CurveFit

Aptech Systems, Inc.
Maple Valley, WA

Documentation Version: February 25, 2002

Part Number: 000024

Contents

Chapter 1

Installation

1.1 UNIX

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

1.1.1 Download

1. Copy the `.tar.gz` file to `/tmp`.
2. Unzip the file.

   `gunzip` *appxxx*`.tar.gz`
3. `cd` to the **GAUSS** or **GAUSS Engine** installation directory. We are assuming `/usr/local/gauss` in this case.

   ```
   cd /usr/local/gauss
   ```
4. Untar the file.

   `tar xvf /tmp/`*appxxx*`.tar`

1.1.2 Floppy

1. Make a temporary directory.

   ```
   mkdir /tmp/workdir
   ```

2. `cd` to the temporary directory.

   ```
   cd /tmp/workdir
   ```

3. Use `tar` to extract the files.

   `tar xvf` *device_name*

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

4. Read the README file.

   ```
   more README
   ```

5. Run the `install.sh` script in the work directory.

   ```
   ./install.sh
   ```

   The directory the files are install to should be the same as the install directory of **GAUSS** or the **GAUSS Engine**.

6. Remove the temporary directory (optional).

The following device names are suggestions. See your system administrator. If you are using Solaris 2.x, see Section 1.1.3.

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

1.1.3 Solaris 2.x Volume Management

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

```
volcheck
```

to signal the system to mount the floppy.

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

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

To turn on volume management, become the superuser and type

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

1.2 Windows/NT/2000

1.2.1 Download

Unzip the `.zip` file into the **GAUSS** or **GAUSS Engine** installation directory.

1.2.2 Floppy

1. Place the diskette in a floppy drive.
2. Call up a DOS window
3. In the DOS window log onto the root directory of the diskette drive. For example:

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

4. Type: **ginstall** *source_drive target_path*

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

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

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

1.3 Differences Between the UNIX and Windows/NT/2000 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.

- On the Intel math coprocessors used by the Windows/NT/2000 machines, intermediate calculations have 80-bit precision, while on the current UNIX machines, all calculations are in 64-bit precision. For this reason, **GAUSS** programs executed under UNIX may produce slightly different results, due to differences in roundoff, from those executed under Windows/NT/2000.

Chapter 2

Using CurveFit

The *CurveFit* module includes procedures for the nonlinear least squares fitting of data with or without weighting. Additional procedures provide for computing bootstrapped estimates and their distributions.

2.1 Getting Started

GAUSS 3.2.0+ is required to use these routines.

2.1.1 README Files

The file `README.cv` contains any last minute information on the **CurveFit** procedures. Please read it before using them.

2.1.2 Setup

In order to use these procedures the **CurveFit** library must be active. This is done by including `cvfit` in the **LIBRARY** statement at the top of your program:

```
library cvfit,pgraph;
```

This enables **GAUSS** to find the **CurveFit** procedures. If you plan to make any right-hand references to the global variables (described under the **CurveFit** function definition in chapter 3), you will also need the statement:

```
#include cvfit.ext;
```

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

```
CurveFitSet;
```

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

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

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

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

2.2 Estimation

Suppose we wish to model an observed "response", or dependent variable, given observations on independent variables. For example, a two compartment model of the metabolism of tetracycline (Bates and Watts, page 281) in the blood might be modelled using the function,

$$h(\theta, x) = \theta_3[e^{-\theta_1(x-\theta_4)} - e^{-\theta_2(x-\theta_4)}]$$

where f is the predicted tetracycline hydrochloride concentration and x is time. θ_1 is a coefficient measuring the movement of the drug into the blood and θ_2 represents the metabolism or elimination of the drug.

CurveFit is a module for the estimation of the coefficients of nonlinear functions such as this one, as well as their distributions. The user must provide a **GAUSS** proc to compute the response or dependent variable given values for the coefficients and the independent variables. The function must be twice differentiable, though **GAUSS** procs to compute the derivatives are not required.

A **GAUSS** proc for the estimation of the coefficients of the model above would look like this:

```
proc fct(th,x)
    retp(th[3]*exp(-th[1]*(x-th[4]))-exp(-th[2]*(x-th[4])));
endp;
```

This **GAUSS** proc and a set of observations on concentration of tetracycline hydrochloride at different time intervals is all that is needed for **CurveFit** to generate estimates of the coefficients.

Objective Function

Define the $N \times 1$ residual matrix Z, the i^{th} element of which is

$$z_i = y_i - h(\theta, x_i),$$

where N is the number of observations. For a single dependent variable Y and Z are vectors, and the objective function is

$$F = Z'Z = (Y - h(\theta, X))'(Y - h(\theta, X))$$

which may be interpreted as the sum of the squared residuals. Minimizing F amounts to finding values for θ that minimize the residual variance. If the assumption that the z_i are independently Gaussian distributed with equal variance (i.e., homoskedastic), the coefficient estimates are maximum likelihood.

Example

Clarke (1987) fits a Micherlitz model

$$h(t, b_0, b_1, b_2) = b_1 + b_2 e^{-b_2 t}$$

to the weight of cut grass from 10 randomly sited quadrants as a function of weeks after the start of grazing in a pasture. The command file for this analysis is:

```
library cvfit;
#include cvfit.ext;
CurveFitset;

proc Micherlitz(b,x);
    retp(b[1] + b[2]*exp(-b[3]*x));
endp;

b0 = { 1, 2.5, .1 };

__title = "Micherlitz Model";

y = { 3.183, 3.059, 2.871, 2.622, 2.541, 2.184, 2.110, 2.075,
      2.018, 1.903, 1.770, 1.762, 1.550 };

x = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };

call CurveFitPrt(CurveFit("",y,x,&Micherlitz,b0));
```

and the results are

```
================================================================================
                              Micherlitz Model
================================================================================
 CurveFit Version 3.1.1                                        8/01/94  10:15 am
================================================================================

return code =    0
normal convergence

Number of cases      13

estimated residual variance      0.00534536

Parameters  Estimates   Std. err.  Est./s.e.  Prob.    Gradient
------------------------------------------------------------------
P01          0.963133    0.321581    2.995   0.0014    0.000001
P02          2.518989    0.265764    9.478   0.0000   -0.000000
P03          0.103056    0.025504    4.041   0.0000   -0.000001

Covariance matrix of parameters computed from
cross-product of first derivatives

Correlation matrix of the coefficients
   1.000  -0.972   0.984
  -0.972   1.000  -0.923
   0.984  -0.923   1.000

Number of iterations    8
Minutes to convergence     0.00183
```

2.2.1 Multiple Dependent Variables

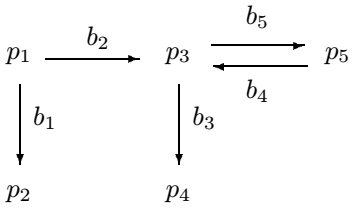
For more than one dependent variable the objective function is

$$F = \ln \det(Z'Z)$$

Z is now an $N \times L$ matrix where L is the number of dependent variables. It is important to note that this objective function does not provide for models in which the dependent variable in one equation is independent in another. The dependent variables, therefore, must not occur on the right hand side of any equation in the model.

Example

The following diagram describes a compartment model with five compartments representing concentrations of *a-pinene*, *dipentene*, *aloocimene*, *pyronene*, and *dimer*. This example is taken from Bates and Watts (1988), page 147 ff.



The diagram implies the following set of simultaneous first order differential equations:

$$\begin{aligned}
\frac{dp_1}{dt} &= -(b_1 + b_2)p_1 \\
\frac{dp_2}{dt} &= b_1 p_1 \\
\frac{dp_3}{dt} &= b_2 p_1 - (b_3 + b_4)p_3 + b_5 p_5 \\
\frac{dp_4}{dt} &= b_3 p_3 \\
\frac{dp_5}{dt} &= b_4 p_3 - b_5 p_5
\end{aligned}$$

Arranging the coefficients into a matrix B:

$$\begin{bmatrix}
-b_1 - b_2 & 0 & 0 & 0 & 0 \\
b_1 & 0 & 0 & 0 & 0 \\
b_2 & 0 & -b_3 - b_4 & 0 & b_5 \\
0 & 0 & b_3 & 0 & 0 \\
0 & 0 & b_4 & 0 & -b_5
\end{bmatrix}$$

the system of equations can be described

$$\frac{dP}{dt} = BP$$

and the solution

$$\begin{aligned}
P(t) &= e^{Bt} \\
&= Ue^{Lt}U^{-1}P(0)
\end{aligned}$$

where ULU^{-1} is an eigendecomposition of B and where $P(0)$ is the initial condition.

The following is a **GAUSS** command file to estimate this model.

```
library cvfit;
#include cvfit.ext;
CurveFitset;



proc fct(b0,x);
   local p, b, p0, u, e, ui, i;

   p = zeros(rows(x),5);

   b0 = exp(b0);

   b0 = exp(b0);  /* The coefficients are strictly positive and therefore */
                  /* the model will estimate the log of the coefficients  */

   b = zeros(5,5);
   b[1,1] = -b0[1] -b0[2];
   b[2,1] =  b0[1];
   b[3,1] =  b0[2];
   b[3,3] = -b0[3] -b0[4];
   b[3,5] =  b0[5];
   b[4,3] =  b0[3];
   b[5,3] =  b0[4];
   b[5,5] = -b0[5];

   p0 = { 1, 0, 0, 0, 0 };

   { e, u } = eigv(b);
   e = real(e);
   u = real(u);
   ui = inv(u);

   i = 1;
   do until i > rows(x);
     p[i,.] = ( ( (u.*exp(e*x[i])') * ui) * p0 )';
     i = i + 1;
   endo;

   retp(p);
endp;


 /* minutes times 1e3 */
_time = { 1.2, 2.1, 3.0, 3.9, 4.9, 6.3, 7.8, 9.2, 10.6, 12.8, 15.0,
            18.8, 22.6, 29.5, 36.4};
```

```
_conc = {
     /* a-pinene  dipentene  aloocimene  pyronene  dimer */
         0.75   0.17   0.06  -0.01   0.00,
         0.62   0.24   0.10   0.01   0.01,
         0.51   0.32   0.11   0.02   0.03,
         0.39   0.37   0.13  -0.01   0.06,
         0.32   0.44   0.13   0.01   0.07,
         0.25   0.50   0.14   0.04   0.08,
         0.16   0.53   0.15   0.05   0.10,
         0.12   0.57   0.16   0.06   0.09,
         0.06   0.59   0.13   0.06   0.11,
         0.07   0.63   0.13   0.06   0.10,
         0.02   0.62   0.12   0.11   0.08,
         0.01   0.64   0.13   0.12   0.10,
         0.02   0.67   0.12   0.15   0.09,
         0.00   0.65   0.09   0.18   0.06,
         0.01   0.66   0.07   0.20   0.05
};


b0 = { -1.9, -2.5, -3.0, -1.4, -1.0 };

{ c,f,g,h,ret } = CurveFit("",_conc,_time,&fct,b0);

call CurveFitPrt(c,f,g,h,ret);
```

The results are:

```
=====================================================================
 CurveFit Version 3.1.1                              7/21/94  10:33 am
=====================================================================

return code =    0
normal convergence

Number of cases      15

estimated residual variance-covariance matrix
   0.000   0.000   0.000   0.000  -0.000
   0.000   0.000   0.000  -0.000   0.000
   0.000   0.000   0.000   0.000   0.000
   0.000  -0.000   0.000   0.000  -0.000
  -0.000   0.000   0.000  -0.000   0.000

Parameters    Estimates  Std. err.  Est./s.e.  Prob.    Gradient
```

```
-------------------------------------------------------------------
P01           -1.906199   0.014890 -128.021   0.0000    0.000051
P02           -2.564489   0.024142 -106.226   0.0000    0.000004
P03           -2.974562   0.042869  -69.388   0.0000    0.000026
P04           -1.353696   0.163991   -8.255   0.0000    0.000009
P05           -1.005758   0.180214   -5.581   0.0000    0.000085

Covariance matrix of parameters computed from
cross-product of first derivatives

Correlation matrix of the coefficients
   1.000   0.588  -0.080   0.208   0.187
   0.588   1.000   0.049   0.310   0.299
  -0.080   0.049   1.000   0.201   0.218
   0.208   0.310   0.201   1.000   0.957
   0.187   0.299   0.218   0.957   1.000

Number of iterations    8
Minutes to convergence     0.02100
```

2.2.2 Aiding Convergence

The example of the preceding section is a good example of a very difficult problem. To achieve rapid convergence consider

scaling failure to scale properly is the cause of nearly all problems in convergence. Check the Hessian - its diagonal elements should all be about the same order of magnitude

start values If convergence eludes you even with good scaling, work with the start values

descent methods toggle descent methods - press P during iterations to print iteration information to the screen, and the press D to toggle descent methods.

The example in the preceding section required very good start values to converge. However, even poor start values will work if the descent method is toggled first to the PRCG method and then to the Levenberg-Marquardt method after 10 or so iterations.

2.2.3 Frequencies

Let f_i be the *frequency* of the i^{th} observation. Further define the $N \times N$ matrix, Ω, the i^{th} diagonal element of which is f_i. The objective function for the single dependent variable becomes

$$F_f = Z'\Omega Z$$

and for multiple dependent variables

$$F_f = \ln\det(Z'\Omega Z)$$

The number of observations is computed, when there are frequencies, as the sum of the frequencies. If this is not the case, the covariance matrix of the coefficients, and thus the t-statistics, will not be correct. This feature permits the analysis of nonlinear models of tabulated data.

For computational purposes the frequencies are entered as elements of a column vector stored in the **GAUSS** global variable, **__weight**. The i^{th} row of **__weight** is the frequency of the i^{th} observation. Zero frequencies are allowed.

Example 1

In the example in Section 2.2 there are thirteen points or observations. To re-estimate the model without the, say, 4th and 5th observations, add the following line to the command file before the call to **CurveFit**:

```
__weight = { 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1 };
```

The results are:

```
=============================================================================
                              Micherlitz Model
=============================================================================
 CurveFit Version 3.1.1                            8/01/94   3:56 pm
=============================================================================

return code =    0
normal convergence

Number of cases      11

estimated residual variance      0.00586157

Parameters    Estimates  Std. err.  Est./s.e.  Prob.    Gradient
-----------------------------------------------------------------------
P01            1.077901   0.310968    3.466   0.0003   -0.000000
P02            2.418233   0.254753    9.492   0.0000    0.000000
P03            0.113281   0.029966    3.780   0.0001   -0.000001

Covariance matrix of parameters computed from
cross-product of first derivatives
```

```
Correlation matrix of the coefficients
   1.000  -0.962   0.984
  -0.962   1.000  -0.909
   0.984  -0.909   1.000

Number of iterations    9
Minutes to convergence     0.00267
```

Example 2

Suppose that based on some analysis of the residuals from an initial run on the preceding analysis, we decide to reduce the influence of the 4th and 5th observations on the estimation. To reduce that influence to, say, 50% of the rest of the observations, add the following lines before the call to **CurveFit**:

```
wgt = { 1, 1, 1, 1, .5, .5, 1, 1, 1, 1, 1, 1, 1 };
__weight = (rows(wgts)/sumc(wgts))*wgts;
```

The results are:

```
==============================================================================
                              Micherlitz Model
==============================================================================
 CurveFit Version 3.1.1                            7/20/94   3:23 pm
==============================================================================

return code =    0
normal convergence

Number of cases      13

estimated residual variance      0.00446591

Parameters    Estimates  Std. err.  Est./s.e.  Prob.    Gradient
------------------------------------------------------------------------
P01            0.908592   0.324097    2.803   0.0025    0.000000
P02            2.562695   0.273074    9.385   0.0000   -0.000000
P03            0.098521   0.023602    4.174   0.0000   -0.000000

Covariance matrix of parameters computed from
cross-product of first derivatives

Correlation matrix of the coefficients
   1.000  -0.980   0.987
```

```
  -0.980   1.000  -0.942
   0.987  -0.942   1.000

Number of iterations    9
Minutes to convergence     0.00267
```

2.2.4 Weights

The assumption of homoskedasticity is often untenable. Heteroskedastic residuals result in inefficient estimates and biased estimates of the covariance matrix of the estimates. If some information is available about the heteroskedasticity of the residual, it may be incorporated into the estimates using the frequencies feature of **CurveFit**.

Define the N $\times$ 1 vector W, the i^{th} element of which is a consistent estimate of the variance of the i^{th} residual up to a constant of proportionality. Because **CurveFit** expects frequencies, the weights must be first inverted and then "normalized", i.e., transformed to sum to N.

Define the vector F, the i^{th} element of which is

$$f_i = \frac{A}{w_i}$$

where

$$A = \frac{\text{No. of observations}}{\sum \frac{1}{w_i}}.$$

and where w_i is the i^{th} element of W. Then set the global variable **__weight** equal to F.

The resulting coefficient estimates will be efficient, provided the weights are proportional to consistent estimates of the variances of the residuals, and the estimate of the covariance matrix of the coefficients will be unbiased.

If little is known about the source of heteroskedasticity, weights may be computed from results of an unweighted estimation, or from a theoretical derivation (Seber and Wild, page 77). Alternatively, one could transform the variables (Seber and Wild, page 68, Bates and Watts, page 28), or compute the heteroskedastic-consistent standard errors described in Section 2.3.1.

Example

In the example in Section 2.2, there are thirteen points or observations. Suppose that based on some analysis of the residuals from an initial analysis, we have determined that the observations are heteroskedastic with variance proportional to time squared, i.e., to the square of the independent variable. To correct for this the following lines are added to the command file before the call to **CurveFit**:

```
wgt = 1/x^2;
__weight = (rows(wgt)/sumc(wgt))*wgt;
```

The results are:

```
=============================================================================
                              Micherlitz Model
=============================================================================
 CurveFit Version 3.1.1                              7/20/94   4:04 pm
=============================================================================

return code =    0
normal convergence

Number of cases      13

estimated residual variance      0.00201156

Parameters    Estimates  Std. err.  Est./s.e.  Prob.    Gradient
-------------------------------------------------------------------
P01           -0.137816   1.228786   -0.112   0.4553    0.000000
P02            3.527866   1.205738    2.926   0.0017    0.000000
P03            0.057360   0.025968    2.209   0.0136    0.000001

Covariance matrix of parameters computed from
cross-product of first derivatives

Correlation matrix of the coefficients
   1.000  -1.000   0.995
  -1.000   1.000  -0.993
   0.995  -0.993   1.000

Number of iterations    13
Minutes to convergence     0.00450
```

2.2.5 Fixed Coefficients

Coefficients can be fixed to their starting values by setting the **CurveFit** global variable, **__cv_active** to a vector of zeros and ones, where a zero indicates that the corresponding element in the starting vector is fixed to that value, and a one indicates that the corresponding element is free to be estimated. This feature allows the estimation of different types of models with one function procedure.

Example

Suppose that we have some prior knowledge about a parameter in the model presented in Section 2.2 - that the second parameter is equal to 2. This can be accomplished by adding the following line to the command file before the call to **CurveFit**:

```
_cv_Active = { 1, 0, 1 };
```

and revise the starting point to

```
b0 = { 1, 2.0, .1 };
```

The results are:

```
==============================================================================
                               Micherlitz Model
==============================================================================
 CurveFit Version 3.1.1                                  7/20/94   4:14 pm
==============================================================================

return code =    0
normal convergence

Number of cases      13

estimated residual variance       0.0100550

Parameters    Estimates  Std. err.  Est./s.e.  Prob.    Gradient
------------------------------------------------------------------
P01            1.480062   0.134828   10.977   0.0000    0.000005
P02            2.000000          .        .        .   -0.308611
P03            0.153671   0.033091    4.644   0.0000   -0.000010

Covariance matrix of parameters computed from
cross-product of first derivatives

Correlation matrix of the coefficients
   1.000       .   0.978
       .       .       .
   0.978       .   1.000

Number of iterations    9
Minutes to convergence     0.00367
```

The derivative for the fixed parameter is included so that it may be used to determine the reasonableness of the constraint. This value has the interpretation of a Lagrangian coefficient in a constrained model, that is, the size of the derivative is indicative of the reasonableness of the constraint.

2.2.6 Descent Methods

Given starting values for the coefficients, **CurveFit** seeks estimates through an iterative process which steadily improves them until convergence criteria have been satisfied. **CurveFit** employs two types of descent methods, the Levenberg-Marquardt variation of the Gauss-Newton method, and the Polak-Ribiere variation of the conjugate gradient.

Levenberg-Marquardt Descent

Let

$$G = \partial F/\partial\theta_0$$

be the $N \times k$ matrix of the first derivatives of the objective function evaluated at the initial values of the coefficients for each observation. Further define $Z = Y - h(\theta_0, X)$, the vector of the residuals, where Y is an N $\times$ L column vector containing the observations on the dependent variable, and $h(\theta_0, X)$ is an N $\times$ L column vector of predicted values.

An improved vector of the coefficients, in the sense that they decrease the objective function, is

$$\theta_1 = \theta_0 + \alpha\delta_\ell$$

where

$$\delta_\ell = -(G'G + (\epsilon + 1)diag(G'G))G'Z.$$

where ϵ is a constant initially set to .01. If, on each iteration, the function increases, ϵ is set to 10ϵ, otherwise to 0.4ϵ.

α is a scalar step factor found by "step-halving": first, α is set to 0.5. If

$$F(\theta_0 + \alpha\delta_\ell) < F(\theta_0) \tag{2.1}$$

the coefficients are updated:

$$\theta_1 = \theta_0 + \alpha\delta_\ell$$

and **CurveFit** proceeds to the next iteration. If not, α is set to $\alpha/2$, and a new evaluation of Equation 2.1 is made. This is continued until Equation 2.1 is satisfied.

Polak-Ribiere Conjugate Gradient Descent

The conjugate gradient descent method is quite slow to converge. However, it doesn't require the computation or storage of the Hessian matrix, which can be advantageous when the number of coefficients is very large.

On the first iteration, a steepest descent is used to the update of the starting values:

$$\theta_1 = \theta_0 - \alpha G$$

where α is a scalar step factor found as described above by step-halving. On subsequent iterations

$$\theta_{\ell+1} = \theta_\ell + \alpha\delta_\ell$$

where

$$\delta_\ell = -G_\ell + \frac{(G_\ell - G_{\ell-1})G_\ell}{G'_\ell G_\ell}\delta_{\ell-1}$$

2.2.7 Gradients

CurveFit finds coefficient estimates such that the gradient, G, is zero. If one or more coefficients have been fixed to their start values (see 2.2.5), their corresponding gradients will be nonzero.

By default **CurveFit** uses a forward difference numerical gradient. You may provide a function to compute the gradient. This will speed up the convergence because of the greater accuracy and the fewer number of function calls. The downside is that user-provided gradient functions are notorious for being difficult to debug.

If you decide to provide a gradient function, you must ensure that the function returns a $T \times K$ matrix, were T is the size of the matrix of independent variables passed to the procedure, and k is the number of coefficients, including any that are fixed to their start values

For example, suppose the function being fitted is

$$h(x, b_1, b_2, b_3) = b_1 + b_2 e^{-b_3 x}$$

The function and gradient procedure are

```
proc fct(b,x);
    retp(b[1] + b[2]*exp(-b[3]*x));
endp;
```

```
proc grd(b,x);
    local g, r;
    g = zeros(rows(x),rows(b));
    r = exp(-b[3]*x);
    g[.,1] = ones(rows(x),1);
    g[.,2] = r;
    g[.,3] = -b[2]*x .* r;
    retp(g);
endp;
```

Multiple Dependent Variables

When there are more than one dependent variable, the gradient procedure must return a T×K matrix of gradients for each dependent variable. Thus for L dependent variables, the gradient procedure returns L T×k matrixes concatenated horizontally.

2.2.8 Convergence Criteria

Relative Gradient

The primary method for convergence testing is the relative gradient:

$$G_{rel} = |G| \frac{|\theta|}{|F|}$$

The gradient is multiplied by the vector of current estimates of the coefficients divided by the current value of the objective function in order to remove the effects of scale.

G itself is also tested against machine epsilon. When the absolute gradients are less than machine precision, there isn't enough machine accuracy to tell when it has passed the relative gradient test, and therefore convergence is declared when $G < 1e-15$ whether or not the relative gradient test succeeds.

ROCC

A secondary test is available, due to Bates and Watts (1988, page 49), called the *relative offset convergence criterion* (ROCC). This criterion is available only when the entire data set fits in memory, and the matrices themselves are passed into **CurveFit**. This is necessary because the ROCC requires the calculation of the QR factorization of the gradient matrix, and that requires the availability of the entire data matrix.

For the ROCC, **CurveFit** first computes the QR factorization of the N × K matrix of gradients, i.e., the gradient computed for each observation. Let the Q matrix from that factorization be called Q_g. Compute

$$w = Q_g'(y - h(\theta, x))$$

and partition the vector w into two parts, $w_1 = w[1:K]$ and $w_2 = w[K+1:N]$. Then

$$ROCC = \frac{w_1' w_1/\sqrt{K}}{w_2' w_2/\sqrt{N-K}}.$$

Convergence is achieved when $ROCC < .001$

2.2.9 Calling CurveFit Recursively

The procedure that performs the least squares estimation may itself call **CurveFit**. This version of **CurveFit** nested inside the procedure is actually a separate copy of **CurveFit** with its own set of globals and must have its own nonlinear function (or otherwise you would have infinite recursion).

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

- Data sets can be opened by nested copies of **CurveFit**. If a nested copy of **CurveFit** is going to use the data set opened by the outer copy of **CurveFit**, 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 the dependent and independent variables in the second two arguments in the nested call to **CurveFit**.
- Before the call to the nested copy of **CurveFit**, the global variables should be reset by calling **CurveFitClr**. You must not use **CurveFitSet** because that will clear information about the data sets opened and processed in the outer copy. The only differences between **CurveFitSet** and **CurveFitClr** 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 **__cv_key** = 0 after the call to **CurveFitClr** and before the call to the nested **CurveFit**.

2.2.10 Using _CurveFit Directly

When **CurveFit** is called, it directly references all the necessary globals and passes its arguments and the values of the globals to a function called **__CurveFit**. When

_CurveFit returns, **CurveFit** then sets the output globals to the values returned by **_CurveFit** and returns its output arguments directly to the user. **_CurveFit** makes no global references to matrices or strings.

_CurveFit can be used directly in situations where you do not want any of the global matrices and strings in your program. If **CurveFit**, **CurveFitPrt**, **CurveFitSet**, and **CurveFitClr** are not referenced, the global matrices and strings in cvfit.dec will not be included in your program.

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

2.3 Inference

Statistical inference in nonlinear least squares models is hampered by the fact that the standard error of the estimates may poorly describe their distributions. This is because the asymptotic normality of the estimates of nonlinear models is not generally established. Thus for a particular nonlinear model the distributions of the coefficients may not have the advantage of being described uniquely by the first two moments.

CurveFit has a variety of ways of describing the distributions of the coefficients. First, the usual covariance matrix of the coefficients based on the information matrix may be computed. There are two variations on this that may be selected, a covariance matrix based on the Hessian, or the heteroskedastic-consistent covariance matrix (White, 1980).

When the distributions of the coefficients fail to be asymptotically Normal, the standard error is not an adequate description, and moreover its associated t-statistic can be quite misleading. To diagnose this situation, **CurveFit** provides a special function, **CurveProfile**, for computing *profile t plots* which are plots of a type of t-statistic across values of the coefficients. The closer this trace is to a straight line, the more reasonable will be the usual inference based on t-statistics.

Just as the standard errors might be misleading in nonlinear models, so also might be the correlations of the coefficients computed in the usual way from the covariance matrix of the coefficients. To provide a description of the "correlatedness" of the coefficients, *likelihood profile traces* are plotted along with the profile t plots.

Bootstrapping is a third method for assessing the distribution of the estimated coefficients. Using a "model-free" method of resampling, a user-controlled number of estimated coefficients is generated. Two procedures are provided in **CurveFit**: **CurveBoot** which returns the mean and variance-covariance matrix of the resampled coefficients, and **CurveHist** which plots univariate histograms and bivariate surface plots of the distributions of the coefficients.

2.3.1 Covariance Matrix of Coefficients

In **CurveFit** you may select from three methods for the calculation of the covariance matrix of the coefficients, (a) the information matrix, (b) the Hessian matrix, or (c) the heteroskedastic-consistent method which uses both the information matrix and the Hessian.

Information Matrix Method

Provided that the residuals are approximately Gaussian distributed, i.e., $Z \sim N(0, \sigma^2 I_N)$, we have for large N

$$\hat{\theta} - \theta* \sim N_K[0, \sigma^2 (G'G)^{-1}]$$

where

$$G = \partial F / \partial \theta$$

is the $N \times k$ matrix of first derivatives for each observation (Jennrich, 1969, Gallant, 1987, Seber and Wild, 1989, page 24).

An estimate of σ^2 is

$$\hat{\sigma}^2 = F/(N - K)$$

CurveFit will return $\hat{\sigma}^2 (G'G)^{-1}$ when the **CurveFit** global variable, **_cv_CovMatrix** is set to 1. This is the default value of **_cv_CovMatrix**.

Hessian Method

When the model is correctly specified, the Hessian matrix is also a consistent estimate of the covariance matrix of the coefficients. When **_cv_CovMatrix** is set to 2,

$$\hat{\sigma}^2 (\partial^2 F / \partial\theta \partial\theta')^{-1}$$

is returned by **CurveFit** in the fourth argument. The Hessian is computed numerically.

Heteroskedastic-Consistent Method

Both the inverse of the information matrix and the inverse of the Hessian are consistent and unbiased estimates of the covariance matrix of the coefficients when the model is correctly specified. Under misspecification, however, neither of them are consistent or

unbiased. The heteroskedastic-consistent method uses both the first and second derivatives to produce a covariance matrix of the coefficients which is consistent under misspecification (White, 1980).

Define the $k \times k$ Hessian,

$$H = \partial^2 F / \partial\theta \partial\theta',$$

the $N \times k$ matrix of first derivatives,

$$G = \partial F / \partial\theta,$$

and the $N \times N$ matrix, Σ, with diagonal elements

$$\sigma_{ii}^2 = (y_i - h(\theta, x_i))^2.$$

Then the heteroskedastic-consistent covariance matrix of the parameters is

$$H^{-1} G' \Sigma G H^{-1}$$

This matrix is returned when **_cv_CovMatrix** is set to 3.

When Inversion Fails

When not enough information is available for one or more coefficients in the model to be properly estimated, the matrix used to compute the covariance matrix of the coefficients will fail to invert. This situation is analogous to multicollinearity in linear regression. When this happens it may be useful to look at the matrices which **CurveFit** attempted to invert in order to diagnose the problem.

When **_cv_CovMatrix** = 1, $G'G$ is stored in the **CurveFit** global variable **_cv_InfoMatrix**. When **_cv_CovMatrix** = 2, H is stored in **_cv_HessMatrix**, and when **_cv_CovMatrix** = 3, $G'\Sigma G$ is stored in **_cv_InfoMatrix** and H in **_cv_HessMatrix**.

The problem may be immediately evident upon inspection of these matrices. For example, if a single parameter is the source of the problem, the entries in the matrix corresponding to that parameter may be very small.

If it appears to be more complex, the following procedure may be used to diagnose the problem: first, produce the *pivoted* QR factorization of the matrix. There will be $K - r$ zeros at the end of the diagonal of R, where r is the rank of the matrix and K the order.

The failure to invert implies a linear dependency in the matrix to be inverted. These dependencies can be derived from the factorization. Compute

$$B = R_{12}' R_{11}^{-1}$$

where R is partitioned

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}$$

and R_{11} is $r \times r$ and R_{12} is $r \times (K - r)$.

Partition the pivot vector,

$$P = \begin{bmatrix} P_1 \\ P_2 \end{bmatrix}$$

where P_1 is $r \times 1$ rows, and P_2 is $K - r$ rows. B gives the linear dependencies of the rows in P_1 of the original matrix being inverted as a function of the rows in P_2. The rows of the matrix being inverted, whether $G'G$ or H, are associated directly with the coefficients in the model, the first row of $G'G$ is associated with the first coefficient, the second row with the second coefficient, and so on.

The matrix B may suggest ways of refining the model to eliminate the problem with the model. If the second coefficient is linearly dependent on the first and third coefficients, it might be constrained to be equal to one or the other, or to some function of the two, or to a constant.

2.4 Bootstrap

The bootstrap method is used to generate empirical distributions of the coefficients, thus avoiding the problems with the methods of statistical inference described above. The resampling method used in **CurveFit** is a *model-free* method. The N observations are randomly sampled with replacement N_r times, and coefficients are estimated for each of these samples. N_r is set by the **CurveFit** global variable, **_cv_NumSample**.

2.4.1 CurveBoot

CurveBoot is a procedure in the **CurveFit** module which returns the mean and variance-covariance matrix of the bootstrapped coefficients. **CurveBoot** generates **_cv_NumSample** random samples of size **_cv_NumObs** from the data set with replacement and calls **CurveFit**. **CurveBoot** returns the mean vector and covariance matrix of the estimates.

Example

To bootstrap the example in Section 2.2, the only necessary alteration is the change of the call to **CurveFit** to a call to **CurveBoot**:

```
    call CurveFitPrt(CurveBoot("",y,x,&Micherlitz,b0));
```

The results are

```
================================================================================
                          bootstrapped Micherlitz Model
================================================================================
 CurveFit Version 3.1.1                                          8/01/94  10:16 am
================================================================================

return code =    0
normal convergence

Number of cases      13

estimated residual variance       0.0349228

Parameters  Estimates   Std. err.  Est./s.e.  Prob.    Gradient
-------------------------------------------------------------
P01          0.988809    0.173910    5.686   0.0000    0.000000
P02          2.513743    0.139890   17.969   0.0000    0.000000
P03          0.107700    0.014842    7.256   0.0000   -0.000000

Covariance matrix of parameters computed from
cross-product of first derivatives

Correlation matrix of the coefficients
   1.000  -0.974   0.970
  -0.974   1.000  -0.901
   0.970  -0.901   1.000
```

2.4.2 Coefficient Distribution Histogram

If the distribution of the coefficients is highly nonlinear, the mean and variance-covariance matrix of the coefficients may not be satisfactory for inference. The profile t plots and likelihood profile traces would provide evidence for this, and if the coefficient distributions appear to be problematic, then simply reporting the variance-covariance matrix of the coefficients would not be sufficient. For this reason a procedure, **CurveHist**, is provided in the **CurveFit** module for visually displaying the empirical distributions of the coefficients. **CurveHist** produces univariate histograms as well as bivariate surface plots of each of the bootstrapped coefficients in pairs. The tables used to generate the histograms and surface plots are stored in **_cv_CrossTab**. Each column of the matrix stored in **_cv_CrossTab** contains a *vec*-ed **_cv_NumCat** $\times$ **_cv_NumCat** matrix tabulating the distribution of the i^{th} coefficient against the j^{th}

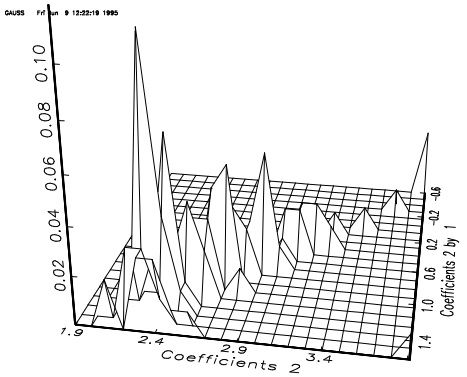
coefficient, where the order of the columns is determined by
$(i, j) \Rightarrow (2,1)(3,1)(3,2)(4,1)\cdots$, for columns 1, 2, 3, 4, 5, 6,...and so on. The cutting points for these tables are stored in **_cv_CutPoint**.

Example

To generate univariate histograms and bivariate surface plots from the bootstrapped example in Section 2.4.1, it is necessary only to change the call to **CurveBoot** to a call to **CurveHist**:

```
call CurveFitPrt(CurveHist("",y,x,&Micherlitz,b0));
```

The identical summary statistics are returned by the procedure which may be printed out using **CurveFitPrt**. In addition, a series of plots of histograms are generated for each combination of pairs of parameters. Each plot contains two univariate histograms and a bivariate surface plot. In this example there are three coefficients and thus three plots are produced of coefficient 1 against 2, 1 against 3, and 2 against 3. The following is the first plot of the histograms and surface plot for coefficients 1 and 2:



2.5 Diagnosing Coefficient Distribution

The **CurveFit** proc, **CurveProfile** generates profile t plots as well as plots of the likelihood profile traces for all of the coefficients in the model in pairs. The profile t plots are used to assess the nonlinearity of the distributions of the individual coefficients, and the likelihood profile traces are used to assess the bivariate distributions.

The input and output arguments to **CurveProfile** are identical to those of **CurveFit**. But in addition to providing the least squares estimates and covariance matrix of the coefficients, a series of plots are printed to the screen using **GAUSS**' Publication Quality Graphics. A screen is printed for each possible pair of coefficients. There are three plots, a profile t plot for each parameter, and a third plot containing the likelihood profile traces for the two coefficients.

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.

2.5.1 The Profile t Plot

Define

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

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

$$\tau(\theta_k) = sign(\theta_k - \hat{\theta}_k)(N-K)\sqrt{F(\tilde{\theta}_k)/F(\hat{\theta}_k) - 1}$$

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

2.5.2 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 coefficient estimates affect each other and how the projection of the likelihood contours onto the (θ_k, θ_ℓ) plane might look. For the (θ_k, θ_ℓ) plot, two lines are plotted, $F(\tilde{\theta}_k)$ against θ_k and $F(\tilde{\theta}_\ell)$ against θ_ℓ.

If the likelihood surface contours are long and thin, indicating the coefficients to be collinear, the profile traces will be close together. If the contours are fat, indicating the coefficients to be more uncorrelated, the profile traces will tend to be perpendicular. And if the contours are nearly elliptical, the profile traces will be 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.

2.5.3 Example

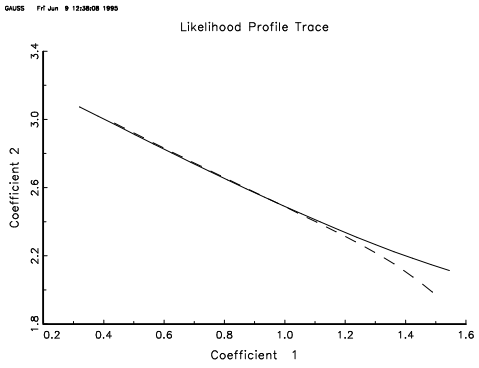
To generate profile t plots and likelihood profile traces from the example in Section 2.2, it is necessary only to change the call to **CurveFit** to a call to **CurveProfile**:

```
call CurveFitPrt(CurveProfile("",y,x,&Micherlitz,b0));
```

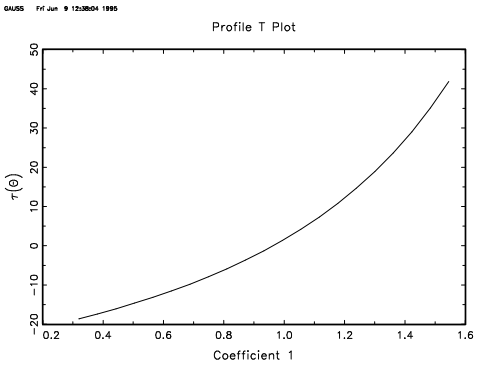
CurveProfile produces the same output as **CurveFit** which can be printed out using a call to **CurveFitPrt**.

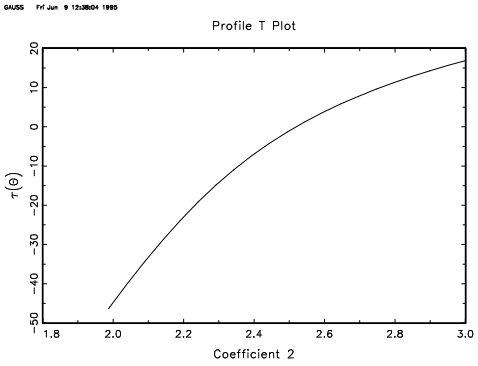
As illustrated in the Figures below, for each pair of coefficients a plot is generated containing an xy plot of the likelihood profile traces of the two coefficients, and two profile t plots, one for each coefficient.

The likelihood profile traces in the following Figure indicate that the distributions of coefficients 1 and 2 are highly correlated. Ideally, the traces would be perpendicular and the trace in this example is far from ideal.



The profile t plots in the following Figures indicate that the coefficient distributions are somewhat nonlinear. Ideally the profile t plots would be straight lines and this example exhibits noticeable nonlinearity. It is clear that any interpretations of the coefficients of this model must be made quite carefully.





2.6 References

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

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

Clarke, G.P.Y. "Approximate Confidence Limits for a Parameter Function in Nonlinear Regression," *Journal of the American Statistical Association*, 82:221:230.

Efron, Bradley. *The Jackknife, the Bootstrap, and Other Resampling Plans*, Philadelphia: SIAM, 1982.

Jennrich, R.I. "Asymptotic Properties of Non-Linear Least Squares Estimators," *The Annals of Mathematical Statistics*, 40:633-643, 1969.

Judge, George C., R. Carter Hill, William E. Griffiths, Helmut Lütkepohl, and Tsoung-Chao Lee. *Introduction to the Theory and Practice of Econometrics. Second Edition*, New York: Wiley, 1988.

Gallant, A.R. *Nonlinear Statistical Models*, New York: Wiley, 1987.

Nash, J.C. *Compact Numerical Methods for Computers, Second Edition*, New York: Adam Hilger, 1990.

Seber, G.A.F, and C.J. Wild. *Nonlinear Regression*, New York: Wiley, 1988.

White, H., "Consequences and Detection of Misspecified Nonlinear Regression Models," *Journal of the American Statistical Association*, 76:419-433, 1981.

Chapter 3

CurveFit Reference

Library

cvfit

Purpose

Generates bootstrapped estimates of a nonlinear regression model

Format

{ *b*,*f*,*g*,*cov*,*retcode* } = **CurveBoot(***dataset*,*depv*,*indv*,&*fct*,*start***)**

Input

dataset	either string containing name of **GAUSS** data set, or null string.
depv	either L × 1 vector of labels of dependent variables, or, if dataset $== "", N×L matrix of dependent variables
indv	either P × 1 vector of labels of independent variables, or, if dataset $== "", N×P matrix of independent variables
fct	the name of a procedure that returns predicted values for a matrix of observations
start	either a K×1 vector of start values, or a procedure name (not a pointer) that returns a K×1 vector of start values

Output

b	K×1 mean vector of re-sampled coefficients
f	L×L matrix, mean re-sampled residual covariance matrix
g	K×1 mean vector of re-sampled gradients
cov	K×K covariance matrix of re-sampled coefficients
retcode	return code:

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

6 step length calculation failed

7 function cannot be evaluated at initial coefficient values

8 number of elements in the gradient vector inconsistent with number of starting values

9 gradient function returned a column vector rather than the required row vector

10 rows(**_cv_Active**) $\neq$ 1 and $\neq$ rows(**start**)

11 maximum time exceeded

12 weights not found

34 data set could not be opened

99 termination condition unknown

Globals

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

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

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

Remarks

CurveBoot generates **_cv_NumSample** random samples of size **_cv_NumObs** from the data set with replacement and calls **CurveFit**. **CurveBoot** returns the mean vector of the estimates in the first argument and the covariance matrix of the estimates in the third argument.

Source

`cvboot.src`

Library

cvfit

Purpose

Computes coefficient estimates of nonlinear regression models

Format

{ *b*,*f*,*g*,*cov*,*retcode* } = **CurveFit**(*dataset*,*depv*,*indv*,&*fct*,*start*)

Input

dataset	either string containing name of **GAUSS** data set, or null string.
depv	either L × 1 vector of labels of dependent variables, or, if dataset $== "", N×L matrix of dependent variables
indv	either P × 1 vector of labels of independent variables, or, if dataset $== "", N×P matrix of independent variables
fct	the name of a procedure that returns predicted values for a matrix of observations
start	either a K×1 vector of start values, or a procedure name (not a pointer) that returns a K×1 vector of start values

Output

b	K×1 vector of coefficients
f	L×L matrix, residual covariance matrix
g	K×1 vector, gradient
cov	K×K covariance matrix of coefficients
retcode	return code:

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

6 step length calculation failed

7 function cannot be evaluated at initial coefficient values

8 number of elements in the gradient vector inconsistent with number of starting values

9 gradient function returned a column vector rather than the required row vector

10 rows(**_cv_Active**) $\neq 1$ and $\neq$ rows(**start**)

11 maximum time exceeded

12 weights not found

34 data set could not be opened

99 termination condition unknown

Globals

_cv_Active scalar or K×1 vector, determines active coefficients. If all coefficients are active set to scalar 1 (default), otherwise set to a K×1 vector of zeros and ones, with zeros were the corresponding element of the coefficient vector is to be fixed to its starting value.

_cv_CovMatrix scalar. If 0, covariance matrix of coefficients is not computed, otherwise if 1, covariance matrix computed from cross-product of first derivatives, if 2, computed from the inverse of the Hessian, and if 3, the heteroskedastic- consistent covariance matrix of the coefficients is computed. Default = 1.

_cv_Criterion scalar. If 0, convergence is determined by testing the relative gradient against **_cv_RelGradTol**. If 1, it is determined by testing the ROCC against **_cv_ROCC**. If 2, convergence is declare when either criterion is satisfied, and if 3, when both are.

_cv_DescMethod scalar, descent method, if nonzero Levenberg-Marquardt method, otherwise conjugate gradient. Conjugate gradient method is best for problems with large numbers of coefficients because it doesn't generate a Hessian matrix.

_cv_GradInc increment size for computing gradient.

_cv_GradMethod scalar, method for computing numerical gradient. = 1, forward difference (default) = else, central difference

_cv_GradProc scalar, pointer to a procedure that computes the gradient of the function with respect to the coefficients. For example, the instruction:

```
_cv_GradProc=&gradproc
```

will tell **CurveFit** that a gradient procedure exists as well as where to find it. The user-provided procedure has a two input arguments, a vector of coefficient values and a matrix of indepencent variables, and a single output argument, a matrix of gradients of the function with respect to the coefficients evaluated at the vector of coefficient values for each observation. For example, suppose the procedure is named gradproc and the function is a quadratic with two coefficients: $y = b_1x^2 + 2b_2x + 1$, then

```
proc gradproc(b,x);
    retp(x^2~2*x);
endp;
```

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

_cv_IterInfo scalar, if nonzero, print iteration information. Can be toggled on and off by pressing P or p on the keyboard.

_cv_Key scalar, controls keyboard capture. Useful for recursively nested version of **CurveFit**. Setting **_cv_Key** = 0 for the nested versions will turn off their key board captures permitting the outside version to retain control of the keyboard.

CurveFit has two actions controlled by keypresses. Pressing C or c will force convergence, and pressing P or p will toggle printing iteration information to the screen.

_cvMaxIter scalar, maximum number of iterations.

_cv_MaxTime scalar, maximum time in iterations in minutes. Default = 1e5, about 10 weeks.

_cv_MaxTry scalar, maximum number of step-half attempts.

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

_cv_ParName K×1 character vector, coefficient labels.

_cv_RelGradTol scalar, convergence tolerance for gradient of estimated coefficients. Default = 1e-5. When this criterion has been satisifed **CurveFit** will exit the iterations.

_cv_ROCC scalar, if nonzero, use the Relative Offset Convergence Criterion as described in Bates and Watts, *Nonlinear Regression Analysis and Its Applications*, page 49.

__row determines the number of rows in the data set to be passed to the user-provided procedures. Default = 0 (number of rows will be computed by **CurveFit**.

__weight either N×1 vector of frequencies, or name of variable in **GAUSS** data set, or column number of variable in **GAUSS** data set. NOTE: frequencies must sum to N, the sample size. See Section 2.2.4 for discussion of the use of this global for weighting observations.

Global Output

_cv_Coefficent K×1 vector, the current estimates of the estimated coefficients will be stored in **_cv_Coefficents**. If **CurveFit** terminates abnormally then the current estimates can be retrieved from this global variable.

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

_cv_HessMatrix K×K matrix, the Hessian used to compute the covariance matrix of the coefficients when **_cv_CovMatrix** equals 2 or 3. See Section 2.3.1 for discussion about the use of this global.

_cv_InfoMatrix K×K matrix, the cross-product of the first derivatives used to compute the covariance matrix of the coefficients when **_cv_CovMatrix** equals 1 or 2. See Section 2.3.1 for discussion about the use of this global.

_cv_IterData 2×1 vector, contains information about the iterations. The first element contains the elapsed time in minutes of the iterations, the second element contains the number of iterations.

Remarks

There are three keyboard toggles:

C	forces convergence
D	toggles between descent methods
P	toggles printing the iteration information to the screen on or off

These are case insensitive. The printing toggle is useful in determining the progress of the iterations. The default is not to print anything to the screen, speeding up computations. However, to reassure yourself that the computations are actually going on, you may press P to check the progress, and then re-press P to turn printing back off to restore the speed of the calculations.

Source

`cvfit.src`

Library

`cvfit`

Purpose

Computes bootstrapped estimates and presents the results in a histogram and surface plot

Format

`{` *b*,*f*,*g*,*cov*,*retcode* `}` = **CurveHist(***dataset*,*depv*,*indv*,**&***fct*,*start***)**

Input

dataset	either string containing name of **GAUSS** data set, or null string.
depv	either L × 1 vector of labels of dependent variables, or, if dataset $== "", N×L matrix of dependent variables
indv	either P × 1 vector of labels of independent variables, or, if dataset $== "", N×P matrix of independent variables
fct	the name of a procedure that returns predicted values for a matrix of observations
start	either a K×1 vector of start values, or a procedure name (not a pointer) that returns a K×1 vector of start values

Output

b	K×1 mean vector of re-sampled coefficients
f	L×L matrix, mean re-sampled residual covariance matrix
g	K×1 mean vector of re-sampled gradients
cov	K×K covariance matrix of re-sampled coefficients
retcode	return code:

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

6 step length calculation failed

7 function cannot be evaluated at initial coefficient values

8 number of elements in the gradient vector inconsistent with number of starting values

9 gradient function returned a column vector rather than the required row vector

10 rows(**_cv_Active**) $\neq 1$ and $\neq$ rows(**start**)

11 maximum time exceeded

12 weights not found

34 data set could not be opened

99 termination condition unknown

Globals

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

_cv_NumSample scalar, number of samples to be drawn from data set. Default = 100.

_cv_NumCat scalar, number of categories for cross-tabulation. Default = 10.

_cv_Increment K × 1 vector, category increments for each histogram. If scalar zero, increments are computed by **CurveHist**.

_cv_Center K × 1 vector, center points for each histogram. If scalar zero, **CurveHist** computes them.

_cv_Width scalar, width of histogram is this global times standard deviations of coefficients. Default = 2.

Global Output

_cv_CutPoint **_cv_NumCat** × K matrix of category cutting points for crosstabulation of the K coefficients.

_cv_CrossTab **_cv_NumCat** * **_cv_NumCat** by $K(K-1)/2$ matrix containing crosstabulations of the re-sampled coefficients.

Remarks

The tables used to generate the histograms and surface plots are stored in **_cv_CrossTab**. Each column of the matrix stored in **_cv_CrossTab** contains a *vec*-ed **_cv_NumCat** × **_cv_NumCat** matrix tabulating the distribution of the i^{th} coefficient against the j^{th} coefficient, where the order of the columns is determined by $(i, j) \Rightarrow (2,1)(3,1)(3,2)(4,1)(4,2)(4,3)\cdots$, for columns 1, 2, 3, 4, 5, 6,...and so on. The cutting points for these tables are stored in **_cv_CutPoint**.

Source

`cvhist.src`

Library

cvfit

Purpose

Computes profile t plots and likelihood profile traces

Format

{ *b*,*f*,*g*,*cov*,*retcode* } = **CurveProfile(***dataset*,*depv*,*indv*,&*fct*,*start***)**

Input

dataset	either string containing name of **GAUSS** data set, or null string.
depv	either L × 1 vector of labels of dependent variables, or, if dataset $== "", N×L matrix of dependent variables
indv	either P × 1 vector of labels of independent variables, or, if dataset $== "", N×P matrix of independent variables
fct	the name of a procedure that returns predicted values for a matrix of observations
start	either a K×1 vector of start values, or a procedure name (not a pointer) that returns a K×1 vector of start values

Output

b	K×1 vector of coefficients
f	L×L matrix, residual covariance matrix
g	K×1 vector, gradient
cov	K×K covariance matrix of coefficients
retcode	return code:

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

6 step length calculation failed

7 function cannot be evaluated at initial coefficient values

8 number of elements in the gradient vector inconsistent with number of starting values

9 gradient function returned a column vector rather than the required row vector

10 rows(**_cv_Active**) $\neq$ 1 and $\neq$ rows(**start**)

11 maximum time exceeded

12 weights not found

34 data set could not be opened

35 no observations left after transformation and selection

99 termination condition unknown

Globals

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

_cv_NumCat scalar, number of points for plots. Default = 10.

_cv_Increment K $\times$ 1 vector, increments for each plot. If scalar zero, increments are computed by **CurveProfile**.

_cv_Center K $\times$ 1 vector, center points for each plot. If scalar zero, **CurveProfile** computes them.

_cv_Width scalar, width of plot is this global times standard deviations of coefficients. Default = 2.

Source

`cvprof.src`

Library

`cvfit`

Purpose

Initializes **CurveFit** global variables to default values.

Format

CurveFitSet

Input

None

Output

None

Remarks

Putting this instruction at the top of all programs that invoke **CurveFit** is generally good practice. This will prevent globals from being inappropriately defined when a program is run either several times or after another program that also calls **CurveFit**.

CurveFitSet calls **GAUSSET**.

Source

`cvfit.src`

Library

`cvfit`

Purpose

Initializes **CurveFit** global variables to default values.

Format

CurveFitClr

Input

None

Output

None

Remarks

CurveFitClr is used to reset global variables to default values for "nested" versions of **CurveFit**, i.e., versions called by function procedures of "outer" versions of **CurveFit**. **CurveFitClr** is identical to **CurveFitSet** except that certain global variables used to pass information from outer versions of **CurveFit** to the nested versions are not reset.

CurveFitClr calls **GAUSSET**.

Source

`cvfit.src`

Purpose

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

Library

`cvfit`

Format

`{` *b*,*f*,*g*,*cov*,*retcode* `}` = **CurveFitPrt(** *b*,*f*,*g*,*cov*,*retcode* **)**

Input

b	K×1 vector of coefficients
f	L×L matrix, residual covariance matrix
g	K×1 vector, gradient
cov	K×K covariance matrix of coefficients
retcode	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	Example:
"v"	print version number of program	
"f"	print file name being analyzed	

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

Default = "tldvf".

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

Remarks

The call to **CurveFit** can be nested in the call to **CurveFitPrt**:

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

Source

`cvfit.src`

Index