Maximum Likelihood Estimation 5.0

for GAUSS™ Mathematical and Statistical System
Information in this document is subject to change without notice and does not represent a commitment on the part of Aptech Systems, Inc. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. The purchaser may make one copy of the software for backup purposes. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Aptech Systems, Inc.


GAUSS, GAUSS Engine and GAUSS Light are trademarks of Aptech Systems, Inc. Other trademarks are the property of their respective owners.

Part Number: 001307
Version 5.0
Documentation Revision: 2173 June 12, 2012
# Contents

1 Installation

1.1 UNIX/Linux/Mac ................................................. 1-1
   1.1.1 Download .............................................. 1-1
   1.1.2 CD .................................................... 1-2

1.2 Windows ...................................................... 1-2
   1.2.1 Download .............................................. 1-2
   1.2.2 CD .................................................... 1-2
   1.2.3 64-Bit Windows ........................................ 1-3

1.3 Difference Between the UNIX and Windows Versions .......... 1-3

2 Getting Started

3 Maximum Likelihood Estimation

3.1 The Log-likelihood Function .................................... 3-1

3.2 Algorithm .................................................... 3-2
   3.2.1 Derivatives ............................................ 3-3
   3.2.2 The Secant Algorithms ................................ 3-3
   3.2.3 Convergence .......................................... 3-5
   3.2.4 Berndt, Hall, Hall, and Hausman’s (BHHH) Method .... 3-5
   3.2.5 Polak-Ribiere-type Conjugate Gradient (PRCG) ........... 3-5
   3.2.6 Line Search Methods ................................... 3-6
   3.2.7 Random Search ........................................ 3-8
   3.2.8 Weighted Maximum Likelihood ......................... 3-8
   3.2.9 Active and Inactive Parameters ......................... 3-9
   3.2.10 Example .............................................. 3-9

3.3 Managing Optimization ........................................ 3-11
3.3.1 Scaling ................................................................. 3-11
3.3.2 Condition ............................................................ 3-12
3.3.3 Starting Point ....................................................... 3-16
3.3.4 Diagnosis ............................................................ 3-17
3.4 Gradients ............................................................... 3-18
  3.4.1 Analytical Gradient .............................................. 3-18
  3.4.2 User-Supplied Numerical Gradient ......................... 3-19
  3.4.3 Algorithmic Derivatives ....................................... 3-19
  3.4.4 Analytical Hessian ............................................... 3-25
  3.4.5 User-Supplied Numerical Hessian ......................... 3-27
  3.4.6 Switching Algorithms Automatically .................... 3-28
3.5 FASTMAX – Fast Execution MAXLIK ................................ 3-28
  3.5.1 Undefined Function Evaluation .............................. 3-29
3.6 Inference ............................................................. 3-29
  3.6.1 Wald Inference .................................................. 3-30
  3.6.2 Profile Likelihood Inference ................................. 3-33
  3.6.3 Profile Trace Plots ............................................. 3-36
  3.6.4 Bootstrap ......................................................... 3-38
  3.6.5 Pseudo-Random Number Generators ..................... 3-39
  3.6.6 Bayesian Inference ............................................ 3-40
3.7 Run-Time Switches .................................................. 3-43
3.8 Calling MAXLIK Recursively .................................... 3-45
3.9 Using MAXLIK Directly .......................................... 3-45
3.10 Error Handling ...................................................... 3-46
  3.10.1 Return Codes .................................................. 3-46
  3.10.2 Error Trapping ................................................ 3-47
3.11 References .......................................................... 3-48

4 Maximum Likelihood Reference
FASTMAX ................................................................. 4-1
FASTBayes .............................................................. 4-12
FASTBoot ................................................................. 4-14
FASTPflClimits .......................................................... 4-17
FASTProfile .............................................................. 4-19
MAXLIK ................................................................. 4-21
MAXBayes ............................................................... 4-36
MAXBoot ................................................................. 4-38
MAXBlimits .............................................................. 4-41
MAXCLPrt ................................................................. 4-42
MAXDensity ............................................................. 4-43
MAXHist ................................................................. 4-45
MAXProfile .............................................................. 4-47
MAXPflClimits ........................................................... 4-50
MAXPrt ................................................................. 4-51
MAXSet ................................................................. 4-53
MAXTlimits .............................................................. 4-53

5 Event Count and Duration Regression .................................

5.0.1 README Files ......................................................... 5-3
5.0.2 Setup ................................................................. 5-3
5.1 About the COUNT Procedures ........................................ 5-4
5.1.1 Inputs ............................................................... 5-5
5.1.2 Outputs .............................................................. 5-6
5.1.3 Global Control Variables ......................................... 5-6
5.1.4 Statistical Inference ................................................. 5-9
5.1.5 Problems with Convergence ..................................... 5-10
5.2 Annotated Bibliography ............................................... 5-13
6 Count Reference

CountCLPrt .............................................. 6-1
CountPrt .............................................. 6-2
CountSet .............................................. 6-3
Expgam .............................................. 6-3
Expon .............................................. 6-9
Hurdlep .............................................. 6-14
Negbin .............................................. 6-19
Pareto .............................................. 6-27
Poisson .............................................. 6-32
Supreme .............................................. 6-37
Supreme2 ............................................ 6-42

Index
1.1 UNIX/Linux/Mac

If you are unfamiliar with UNIX/Linux/Mac, see your system administrator or system documentation for information on the system commands referred to below.

1.1.1 Download

1. Copy the .tar.gz or .zip file to /tmp.

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

3. cd to your GAUSS or GAUSS Engine installation directory. We are assuming /usr/local/gauss in this case.
   
   `cd /usr/local/gauss`
Maxlik 5.0 for GAUSS

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

\[
tar \ xvf \ /tmp/app\_appname\_vernum\_.revnum\_UNIX.tar
\]

– or –

\[
unzip \ /tmp/app\_appname\_vernum\_.revnum\_UNIX.zip
\]

1.1.2 CD

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

2. Open a terminal window.

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

   \[
   cd \ /usr/local/gauss
   \]

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

\[
tar \ xvf \ /cdrom/apps/app\_appname\_vernum\_.revnum\_UNIX.tar
\]

– or –

\[
unzip \ /cdrom/apps/app\_appname\_vernum\_.revnum\_UNIX.zip
\]

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

1.2 Windows

1.2.1 Download

Unzip the .zip file into your GAUSS or GAUSS Engine installation directory.

1.2.2 CD

1. Insert the Apps CD into your machine’s CD-ROM drive.
2. Unzip the .zip files found on the CD to your GAUSS or GAUSS Engine installation directory.

1.2.3 64-Bit Windows

If you have both the 64-bit version of GAUSS and the 32-bit Companion Edition installed on your machine, you need to install any GAUSS applications you own in both GAUSS installation directories.

1.3 Difference Between the UNIX and Windows Versions

- If the functions can be controlled during execution by entering keystrokes from the keyboard, it may be necessary to press ENTER after the keystroke in the UNIX version.
Getting Started
3.1 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 \( \theta \), a vector of parameters, and \( w_i \) is the weight of the i-th observation.
Maxlik 5.0 for GAUSS

The **Maximum Likelihood** procedure **Maxlik** finds values for the parameters in \( \theta \) 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 \( \theta \), 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.

### 3.2 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 \( \theta_t \) be the current parameter values. Then the succeeding values are

\[
\theta_{t+1} = \theta_t + \rho \delta
\]

where \( \delta \) is a \( k \times 1 \) direction vector, and \( \rho \) a scalar step length.

**Direction**

Define

\[
\Sigma(\theta) = \frac{\delta^2 L}{\partial \theta \partial \theta'}
\]
Maximum Likelihood Estimation

\[ \Psi(\theta) = \frac{\partial L}{\partial \theta} \]

The direction, \( \delta \), is the solution to

\[ \Sigma(\theta_t) \delta = \Psi(\theta_t) \]

This solution requires that \( \Sigma \) be positive definite.

**Line Search**

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

### 3.2.1 Derivatives

The minimization requires the calculation of a Hessian, \( \Sigma \), and the gradient, \( \Psi \). Maxlik computes these numerically if procedures to compute them are not supplied.

If you provide a proc for computing \( \Psi \), the first derivative of \( L \), Maxlik uses it in computing \( \Sigma \), the second derivative of \( L \), i.e., \( \Sigma \) 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.

### 3.2.2 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
Maxlik 5.0 for GAUSS

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 $\Sigma$ 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.

### 3.2.3 Convergence

Convergence is declared when the relative gradient is less than \_max_GradTol. The relative gradient is a scaled gradient and is used for determining convergence in order to reduce the effects of scale. It is defined as the absolute value of the gradient times the absolute value of the parameter vector divided by the larger of zero and the absolute value of the function. By default, \_max_GradTol = 1e-5.

### 3.2.4 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.

### 3.2.5 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.

### 3.2.6 Line Search Methods

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

\[
\theta_{t+1} = \theta_t + \rho \delta
\]

where \( \rho \) 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 \( \rho \). The value of the function to be minimized as a function of \( \rho \) is

\[
L(\theta_t + \rho \delta)
\]

Given \( \theta \) and \( d \), this is a function of a single variable \( \rho \). Line search methods attempt to find a value for \( \rho \) 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).
Maximum Likelihood Estimation

**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 \( m(\theta_i + \rho \delta) \) and computes an \( \rho \) 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 \( \rho \). 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.

3.2.7 Random Search

If the line search fails, i.e., no $\rho$ 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.

3.2.8 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.

### 3.2.9 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`.

### 3.2.10 Example

This example estimates coefficients for a tobit model:

```plaintext
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 5.0.0 5/30/2001 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimates</th>
<th>Std. err.</th>
<th>Est./s.e.</th>
<th>Prob.</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>0.0104</td>
<td>0.0845</td>
<td>0.123</td>
<td>0.4510</td>
<td>-0.0000</td>
</tr>
<tr>
<td>P02</td>
<td>-0.2081</td>
<td>0.0946</td>
<td>-2.200</td>
<td>0.0139</td>
<td>-0.0000</td>
</tr>
<tr>
<td>P03</td>
<td>-0.0998</td>
<td>0.0801</td>
<td>-1.245</td>
<td>0.1065</td>
<td>-0.0000</td>
</tr>
<tr>
<td>P04</td>
<td>0.6522</td>
<td>0.0999</td>
<td>6.531</td>
<td>0.0000</td>
<td>-0.0000</td>
</tr>
</tbody>
</table>

Correlation matrix of the parameters

<table>
<thead>
<tr>
<th></th>
<th>1.000</th>
<th>0.035</th>
<th>0.155</th>
<th>-0.090</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.035</td>
<td>1.000</td>
<td>-0.204</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>0.155</td>
<td>-0.204</td>
<td>1.000</td>
<td>-0.030</td>
<td></td>
</tr>
<tr>
<td>-0.090</td>
<td>0.000</td>
<td>-0.030</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>
3.3 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.

3.3.1 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.
3.3.2 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.

Computer Arithmetic

Computer arithmetic is fundamentally flawed by the fact that the computer number is finite (see Higham, 1996, for a general discussion). The standard double precision number in PCs carries about 16 decimal significant places. A simple operation can destroy nearly all of those places. The most destructive operation on a computer is addition and subtraction. Numbers are stored in a computer in the form of an abscissa and an exponent, e.g., 1.234567890123456e+02. There are about 16 decimal places of precision on most computers. The problem occurs when adding numbers that are of very different size. Before adding the number must be transformed so that the exponents are the same. For example consider adding 1.234567890123456e-07 to 1.0000000000000000e+00:
As you can see eight places were lost in the smaller number. If the exponent in the smaller number was 16 all of the places in that number would be lost.

This problem is due to the finiteness of the computer number, not to the implementation of the operators. It is an inherent problem in all computers and the only solution, adding more bits to the computer number, is only temporary because sooner or later a problem will arise where that quantity of bits won’t be enough. The first lesson to be learned from this is to avoid operations combining very small numbers with relatively large numbers. And for very small numbers, 1 can be a large number, as the example shows.

The standard method for evaluating the precision lost in computing a matrix inverse is the ratio of the largest to the smallest eigenvalue of the matrix. This quantity is sometimes called the condition number. The log of the condition number to the base 10 is approximately the number of decimal places lost in computing the inverse. A condition number greater than $1e16$ therefore indicates that all of the 16 decimal places are lost that are available in the standard double precision floating point number.

The BFGS optimization method in Maxlik has been successful primarily because its method of generating an approximation to the Hessian encourages better conditioning. The implementation of the NEWTON method involves a numerical calculation of the Hessian. A numerical Hessian, like all numerical derivatives, are computed by first computing a difference, the most destructive operation as we’ve seen, and then compounding that by dividing the difference by a very small quantity. In general, when using double precision with 16 places of accuracy, about four places are lost in calculating a first derivative and another four with the second derivative. The numerical Hessian therefore begins with a loss of eight places of precision. If there are any problems computing the function itself, or if the model itself contains any problems of condition, there may be nothing left at all.
The BFGS method avoids much of the problems in computing a numerical Hessian. It produces an approximation by building information slowly with each iteration. Initially the Hessian is set to the identity matrix, the matrix with the best condition but the least information. Information is increased at each iteration with a method that guarantees a positive definite result. This provides for stabler, though slower, progress towards convergence.

The implementation of has been designed to minimize the damage to the precision of the optimization problem. The BFGS method avoids a direct calculation of the numerical Hessian, and uses sophisticated techniques for calculating the direction that preserve as much precision as possible. However, all of this can be defeated by a poorly scaled problem or a poorly specified model. When the objective function being optimized is a log-likelihood, the inverse of the Hessian is an estimate of the covariance matrix of the sampling distribution of the parameters. The condition of the Hessian is related to (i) the scaling of the parameters, and (ii) the degree with which there are linear dependencies in the sampling distribution of the parameters.

**Scaling**

Scaling is under the direct control of the investigator and should never be an issue in the optimization. It might not always be obvious how to do it, though. In estimation problems scaling of the parameters is usually implemented by scaling the data. In regression models this is simple to accomplish, but in more complicated models it might be more difficult to do. It might be necessary to experiment with different scaling to get it right. The goal is to optimize the condition of the Hessian. The definition of the condition number implies that we endeavor to minimize the difference of the largest to the smallest eigenvalue of the Hessian. A rule of thumb for this is to scale the Hessian so that the diagonal elements are all about the same magnitude.

If the scaling of the Hessian proves too difficult, an alternative method is to scale the parameters directly in the procedure computing the log-likelihood. Multiply or divide the parameter values being passed to the procedure by setting quantities before their use in the calculation of the log-likelihood. Experiment with different values until the diagonal
elements of the Hessian are all about the same magnitude.

**Linear Dependencies or Nearly Linear Dependencies in the Sampling Distribution**

This is the most common difficulty in estimation and arises because of a discrepancy between the data and the model. If the data do not contain sufficient information to “identify” a parameter or set of parameters, a linear dependency is generated. A simple example occurs in regressors that cannot be distinguished from the constant because its variation is too small. When this happens, the sampling distribution of these two parameters becomes highly collinear. This collinearity will produce an eigenvalue approaching zero in the Hessian, increasing the number of places lost in the calculation of the inverse of the Hessian, degrading the optimization.

In the real world the data we have available will frequently fail to contain the information we need to estimate all of the parameters of our models. This means that it is a constant struggle to a well-conditioned estimation. When the condition sufficiently deteriorates to the point that the optimization fails, or the statistical inference fails through a failure to invert the Hessian, either more data must be found, or the model must be re-specified. Re-specification means either the direct reduction of the parameter space, that is, a parameter is deleted from the model, or some sort of restriction is applied to the parameters.

**Diagnosing the Linear Dependency**

At times it may be very difficult to determine the cause of the ill-conditioning. If the Hessian being computed at convergence for the covariance matrix of the parameters fails to invert, try the following: first generate the pivoted QR factorization of the Hessian,

\[
\{ R, E \} = \text{qre}(H);
\]
The linearly dependent columns of $H$ are pivoted to the end of the $R$ matrix. $E$ contains the new order of the columns of $H$ after pivoting. The number of linearly dependent columns is found by looking at the number of nearly zero elements at the end of the diagonal of $R$.

We can compute a coefficient matrix of the linear relationship of the dependent columns on the remaining columns by computing $R_{11}^{-1}R_{12}$ where $R_{11}$ is that portion of the $R$ matrix associated with the independent columns and $R_{12}$ the independent with dependent. Rather than use the inverse function in GAUSS, we use a special solve function that takes advantage of the triangular shape of $R_{11}$. Suppose that the last two elements of $R$ are nearly zero, then

\[
\begin{align*}
    r0 &= \text{rows}(R); \\
    r1 &= \text{rows}(R) - 1; \\
    r2 &= \text{rows}(R) - 2; \\
    B &= \text{utrisol}(R[1:r2,r1:r0], R[1:r2,1:r2]);
\end{align*}
\]

$B$ describes the linear dependencies among the columns of $H$ and can be used to diagnose the ill-conditioning in the Hessian.

### 3.3.3 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.
3.3.4 Diagnosis

When the optimization is not proceeding well, it is sometimes useful to examine the function, the gradient $\Psi$, the direction $\delta$, the Hessian $\Sigma$, the parameters $\theta$, or the step length $\rho$, 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 them 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,

```plaintext
_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:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta$</td>
<td>&quot;params&quot;</td>
</tr>
<tr>
<td>$\delta$</td>
<td>&quot;direct&quot;</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>&quot;hessian&quot;</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>&quot;gradient&quot;</td>
</tr>
<tr>
<td>$\rho$</td>
<td>&quot;step&quot;</td>
</tr>
</tbody>
</table>

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.
3.4 Gradients

3.4.1 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,

```gauss
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.

### 3.4.2 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;
```

### 3.4.3 Algorithmic Derivatives

Algorithmic Derivatives is a program that can be used to generate a GAUSS procedure
to compute derivatives of the log-likelihood function. If you have Algorithmic
Derivatives, be sure to read its manual for details on doing this.

First, copy the procedure computing the log-likelihood to a separate file. Second, from the
command line enter
Maxlik 5.0 for GAUSS

ad file_name d_file_name

where file_name is the name of the file containing the input function procedure, and d_file_name is the name of the file containing the output derivative procedure.

If the input function procedure is named lpr, the output derivative procedure has the name d_1_lpr where the addition to the “_1_” indicates that the derivative is with respect to the first of the two arguments.

For example, put the following function into a file called lpr.fct

```gauss
proc lpr(x,z);
    local s,m,u;
    s = x[4];
    m = z[.,2:4]*x[1:3,.];
    u = z[.,1] ./= 0;
    retp(u.*lnpdfmvn(z[.,1]-m,s) + (1-u).*(lncdfnc(m/sqrt(s))));
endp;
```

Then enter the following at the GAUSS command line

```gauss
library ad;
ad lpr.fct d_lpr.fct;
```

If successful, the following is printed to the screen

```
java -jar d:\gauss6.0\src\GaussAD.jar lpr.fct d_lpr.fct
```

and the derivative procedure is written to file named d_lpr.fct:
Maximum Likelihood Estimation

/* Version:1.0 - May 15, 2004 */
/* Generated from:lpr.fct */

/* Taking derivative with respect to argument 1 */
Proc(1)=d_1_lpr(x, z);
Clearg _AD_fnValue;
Local s, m, u;
s = x[(4)];
Local _AD_t1;
_AD_t1 = x[(1):(3),.];
m = z[.,(2):(4)] * _AD_t1;
u = z[.,(1)] ./= 0;
_AD_fnValue = (u .* lnpdfmvn( z[.,(1)] - m, s)) + ((1 - u) .* lncdfnc(m / sqrt(s)));
/* retp(_AD_fnValue); */
/* endp; */
struct _ADS_optimum _AD_d__AD_t1 ,_AD_d_x ,_AD_d_s ,_AD_d_m ,_AD_d__AD_fnValue;
/* _AD_d__AD_t1 = 0; _AD_d_s = 0; _AD_d_m = 0; */
_AD_d__AD_fnValue = _ADP_d_x_dx(_AD_fnValue);
_AD_d_s = _ADP_DtimesD(_AD_d__AD_fnValue,
_ADP_DplusD(_ADP_DtimesD(_ADP_d_xplusy_dx(u .* lnpdfmvn( z[.,(1)] - m, s), (1 - u) .* lncdfnc(m / sqrt(s))), _ADP_DtimesD(_ADP_d_ydotx_dx(1 - u, lncdfnc(m / sqrt(s))), _ADP_DtimesD(_ADP_d_sqrt(s), _ADP_d_x_dx(m))))),
_ADP_DtimesD(_ADP_d_ydotx_dx(1 - u, lncdfnc(m / sqrt(s))),
_ADP_DtimesD(_ADP_d_lncdfnc(m / sqrt(s)), _ADP_DtimesD(_ADP_d_xdivy_dx(m, sqrt(s)), _ADP_DtimesD(_ADP_d_sqrt(s), _ADP_d_ydivx_dx(m, sqrt(s)), _ADP_DtimesD(_ADP_d_ydotx_dx(1 - u, lncdfnc(m / sqrt(s))), _ADP_DtimesD(_ADP_d_sqrt(s), _ADP_d_x_divx(m)))))));
_AD_d_m = _ADP_DtimesD(_AD_d__AD_fnValue,
_ADP_DplusD(_ADP_DtimesD(_ADP_d_xplusy_dx(u .* lnpdfmvn( z[.,(1)] - m, s), (1 - u) .* lncdfnc(m / sqrt(s))), _ADP_DtimesD(_ADP_d_ydotx_dx(1 - u, lncdfnc(m / sqrt(s))),
_ADP_DtimesD(_ADP_d_lncdfnc(m / sqrt(s)), _ADP_DtimesD(_ADP_d_xdivy_dx(m, sqrt(s)), _ADP_DtimesD(_ADP_d_sqrt(s), _ADP_d_ydivx_dx(m, sqrt(s)), _ADP_DtimesD(_ADP_d_sqrt(s), _ADP_d_x_divx(m)))))));
/* u = z[.,(1)] ./= 0; */
_AD_d__AD_t1 = _ADP_DtimesD(_AD_d_m, _ADP_DtimesD(_ADP_d_yx_dx(z[.,(2):(4)], _AD_t1), _AD_d_x_dx(_AD_t1)));
Maxlik 5.0 for GAUSS

```gauss
_AD_sr_x = _ADP_seqaMatrixRows(x);
_AD_sc_x = _ADP_seqaMatrixCols(x);
_AD_d_x = _ADP_DtimesD(_AD_d__AD_t1, _ADP_d_x2Idx_dx(x,
   _AD_sr_x[(1):(3)] , _AD_sc_x[0] ));
Local _AD_s_x;
_AD_s_x = _ADP_seqaMatrix(x);
_AD_d_x = _ADP_DplusD(_ADP_DtimesD(_AD_d_s, _ADP_d_xIdx_dx(x,
   _AD_s_x[(4)] )), _AD_d_x);
retp(_ADP_external(_AD_d_x));
endp;
```

If there’s a syntax error in the input function procedure, the following is written to the screen

```bash
java -jar d:\gauss6.0\src\GaussAD.jar lpr.fct d_lpr.fct
```

Command 'java -jar d:\gauss6.0\src\GaussAD.jar lpr.fct d_lpr.fct' exit status 1 indicating that an error has occurred. The output file then contains the reason for the error:

```bash
/* Version:1.0 - May 15, 2004 */
/* Generated from:lpr.fct */
/* Taking derivative with respect to argument 1 */
proc lpr(x,z);
local s,m,u;
s = x[4];
m = z[.,2:4]*x[1:3,.];
u = z[.,1] ./= 0;
retp(u.*lnpdfmvn(z[.,1]-m,s) + (1-u).*(lncdfnc(m/sqrt(s)));
Error: lpr.fct:12:63: expecting ')', found ';
```

Finally, set the global, `max_GradProc` equal to a pointer to this above procedure, for example,
library maxlik,ad;

#include ad.sdf

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

_max_Bounds = { -10 10,
               -10 10,
               -10 10,
               .1 10 };__title = "tobit example";

_max_GradProc = &d_1_lpr;

Maxlik("tobit",0,&lpr,x0);

### Speeding Up the Algorithmic Derivative

A slightly faster derivative procedure can be generated by modifying the log-likelihood proc to return a scalar sum of the log-likelihoods in the input file in the call to AD. It is important to note that this derivative function based on a scalar return cannot be used for computing the QML covariance matrix of the parameters. Thus if you want both a derivative procedure based on a scalar return and QML standard errors you will need to provide both types of gradient procedures. To accomplish this first copy both versions of the log-likelihood procedure into separate files and run AD on both of them with different output files. Then copy both of these derivatives procedures to the command file. Note: the log-likelihood procedure that returns a vector of log-likelihoods should remain in the command file, i.e., don’t use the version of the log-likelihood that returns a scalar in the command file.

For example, enlarging on the example in the previous section, put the following into a separate file,
Maxlik 5.0 for **GAUSS**

```gauss
proc lpr2(x,z);
    local s,m,u,logl;
    s = x[4];
    m = z[.,2:4]*x[1:3,.];
    u = z[.,1] ./= 0;
    logl = u.*lnpdfmvn(z[.,1]-m,s) + (1-u).*lncdfnc(m/sqrt(s));
    retp(sumc(logl));
endp;
```

Then enter on the command line

```
ad lpr2.src d_lpr2.src
```

and copy the contents of d_lpr2.src into the command file.

Our command file now contains two derivative procedures, one based on a scalar result and another on a vector result. The one in the previous section `d_1_lpr` is our vector result derivative, and the from run above, `d_1_lpr2` is our scalar result derivative. We want to use `d_1_lpr2` for the iterations because it will be faster (it is computing a $1 \times K$ vector gradient), and for the QML covariance matrix of the parameters we will use `d_1_lpr` which returns a $N \times K$ matrix of derivatives as required for the QML covariance matrix.

Our command file will be

```gauss
library maxlik,ad;

#include ad.sdf

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

_max_Bounds = { -10 10,
               -10 10,
               ...
```
-10 10,
.1 10 \};

_max_QMLProc = &d_1_lpr;
_max_GradProc = &d_1_lpr2;

Maxlik("tobit",0,&lpr,x0);

in addition to the two derivative procedures.

### 3.4.4 Analytical Hessian

You may provide a procedure for computing the Hessian, \( \Sigma(\theta) = \frac{\partial^2 L}{\partial \theta \partial \theta'} \). 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.

```plaintext
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);
```
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
3.4.5 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 $K \times K$ matrix which is the estimated Hessian evaluated at the parameter vector.
3.4.6 Switching Algorithms Automatically

The global variable _max_Switch can be used to switch algorithms automatically during the iterations. If _max_Switch has one column, the algorithm is switched once during the iterations, and if it has two columns it is switched back and forth. The conditions for the switching is determined by the elements of _max_Switch in the second through fourth rows. If these are rows are not supplied default values are entered. The first row contains the algorithm numbers to switch to, or if two columns to switch to and from. The algorithm switches if the log-likelihood function improves by less than the quantity in the second row, or if the number of iterations exceeds the quantity in the third row, or if the line search changes by less than the quantity in the fourth row.

If only the first row is specified in the command file, that is, if only the algorithm numbers are entered, the second, third and fourth rows are set by default to .001, 10, .001 respectively.

3.5 FASTMAX – Fast Execution MAXLIK

Depending on the type of problem FASTMAX, the fast version of Maxlik, can be called with speed-ups from 10 percent to 500 percent over the regular version of Maxlik. This is achieved at the expense of losing some features, in particular, it won’t print any iteration information to the screen, the globals cannot be modified on the fly, it can’t print or store diagnostic information. Moreover, the dataset must be entirely storable in RAM.

The gain in time depends on the type of problem. The greatest speedup occurs with problems that are function call intensive. The speedup will be less if gradients and/or Hessians are provided. The least speedup occurs for problems where convergence is quick, and the most where convergence is slow. Thus FASTMAX will least affect a bootstrap or profile likelihood estimation for models that converge quickly, and most affect those that don’t.

FASTMAX is most useful for problems that will be repeated in some way such as in a Monte
Maximum Likelihood Estimation

Carlo study or a bootstrap. The initial runs would use **Maxlik** where monitoring the progress is most important, and subsequent runs would use **FASTMAX**.

**FASTMAX** has the same arguments and returns as **Maxlik** and thus to call it you may change the name **Maxlik** in your command file to **FASTMAX**. **FASTMAX** does require that the dataset be storable in memory in its entirety, however, and if that isn’t possible **FASTMAX** will fail.

In a similar way, for the fast versions of **MAXBOOT**, **MAXPROFILE**, and **MAXBAYES**, change the calls to **FASTBOOT**, **FASTPROFILE**, and **FASTBAYES**, respectively. No changes in input or output arguments are necessary.

### 3.5.1 Undefined Function Evaluation

On occasion the log-likelihood function will evaluate to an undefined value, for example, the log-likelihood procedure may attempt to take the log of a negative quantity for one or more observations. If you have written your procedure to return a scalar missing value when this happens, **Maxlik** will succeed in recovering in most cases. That is, depending on circumstances it will find another set of parameter values or use a different line search method.

If you are using **FASTMAX**, you can try a different strategy. Write your procedure to enter a missing value in the log-likelihood vector for that observation for which the calculation is undefined. **FASTMAX** will compute gradients and function values by list-wise deletion. In other words it will compute the function and gradient from the available observations.

### 3.6 Inference

**Maxlik** includes four classes of methods for analyzing the distributions of the estimated parameters:
Maxlik 5.0 for GAUSS

- Wald
- Profile likelihood
- Bootstrap
- 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 preferred 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-likelihood surface.

3.6.1 Wald Inference

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

\[
\hat{\theta} \rightarrow N(\theta, A^{-1}BA^{-1})
\]
Maximum Likelihood Estimation

where

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

Estimates of A and B are

\[ \hat{A} = \frac{1}{N} \sum_i^N \frac{\partial^2 L_i}{\partial \theta \partial \theta'} \]
\[ \hat{B} = \frac{1}{N} \sum_i^N \left( \frac{\partial L_i}{\partial \theta} \right)' \left( \frac{\partial L_i}{\partial \theta} \right) \]

Assuming the correct specification of the model \( \text{plim}(A) = \text{plim}(B) \) and thus

\[ \hat{\theta} \rightarrow N(\theta, \hat{A}^{-1}) \]

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.
Maxlik 5.0 for GAUSS

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.
3.6.2 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 preferred.

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 $\theta$.

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

\[
G(\phi) = \min(\text{Logl}(\theta) \mid \eta'_i\theta = \phi)
\]

where $\eta_i$ 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 $\phi$ such that

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

and the upper limit is found by redefining Equation 1 as a maximum.
Example

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

```gauss
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)))) +
         (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";
c11 = maxtlimits(x,cov);
call maxclprt(x,f,g,c11,ret);
__title = "Profile Likelihood Confidence Limits";
c12 = maxpflclimits(x,f,"tobit",0,&lpr);
call maxclprt(x,f,g,c12,ret);
```

The output is:

==========================================================================

3-34
### Wald Confidence Limits

MAXLIK Version 5.0.0 5/30/2001 1:16 pm

Data Set: tobit

---

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimates</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>0.0104</td>
<td>-0.1573</td>
<td>0.1781</td>
<td>-0.0000</td>
</tr>
<tr>
<td>P02</td>
<td>-0.2081</td>
<td>-0.3958</td>
<td>-0.0203</td>
<td>-0.0000</td>
</tr>
<tr>
<td>P03</td>
<td>-0.0998</td>
<td>-0.2588</td>
<td>0.0593</td>
<td>-0.0000</td>
</tr>
<tr>
<td>P04</td>
<td>0.6522</td>
<td>0.4540</td>
<td>0.8505</td>
<td>-0.0000</td>
</tr>
</tbody>
</table>

Number of iterations: 17
Minutes to convergence: 0.03200

### Profile Likelihood Confidence Limits

MAXLIK Version 5.0.0 5/30/2001 1:16 pm

Data Set: tobit

---

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimates</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>0.0104</td>
<td>-0.1560</td>
<td>0.1720</td>
<td>-0.0000</td>
</tr>
<tr>
<td>P02</td>
<td>-0.2081</td>
<td>-0.3918</td>
<td>-0.0245</td>
<td>-0.0000</td>
</tr>
<tr>
<td>P03</td>
<td>-0.0998</td>
<td>-0.2562</td>
<td>0.0549</td>
<td>-0.0000</td>
</tr>
</tbody>
</table>
In this example, the model is conditionally linear and we see that the Wald and profile likelihood limits are quite similar.

### 3.6.3 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

\[ \tilde{\theta}_k = (\tilde{\theta}_1, \tilde{\theta}_2, ..., \tilde{\theta}_{k-1}, \theta_k, \tilde{\theta}_{k+1}, ..., \tilde{\theta}_K) \]
Maximum Likelihood Estimation

This is the vector of maximum likelihood estimates \textit{conditional} on \( \theta_k \), i.e., where \( \theta_k \) is fixed to some value. Further define the profile t function

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

For each parameter in the model, \( \tau \) is computed over a range of values for \( \theta_k \). These plots provide exact likelihood intervals for the parameters, and reveal how nonlinear the estimation is. For a linear model, \( \tau \) 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 \((\theta_k, \theta_\ell)\) plane might look. For the \((\theta_k, \theta_\ell)\) plot, two lines are plotted, \( L(\hat{\theta}_k) \) against \( \theta_k \) and \( L(\hat{\theta}_\ell) \) against \( \theta_\ell \).

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 3.2.10, it is necessary only to change the call to \texttt{Maxlik} to a call to \texttt{MAXProfile}:

\[
\text{call MAXPrt(MAXProfile("tobit",0,&lpr,x0));}
\]
**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.

### 3.6.4 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 100 re-samplings. Efron and Tibshirani (1993:52) suggest that 100 is satisfactory, 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,
**Maximum Likelihood Estimation**

_\texttt{\_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, _\texttt{\_max_Smoothing}, or \texttt{MAXDensity} will compute them.

**MAXHist**

\texttt{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 \texttt{MAXHist} in a matrix.

**Example**

To bootstrap the example in Section 3.2.10, the only necessary alteration is the change the call to \texttt{Maxlik} to a call to \texttt{MAXBoot}:

```plaintext
\_\texttt{\_max_BootFname} = "bootdata";

\texttt{call MAXPrt(maxlikboot("tobit",0,&lpr,x0))};

\texttt{call MAXDensity("bootdata",0)};
\texttt{call MAXHist("bootdata",0)};
```

**3.6.5 Pseudo-Random Number Generators**

Pseudo-Random numbers are generated by \texttt{Maxlik} and \texttt{FASTMAX} in the random line search, by \texttt{MAXBoot} and \texttt{FASTBoot} for re-sampling, and by \texttt{MAXBayes} and \texttt{FASTBayes} also for re-sampling. There two types of pseudo-random generators, the linear congruential (LC) and another based on Marsaglia’s \texttt{Kiss-Monster} algorithm (KM). The
Maxlik 5.0 for GAUSS

LC generators are faster but have shorter period \(2^{32}\), whereas the KM generators are slower but have much longer periods \(2^{3859}\).

The global variable \_max_RandType chooses between these. By default the LC generators are used.

The seed for these generators is kept in the global \_max_State. The default value is 345678. You may set this to any integer value in your command file.

### 3.6.6 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:

\[
\begin{align*}
    r(\hat{\theta}) &= \pi(\hat{\theta})e^{L(\hat{\theta})} / \hat{g}(\hat{\theta})
\end{align*}
\]

where \(\pi(\hat{\theta})\) is the prior distribution of the parameters, and \(\hat{g}(\hat{\theta})\) is a normal kernel density estimate of the parameters using Terrell’s (1990) method of maximum smoothing. The SIR algorithm, described in Rubin (1988), is applied 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.
Maximum Likelihood Estimation

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.

```plaintext
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),.]);
```

3-41
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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimates</th>
<th>Std. err.</th>
<th>Est./s.e.</th>
<th>Prob.</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>0.0104</td>
<td>0.0873</td>
<td>0.119</td>
<td>0.4525</td>
<td>0.0000</td>
</tr>
<tr>
<td>P02</td>
<td>-0.2081</td>
<td>0.0946</td>
<td>-2.200</td>
<td>0.0139</td>
<td>0.0000</td>
</tr>
<tr>
<td>P03</td>
<td>-0.0998</td>
<td>0.0800</td>
<td>-1.247</td>
<td>0.1062</td>
<td>0.0000</td>
</tr>
<tr>
<td>P04</td>
<td>0.6522</td>
<td>0.0999</td>
<td>6.531</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Correlation matrix of the parameters

<table>
<thead>
<tr>
<th></th>
<th>1.000</th>
<th>0.030</th>
<th>0.151</th>
<th>-0.092</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.030</td>
<td>1.000</td>
<td>-0.205</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>0.151</td>
<td>-0.205</td>
<td>1.000</td>
<td>-0.029</td>
<td></td>
</tr>
<tr>
<td>-0.092</td>
<td>0.000</td>
<td>-0.029</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>

Number of iterations 17
Minutes to convergence 0.01462
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

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

Correlation matrix of the parameters

\[
\begin{matrix}
1.000 & -0.165 & 0.362 & 0.465 \\
-0.165 & 1.000 & -0.473 & 0.026 \\
0.362 & -0.473 & 1.000 & 0.322 \\
0.465 & 0.026 & 0.322 & 1.000 \\
\end{matrix}
\]

Number of iterations  7
Minutes to convergence  0.00354

3.7 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
Maxlik 5.0 for **GAUSS**

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)
3.8 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.

3.9 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.

3.10 Error Handling

3.10.1 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 following table describes their meanings:
3.10.2 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.
3.11 References


FASTMAX

PURPOSE Computes estimates of parameters of a maximum likelihood function.

LIBRARY maxlik

FORMAT \[ x, f, g, cov, retcode = \text{FASTMAX}(data, vars, &fct, start) \]

INPUT

- \( data \) \( N \times NV \) 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.
vars may be a character vector containing either the standard labels created by **FASTMAX** (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 **FASTMAX** 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_Switch, _**_max_RandType, _**_max_State

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

**Gradient**  
_**_max_GradMethod, _**_max_GradProc, _**_max_UserNumGrad, _**_max_HessProc, _**_max_GradStep

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

**Data**  
_**_max_NumObs, __weight

**Parameters**  
_**_max_Active, _**_max ParNames

**Miscellaneous**  
__title, _**_max_IterData,

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. computed from cross-product of Jacobian.
3. 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_GradTol_** scalar, convergence tolerance for gradient of estimated coefficients. When this criterion has been satisfied FASTMAX 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 FASTMAX 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_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 FASTMAX 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 $N \times K$ 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:

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

_max_GradProc = &ldg;
```

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:

_\text{max}_\text{HessProc} = \&\text{hessproc};

tells FASTMAX 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 $N \times P$ 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.

_\text{max}_\text{Interp} scalar, interpolation constant in BRENT. Default $= .25$.

_\text{max}_\text{IterData} 3\times1$ 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.

_\text{max}_\text{LineSearch} scalar, selects method for conducting line search. The result of the line search is a \textit{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 _\text{max}_\text{LineSearch} $= 2$ is best. If the optimization bogs down, try setting _\text{max}_\text{LineSearch} $= 1$, $4$ or $5$. _\text{max}_\text{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, FASTMAX 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.

\_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 FASTMAX options in a single global using identifiers. The following

\_max\_Options = { bfgs stepbt forward };

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.

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
_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 FASTMAX attempts _max_MaxTry tries to find a random direction within radius determined by _max_RandRadius that is a descent. Default = 1e-2.

_max_RandType  scalar, if nonzero, pseudo-random numbers of the linear congruential type are generated, otherwise, they are generated by Marsaglia’s Kiss-Monster method. The latter method is slower but has a much larger period. Random numbers are generated for the random line search.

_max_State  scalar or vector, state vector for pseudorandom number generators containing seed. By default it is set to 345678. If you wish to select a seed, set to a different value.

_max_Switch  \( 4 \times 1 \) or \( 4 \times 2 \) vector, controls algorithm switching. If _max_Switch is \( 4 \times 1 \), set its elements in the following way:

1  algorithm number to switch to.
2  FASTMAX will switch to algorithm in the first element when the function value is less than the value entered here.
3  FASTMAX switches if the number of iterations exceeds the number entered here.
4  FASTMAX switches if line search step changes less than the amount entered here.

If _max_Switch is \( 4 \times 2 \), FASTMAX switches between the algorithms in column 1 and column 2 subject to the conditions specified for the \( 4 \times 1 \) vector.

Thus if _max_Switch is a \( 4 \times 1 \) vector, FASTMAX will switch algorithms no more than once during the iterations, whereas if it is \( 4 \times 2 \) it may switch back and forth between the two algorithms throughout the iterations.

__title  string title of run
**__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.

** maxi 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 a matrix of observations. The procedure must have two input arguments: first, a vector of parameter values, and second, the data matrix. The output argument is the log-likelihood for the 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:

\[
\text{logprob} = \text{pfct}(x,y);
\]

where

$x$ column vector of parameters of model.

$y$ data.

The output from the procedure *pfct* is the vector of log-likelihoods for a set of observations.

**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,

$$ g = \text{gradproc}(x,y); $$

where the input arguments are

- $x$: vector of coefficients.
- $y$: matrix, dataset.

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).

It is important to note that the gradient is row oriented. `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, `FASTMAX` 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. For best results use `Maxlik` with
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.
- \(y\) matrix containing data set.

and the output argument is

\[
h \quad K \times K \text{ matrix of second order partial derivatives evaluated at the coefficients in } x.
\]

In practice much of the time spent on writing the Hessian procedure is devoted to debugging. To help in this debugging process, use the Maxlik procedure with _max_GradCheckTol is set to a small nonzero value.

SOURCE fastmax.src
FASTBayes

PURPOSE
Computes a simulated posterior of the parameters of a maximum likelihood function using FASTMAX.

LIBRARY
maxlik

FORMAT
{ \textit{x}, f, g, cov, retcode } = \textbf{FASTBayes}(\textit{data}, \textit{vars}, \&\textit{fct}, \textit{start})

INPUT
\textit{data} \quad N \times NV \text{ matrix, dataset.}
\textit{vars} \quad NV \times 1 \text{ character vector, labels of variables selected for analysis.}
\quad \text{– or –}
\quad NV \times 1 \text{ numeric vector, indices of variables selected for analysis.}
\textit{vars} \text{ may be a character vector containing either the standard labels created by FASTBayes (i.e., either V1, V2,..., or V01, V02,...) See discussion of the global variable \_\_vpad below, or the user-provided labels in \_\_altnam).}
\&\textit{fct} \quad \text{a pointer to a procedure that returns the log-likelihood for a vector of log-likelihoods for a matrix of observations.}
\textit{start} \quad K \times 1 \text{ vector, start values.}

OUTPUT
\textit{x} \quad K \times 1 \text{ vector, means of simulated posterior.}
\textit{f} \quad \text{scalar, mean weighted bootstrap log-likelihood.}
\textit{g} \quad K \times 1 \text{ vector, means gradient of weighted bootstrap.}
\textit{h} \quad K \times K \text{ matrix, covariance matrix of simulated posterior.}
\textit{retcode} \quad \text{scalar, return code. If normal convergence is achieved, then \texttt{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 **FASTMAX** 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, **FASTBayes** 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.

_max_RandType scalar, if nonzero, pseudo-random numbers of the linear congruential type are generated, otherwise, they are generated by Marsaglia’s Kiss-Monster method. The latter method is slower but has a much larger period. Random numbers are generated for the random line search.

_max_State scalar or vector, state vector for pseudorandom number generators containing seed. By default it is set to 345678. If you wish to select a seed, set to a different value.

REMARKS FASTBayes 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 FASTBayes 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 procedures MAXHist and MAXDensity for further analysis.

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

SOURCE fastbayes.src
likelihood function using **FASTMAX**.

**LIBRARY** maxlik

**FORMAT** { x,f,g,cov,retcode } = **FASTBoot**(*data*,*vars*,&*fct*,*start*)

**INPUT**  
*data*  $N \times NV$ matrix, dataset.  
*vars*  $NV \times 1$ character vector, labels of variables selected for analysis.  
– or –  
$NV \times 1$ numeric vector, indices of variables selected for analysis.  
*vars* may be a character vector containing either the standard labels created by **FASTBoot** (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 the log-likelihood for a matrix of observations.  
*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.
FASTBoot

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 FASTMAX 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, FASTBoot 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_RandType scalar, if nonzero, pseudo-random numbers of the linear congruential type are generated, otherwise, they are generated by Marsaglia’s Kiss-Monster method. The latter method is slower but has a much larger period. Random numbers are generated for the random line search.
_max_State scalar or vector, state vector for pseudorandom number generators containing seed. By default it is set to 345678. If you wish to select a seed, set to a different value.

REMARKS FASTBoot generates _max_NumSample random samples of size _max_NumObs from the data set with replacement and calls FASTMAX. FASTBoot 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 FASTBoot 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 procedures MAXHist and MAXDensity for further analysis.

SOURCE fastboot.src

PURPOSE Computes profile likelihood confidence limits using FASTMAX.

LIBRARY maxlik

FORMAT \( cl = \text{FASTPflLimits}(b,f,\text{data},\text{vars},\&fct) \)

INPUT \( b \) \( K \times 1 \) vector, maximum likelihood estimates.

\( f \) scalar, function at minimum (mean log-likelihood).

\( \text{data} \) \( N \times NV \) matrix, data.

\( \text{vars} \) \( NV \times 1 \) character vector, labels of variables selected for analysis.
FASTPflClimits

– or –

$NV \times 1$ numeric vector, indices of variables selected for analysis.

`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 a vector of log-likelihoods for a matrix of observations.

**OUTPUT**

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

**GLOBALS**

The `FASTMAX` procedure global variables are also applicable.

`_max_Asym` (1-/commandname_max_Asym)% 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,

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

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

**REMARKS**

`FASTPflClimits` computes profile likelihood confidence limits given a maximum likelihood estimation. $b$ and $f$ should be returns from a call to `FASTMAX`. This will also properly set up `max_NumObs` for `FASTPflClimits`.

`FASTPflClimits` solves for the confidence limits as a parametric likelihood problem. Thus it itself calls `FASTMAX` several times for each confidence limit.
FASTProfile

SOURCE fastpflcl.src

PURPOSE Computes profile t plots and likelihood profile traces for maximum likelihood models using FASTMAX.

LIBRARY maxlik

FORMAT \{ x,f,g,cov,retcode \} = FASTProfile(data,vars,&fct,start)

INPUT data $N \times NV$ matrix, dataset.
vars $NV \times 1$ character vector, labels of variables selected for analysis.
– or – $NV \times 1$ numeric vector, indices of variables selected for analysis.
vars may be a character vector containing either the standard labels created by FASTProfile (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 a vector of log-likelihoods for a matrix of observations
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
FASTProfile

$h$  $K \times K$ 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 **FASTMAX** procedure global variables are also relevant.

_\_max_NumCat_ scalar, number of categories in profile table. Default = 16.

_\_max_Increment_ $K \times 1$ vector, increments for cutting points, default is $2 * _\_max_Width * \text{std dev} / _\_max_NumCat$. If scalar zero, increments are computed by **FastProfile**.

_\_max_Center_ $K \times 1$ 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 fastprof.src

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 x 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>normal convergence</td>
</tr>
<tr>
<td>1</td>
<td>forced exit.</td>
</tr>
<tr>
<td>2</td>
<td>maximum iterations exceeded.</td>
</tr>
<tr>
<td>3</td>
<td>function calculation failed.</td>
</tr>
<tr>
<td>4</td>
<td>gradient calculation failed.</td>
</tr>
<tr>
<td>5</td>
<td>Hessian calculation failed.</td>
</tr>
<tr>
<td>6</td>
<td>line search failed.</td>
</tr>
</tbody>
</table>
function cannot be evaluated at initial parameter values.
error with gradient
gradient vector transposed
secant update failed
maximum time exceeded
error with weights
data set could not be opened.
termination condition unknown.

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**
**_max_Switch, _max_RandType, _max_State,**

**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**
**_max_Key,**

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  computed from cross-product of Jacobian calculated after iterations
3  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
the current quantities are stored in \_max\_Diagnostic using the vput command. Use the following strings to extract from \_max\_Diagnostic using vread:

<table>
<thead>
<tr>
<th>function</th>
<th>&quot;function&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimates</td>
<td>&quot;params&quot;</td>
</tr>
<tr>
<td>direction</td>
<td>&quot;direct&quot;</td>
</tr>
<tr>
<td>Hessian</td>
<td>&quot;hessian&quot;</td>
</tr>
<tr>
<td>gradient</td>
<td>&quot;gradient&quot;</td>
</tr>
<tr>
<td>step</td>
<td>&quot;step&quot;</td>
</tr>
</tbody>
</table>

When \_max\_Diagnostic is nonzero, \_output is forced to 1.

\_max\_GradTol scalar, convergence tolerance for gradient of estimated coefficients. When this criterion has been satisfied 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.

\_\texttt{max\_GradMethod} scalar, method for computing numerical gradient.

0 central difference

1 forward difference (default)

\_\texttt{max\_GradProc} scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. For example, the statement:

\_\texttt{max\_GradProc} = &\texttt{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 \( N \times K \) 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:

\texttt{proc \_lgd(b,z);}\
\quad \texttt{retp((z[.,1]-exp(z[.,2:4]*b)).*z[.,2:4]);}\
\texttt{endp;}

\_\texttt{max\_GradProc} = &\texttt{lgd};

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

\_\texttt{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.

\_\texttt{max\_HessCov} \( K \times K \) matrix. When \_\texttt{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 $N \times P$ 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 BREP. 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_Key scalar, if nonzero, the keyboard is polled for keystrokes for modifying globals, and if zero, polling is turned off. Default = 1.

_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_MaxIter  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.
**Maxlik**

_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 x 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
**MAXLIK**

_max_MaxTry_ tries to find a random direction within radius determined by _max_RandRadius_ that is a descent. Default = 1e-2.

_max_RandType_ scalar, if nonzero, pseudo-random numbers of the linear congruential type are generated, otherwise, they are generated by Marsaglia’s **Kiss-Monster** method. The latter method is slower but has a much larger period. Random numbers are generated for the random line search.

_max_State_ scalar or vector, state vector for pseudorandom number generators containing seed. By default it is set to 345678. If you wish to select a seed, set to a different value.

_max_Switch_ 4 × 1 or 4 × 2 vector, controls algorithm switching. If _max_Switch_ is 4 × 1, set its elements in the following way,

1. algorithm number to switch to
2. _Maxlik_ will switch to algorithm in the first element when the function value is less than the value entered here
3. _Maxlik_ switches if the number of iterations exceeds the number entered here
4. _Maxlik_ switches if line search step changes less than the amount entered here

If _max_Switch_ is 4 × 2, _Maxlik_ switches between the algorithms in column 1 and column 2 subject to the conditions specified for the 4 × 1 vector.

Thus if _max_Switch_ is a 4 × 1 vector, _Maxlik_ will switch algorithms no more than once during the iterations, whereas if it is 4 × 2 it may switch back and forth between the two algorithms throughout the iterations.

_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:

```gauss
_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

```gauss
__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 $N\times P$ matrix of data. The procedure returns a single output argument, a $K \times K$ Hessian matrix of the function with respect to the parameters.

_\_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.

_\_\_XprodCovr index_max_XprodCov@_\_\_XprodCov $K \times K$ matrix. When _\_\_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 _\_\_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 ≠ 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,

\[ g = \text{gradproc}(x, y); \]

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 \texttt{max\_Lag} \( \geq 1 \) 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 (\texttt{__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 \texttt{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
MAXBayes

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

LIBRARY
maxlik

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

INPUT
dataset
string containing name of \texttt{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 \texttt{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

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

0 \quad \text{normal convergence}

1 \quad \text{forced exit.}

2 \quad \text{maximum iterations exceeded.}

3 \quad \text{function calculation failed.}

4 \quad \text{gradient calculation failed.}

5 \quad \text{Hessian calculation failed.}

6 \quad \text{line search failed.}

7 \quad \text{function cannot be evaluated at initial parameter values.}

8 \quad \text{error with gradient}

9 \quad \text{gradient vector transposed}

10 \quad \text{secant update failed}

11 \quad \text{maximum time exceeded}

12 \quad \text{error with weights}

34 \quad \text{data set could not be opened.}

99 \quad \text{termination condition unknown.}

\text{GLOBALS} \quad \text{The Maxlik procedure global variables are also applicable.}


\_max\_BootFname \quad \text{string, file name of GAUSS data set (do not include .DAT extension) containing bootstrapped parameter estimates. If not specified, MAXBayes selects a temporary name.}
**MAXBayes**

__MAXBoot__

__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

**MAXBoot**

**PURPOSE** Computes bootstrapped estimates of parameters of a maximum likelihood function.

**LIBRARY** maxlik
FORMAT  \{ x,f,g,cov,retcode \} = \texttt{MAXBoot}(\texttt{dataset},\texttt{vars},&\texttt{fct},\texttt{start})

INPUT  \texttt{dataset}    string containing name of \texttt{GAUSS} data set.
– or –  \( N \times NV \) matrix, data.
\texttt{vars} \( NV \times 1 \) character vector, labels of variables selected for analysis.
\( NV \times 1 \) numeric vector, indices of variables selected for analysis.
If \texttt{dataset} is a matrix, \texttt{vars} may be a character vector containing either the standard labels created by \texttt{MAXBoot} (i.e., either V1, V2, ..., or V01, V02, ...). See discussion of the global variable \texttt{__vpad} below, or the user-provided labels in \texttt{__altnam}).
\&\texttt{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 \texttt{__row} in global variable section below).
\texttt{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
\texttt{retcode} scalar, return code. If normal convergence is achieved, then \texttt{retcode} = 0, otherwise a positive integer is returned indicating the reason for the abnormal termination:
0   normal convergence
1   forced exit.
MAXBoot

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**  
\[ cl = \text{MAXBlimits}(\text{dataset}) \]

**INPUT**  
\text{dataset}  
string containing name of **GAUSS** data set.  
– or –  
\( N \times K \) matrix, data.

**OUTPUT**  
\text{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.
MAXCLPrt

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

MAXCLPrt

PURPOSE  Formats and prints the output from a call to **Maxlik** along with 
confidence limits.

LIBRARY  maxlik

FORMAT  
{ x,f,g,cl,retcode } = MAXCLPrt(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

__header = \commandname{tld};

Default = “tldvf”.

__title string, message printed at the top of the screen and printed out by MAXCLPrt. Default = “”.

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

{ 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
**MAXDensity**

**INPUT**

*d\textit{dataset}* string containing name of \textsc{gauss} data set.

– or –

N×K matrix, data.

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

**OUTPUT**

\texttt{_\textit{px}} $\_\textit{max\_NumPoints} \times K$ matrix, abscissae of plotted points

\texttt{_\textit{py}} $\_\textit{max\_NumPoints} \times K$ matrix, ordinates of plotted points

\texttt{smth} $K \times 1$ vector, smoothing coefficients

**GLOBALS** The \textsc{Maxlik} procedure global variables are also applicable.

\texttt{\_\textit{max\_Kernel}} $K \times 1$ character vector, type of kernel:

\begin{itemize}
  \item \texttt{NORMAL} normal kernel
  \item \texttt{EPAN} Epanechnikov kernel
  \item \texttt{BIWGT} biweight kernel
  \item \texttt{TRIANG} triangular kernel
  \item \texttt{RECTANG} rectangular kernel
  \item \texttt{TNORMAL} truncated normal kernel
\end{itemize}

If \texttt{\_\textit{max\_Kernel}} is scalar, the kernel is the same for all parameter densities. Default = \texttt{NORMAL}.

\texttt{\_\textit{max\_NumPoints}} scalar, number of points to be computed for plots

\texttt{\_\textit{max\_EndPoints}} $K \times 2$ matrix, lower (in first column) and upper (in second column) endpoints of density. Default is minimum
and maximum, respectively, of the parameter values. If $1 \times 2$ matrix, endpoints are the same for all parameters.

`max_Smoothing` $K \times 1$ 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` $K \times 2$ matrix, lower (in first column) and upper (in second column) truncation limits for truncated normal kernel. If $1 \times 2$ 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
MAXHist

\( K \times 1 \) 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**: \(_\text{max\_NumCat} \times K\) matrix, univariate distributions of bootstrapped parameters
- **cut**: \(_\text{max\_NumCat} \times K\) matrix, cutting points

**GLOBALS**
The Maxlik procedure global variables are also applicable.

- \(_\text{max\_Center}\) \( K \times 1 \) value of center category in histograms. Default is initial coefficient estimates.
- \(_\text{max\_CutPoint}\) \(_\text{max\_NumCat} \times 1\) vector, output, cutting points for histograms
- \(_\text{max\_Increment}\) \( K \times 1\) vector, increments for cutting points of the histograms. Default is \(2 \times _\text{max\_Width} \times \text{std dev} / _\text{max\_NumCat}\).
- \(_\text{max\_NumCat}\) scalar, number of categories in the histograms
- \(_\text{max\_Width}\) 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, \(_\text{max\_Center}, _\text{max\_Width}, _\text{max\_Increment}_\) 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

MAXProfile

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

LIBRARY maxlik

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

INPUT \( \text{dataset} \) string containing name of GAUSS data set.
– or –
\( N \times NV \) matrix, data.

\( \text{vars} \) \( NV \times 1 \) character vector, labels of variables selected for analysis.
– or –
\( NV \times 1 \) numeric vector, indices of variables selected for analysis.

If \( \text{dataset} \) is a matrix, \( \text{vars} \) 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).
MAXProfile

&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.
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`
MAXPfIClimits

PURPOSE Computes profile likelihood confidence limits.

LIBRARY maxlik

FORMAT \( cl = \text{MAXPfIClimits}(b,f,\text{dataset},\text{vars},\&fct) \)

INPUT
- \( b \) \( K \times 1 \) vector, maximum likelihood estimates
- \( f \) scalar, function at minimum (mean log-likelihood)
- \( \text{dataset} \) string containing name of GAUSS data set.
  - or – \( N \times NV \) matrix, data.
- \( \text{vars} \) \( NV \times 1 \) character vector, labels of variables selected for analysis.
  - or – \( NV \times 1 \) numeric vector, indices of variables selected for analysis.
  If \( \text{dataset} \) is a matrix, \( \text{vars} \) may be a character vector containing either the standard labels created by \text{MAXPfIClimits} (i.e., either V1, V2,..., or V01, V02,......
  See discussion of the global variable \text{__vpad} below, or the user-provided labels in \text{__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 \text{__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.
$\text{MAXSet}$

LIBRARY maxlik

FORMAT \{ $x, f, g, h, \text{retcode}$ \} = $\text{MAXPr}t(x, f, g, h, \text{retcode})$;

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  
    $\text{retcode}$ 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:

$$\text{__header} = \text{"tld"};$$

Default = “tldvf”.

__title string, message printed at the top of the screen and printed out by $\text{MAXPr}t$. Default = “”.

REMARKS The call to $\text{Maxlik}$ can be nested in the call to $\text{MAXPr}t$:

\{ $x, f, g, h, \text{retcode}$ \} = $\text{MAXPr}t(\text{MAXLIK(dataset, vars, &fct, start)})$;

SOURCE maxlik.src
### MAXSet

**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

### MAXTlimits

**PURPOSE** Computes Wald confidence limits.

**LIBRARY** maxlik

**FORMAT**

\[
cl = \text{MAXTlimits}(b, cov)
\]
### MAXTlimits

**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**  
- $\_\text{max}_\text{Alpha}$ \( (1-\_\text{max}_\text{Alpha})\% \) confidence limits are computed. The default is .05  
- $\_\text{max}_\text{NumObs}$ scalar, number of observations. Must be set.  
- $\_\text{max}_\text{Select}$ selection vector for selecting coefficients to be included in profiling, for example  
  
  $\_\text{max}_\text{Select} = \{ 1, 3, 4 \};$

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

**REMARKS**  
- **MAXTlimits** returns  
  
  \[
  b[i] \pm t(\_\text{max}_\text{NumObs} - K; \_\text{max}_\text{Alpha}/2) \times \sqrt{\text{cov}[i,i]}
  \]

The global $\_\text{max}_\text{NumObs}$ must be set. If **MAXTlimits** is called immediately after a call to **Maxlik**, $\_\text{max}_\text{NumObs}$ will be set by **Maxlik**.

**SOURCE**  
- maxlik.src
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’état. 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.

<table>
<thead>
<tr>
<th>Program</th>
<th>Models and citations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hurdlep</td>
<td>Hurdle Poisson regression model (1989d: Section 4).</td>
</tr>
<tr>
<td>Supreme</td>
<td>Seemingly unrelated Poisson regression model (1989c).</td>
</tr>
<tr>
<td>Supreme2</td>
<td>Poisson regression model with unobserved dependent variables (1989d: Section 6).</td>
</tr>
<tr>
<td>Expon</td>
<td>Exponential duration model with or without censoring (King, Alt, Burns, and Laver, 1989).</td>
</tr>
<tr>
<td>Expgam</td>
<td>Exponential-Gamma duration model with or without censoring (King, Alt, Burns, and Laver, 1989).</td>
</tr>
<tr>
<td>Pareto</td>
<td>Pareto duration model with or without censoring (King, Alt, Burns, and Laver, 1989).</td>
</tr>
</tbody>
</table>
5.0.1 README Files

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

5.0.2 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:

```gauss
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:

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

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

```gauss
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:
Maxlik 5.0 for GAUSS

.cn_ver  3×1 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.

5.1 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.
5.1.1 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.
5.1.2 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 $\text{SQRT(DIAG}(vc))$. If you choose the “” global __CovPar = 3, $vc$ contains heteroskedastic-consistent parameter estimates. See Section 3.6 for more discussion of options for statistical inference in maximum likelihood models.

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

5.1.3 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 bootstrapped 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

_\texttt{cn\_Censor} = \textit{“varname”} if you are using a \texttt{GAUSS} data set, or a number

(_\texttt{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.

_\texttt{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 _\texttt{cn\_Fix} to a variable name or number (_\texttt{cn\_Fix} = \textit{“totals”} or

_\texttt{cn\_Fix} = 12). The default (_\texttt{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 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.

_\texttt{cn\_Dispersion} scalar, set this to a value to change the starting value for only the

dispersion parameter in the negative binomial (\textit{Negbin}), generalized event

count (\textit{Hurdlep}), exponential-gamma (\textit{Expgam}), Pareto (\textit{Pareto}), and

seemingly-unrelated Poisson models (\textit{Supreme}, \textit{Supreme2}). By default, a

special starting value is not used for the dispersion parameter.
Maxlik 5.0 for GAUSS

__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.

5.1.4 Statistical Inference

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>maxlik</td>
<td>maximum likelihood estimates</td>
</tr>
<tr>
<td>boot</td>
<td>bootstrapped estimates</td>
</tr>
</tbody>
</table>
That is to generate bootstrapped estimates, set

\[ _\text{cn}_\text{Inference} = "\text{boot}"; \]

**Bootstrapping**

In addition to the usual standard errors, you may generate bootstrap standard errors. Setting \(_\text{cn}_\text{Inference} = \text{BOOT}\) causes \textsc{COUNT} to call \textsc{MAXBoot}. This generates bootstrapped estimates and covariance matrices of the estimates.

The bootstrapped parameters are also stored in a \textsc{Gauss} data set. The name of the data set can be determined by setting \(_\text{max}_\text{BootFname}\) to a file name, or by default it will be set to \textsc{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 3.6.4.

The data set thus generated can be used for computing confidence intervals of the coefficients using \textsc{MAXBlimits}. Also, density estimates and plots can be generated using \textsc{MAXDensity}, and histograms and surface plots of the coefficients can be produced using \textsc{MAXHist}. For further details about \textsc{MAXDensity}, see Section 3.6.4, and for further details about \textsc{MAXHist} see Section 3.6.4.

**5.1.5 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:

```gauss
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).

5.2 Annotated Bibliography


11. 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.]


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) × 2 matrix, confidence limits

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

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:

\[
_{\text{cn}}\text{_Inference} = \{ \text{boot} \};
\{ b, vc, llik \} = \text{Expgam}(\text{dataset, dep, ind});
cl = \text{MAXBlimits}(\_\text{max}_\text{_BootFname});
call \text{CountCLPrt}(b, cl, llik);
\]

SOURCE  count.src

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

Library  count

Format  \{ \text{b, vc, llik} \} = \text{CountPrt}(\text{b, vc, llik});

Input  \text{b} \quad \text{(K+1)\times1} \text{ vector, maximum likelihood estimates of the effect parameters stacked on top of the dispersion parameter.}
\text{vc} \quad \text{K+1}\times(K+1) \text{ matrix, variance-covariance matrix of the estimated parameters}
\text{llik} \quad \text{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 \} = \text{countprt}(\text{Expgam}($\text{dataset}$, $\text{dep}$, $\text{ind}$));

\text{SOURCE:} \text{count.src}

**Purpose:** Resets \texttt{COUNT} global variables to default values.

**Library:** \texttt{count}

**Format:** \texttt{CountSet;}

**Input:** None

**Output:** None

**Remarks:** Putting this instruction at the top of all command files that invoke \texttt{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 \texttt{COUNT} procedure.

\texttt{CountSet} calls \texttt{Set} which calls \texttt{GAUSSET}.

**Source:** \texttt{count.src}
Expgam

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

**LIBRARY**
count

**FORMAT**
\[
\{ b, vc, llik \} = \text{Expgam}(\text{dataset}, \text{dep}, \text{ind});
\]

**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-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 **MAXLimits** 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.
string, message printed at the top of the screen and printed out by `CountPrt`. Default = “”.

_scalar, if `dataset` is a matrix in memory, the variable names are automatically created by “”

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 \( \mu_i \) and variance \( \mu_i^2 \sigma^2 \). Let the mean \( \mu_i \) be an exponential-linear function of a vector of explanatory variables, \( x_i \):

\[
E(y_i) \equiv \mu_i = \exp(x_i \beta)
\]  

(2)

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 \( \mu_i \) 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 \( \sigma^2 \) is parametrized as follows:

\[
\sigma_i^2 = \exp(\gamma)
\]  

(3)
Expgam

EXPGAM reports estimates of $\beta$ and $\gamma$.

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 };

Expon

_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 \textit{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 \} = \textbf{Expon}(dataset, dep, ind);

INPUT \textit{dataset} string, name of \textbf{GAUSS} data set.  
\quad – or –  
\quad N\times K matrix, data.

\textit{dep} string, the name of the dependent variable.  
\quad – or –  
\quad scalar, the index of the dependent variable.

\textit{ind} K\times 1 character vector, names of the independent variables.
Expon

– 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 $\mu_i$. Also let
$E(y_i) \equiv \mu_i = \exp(x_i \beta)$. Note that $\mu_i$ 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 Kalbflieisch and Prentice

EXAMPLE  Exponential Regression Model

```plaintext
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 bootstrapped 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.
Hurdlep

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 \( \lambda_0i \) be the rate of the first event occurrence and \( \lambda_{+i} \) 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:

\[
\lambda_0i = \exp(x_i \beta) \tag{4}
\]

and

\[
\lambda_{+i} = \exp(z_i \gamma) \tag{5}
\]

where \( x_i \) and \( z_i \) are (possibly) different vectors of explanatory variables. The program produces estimates of \( \beta \) and \( \gamma \). 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 

\textit{dataset} \hspace{1em} \text{string, name of \texttt{GAUSS} data set.}
\hspace{1em} \text{– or –}
\hspace{1em} N\times K \text{ matrix, data.}
\hspace{1em} 
\textit{dep} \hspace{1em} \text{string, the name of the dependent variable.}
Negbin

– 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.
Negbin

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 [-1/2, 1/2].

Default = 0.

(cn_StartValue) (K+1)x1 or (K+L)x1 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
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 $\lambda_i$ and variance $\lambda_i\sigma^2$. Let the mean $\lambda_i$ (the rate of event occurrence, which must be greater than zero) be an exponential-linear function of a vector of explanatory variables, $x_i$:

$$E(y_i) \equiv \lambda_i = \exp(x_i\beta)$$  \hspace{1cm} (6)

The program includes a constant term as the first column of $x_i$ and allows one to include any number of explanatory variables.
\( \sigma^2 \) is parametrized as follows:

\[
\sigma^2_i = 1 + \exp(z_i \gamma) \tag{7}
\]

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 \( \beta \) and \( \gamma \).

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

```plaintext
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

```r
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 \( \text{ind1} \) and one for the variance function \( \text{ind2} \). 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

```r
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);
```

---

Maxlik Command Reference  6-25
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

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 \textit{ind1} pertain to the expected value and \textit{ind2} to the variance. Seven parameters are estimated.

\textbf{SOURCE} \hspace{1cm} negbin.src
PURPOSE  Estimates a Pareto regression model, for the analysis of duration data, with maximum likelihood.

LIBRARY  count

FORMAT  \{ b, vc, llik \} = \textbf{Pareto}(\textit{dataset}, \textit{dep}, \textit{ind});

INPUT  \textit{dataset} string, name of \textbf{GAUSS} data set.
– or –
N×K matrix, data.

\textit{dep} string, the name of the dependent variable.
– or –
scalar, the index of the dependent variable.

\textit{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 \textit{dataset} is a matrix, \textit{dep} and \textit{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 \texttt{__vpad}), or the user-provided labels in \texttt{__altnam}.

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

\textit{vc} (K+1)×(K+1) matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If the “” global \texttt{__CovPar} is set to 3, \textit{vc} will contain heteroskedastic-consistent parameter estimates.
Pareto

\( llik \) 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 \((K+1)\times1\) 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
Pareto

__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 $\mu_i$ and variance $\mu_i \sigma^2 + \mu_i^2$. Let the mean $\mu_i$ be an exponential-linear function of a vector of explanatory variables, $x_i$:

$$E(y_i) = \mu_i = \exp(x_i \beta)$$  \hspace{1cm} (8)

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 $\mu_i$ 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 $\sigma^2$ is parametrized as follows:

$$\sigma_i^2 = \exp(\gamma)$$

The program gives estimates of $\beta$ and $\gamma$.

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

```plaintext
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 \textit{ind} and a constant term. Five parameters are estimated in this example.

\textbf{SOURCE} \hspace{1cm} pareto.src

\begin{verbatim}

\end{verbatim}
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 bootstrapped 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,....
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:

$$f_p(y_i | \lambda_i) = \begin{cases} \frac{e^{-\lambda_i} (\lambda_i)^{y_i}}{y_i!} & \text{for } \lambda_i > 0 \text{ and } y_i = 0, 1, \ldots \\ 0 & \text{otherwise} \end{cases}$$ (10)

with expected value and variance $\lambda_i$. Under the Poisson regression model, $\lambda_i$ (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, $x_i$:

$$E(y_i) = \lambda_i = \exp(x_i \beta)$$ (11)

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).
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
SUPREME

Purpose

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

Library

count

Format

\[
\{ b, vc, llik \} = \text{Supreme}(\text{dataset}, \text{dep1}, \text{dep2}, \text{ind1}, \text{ind2});
\]

\textit{dataset} string, name of GAUSS data set.
- or -
N×K matrix, data.

\textit{dep1} string, name of the first dependent variable.
- or -
scalar, index of the first dependent variable.

\textit{dep2} string, name of the second dependent variable.
- or -
scalar, index of the second dependent variable.

\textit{ind1} \( K \times 1 \) character vector, names of first event independent variables.
- or -
\( K \times 1 \) numeric vector, indices of first event independent variables.
Set to 0 to include only a constant term.

\textit{ind2} \( K \times 1 \) character vector, names of second event independent variables.
- or -
\( K \times 1 \) numeric vector, indices of second event independent variables.
Set to 0 to include only a constant term.

If \textit{dataset} is a matrix, \textit{dep1}, \textit{dep2}, \textit{ind1} and \textit{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  

\( b \)  
\((K+L+2)\times1\) vector, maximum likelihood estimates of the effect parameters of \( \beta \) and \( \gamma \) stacked on top of the covariance parameter \( \xi \).

\( vc \)  
\((K+L+2)\times(K+L+2)\) matrix, variance-covariance matrix of the estimated parameters evaluated at the maximum. If you choose the global \texttt{CovPar} = 3, \( vc \) will contain heteroskedastic-consistent parameter estimates.

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

GLOBALS  

\textbf{Maxlik} globals are also relevant.

\_cn\_Inference  
string, determines the type of statistical inference.

\textit{boot}  
generates bootstrapped estimates and covariance matrix of estimates

\textit{maxlik}  
generates maximum likelihood estimates

Setting \texttt{cn\_Inference} to \texttt{BOOT} generates a \texttt{GAUSS} data set containing the bootstrapped parameters. The file name of this data set is either a temporary name, or the name in the \texttt{Maxlik} global variable, \texttt{max\_BootFname}. This data set can be used with \texttt{MAXBlimits} for generating confidence limits, with \texttt{MAXDensity} for generating density estimates and plots of the boostrapped parameters, or with \texttt{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 \texttt{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:

\[
\lambda_{0i} = \exp(x_i \beta) \quad (12)
\]

and

\[
\lambda_{+i} = \exp(z_i \gamma) \quad (13)
\]
where $x_i$ and $z_i$ are (possibly) different vectors of explanatory variables. The covariance parameter is $\xi$.

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)

```plaintext
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(\textit{dataset}, \textit{dep1}, \textit{dep2}, \textit{ind1}, \textit{ind2}, \textit{ind3});

INPUT  \textit{dataset}  string, name of \textbf{GAUSS} data set.
– or –
N×K matrix, data.

\textit{dep1}  string, name of the first dependent variable.
– or –
scalar, index of the first dependent variable.

\textit{dep2}  string, name of the second dependent variable.
– or –
scalar, index of the second dependent variable.

\textit{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.

\textit{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 nd3 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 \texttt{MAXHist} for generating histogram and surface plots.

\texttt{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 \texttt{cn\_StartValue}.

2 uses a vector of zeros.

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

Default = 0.

\texttt{cn\_StartValue} \hspace{1em} (K+L+M)×1 vector, start values if \texttt{cn\_Start} = 1.

\texttt{cn\_Precision} scalar, number of decimal points to print on output.
Default = 4.

\texttt{__altnam} K×1 vector, alternate names for variables when a matrix is passed to \texttt{Supreme2}. When a data matrix is passed to \texttt{Supreme2} and the user is selecting from that matrix, the global variable \texttt{__altnam} if it is used, must contain names for the columns of the original matrix.

\texttt{__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.

\texttt{__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:

\[
\begin{align*}
y_{1i} &= y_{1i}^* + y_{3i}^* \\
y_{2i} &= y_{2i}^* + y_{3i}^*
\end{align*}
\]

The procedure estimates three separate regression functions, one for the expected value of each of the unobserved variables:

\[
\begin{align*}
\lambda_{1i} &= \exp(x_{1i}\beta_1) \\
\lambda_{2i} &= \exp(x_{2i}\beta_2) \\
\lambda_{3i} &= \exp(x_{3i}\beta_3)
\end{align*}
\]

where $x_{1i}$, $x_{2i}$, and $x_{3i}$ are (possibly) different sets of explanatory variables and $\beta_1$, $\beta_2$, and $\beta_3$ are separate parameter vectors. This option produces maximum likelihood estimates for these three parameter vectors.

**EXAMPLE**  Poisson Regression Model with Unobserved Dependent Variables

```plaintext
library count;
#include count.ext;
Countset;
dataset = "wars";
dep1 = { wars };
ind1 = { unem, poverty, allianc };
dep2 = { coups };
ind2 = { unem, age, sex, race };
```
Supreme2

\[
\text{ind3} = \{ \text{us, sov} \}; \\
\{ \text{b,vc,llik} \} = \text{Supreme2}(\text{dataset,dep1,dep2,ind1,ind2,ind3}); \\
\text{output file} = \text{count.out reset}; \\
\text{call CountPrt(b,vc,llik)}; \\
\text{output off};
\]

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
active parameters, 3-9
AD, 3-19
algorithm, 3-44
algorithmic derivatives, 3-19
Alt-1, 3-44
Alt-2, 3-44
Alt-3, 3-44
Alt-4, 3-44
Alt-5, 3-44
Alt-6, 3-44
Alt-A, 3-44
Alt-H, 3-43
__altnam, 5-6, 5-8, 6-5, 6-11, 6-16, 6-22, 6-29, 6-34, 6-40, 6-45

B __________________________

Bayesian inference, 3-29, 4-12, 4-36
BFGS, 3-4, 3-44, 4-4, 4-24
BHHH, 3-5, 3-44, 4-4, 4-24
BHHHStep, 3-8
bootstrap, 3-38, 5-6, 5-10
BRENT, 3-6, 3-7

C __________________________

_cn_Censor, 5-6
_cn_Dispersion, 5-6
_cn_Fix, 5-6
_cn_Inference, 5-6, 5-9
_cn_Precision, 5-6
_cn_Start, 5-6
_cn_StartValue, 5-8, 6-5, 6-11, 6-16, 6-21, 6-28, 6-34, 6-40, 6-45
_cn_ZeroTruncate, 5-6
condition of Hessian, 3-12
conjugate gradient, 3-5
convergence, 3-5, 4-7, 4-28
count.src, 6-2, 6-3
CountCLPrt, 6-1
CountPrt, 6-2
CountSet, 6-3
covariance matrix, parameters, 3-29, 3-31, 3-36, 4-4, 4-24
__CovPar, 5-6
cubic step, 4-6, 4-28

D __________________________
density, 5-6
### Index

<table>
<thead>
<tr>
<th>Derivatives, 3-3, 3-18, 4-4, 4-24, 4-34</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFP, 3-4, 3-44, 4-4, 4-24</td>
</tr>
<tr>
<td>Diagnosis, 3-17</td>
</tr>
<tr>
<td>Direction, 3-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expgam</strong>, 6-3</td>
</tr>
<tr>
<td>expgam.src, 6-9</td>
</tr>
<tr>
<td><strong>Expon</strong>, 6-9</td>
</tr>
<tr>
<td>expon.src, 6-14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FASTBAYES</strong>, 3-29</td>
</tr>
<tr>
<td><strong>FASTBayes</strong>, 4-12</td>
</tr>
<tr>
<td>fastbayes.src, 4-14</td>
</tr>
<tr>
<td><strong>FASTBOOT</strong>, 3-29</td>
</tr>
<tr>
<td><strong>FASTBoot</strong>, 4-14</td>
</tr>
<tr>
<td>fastboot.src, 4-17</td>
</tr>
<tr>
<td><strong>FASTMAX</strong>, 3-28, 4-1</td>
</tr>
<tr>
<td>fastmax.src, 4-11</td>
</tr>
<tr>
<td>fastpflcl.src, 4-19</td>
</tr>
<tr>
<td><strong>FASTPflClimits</strong>, 4-17</td>
</tr>
<tr>
<td>fastprof.src, 4-21</td>
</tr>
<tr>
<td><strong>FASTPROFILE</strong>, 3-29</td>
</tr>
<tr>
<td><strong>FASTProfile</strong>, 4-19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global variables, 3-43</td>
</tr>
<tr>
<td>Gradient, 4-2, 4-12, 4-15, 4-19, 4-22,</td>
</tr>
<tr>
<td>4-37, 4-39, 4-42, 4-48, 4-52</td>
</tr>
<tr>
<td>Gradient procedure, 3-18, 4-5, 4-9, 4-10,</td>
</tr>
<tr>
<td>4-26, 4-33</td>
</tr>
<tr>
<td><strong>GRADRE</strong>, 4-31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>HALF, 3-7</td>
</tr>
<tr>
<td>Hessian, 3-3, 3-12, 3-44</td>
</tr>
<tr>
<td>Hessian procedure, 3-25, 3-27, 4-11,</td>
</tr>
<tr>
<td>4-35</td>
</tr>
<tr>
<td><strong>Hurdlep</strong>, 6-14</td>
</tr>
<tr>
<td>hurdlep.src, 6-19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inactive parameters, 3-9</td>
</tr>
<tr>
<td>Installation, 1-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kiss-Monster, 3-40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likelihood profile trace, 3-36, 3-37</td>
</tr>
<tr>
<td>Line search, 3-3, 3-6, 3-44</td>
</tr>
<tr>
<td>Linear congruential, 3-40</td>
</tr>
<tr>
<td>Log-likelihood function, 3-1, 3-2, 3-27,</td>
</tr>
<tr>
<td>4-2, 4-9, 4-10, 4-12, 4-15, 4-22,</td>
</tr>
<tr>
<td>4-32, 4-33, 4-34, 4-36, 4-39,</td>
</tr>
<tr>
<td>4-48, 4-50</td>
</tr>
<tr>
<td>Log-linear, 5-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_Active, 3-9, 4-3, 4-23, 4-24</td>
</tr>
<tr>
<td>max_Algorithm, 4-3, 4-4, 4-23, 4-24</td>
</tr>
<tr>
<td>max_Alpha, 4-18, 4-41, 4-51, 4-54</td>
</tr>
<tr>
<td>max_BayesAlpha, 3-40, 4-13, 4-37</td>
</tr>
<tr>
<td>max_BootFname, 4-13, 4-14, 4-16,</td>
</tr>
<tr>
<td>4-17, 4-37, 4-38, 4-40, 4-41</td>
</tr>
</tbody>
</table>

---

Index-2
_max_Center, 4-20, 4-46, 4-49
_max_CovPar, 3-31, 4-2, 4-3, 4-4, 4-22, 4-23, 4-24
_max_CutPoint, 4-46
_max_Delta, 4-3, 4-4, 4-23, 4-24
_max_Diagnostic, 3-17, 4-23, 4-24
_max_Diagnostic
_max_Extrap, 4-3, 4-4, 4-23, 4-25
_max_FinalHess, 3-32, 4-3, 4-4, 4-23, 4-25
_max_GradCheckTol, 3-25, 4-11, 4-23, 4-25, 4-35
_max_GradMethod, 3-44, 4-3, 4-4, 4-23, 4-26
_max_GradProc, 4-3, 4-5, 4-10, 4-23, 4-26, 4-33, 4-34
_max_GradStep, 4-3, 4-5, 4-23, 4-26
_max_GradTol, 3-5, 3-44, 4-3, 4-4, 4-23, 4-25
_max_HessCov, 3-32, 4-3, 4-5, 4-23, 4-26
_max_HessProc, 3-25, 4-3, 4-6, 4-11, 4-23, 4-27, 4-35
_max_Increment, 4-20, 4-46, 4-49
_max_Interp, 4-3, 4-6, 4-23, 4-27
_max_IterData, 4-3, 4-6, 4-23, 4-27
_max_Kernel, 3-39, 4-44
_max_Key, 4-23, 4-27
_max_Lag, 4-23, 4-27
_max_Lagrange, 3-31
_max_LineSearch, 4-3, 4-6, 4-23, 4-28
_max_MaxIters, 4-3, 4-7, 4-23, 4-28
_max_MaxTime, 4-3, 4-7, 4-13, 4-16, 4-23, 4-28, 4-37, 4-40
_max_MaxTry, 3-44
_max_Maxtry, 4-3
_max_MaxTry, 4-7
_max_MaxTry, 4-23
_max_MaxTry, 4-28
_max_NumCat, 4-20, 4-46, 4-49
_max_NumObs, 3-38, 4-3, 4-14, 4-17, 4-23, 4-38, 4-40
_max_NumPoints, 4-44
_max_NumSample, 3-38, 4-7, 4-13, 4-16, 4-20, 4-28, 4-37, 4-40, 4-49
_max_Options, 4-3, 4-7, 4-23, 4-29
_max_ParNames, 4-3, 4-8, 4-23, 4-29
_max_PriorProc, 4-13, 4-37
_max_RandRadius, 3-6, 4-3, 4-7, 4-8, 4-23, 4-28, 4-29
_max_RandType, 3-40, 4-3, 4-8, 4-14, 4-16, 4-23, 4-30
_max_Select, 4-20, 4-41, 4-49, 4-54
_max_Smoothing, 3-39, 4-44
_max_State, 4-3, 4-8, 4-14, 4-17, 4-23, 4-30
_max_Switch, 3-28, 4-3, 4-8, 4-23, 4-30
_max_Truncate, 4-44
_max_UserHess, 3-27
_max_UserNumGrad, 4-3, 4-23, 4-30
_max_UserNumHess, 4-23, 4-31
_max_UserSearch, 4-23, 4-32
_max_Width, 4-20, 4-46, 4-49
_max_XprodCov, 3-32, 4-3, 4-9, 4-23
MAXBayes, 4-36
maxbayes.src, 4-38
maxblim.src, 4-42
MAXBlimits, 4-41
Index

MAXBoot, 3-38, 4-38
maxboot.src, 4-41
MAXCLPrt, 4-42
maxdens.src, 4-45
MAXDensity, 3-38, 4-43, 5-10
MAXHist, 3-39, 4-45, 5-6, 5-10
maxhist.src, 4-47
maximum likelihood, 3-1, 4-1, 4-14, 4-21, 4-38, 5-1
Maxlik, 4-21
maxlik.src, 4-35, 4-43, 4-52, 4-53, 4-54
maxpflcl.src, 4-51
MAXPflLimits, 4-50
maxprof.src, 4-49
MAXProfile, 4-47
MAXPrt, 4-51
MAXSet, 4-53
MAXTlimits, 4-53
__miss, 6-6, 6-11, 6-16, 6-22, 6-29, 6-34, 6-40, 6-45

N ________________

Negbin, 6-19
negbin.src, 6-26
NEWTON, 3-4, 3-44, 4-4, 4-11, 4-24, 4-35
NR, 3-44

O ________________

__output, 3-44, 4-29, 4-44, 4-46, 5-6, 5-8, 6-6, 6-11, 6-17, 6-22, 6-29, 6-35, 6-40, 6-45

P ________________
Pareto, 6-27
pareto.src, 6-32
Poisson, 6-32
poisson.src, 6-37
PRCG, 3-5, 3-44, 4-4, 4-24
profile likelihood, 3-29, 3-33, 4-17, 4-50
profile t plot, 3-36
pseudo-random numbers, 3-40

Q ________________

quadratic step, 4-6, 4-28
quasi-maximum likelihood covariance matrix, 3-31, 4-4, 4-24
quasi-Newton, 3-4

R ________________

random search, 3-8
regression, Hurdle Poisson, 5-2
regression, negative binomial, 5-2
regression, seemingly unrelated Poisson, 5-2
regression, truncated negative binomial, 5-2
regression, truncated Poisson, 5-2
resampling, 3-38
Richardson Extrapolation, 4-31
__row, 3-2
__row, 4-2, 4-22, 4-23, 4-27, 4-31, 4-33, 4-34, 4-36, 4-39, 4-48, 4-50
__rowfac, 4-23, 4-31, 5-9, 6-6, 6-12, 6-17, 6-22, 6-29, 6-35, 6-40, 6-46

Index-4
run-time switches, 3-43

S __________________________

scaling, 3-11
Shift-1, 3-44
Shift-2, 3-44
Shift-4, 3-44
Shift-3, 3-44
Shift-5, 3-44
starting point, 3-16
statistical inference, 3-29, 5-9
STEEP, 3-44, 4-4, 4-24
step length, 3-6, 3-44, 4-6, 4-28
STEPBT, 3-6
Supreme, 6-37
supreme.src, 6-42
Supreme2, 6-43
supreme2.src, 6-48
switching algorithms, 3-28

T __________________________

__title, 4-3, 4-8, 4-23, 4-31

U __________________________

UNIX, 1-3
UNIX/Linux/Mac, 1-1

V __________________________

vput, 3-17
vread, 3-17

W __________________________

Wald, 3-29, 4-53
Wald inference, 3-30
__weight, 3-8, 4-3, 4-9, 4-23, 4-32
weighted maximum likelihood, 3-8
Windows, 1-2, 1-3