Nonlinear Equations

Documentation Version: January 15, 2001

Contents

1 Installation **1**

1.1 UNIX 1

1.1.1 Solaris 2.x Volume Management 2

1.2 DOS 2

1.3 Differences Between the UNIX and DOS Versions 3

2 Nonlinear Equations **5**

2.1 Introduction 5

2.2 Getting Started 5

2.2.1 README Files 5

2.2.2 Setup 5

2.3 About the NLSYS Procedure 6

2.4 Solution Method 7

2.5 Using _NLSYS Directly 8

2.6 An Example 8

2.6.1 Setting Up the System 8

2.6.2 Starting Values 9

2.6.3 Global Parameters 9

2.6.4 The Complete Example 10

2.7 References 11

3 Nonlinear Equations Reference **13**

NLSET 14

NLSYS 15

NLPRT 20

Index **23**

Chapter 1

Installation

1.1 UNIX

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

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

   `tar xvf` *device_name*

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

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

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

1.1.1 Solaris 2.x Volume Management

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

```
volcheck
```

to signal the system to mount the floppy.

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

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

To turn on volume management, become the superuser and type

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

1.2 DOS

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

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

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

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

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

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

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

   DEVICE=C:\DOS\ANSI.SYS

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

1.3 Differences Between the UNIX and DOS Versions

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

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

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

Chapter 2

Nonlinear Equations

2.1 Introduction

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

$$F(x) = 0$$

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

2.2 Getting Started

GAUSS 3.1.0+ is required to use these routines.

2.2.1 README Files

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

2.2.2 Setup

In order to use the procedures in the *NONLINEAR EQUATIONS* Module, the **NLSYS** library must be active. This is done by including `nlsys` in the **LIBRARY** statement at the top of your program:

```
library nlsys,simplex,pgraph;
```

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

```
#include nlsys.ext;
```

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

```
nlset;
```

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

The version number of each module is stored in a global variable. For *NONLINEAR EQUATIONS*, this global is:

_nl_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.

2.3 About the NLSYS Procedure

To call **NLSYS**, all that is needed is the statement

```
{ x, fvc, jc, tcode } = NLSYS(&f,x0);
```

where `&f` is a pointer to the procedure describing the system of equations (a discussion on how to set up this procedure follows), and `x0` is a vector of start values. The output arguments are, respectively, the final solution (x), the value of the system of equations at the final solution (fvc), the final Jacobian (jc), and a return code ($tcode$).

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.

2.4 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 global variables **_nlchpf**, **_nlalgr**, **_nlstjc**, and **_nlajac** 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.

2.5 Using _NLSYS Directly

When **NLSYS** is called, it directly references all the necessary globals and passes its arguments and the values of the globals to a function called **_NLSYS**. When **_NLSYS** returns, **NLSYS** then sets the output globals to the values returned by **_NLSYS** and returns its arguments directly to the user. **_NLSYS** makes no global references to matrices or strings, and all procedures it references have names that begin with an underscore "_".

_NLSYS can be used directly in situations where you do not want any of the global matrices and strings in your program. If **NLSYS**, **NLPRT** and **NLSET** are not referenced, the global matrices and strings in `nlsys.dec` will not be included in your program.

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

2.6 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 `nl4.e`.

2.6.1 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 global references 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**.

2.6.2 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 };
```

2.6.3 Global Parameters

Optional parameters may be specified which set the convergence, scaling and global returns. These can be specified as follows:

```
_nlalgr = 2;
__altnam = { co, co2, h2o, h2, ch4, o2/ch4, total };
__title = "Chemical Equilibrium Problem";
__output = 1;
```

See the **NLSYS** function definition in Chapter 3 for a complete listing of these options.

2.6.4 The Complete Example

Here is a complete example illustrating the use of **NLSYS** and the printing procedure **NLPRT**.

```
library nlsys;
#include nlsys.ext;
nlset;

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 };
_nlalgr = 2;
__altnam = { co, co2, h2o, h2, ch4, o2/ch4, total };
__title = "Chemical Equilibrium Problem";
__output = 1;
output file = nl4.out reset;
{ x,f,j,tcode } = nlprt(nlsys(&fsys,x0));
output off;
```

The procedure **LPPRT** nested around the call to **NLSYS** prints the results to the current output device: in this case, the screen and the output file `nl4.out`. The alternative names set in **__altnam** will be used to label the variables.

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

```
-------------------------------------------------------------------------------
                        Chemical Equilibrium Problem
===============================================================================
 NLSYS:  Version 2.01 (R1)                                 8/08/90  11:32 am
===============================================================================

Number of iterations required:                                             6
||F(x)|| at final solution:                                  5.9710634e-09
```

```
Algorithm used:                                                  HOOK STEP
Jacobian calculated using:                                FORWARD DIFFERENCE

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

Norm of the scaled function value is less than _nlfvtol;
Xp given is an approximate root of F(x) (unless _nlfvtol
is too large).
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
VARIABLE             START                ROOTS                F(ROOTS)
--------------------------------------------------------------------------------
CO                 0.50000              0.32287084           2.1804787e-13
CO2                0.00000           0.0092235435           -1.6312507e-12
H2O                0.00000             0.046017091          -8.4510177e-13
H2                 0.50000              0.61817168           9.9998942e-09
CH4                0.00000             0.003716851          -7.571721e-14
O2/CH4             0.50000               0.5767154          -6.8443029e-11
TOTAL              2.00000               2.9778635           2.5942616e-13
--------------------------------------------------------------------------------
```

2.7 References

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

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

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

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

Chapter 3

Nonlinear Equations Reference

Purpose

Resets *NONLINEAR EQUATIONS* global variables to default values.

Library

NLSYS

Format

NLSET;

Input

None

Output

None

Remarks

Putting this instruction at the top of all programs that invoke **NLSYS** 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 **NLSYS**.

NLSET calls **GAUSSET**.

Source

`nlsys.src`

Purpose

Solves a system of nonlinear equations.

Library

NLSYS

Format

{ *xp*,*fvc*,*jc*,*tcode* } = **NLSYS(&***F*,*x0*);

Input

x0 Nx1 vector of the starting values for the equation solution algorithm. There should be as many elements in this vector as equations to be solved.

&*F* A pointer to a procedure which computes the value at x of the equations to be solved. This procedure should return an Nx1 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;
```

Output

xp Nx1 vector which represents a solution to the problem $F(x_p) = 0$. Check *tcode* to confirm that the algorithm converged to a proper solution.

fvc Nx1 vector, the final function results, $F(x_p)$.

jc NxN matrix, the final Jacobian results.

tcode scalar, the termination code. 1 is successful. Others may represent failure.

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

2 The scaled distance between the last two steps is less than the step-tolerance (**_nlstol**). 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 (**_nlstol**) 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 (**_nlstol**) 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 **_nlmtol** is too small). To find a root of $F(x)$, **NLSYS** should be restarted from a different region.

Global Input

_nlalgr 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 **_nlchpf** = 1, or to supply an analytic Jacobian (see **_nlajac**). 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.

_nlajac pointer to a procedure which computes the analytic Jacobian. By default, **NLSYS** will compute the Jacobian numerically.

_nlchpf 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.

_nlfdig 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:

$eta = \max(macheps, 10^{-\_nlfdig})$

Default = 14.

_nlfvtol scalar, the tolerance of the scalar function $f = \frac{1}{2} \parallel F(x) \parallel_2$ required to terminate algorithm. That is, $|f(x)|$ must be less than **_nlfvtol** before the algorithm can terminate successfully. Default = $macheps^{1/3}$.

_nlmaxit scalar, the maximum number of iterations. Default = 100.

_nlmtol 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 $\leq$ **_nlmtol**.

Default = $macheps^{2/3}$.

_nlstol scalar, the step tolerance. Default = $macheps^{2/3}$.

_nltypf Nx1 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.

_nltypx Nx1 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.

_nlstjc NxN 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.

__altnam Nx1 character vector of alternate names to be used by the printed output. By default, the names X1, X2, X3... or X01, X02, X03... (depending on how **__vpad** is set) will be used.

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

t	print title (see **__title**)
l	bracket title with lines
d	print date and time
v	print procedure name and version number
f	print file name being analyzed (NOT USED BY **NLSYS**)

Example:

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

Default = "tldvf".

__output scalar. If nonzero, output produced during iterations is sent to the screen and/or output device such as a printer or output file.

__title string, a custom title to be printed at the top of the iterations report. By default, no title will be printed.

__vpad scalar. If **__altnam** = 0, 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.

Global Output

_nlitnum scalar, the number of iterations required by **NLSYS** to arrive at the final 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:

$$\begin{aligned} \frac{1}{2}sin(x_1x_2) - \frac{x_2}{4\pi} - \frac{x_1}{2} &= 0 \\ \left(1 - \frac{1}{4\pi}\right)(e^{2x_1-1} - 1) + \frac{x_2}{\pi} - 2x_1 &= 0 \end{aligned}$$

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 nlsys;
nlset;

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 = nl.out reset;
{ xp,fvc,jc,tcode } = nlprt(nlsys(&f,x0));
output off;
```

Source

nlsys.src

Purpose

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

Library

NLSYS

Format

NLPRT(*x*,*fvc*,*jc*,*tcode***);**

Input

xp	An Nx1 vector which represents a solution to the problem $F(x_p) = 0$.
fvc	Nx1 vector, the final function results, $F(x_p)$.
jc	NxN matrix, the final Jacobian results.
tcode	scalar, the termination code returned from **NLSYS**.

Globals

__altnam Kx1 character vector, alternate names for variables created by **NLSYS**. These names are used with the display produced by **NLPRT**. For example,

```
__altnam = { CO, CO2, H2O, H2, CH4, O2/CH4, Total };
```

By default, the names X1, X2, X3... or X01, X02, X03... (depending on how **__vpad** is set) will be used.

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

t	print title (see **__title**)
l	bracket title with lines
d	print date and time
v	print procedure name and version number
f	print file name being analyzed (NOT USED BY **NLSYS**)

Example:

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

Default = "tldvf".

___title string, a custom title to be printed at the top of the iterations report. By default, no title will be printed.

___vpad scalar. If **___altnam** = 0, 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, X01, X02 ... X10, X11....

Default = 1.

Remarks

The call to **NLSYS** can be nested inside the call to **NLPRT**. For example:

```
{ x,fvc,jc,tcode } = nlprt(nlsys(&f,x0));
```

NLPRT requires that the output globals used and returned by **NLSYS** still be in memory. Either nest the call to **NLSYS** inside the call to **LPPRT** (the safest way), or call **NLPRT** directly after the call to **NLSYS**.

Globals

_nlajac, **_nlchpf**, **_nlalgr**, **_nlitnum**, **_nl_x0**

Source

nlsys.src

Index

__**altnam**, 20

C

Carnahan, 9
convergence, 9

D

derivatives, analytic, 19
derivatives, numeric, 19
derivatives, partial, 6
DOS, 2, 3

F

forward difference, 7

G

GAUSS, procedure, 8
global returns, 9

H

hookstep algorithm, 7

I

Installation, 1

J

Jacobian, 6

L

library, NLSYS, 5
line-search, 19
line-search algorithm, 7

N

_**nlajac**, 16
_**nlalgr**, 16
_**nlchpf**, 16
_**nlfdig**, 17
_**nlfvtol**, 17
_**nlmaxit**, 17
_**nlmtol**, 17
NLPRT, 10, 20
NLSET, 14
_**nlstjc**, 17
_**nlstol**, 17
NLSYS, 6, 15
_**nltypf**, 17
_**nltypx**, 17
noise, 17

Q

quasi-Newton, 7, 19

S

scaling, 9, 17
secant, Broyden's, 19
starting values, 9, 19

T

tolerance levels, 7

U

UNIX, 1, 3