. When a data matrix is passed to **Pareto** and the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__miss scalar, determines how missing data will be handled.

0 Missing values will not be checked for, and so the data set must not have any missings. This is the fastest option.

1 Listwise deletion. Removes from computation any observation with a missing value on any variable included in the analysis.

Default = 0.

__output scalar, determines printing of intermediate results.

0 nothing is written.

1 serial ASCII output format suitable for disk files or printers.

2 (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set will be read per iteration of the read loop. By default, the number of rows to be read will be calculated automatically.

__rowfac scalar, "row factor". If **Pareto** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

will cause **GAUSS** to read in 80% of the rows originally calculated.

This global only has an affect when **__row** = 0.

Default = 1.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

0 Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....

1 Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

Remarks

Let the n duration observations (non-negative real numbers) for the dependent variable be denoted as $y_1, \ldots, y_n$. Assume that y_i follows a Pareto distribution with expected value μ_i and variance $\mu_i\sigma^2 + \mu_i^2$. Let the mean μ_i be an exponential-linear function of a vector of explanatory variables, x_i:

$$E(y_i) \equiv \mu_i = \exp(x_i\beta) \tag{5.7}$$

The program includes a constant term as the first column of x_i and allows one to include any number of explanatory variables. Note that μ_i from a duration model equals $1/\lambda_i$ from an event count model; thus, one need only change the sign of the effect parameters to get estimates of the same parameters from these different kinds of data.

The dispersion σ^2 is parametrized as follows:

$$\sigma_i^2 = \exp(\gamma) \tag{5.8}$$

The program gives estimates of β and γ.

For an introduction to the Pareto regression model see Hannan and Tuma (1984) and King, Alt, Burns, and Laver (1989).

Example

Pareto Regression Model of Duration Data

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind = { unem, poverty, allianc };
{ b,vc,llik } = Pareto(dataset,dep,ind);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

A vector of effect parameters and a scalar dispersion parameter are estimated. The vector includes one element corresponding to each explanatory variable named in *ind* and a constant term. Five parameters are estimated in this example.

Censored Pareto Regression Model of Duration Data

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind = { unem, poverty };
_cn_Censor = { cvar };
{ b,vc,llik } = Pareto(dataset,dep,ind);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

A vector of effect parameters and a scalar dispersion parameter are estimated. The vector includes one element corresponding to each explanatory variable named in *ind* and a constant term. Five parameters are estimated in this example.

Source

pareto.src

Purpose

Estimates a Poisson regression model or truncated-at-zero Poisson regression model with maximum likelihood.

Library

`count`

Format

{ *b*,*vc*,*llik* } = **Poisson(***dataset*,*dep*,*ind***);**

Input

dataset string, name of **GAUSS** data set.
– or –
N×K matrix, data

dep string, the name of the dependent variable
– or –
scalar, the index of the dependent variable

ind K×1 character vector, names of the independent variables
– or –
K×1 numeric vector, indices of independent variables
Set to 0 to include only a constant term.

If *dataset* is a matrix, *dep* and *ind* may be a string or character variable containing either the standard labels created by (V1, V2,..., or V01, V02,...., depending on the value of **__vpad**), or the user-provided labels in **__altnam**.

Output

b K×1 vector, maximum likelihood estimates of the effect parameters.

vc K×K matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If you choose the global **__CovPar** = 3, *vc* will contain heteroskedastic-consistent parameter estimates.

llik scalar, value of the log-likelihood function at the maximum.

Globals

MAXLIK globals are also relevant.

_cn_Inference string, determines the type of statistical inference.

boot generates bootstrapped estimates and covariance matrix of estimates

maxlik generates maximum likelihood estimates

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Fix scalar, name of index number of logged variable among the regressors with coefficient constrained to 1.0 By default, no logged variables are included.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

0 calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.

1 will use a vector of user supplied start values stored in the global variable **_cn_StartValue**.

2 uses a vector of zeros.

3 uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue K×1 vector, start values if **_cn_Start** = 1.

_cn_Precision scalar, number of decimal points to print on output. Default = 4.

_cn_ZeroTruncate scalar, specifies which model is used:

0 truncated-at-zero negative binomial model

1 negative binomial model is used.

Default = 1.

__altnam K×1 vector, alternate names for variables when a matrix is passed to **Poisson**. When a data matrix is passed to **Poisson** and the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__miss scalar, determines how missing data will be handled.

0 Missing values will not be checked for, and so the data set must not have any missings. This is the fastest option.

1 Listwise deletion. Removes from computation any observation with a missing value on any variable included in the analysis.

Default = 0.

__output scalar, determines printing of intermediate results.

0 nothing is written.

1 serial ASCII output format suitable for disk files or printers.

2 (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set will be read per iteration of the read loop. By default, the number of rows to be read will be calculated automatically.

__rowfac scalar, "row factor". If **POISSON** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

will cause **GAUSS** to read in 80% of the rows originally calculated.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

0 Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....

1 Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

Remarks

Let the n event count observations (non-negative integers) for the dependent variable be denoted as $y_1, \ldots, y_n$. y_i is then a random dependent variable representing the number of events that have occurred during observation period i. By assuming that the events occurring within each period are independent and have constant rates of occurrence, y_i can be shown to follow a Poisson distribution:

$$f_p(y_i|\lambda_i) = \begin{cases} \frac{e^{-\lambda_i}(\lambda_i)^{y_i}}{y_i!} & \text{for } \lambda_i > 0 \text{ and } y_i = 0, 1, \ldots \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

with expected value and variance λ_i. Under the Poisson regression model, λ_i (the rate of event occurrence, which must be greater than zero) is assumed to be an exponential-linear function of a vector of explanatory variables, x_i:

$$E(y_i) \equiv \lambda_i = \exp(x_i\beta) \tag{5.10}$$

The program includes a constant term as the first element of x_i and allows one to include any number of explanatory variables.

For an introduction to the Poisson regression model see King (1988); on the truncated model, see Grogger and Carson (1988) and King (1989d).

Example

Poisson Regression Model

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep = { wars };
ind = { unem, poverty, allianc };
{ b,vc,llik } = Poisson(dataset,dep,ind);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

Truncated-at-zero Poisson Regression Model

```
library count;
#include count.ext;
Countset;
dataset = "wars";
```

```
dep = { wars };
ind = { unem, poverty, allianc };
_cn_ZeroTruncate = 0;
{ b,vc,llik } = Poisson(dataset,dep,ind);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

Source

poisson.src

Purpose

Estimates a seemingly unrelated Poisson regression model, for the analysis of two event **COUNT** variables, with maximum likelihood.

Library

`count`

Format

{ *b*,*vc*,*llik* } = **Supreme(***dataset*,*dep1*,*dep2*,*ind1*,*ind2***);**

Input

dataset string, name of **GAUSS** data set.
– or –
N×K matrix, data

dep1 string, name of the first dependent variable
– or –
scalar, index of the first dependent variable

dep2 string, name of the second dependent variable
– or –
scalar, index of the second dependent variable

ind1 K×1 character vector, names of first event independent variables
– or –
K×1 numeric vector, indices of first event independent variables
Set to 0 to include only a constant term.

ind2 K×1 character vector, names of second event independent variables
– or –
K×1 numeric vector, indices of second event independent variables
Set to 0 to include only a constant term.

If *dataset* is a matrix, *dep1*, *dep2*, *ind1* and *ind2* may be a string or character variable containing either the standard labels created by (V1, V2,..., or V01, V02,...., depending on the value of **__vpad**), or the user-provided labels in **__altnam**.

Output

b (K+L+2)×1 vector, maximum likelihood estimates of the effect parameters of β and γ stacked on top of the covariance parameter ξ.

vc (K+L+2)×(K+L+2) matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If you choose the global **__CovPar** = 3, *vc* will contain heteroskedastic-consistent parameter estimates.

llik scalar, value of the log-likelihood function at the maximum.

Globals

MAXLIK globals are also relevant.

_cn_Inference string, determines the type of statistical inference.

boot generates bootstrapped estimates and covariance matrix of estimates

maxlik generates maximum likelihood estimates

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

0 calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.

1 will use a vector of user supplied start values stored in the global variable **_cn_StartValue**.

2 uses a vector of zeros.

3 uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue (K+L+2)×1 vector, start values if **_cn_Start** = 1.

_cn_Precision scalar, number of decimal points to print on output. Default = 4.

__altnam K×1 vector, alternate names for variables when a matrix is passed to **Supreme**. When a data matrix is passed to **Supreme** and the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__miss scalar, determines how missing data will be handled.

0 Missing values will not be checked for, and so the data set must not have any missings. This is the fastest option.

1 Listwise deletion. Removes from computation any observation with a missing value on any variable included in the analysis.

Default = 0.

__output scalar, determines printing of intermediate results.

0 nothing is written.

1 serial ASCII output format suitable for disk files or printers.

2 (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set will be read per iteration of the read loop. By default, the number of rows to be read will be calculated automatically.

__rowfac scalar, "row factor". If **Supreme** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

will cause **GAUSS** to read in 80% of the rows originally calculated.

This global only has an affect when **__row** = 0.

Default = 1.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

0 Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....

1 Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

Remarks

Suppose we observe two event count dependent variables y_{1i} and y_{2i} for n observations. Let these variables be distributed as a bivariate Poisson with $E(y_{1i}) = \lambda_{1i}$ and $E(y_{2i}) = \lambda_{2i}$. These means are parametrized as follows:

$$\lambda_{0i} = \exp(x_i\beta) \tag{5.11}$$

and

$$\lambda_{+i} = \exp(z_i\gamma) \tag{5.12}$$

where x_i and z_i are (possibly) different vectors of explanatory variables. The covariance parameter is ξ.

If you have convergence problems, you might try **Supreme2** with argument *ind3* $= 0$ instead.

For details about this model, see King (1989c).

Example

Seemingly Unrelated Poisson Regression Model (Supreme)

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep1 = { wars };
ind1 = { unem, poverty, allianc };
dep2 = { coups };
ind2 = { unem, age, sex, race };
{ b,vc,llik } = Supreme(dataset,dep1,dep2,ind1,ind2);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

Two vectors of effect parameters and one scalar covariance parameter are estimated. The vectors of effect parameters each include one element corresponding to each explanatory variable and a constant term. In the example, ten parameters are estimated.

Source

supreme.src

Purpose

Estimates a Poisson regression model with unobserved dependent variables, for the analysis of two observed (and three unobserved) event count variables, with maximum likelihood.

Library

`count`

Format

`{` *b*,*vc*,*llik* `}` = **Supreme2(***dataset*,*dep1*,*dep2*,*ind1*,*ind2*,*ind3***);**

Input

dataset string, name of **GAUSS** data set.
– or –
N×K matrix, data

dep1 string, name of the first dependent variable
– or –
scalar, index of the first dependent variable

dep2 string, name of the second dependent variable
– or –
scalar, index of the second dependent variable

ind1 K×1 character vector, names of first event independent variables
– or –
K×1 numeric vector, indices of first event independent variables
Set to 0 to include only a constant term.

ind2 L×1 character vector, names of second event independent variables
– or –
L×1 numeric vector, indices of second event independent variables
Set to 0 to include only a constant term.

ind3 M×1 character vector, names of second event independent variables
– or –
M×1 numeric vector, indices of second event independent variables
Set to 0 to include only a constant term.

If *dataset* is a matrix, *dep1*, *dep2*, *ind1*, *ind2*, or *ind3* may be a string or character variable containing either the standard labels created by (V1, V2,..., or V01, V02,...., depending on the value of **__vpad**), or the user-provided labels in **__altnam**.

Output

b (K+L+M)×1 vector, maximum likelihood estimates of the effect parameters of β and γ stacked on top of the covariance parameter ξ.

vc (K+L+M)×(K+L+M) matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If you choose the global **__CovPar** = 3, *vc* will contain heteroskedastic-consistent parameter estimates.

llik scalar, value of the log-likelihood function at the maximum.

Globals

MAXLIK globals are also relevant.

_cn_Inference string, determines the type of statistical inference.

boot generates bootstrapped estimates and covariance matrix of estimates

maxlik generates maximum likelihood estimates

Setting **_cn_Inference** to BOOT generates a **GAUSS** data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the **MAXLIK** global variable, **_max_BootFname**. This data set can be used with **MAXBlimits** for generating confidence limits, with **MAXDensity** for generating density estimates and plots of the boostrapped parameters, or with **MAXHist** for generating histogram and surface plots.

_cn_Start scalar, selects method of calculating starting values. Possible values are:

0 calculates them by regressing $\ln(y + 0.5)$ on the explanatory variables.

1 will use a vector of user supplied start values stored in the global variable **_cn_StartValue**.

2 uses a vector of zeros.

3 uses random uniform numbers on the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Default = 0.

_cn_StartValue (K+L+M)×1 vector, start values if **_cn_Start** = 1.

_cn_Precision scalar, number of decimal points to print on output. Default = 4.

__altnam K×1 vector, alternate names for variables when a matrix is passed to **Supreme2**. When a data matrix is passed to **Supreme2** and the user is selecting from that matrix, the global variable **__altnam**, if it is used, must contain names for the columns of the original matrix.

__miss scalar, determines how missing data will be handled.

- **0** Missing values will not be checked for, and so the data set must not have any missings. This is the fastest option.
- **1** Listwise deletion. Removes from computation any observation with a missing value on any variable included in the analysis.

Default = 0.

__output scalar, determines printing of intermediate results.

- **0** nothing is written.
- **1** serial ASCII output format suitable for disk files or printers.
- **2** (DOS only) output is suitable for screen only. ANSI.SYS must be active.

Default = 2.

__row scalar, specifies how many rows of the data set will be read per iteration of the read loop. By default, the number of rows to be read will be calculated automatically.

__rowfac scalar, "row factor". If **Supreme2** fails due to insufficient memory while attempting to read a **GAUSS** data set, then **__rowfac** may be set to some value between 0 and 1 to read a *proportion* of the original number of rows of the **GAUSS** data set. For example, setting

```
__rowfac = 0.8;
```

will cause **GAUSS** to read in 80% of the rows originally calculated.

This global only has an affect when **__row** = 0.

Default = 1.

__title string, message printed at the top of the screen and printed out by **CountPrt**. Default = "".

__vpad scalar, if *dataset* is a matrix in memory, the variable names are automatically created by . Two types of names can be created:

- **0** Variable names automatically created by are not padded to give them equal length. For example, V1, V2,...V10, V11,....
- **1** Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, V01, V02, ..., V10, V11,.... This is useful if you want the variable names to sort properly.

Default = 1.

Remarks

This model assumes the existence of three independent unobserved variables, y^*_{1i}, y^*_{2i}, and y^*_{3i}, with means $E(y^*_{ji}) = \lambda_{ji}$, for $j = 1, 2, 3$. Although these are not observed, we do observe y_{1i} and y_{2i}, which are functions of these three variables:

$$\begin{aligned} y_{1i} &= y^*_{1i} + y^*_{3i} \\ y_{2i} &= y^*_{2i} + y^*_{3i} \end{aligned}$$

The procedure estimates three separate regression functions, one for the expected value of each of the unobserved variables:

$$\lambda_{1i} = \exp(x_{1i}\beta_1) \tag{5.13}$$

$$\lambda_{2i} = \exp(x_{2i}\beta_2)$$

$$\lambda_{3i} = \exp(x_{3i}\beta_3)$$

where x_{1i}, x_{2i} and x_{3i} are (possibly) different sets of explanatory variables and β_1, β_2, and β_3 are separate parameter vectors. This option produces maximum likelihood estimates for these three parameter vectors.

Example

Poisson Regression Model with Unobserved Dependent Variables

```
library count;
#include count.ext;
Countset;
dataset = "wars";
dep1 = { wars };
ind1 = { unem, poverty, allianc };
dep2 = { coups };
ind2 = { unem, age, sex, race };
ind3 = { us, sov };
{ b,vc,llik } = Supreme2(dataset,dep1,dep2,ind1,ind2,ind3);
output file = count.out reset;
call CountPrt(b,vc,llik);
output off;
```

Three vectors of effect parameters are estimated. Each includes one element corresponding to each explanatory variable plus a constant term. In the example, twelve parameters are estimated.

Source

`supreme2.src`

Index