Nonlinear Equations MT 1.0

for GAUSS™ Mathematical and Statistical System

Aptech Systems, Inc.

Part Number: `004437`
Version 1.0
Documentation Revision: 1882 June 7, 2010

Contents

Installation 1

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_vernum*.*revnum*_`UNIX.tar.gz`

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

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

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

   ```
   tar xvf /tmp/app_appname_vernum.revnum_UNIX.tar
   ```
   – or –
   ```
   unzip /tmp/app_appname_vernum.revnum_UNIX.zip
   ```

1.1.2 CD

1. Insert the 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 2

2.1 Getting Started

GAUSS 6.0.26+ is required to use these routines. See **_rtl_ver** in `src/gauss.dec`.

2.1.1 Setup

In order to use the procedures in the **Nonlinear Equations MT** module, the **NLEMT** library must be active. This is done by including **nlsysmt** in the **library** statement at the top of your program or command file:

```
library nlsysmt,simplex,pgraph;
```

This enables **GAUSS** to find the **NLEMT** procedures. An **`nlControl`** structure is used to hold control variables that are used by the procedures in this module. To define an instance of this structure and set its members to default values, include the following instructions just after the **`library`** statement at the top of your program:

```
#include nlsysmt.sdf
struct nlControl nlc;
nlc = nlControlCreate;
```

The version number of each module is stored in a global variable. For **Nonlinear Equations MT**, this global is:

`_nlmt_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 this module.

2.1.2 README Files

If it exists, the file `README.nlmt` contains any last minute information on the **Nonlinear Equations MT** procedures. Please read it before using them.

3 Nonlinear Equations MT

3.1 Introduction

This module contains the procedure **nlsys** which solves the system:

$$F(x) = 0$$

where F is a general nonlinear system of equations, $F : R^n \to R^n$. F must have first and second derivatives, although these need not be supplied analytically.

3.2 About the nlsys Procedure

You can call **nlsys**, with the following statement:

```
            { nlc,nlo } = nlsys(nlc,&f,x0);
```

where **`nlc`** is an instance of an **`nlControl`** structure created by **`nlControlCreate`**, *&f* is a pointer to the procedure describing the system of equations (a discussion on how to set up this procedure follows), *x0* is a vector of start values, and *nlo* is an instance of an **`nlOut`** structure. The **`nlOut`** structure contains the final solution, the value of the system of equations at the final solution, the final Jacobian, the number of iterations needed to find the final solution, and a return code.

It is assumed that the function passed to **`nlsys`** is continuous and differentiable. This ensures that the matrix of first partial derivatives of the equations, the Jacobian matrix, exists, even though it may not be possible to calculate it analytically. The Jacobian matrix may be defined as

$$J_{i,j} = \frac{\partial F_i}{\partial x_j}$$

The F_i's must be independent; otherwise, the system will be underdetermined, the equation $F = 0$ will have an infinite number of solutions and the method will fail. Independence of the F_i's, however, is not a sufficient condition to ensure that the solution found by **`nlsys`** will be unique; for many problems, a number of solutions exist. For example, $F(x)$ could be the two equation system

$$\begin{aligned} x_1 + x_2 - 3 &= 0 \\ {x_1}^2 + {x_2}^2 - 9 &= 0 \end{aligned}$$

which has roots at $x_1 = 3$, $x_2 = 0$ and $x_1 = 0$, $x_2 = 3$.

In this case, the particular solution located by **`nlsys`** generally will be the solution closest to the starting values x_0.

Specific tolerance levels may be set to define the accuracy of your equations and, therefore, the accuracy of the solution.

3.3 Solution Method

The program **`nlsys`** uses a quasi-Newton method for finding the zeros of a system of nonlinear equations, if an analytic Jacobian of the system is not supplied by the user. To approximate the Jacobian, you may chose one of two methods: (1) Broyden's secant update of the approximation of the Jacobian, or (2) a forward difference method. Should Broyden's technique fail, the algorithm reverts to the forward difference estimate of the Jacobian and recalculates the step. You may also supply a function to compute the Jacobian analytically which speeds up the solution significantly, especially for large problems.

To provide a globalizing strategy for failures of Newton steps, the user may chose one of two algorithms. The first, a line-search algorithm, uses a backtracking strategy to search in the Newton direction for a step-length which minimizes the local quadratic model of the system. The second, the hookstep algorithm, uses a predetermined step-length, which is at most the Newton step-length, and searches for a direction which minimizes the local quadratic model. Refer to Dennis and Schnabel for a complete discussion of these methods.

The **`chpf`**, **`algr`**, **`stjc`**, and **`ajac`** members of the **`nlControl`** structure are used to specify which method should be used to calculate the Jacobian, a starting Jacobian, if one is desired, and which search strategy should be used.

To initialize the process, the user must supply starting values, and optionally, a starting Jacobian.

3.4 An Example

To illustrate the use of **`nlsys`**, we will solve Example 5.5 of Carnahan *et.al.* in the following sections. The completed program is given in the example file *nlmt4.e*.

3.4.1 Setting Up the System

A **GAUSS** procedure which defines the system of equations F must be defined. There are no special name requirements for this procedure and and functions called through the language interface may be used to define the equations. The procedure must however, accept only the Nx1 vector $x = (x_1, x_2, ...x_n)$ as an argument, and return the Nx1 vector representing $F(x) = (f_1, f_2, f_3...f_n)$.

The system of equations

$$\begin{aligned}
\frac{1}{2}x_1 + x_2 + \frac{1}{2}x_3 - \frac{x_6}{x_7} &= 0 \\
x_3 + x_4 + 2x_5 - \frac{2}{x_7} &= 0 \\
x_1 + x_2 + x_5 - \frac{1}{x_7} &= 0 \\
-28837x_1 - 139009x_2 - 78213x_3 + 18927x_4 + & \\
+8427x_5 + \frac{13492}{x_7} - 10690\frac{x_6}{x_7} &= 0 \\
x_1 + x_2 + x_3 + x_4 + x_5 - 1 &= 0 \\
P^2 x_1 x_4^3 - 1.7837 \times 10^5 x_3 x_5 &= 0 \\
x_1 x_3 - 2.6058 x_2 x_4 &= 0
\end{aligned}$$

which is given by Carnahan *et.al.* represent a methane–oxygen reaction. The task is to solve for the variables $x_* = (x_1 ... x_n)$.

Here is one method for placing this system in a procedure:

```
proc fsys(x);
    local f1,f2,f3,f4,f5,f6,f7,P;
    P = 20;
    f1 = 0.5*x[1] + x[2] + 0.5*x[3] - x[6]/x[7];
    f2 = x[3] + x[4] + 2*x[5] - 2/x[7];
    f3 = x[1] + x[2] + x[5] - 1/x[7];
    f4 =  -28837*x[1] - 139009*x[2] - 78213*x3 + 18927*x[4] +
                 8427*x[5] + 13492/x[7] - 10690*x[6]/x[7];
    f5 = x[1] + x[2] + x[3] + x[4] + x[5] - 1;
    f6 = (P^2)*x[1]*x[4]^3 - 1.7837*1e5*x[3]*x[5];
    f7 = x[1]*x[3] - 2.6058*x[2]*x[4];
    retp(f1|f2|f3|f4|f5|f6|f7);
endp;
```

In this case, one would pass a pointer to the procedure **fsys** to **nlsys**.

3.4.2 Starting Values

Starting values for x_0 are required. These values should be chosen to be as close as possible to the solution. This should be a column vector.

```
x0 = { 0.5, 0, 0, 0.5, 0, 0.5, 2.0 };
```

3.4.3 Control Variables

The members of the **nlControl** structure are control variables which may be specified to control various options, including the convergence, scaling and printed output. These can be specified as follows:

```
struct nlControl nlc;
nlc = nlControlCreate;
nlc.algr = 2;
nlc.altnam = "CO"$|"CO2"$|"H2O"$|"H2"$|"CH4"$|"O2/CH4"$|"TOTAL";
nlc.title = "Chemical Equilibrium Problem";
```

See the **nlsys** function definition in Chapter 4 for a complete listing of these options.

3.4.4 The Complete Example

Here is a complete example illustrating the use of **nlsys** and the printing procedure **nlprt**.

```
library nlsysmt;
#include nlsysmt.sdf
struct nlControl nlc;
struct nlOut nlo;
nlc = nlControlCreate;

proc fsys(x);
    local f1,f2,f3,f4,f5,f6,f7,P;
    P = 20;
    f1 = 0.5*x[1] + x[2] + 0.5*x[3] - x[6]/x[7];
    f2 = x[3] + x[4] + 2*x[5] - 2/x[7];
    f3 = x[1] + x[2] + x[5] - 1/x[7];
    f4 =  -28837*x[1] - 139009*x[2] - 78213*x3 + 18927*x[4] +
                 8427*x[5] + 13492/x[7] - 10690*x[6]/x[7];
    f5 = x[1] + x[2] + x[3] + x[4] + x[5] - 1;
    f6 = (P^2)*x[1]*x[4]^3 - 1.7837*1e5*x[3]*x[5];
    f7 = x[1]*x[3] - 2.6058*x[2]*x[4];
    retp(f1|f2|f3|f4|f5|f6|f7);
endp;

x0 = { 0.5, 0, 0, 0.5, 0, 0.5, 2.0 };
nlc.algr = 2;
nlc.altnam = "CO"$|"CO2"$|"H2O"$|"H2"$|"CH4"$|"O2/CH4"$|"TOTAL";
nlc.title = "Chemical Equilibrium Problem";
output file = nlmt3.out reset;
{ nlc,nlo } = nlprt(nlsys(nlc,&fsys,x0));
output off;
```

The procedure **nlprt** nested around the call to **nlsys** prints the results to the current output device: in this case, the screen and the output file *nlmt4.out*. The alternative names set in *nlc.altnam* will be used to label the variables.

Here is the output produced by **nlprt**:

```
----------------------------------------------------------------------------
           ROOTS                                       F(ROOTS)
----------------------------------------------------------------------------

Iteration #1
CO                    0.22101731           F1(X)             0.00098702
CO2                   0.02592764           F2(X)            -0.00015119
H2O                   0.06756228           F3(X)            -0.00007557
H2                    0.42632756           F4(X)            11.57115067
CH4                   0.25916519           F5(X)            -0.00000001
O2/CH4                0.33432477           F6(X)         -3116.37106748
TOTAL                 1.97555953           F7(X)            -0.01387121


Iteration #2
CO                    0.31014830           F1(X)             0.00449899
CO2                   0.00714194           F2(X)            -0.06125687
H2O                   0.05538274           F3(X)            -0.03062845
H2                    0.57919778           F4(X)           461.33304026
CH4                   0.04812924           F5(X)            -0.00000000
O2/CH4                0.46814654           F6(X)          -451.34526198
TOTAL                 2.52494691           F7(X)             0.00639772


Iteration #3
CO                    0.32028481           F1(X)             0.00094304
CO2                   0.00955481           F2(X)            -0.01374807
```

```
H2O                   0.04671279             F3(X)          -0.00687404
H2                    0.61296646             F4(X)         102.82560720
CH4                   0.01048112             F5(X)          -0.00000000
O2/CH4                0.55332216             F6(X)         -57.82467068
TOTAL                 2.88022758             F7(X)          -0.00030019


Iteration #4
CO                    0.32283808             F1(X)           0.00004890
CO2                   0.00922480             F2(X)          -0.00071492
H2O                   0.04603061             F3(X)          -0.00035746
H2                    0.61809514             F4(X)           5.34555935
CH4                   0.00381137             F5(X)           0.00000000
O2/CH4                0.57582389             F6(X)          -0.79941029
TOTAL                 2.97413952             F7(X)           0.00000267


Iteration #5
CO                    0.32287083             F1(X)           0.00000007
CO2                   0.00922354             F2(X)          -0.00000105
H2O                   0.04601709             F3(X)          -0.00000053
H2                    0.61817165             F4(X)           0.00784925
CH4                   0.00371688             F5(X)          -0.00000000
O2/CH4                0.57671424             F6(X)          -0.00022526
TOTAL                 2.97785864             F7(X)          -0.00000000


Iteration #6
CO                    0.32287084             F1(X)           0.00000000
CO2                   0.00922354             F2(X)          -0.00000000
H2O                   0.04601709             F3(X)          -0.00000000
H2                    0.61817168             F4(X)           0.00000001
CH4                   0.00371685             F5(X)          -0.00000000
O2/CH4                0.57671540             F6(X)          -0.00000000
TOTAL                 2.97786345             F7(X)          -0.00000000


===========================================================================
                       Chemical Equilibrium Problem
===========================================================================
 nlsys Version 1.0.0                                  7/21/2004   2:39 pm
===========================================================================
Number of iterations required:                                          6
||F(x)|| at final solution:                                 6.5585984e-09

Algorithm used:                                                 HOOK STEP
Jacobian calculated using:                             FORWARD DIFFERENCE


-----------------------------------------------------------------------------
Termination Code = 1:

Norm of the scaled function value is less than the value of the fvtol member
of the nlControl structure; xp given is an approximate root of F(x) (unless
fvtol is too large).
-----------------------------------------------------------------------------


-----------------------------------------------------------------------------
VARIABLE              START                ROOTS                   F(ROOTS)
-----------------------------------------------------------------------------
CO                  0.50000           0.32287084              1.1154966e-13
CO2                 0.00000         0.0092235435             -1.7451596e-12
H2O                 0.00000          0.046017091             -8.4909857e-13
H2                  0.50000           0.61817168              1.3117187e-08
CH4                 0.00000          0.003716851             -5.5511151e-15
O2/CH4              0.50000            0.5767154             -1.5589308e-11
TOTAL               2.00000            2.9778635             -5.7322203e-14
-----------------------------------------------------------------------------
```

3.5 References

1. Agresti, A. 1984. *Analysis of Ordinal Categorical Data.* New York:Wiley.

2. Dennis and Schnabel 1983, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* New Jersey:Prentice-Hall.

3. Gill, Murray and Wright 1981, *Practical Optimization.* New York: Academic Press.

4. Carnahan, Luther and Wilkes 1969, *Applied Numerical Methods*. New York: Wiley.

Nonlinear Equations MT Reference 4

nlControlCreate

PURPOSE	Sets the members of an **nlControl** structure to default values.
LIBRARY	**nlsysmt**
FORMAT	*nlc* = **nlControlCreate;**
INPUT	None
OUTPUT	*nlc* — an instance of an **nlControl** structure with its members set to default values.

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

SOURCE `nlsysmt.src`

nlprt

PURPOSE Prints a summary of results from **nlsys**. This printout can be sent to an output file if one is open.

LIBRARY **nlsys**

FORMAT { *nlc*,*nlo* } = **nlprt(***nlc*,*nlo***);**

INPUT *nlc* an instance of an **nlControl** structure. The following members of *nlc* are referenced within the **nlprt** routine:

nlc.*ajac*	pointer to a procedure which computes the analytic Jacobian. By default, **nlsys** will compute the Jacobian numerically.
nlc.*algr*	scalar, indicates which search-direction/step-length algorithm should be used. **1** Use line search **2** Use hook step—a locally constrained search strategy. Default = 1.

`nlc.`*altnam*	$K \times 1$ string array, alternate names for variables created by **`nlsys`**. These names are used with the display produced by **`nlprt`**. For example,
	`nlc.altnam = "CO"$\|"CO2"$|"H2O"$\|"H2"$|`
	`"CH4"$\|"O2/CH4"$|"Total";`
	By default, the names *X1*, *X2*, *X3*... or *X01*, *X02*, *X03*... (depending on how **`nlc`**.*vpad* is set) will be used.

`nlc.`*chpf*	scalar, flag to control the method of Jacobian approximation: **0** use Broyden’s secant approximations **1** use finite difference Jacobians Default = 1.
`nlc.`*header*	string. This is used by **`nlprt`** to display information about the date, time, version of module, etc. The string can contain zero or more of the following characters: **t** print title (see **`nlc`**.*title*) **l** bracket title with lines **d** print date and time **v** print procedure name and version number Example: `nlc.header = "tld";` Default = “tldvf”.
`nlc.`*title*	string, a custom title to be printed at the top of the iterations report. By default, no title will be printed.
`nlc.`*vpad*	scalar. If **`nlc`**.*altnam* = “”, then the variable names used during printout

are automatically created by **nlsys**. Two types of names can be created:

0 Variable names are not padded to give them equal length. For example, *X1*, *X2* ... *X10*, *X11*....

1 Variable names are padded with zeros to give them an equal number of characters. For example, *X1*, *X2* ... *X10*, *X11*....

Default = 1.

nlo an instance of an **nlOut** structure returned by **nlsys**, which contains the following members:

nlo.*fvc*	$N \times 1$ vector, the final function results, $F(x_p)$.
nlo.*itnum*	scalar, the number of iterations required by **nlsys** to arrive at the final solution.
nlo.*jc*	$N \times N$ matrix, the final Jacobian results.
nlo.*startvals*	$N \times 1$ vector, starting values for the equation solution algorithm.
nlo.*tcode*	scalar, the termination code returned from **nlsys**.
nlo.*xp*	$N \times 1$ vector which represents a solution to the problem $F(x_p) = 0$.

OUTPUT *nlc* an instance of an **nlControl** structure that is identical to the one inputted by the user.

nlo an instance of an **nlOut** structure that is identical to the one inputted by the user. See **nlsys** for more information on the members of this structure.

REMARKS The call to **nlsys** can be nested inside the call to **nlprt**. For example:

```
{ nlc,nlo } = nlprt(nlsys(nlc,&f,x0));
```

SOURCE nlsys.src

nlsys

PURPOSE Solves a system of nonlinear equations.

LIBRARY **nlsysmt**

FORMAT { *nlc*,*nlo* } = **nlsys(***nlc*,**&***F*,*x0***);**

INPUT *nlc* an instance of an **nlControl** structure. The following members of *nlc* are referenced within the **nlsys** routine:

nlc.*ajac*	pointer to a procedure which computes the analytic Jacobian. By default, **nlsys** will compute the Jacobian numerically.
nlc.*algr*	scalar, indicates which search-direction/step-length algorithm should be used. **1** Use line search **2** Use hook step—a locally constrained search strategy. The line-search uses a Newton direction, and then determines a step length. The Hook step uses a predetermined step length, and then finds an optimal search direction.

	If using the hook step, it is best to set **nlc**.*chpf* = 1, or to supply an analytic Jacobian (see **nlc**.*ajac*). This particular strategy is sensitive to accuracy of the Hessian being used. If the secant update is being used, the Hessian may fail to invert. Default = 1.
`nlc.`*altnam*	$N \times 1$ string array of alternate names to be used by the printed output. By default, the names *X1*, *X2*, *X3*... or *X01*, *X02*, *X03*... (depending on how **nlc**.*vpad* is set) will be used.
`nlc.`*chpf*	scalar, flag to control the method of Jacobian approximation: **0** use Broyden's secant approximations **1** use finite difference Jacobians Use 0 if the function being evaluated is expensive and not sensitive. Neither of these methods will be used if an analytic Jacobian has been supplied. Default = 1.
`nlc.`*fdig*	scalar, the number of reliable digits in *F(x)*. This is used to compute *eta*, which is used to specify the relative noise in $F(x)$. *eta* is computed as follows: $eta = \max(macheps, 10^{-nlc.fdig})$ Default = 14.
`nlc.`*fvtol*	scalar, the tolerance of the scalar function $f = \frac{1}{2} \parallel F(x) \parallel_2$ required to terminate algorithm. That is, $\lvert f(x) \rvert$

	must be less than **nlc**.*fvtol* before the algorithm can terminate successfully. Default = $macheps^{1/3}$.
nlc.*maxit*	scalar, the maximum number of iterations. Default = 100.
nlc.*mtol*	scalar, the value used to test if the algorithm is stuck at a local minimizer. The algorithm stops if the maximum component of the scaled gradient is ≤ **nlc**.*mtol*. Default = $macheps^{2/3}$.
nlc.*output*	scalar. If 1, output produced during iterations is sent to the screen and/or output device such as a printer or output file. Default = 1.
nlc.*stjc*	$N \times N$ matrix, may be set by user to initial Jacobian of $F(x)$ with respect to coefficients, if one is available. By default, **nlsys** will compute the initial Jacobian using a forward difference method.
nlc.*stol*	scalar, the step tolerance. Default = $macheps^{2/3}$.
nlc.*typf*	$N \times 1$ vector of the typical $F(x)$ values at a point not near a root, used for scaling. This becomes important when the magnitudes of the components of $F(x)$ are expected to be very different. By default, function values are not scaled.
nlc.*typx*	$N \times 1$ vector of the typical magnitude of x, used for scaling. This becomes important when the magnitudes of the components of x are expected to be

very different. By default, variable values are not scaled.

`nlc.`*vpad* — scalar. If **`nlc`**.*altnam* = “”, then the variable names used during printout are automatically created by **`nlsys`**. Two types of names can be created:

- **0** Variable names automatically created by **`nlsys`** are not padded to give them equal length. For example, *X1*, *X2* ... *X10*, *X11*....
- **1** Variable names created by the procedure are padded with zeros to give them an equal number of characters. For example, *X01*, *X02* ... *X10*, *X11*....

Default = 1.

&*F* — A pointer to a procedure which computes the value at x of the equations to be solved. This procedure should return an $N \times 1$ column vector containing the result for each equation. For example:

Equation 1: $x_1^2 + x_2^2 - 2 = 0$

Equation 2: $e^{x_1 - 1} + x_2^3 - 2 = 0$

```
proc G(x);
    local g1,g2;
    g1 = x[1]^2 + x[2]^2 - 2;
    g2 = exp(x[1]-1) + x[2]^3 - 2;
    retp(g1|g2);
endp;
```

x0 — $N \times 1$ vector, starting values for the equation solution algorithm. There should be as many elements in this vector as equations to be solved.

OUTPUT *nlc* — an instance of an **`nlControl`** structure that is identical to the one inputted by the user.

nlo an instance of an **`nlOut`** structure, containing the following members:

`nlo.`*fvc*	$N \times 1$ vector, the final function results, $F(x_p)$.
`nlo.`*itnum*	scalar, the number of iterations required by **`nlsys`** to arrive at the final solution.
`nlo.`*jc*	$N \times N$ matrix, the final Jacobian results.
`nlo.`*tcode*	scalar, the termination code. 1 is successful. Others may represent failure.

1 Norm of the scaled function value is less than **`nlc`**.*fvtol*; the x_p given is an approximate root of $F(x)$ (unless **`nlc`**.*fvtol* is too large).

2 The scaled distance between the last two steps is less than the step-tolerance (**`nlc`**.*stol*). x_p may be an approximate root of $F(x)$, but it is also possible that the algorithm is making very slow progress and is not near a root, or the step-tolerance (**`nlc`**.*stol*) is too large.

3 The last global step failed to decrease $\| F(x) \|_2$ sufficiently; either x_p is close to a root of F and no more accuracy is possible, an incorrectly coded analytic Jacobian is being used, the secant approximation to the Jacobian is inaccurate, or the step-tolerance (**`nlc`**.*stol*) is too large.

4 Iteration limit exceeded.

5 Five consecutive steps of maximum step length have been taken. This probably means that **nlsys** is approaching asymptotically a finite value from above.

6 x_p may be an approximate local minimizer of $\| F(x) \|_2$ that is not a root of $F(x)$ (or **nlc**.*mtol* is too small). To find a root of $F(x)$, **nlsys** should be restarted from a different region.

nlo.*xp*	$N \times 1$ vector which represents a solution to the problem $F(x_p) = 0$. Check **nlo**.*tcode* to confirm that the algorithm converged to a proper solution.

REMARKS

This solves a system of nonlinear equations using a quasi-Newton algorithm with Broyden's secant update method. The algorithm uses a line-search algorithm or a model trust region approach (hookstep) as a globalizing strategy. Numeric derivatives are calculated by default; however, analytic derivatives may be substituted if available.

EXAMPLE

This example illustrates the difficulty of solving certain classes of problems. Try various starting values to get some of the roots of these equations. Certain starting values will not converge to a root.

The equations to be solved are:

$$
\begin{aligned}
\frac{1}{2} sin(x_1 x_2) - \frac{x_2}{4\pi} - \frac{x_1}{2} &= 0 \\
\left(1 - \frac{1}{4\pi}\right)(e^{2x_1 - 1} - 1) + \frac{x_2}{\pi} - 2x_1 &= 0
\end{aligned}
$$

A number of roots can be found—here are a few:

x1	x2
0.5000	3.1416
0.2994	2.8369
-0.2606	0.6225

Here is the code for the above example:

```
library nlsysmt;
#include nlsysmt.sdf
struct nlControl nlc;
struct nlOut nlo;
nlc = nlControlCreate;

proc  F(x);
    local f1,f2;
    f1 = 0.5*sin(x[1]*x[2]) - x[2]/(4*pi) - x[1]/2;
    f2 = (1 - 1/(4*pi))*(exp(2*x[1] - 1) - 1) + x[2]/pi - 2*x[1];
    retp(f1|f2);
endp;

x0 = { 0.2, 0.3 };    /* Starting Values */
output file = nlmt.out reset;
{ nlc,nlo } = nlprt(nlsys(nlc,&f,x0));
output off;
```

SOURCE `nlsys.src`

Index