NNES USER'S MANUAL

Table of Contents

Introduction

This manual describes a nonlinear algebraic equation solver NNES (**N**onmonotonic **N**onlinear **E**quation **S**olver) designed to solve problems of the form:

$$\boldsymbol{F}(\boldsymbol{x})=\mathbf{0} \qquad \boldsymbol{F},\boldsymbol{x} \in \mathbb{R}^n$$

$$boundl_i \le x_i \le boundu_i \qquad i = 1,n$$

given an initial estimate, $\mathbf{x}_0$, plus a function (or residual) vector and, if desired, an analytical Jacobian. It comprises three sections:

Section 1 Using NNES with Default Settings
Section 2 Using NNES's Options
Section 3 General Description of NNES.

Section 1 describes using NNES with minimal user-input. A basic driver (provided) can be modified to run a nonmonotonic, damped-Newton version of the program. If more flexibility is desired, specifically if the default settings fail to generate the correct feasible solution, then the options described in Section 2 may be used. Also provided is a description of tactics which may help if NNES fails to solve the problems using default settings. For the user not familiar with this type of nonlinear equation solver, a description of the various methods is provided in Section 3 along with a description of some features of NNES.

1. Using NNES with Default Settings

NNES can be run with minimal user-input using the default settings. A generic driver, which can be copied as NNESDR.FOR from the subdirectory DRIVERS, is shown in Appendix 1. The following must be modified to fit the particular problem:

1. the number of equations set as a parameter, N=?
2. the unit number, NUNIT, for output (usually 10)
3. the initial estimates (given before the call to NNES) XC(1)= ...
4. the functions given in the function evaluation subroutine (FCN)
5. the name of this subroutine corresponding to the EXTERNAL statement and to the call to NNES (it may be left as FCN)
6. an output file name (called NNESDR.OUT in the generic sample).

The subroutines in subdirectory LIB must be compiled and should be put into a library. The modified driver must then be linked to this library. Note that a dummy Jacobian evaluation subroutine must be attached even if an analytical Jacobian is not being used. The dummy Jacobian provided, JACOB, can be detached and linked separately to avoid continuously recompiling it. If a solution is found the output will list the starting estimates and the solution and detail how many iterations and function evaluations were used.

Along with the generic driver NNESDR, sample drivers and subroutines for the solution of several problems are provided to show how some of the features of NNES are invoked. The solution of Rosenbrock's Banana, Example 1, is shown in Appendix 2 corresponding to OUTPUT=2.

2. Using NNES's Options

In this section a description of NNES's options is given. For a user familiar with Newton-based nonlinear equation solvers, the description of each parameter given in Table 2 should be adequate to understand the implications of adjusting that parameter. A fuller description of the methods and variables is given in Section 3.

NNES includes four basic types of nonlinear equation solver which differ as to whether some acceptance criterion must be met to accept a step and, if so, as to how the search for that acceptable step is conducted. They are

1) Newton's Method
2) Absolute Newton's Method
3) Line Search Method
4) Trust Region Method.

When no step acceptance test is conducted nor bounds enforced, this is the standard Newton's method. The Absolute Newton's Method is a variation of Newton's method where any step which violates a bound is reflected in that bound as shown in Figure 1.

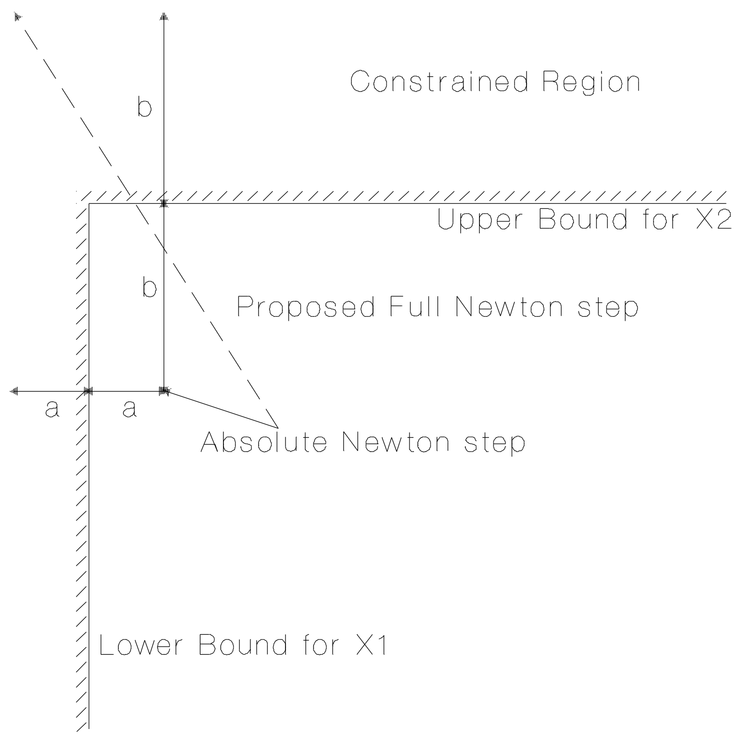

Figure 1 Step for Absolute Newton's Method

The lower bound for X1 has been violated by amount a and the upper bound for X2 has been violated by amount b by the proposed full Newton step. The amount of the violation for each component is reflected in the respective bound to obtain the shorter step of the Absolute Newton's method.

For the other two methods, the nonlinear equations problem is converted into a specialized minimization problem. The usual objective function to be minimized is f=1/2 $\mathbf{F}(\mathbf{x})^T\mathbf{F}(\mathbf{x})$ but NNES also has the capability of employing a second objective function (often called the *natural* objective function). Mathematically, the problem is solved when the objective function, f, reaches zero. The method to find this minimum is different, however, than that which would be used by a general nonlinear minimization program because of the special structure of the objective function and the knowledge that the objective has a minimum at zero.

One of two procedures may be used to enhance convergence: a line search in the direction of the Newton step or a trust region method in which a linear combination of a step in the steepest descent direction for the objective function and the Newton step is found which produces a step of a designated length (the trust region size, DELTA). Thus the trust region method restricts the step length to be within a region in which some confidence is held that the quadratic model for the objective function, upon which the Newton step is based, is a reasonably accurate representation of the actual objective function. The size of the trust region is adjusted at each step depending on how well the model and the actual objective function agree. A detailed description of these methods is given in Section 3.

The Jacobian matrix need not be evaluated explicitly at each iteration as two quasi-Newton update methods are also available: Broyden's (1965) method and a more recent method proposed by Lee and Lee (1989). If the Jacobian matrix is to be evaluated explicitly there are two methods, namely a user-supplied analytical Jacobian or via finite differences.

Within this framework there are many other options which are set to default values by the subroutine SETUP which has the following argument list:

```
SUBROUTINE SETUP(ABSNEW ,CAUCHY ,DEUFLH ,GEOMS  ,LINESR ,
                 NEWTON ,OVERCH ,ACPTCR ,ITSCLF ,ITSCLX ,
                 JACTYP ,JUPDM  ,MAXEXP ,MAXIT  ,MAXNS  ,
                 MAXQNS ,MINQNS ,N      ,NARMIJ ,NIEJEV ,
                 NJACCH ,OUTPUT ,QNUPDM ,STOPCR ,SUPPRS ,
                 TRUPDM ,ALPHA  ,CONFAC ,DELTA  ,DELFAC ,
                 EPSMCH ,ETAFAC ,FDTOLJ ,FTOL   ,LAM0   ,
                 MSTPF  ,NSTTOL ,OMEGA  ,RATIOF ,SIGMA  ,
                 STPTOL ,BOUNDL ,BOUNDU ,SCALEF ,SCALEX ,
                 HELP   )
```

In this subroutine, as is the case for all subroutines in NNES, arguments are given in alphabetical order within each classification of LOGICAL, INTEGER, DOUBLE PRECISION NUMBER, DOUBLE PRECISION MATRIX (includes vectors) and CHARACTER. Note that first-letter conventions for integers are not followed, e.g., ACPTCR is the first INTEGER.

Because more than one method is included in NNES, there is a hierarchy of options which is shown in Table 1. The first section shows how the three key logical variables, NEWTON, ABSNEW and LINESR, must be set to obtain the desired method. The default values are designated with asterisks. The second section lists the options which are available for each method. Not included in the table are those variables which govern output only, namely OUTPUT and SUPPRS, and those variables which are common to each method: OVERCH; JACTYP; JUPDM; MAXEXP; MAXIT; N; NJACCH; QNUPDM; STOPCR; TRMCOD; FTOL; NSTTOL; OMEGA and STPTOL.

There are also variables which are required for special situations only. These are held in COMMON blocks: MATSUP; WRNSUP; BYPASS and NFETOT, the total number of function evaluations. These are also common to each method.

		METHOD		
	NEWTON	ABSOLUTE NEWTON	LINE SEARCH	TRUST REGION
NEWTON	TRUE	FALSE	FALSE*	FALSE
ABSNEW	FALSE	TRUE	FALSE*	FALSE
LINESR	TRUE	TRUE	TRUE*	FALSE
Options			DEUFLH	CAUCHY
			GEOMS	ACPTCR
			ACPTCR	ITSCLF
			ITSCLF	ITSCLX
			ITSCLX	MAXNS
			MAXNS	MAXQNS
			MAXQNS	MINQNS
			MINQNS	NARMIJ
			NARMIJ	NIEJEV
			NIEJEV	TRUPDM
			ALPHA	ALPHA
			CONFAC	CONFAC
			LAM0	DELTA
			MSTPF	DELFAC
			RATIOF	ETAFAC
			SIGMA	MSTPF
			BOUNDL	RATIOF
			BOUNDU	SIGMA
			SCALEF	BOUNDL
			SCALEX	BOUNDU
				SCALEF
				SCALEX

Table 1 Required and Optional Variables for NNES
* designates default settings

If incompatible options are used, a warning will be issued by NNES and execution will usually continue with the options used being declared by NNES.

Each option is briefly described in Table 2. If changes are desired, these must be set between the calls to SETUP and NNES. The possible values are given within each description.

Because of the large number of options the sensitivity of the program to changes in the input parameters is discussed following Table 2. Also given are some suggestions on how problems can be transformed for easier solution.

VARIABLE	DESCRIPTION	DEFAULT
ABSNEW	Switches to Absolute Newton's method; NEWTON should be FALSE and LINESR should be TRUE. (If both NEWTON and ABSNEW are TRUE the Absolute Newton's method is used and a warning issued).	FALSE
BYPASS	Bypasses calculation of condition number and the remedial measures of Dennis and Schnabel (1983) for ill-conditioned Jacobians. Execution always continues as if Jacobian were well-conditioned (unless it is found to be singular). Switching BYPASS to TRUE may be useful in two ways: to save time if you know the system is well-conditioned and don't need the condition number estimation or if you would prefer the possibly inaccurate Newton step from a ill-conditioned system.	FALSE
CAUCHY	The initial trust region size, DELTA, may be set by the user or by the program. If the program is used, DELTA is set to the length of the initial Newton step (CAUCHY is FALSE) or to the distance to the minimum of the quadratic model of the objective function in the steepest descent direction, the Cauchy point (CAUCHY is TRUE). DELTA must be ≤ 0 on entry for either of these two methods to be used. The Newton length is always greater, thus setting CAUCHY to TRUE is, generally, a more conservative and thus a slower approach.	FALSE
DEUFLH	The initial line search relaxation factor, λ, can be set to LAM0 or by a version of Deuflhard's (1974,1975) relaxation factor initialization process which estimates what the relaxation factor should be at each iteration. LAM0 is usually 1.0 but may be made smaller if the problem is extremely nonlinear.	TRUE
GEOMS	Two types of search may be used to reduce either λ or δ. In the geometric search the length is decreased at each stage by multiplying the current λ, by the factor SIGMA (usually 0.5) or dividing δ by DELFAC (usually 2.0). If GEOMS is FALSE then for line searches at the first stage and trust region steps for all stages, a quadratic function of the relaxation factor or DELTA, respectively, is minimized using the initial objective function value, the trial point objective function value and the directional derivative. For line searches, if further cutbacks are needed, a cubic minimization using the two most recent objective function values, the initial objective function value and the directional derivative is used.	TRUE
LINESR	Designates whether a line search (TRUE) or trust region method (FALSE) is to be used. Should be TRUE for Newton's or Absolute Newton's method.	TRUE
MATSUP	Suppresses matrix printing in detailed outputs.	FALSE

NEWTON	Designates that a strict Newton or quasi-Newton method is to be used if set to TRUE. Otherwise, Absolute Newton's method or either a line search or a trust region method will be used. LINESR should also be TRUE if NEWTON is TRUE (LINESR will be switched internally if not). See also JUPDM to choose among Newton or either quasi-Newton method.	FALSE
OVERCH	A check can be made for potential overflows at certain locations within NNES. If a potential overflow is detected, 10^{MAXEXP} is inserted in place. This may prevent a few unnecessary failures because of overflows.	FALSE
WRNSUP	Suppresses printing of warnings. Used only if known warnings will clutter output.	FALSE
ACPTCR	Denotes whether a step may be accepted based on only the usual sum-of-squares function value step acceptance criterion, *f*, whereupon ACPTCR=1, or if the function value and the natural objective function step acceptance criteria are both being used, ACPTCR=12.	12
ITSCLF	When set to 0 it implies no adaptive function scaling to be used. For any integer greater than 0, adaptive scaling begins at that iteration. Scaling factors are updated when the Jacobian is evaluated explicitly. No scaling or user-supplied scaling with no updating usually work best.	0
ITSCLX	When set to 0 it implies no adaptive variable scaling to be used. For any integer greater than 0, adaptive scaling begins at that iteration. Scaling factors are updated when the Jacobian is evaluated explicitly. No scaling or user-supplied scaling with no updating usually work best.	0
JACTYP	Denotes how the Jacobian is to be explicitly evaluated (see JUPDM for choosing how the Jacobian is updated): JACTYP=0 Analytical Jacobian provided JACTYP=1 Forward Difference Jacobian JACTYP=2 Backward Difference Jacobian JACTYP=3 Central Difference Jacobian	1
JUPDM	Jacobian update method: JUPDM=0 Jacobian calculated at every iteration (either using analytical derivatives or finite differences) JUPDM=1 Broyden's (1965) quasi-Newton update JUPDM=2 Lee and Lee's (1989) quasi-Newton update	0
MAXEXP	Maximum exponent allowed (base ten). This is found internally using the subroutine MACHAR along with other machine-specific constants.	-
MAXIT	Maximum number of iterations allowed.	250
MAXNS	In any iteration, the maximum number of line search or trust region steps allowed after an explicit Jacobian update. Used in conjunction with MAXQNS.	50

MAXQNS	In any iteration, the maximum number of line search trial or trust region steps allowed after a quasi-Newton Jacobian update. This is used to prevent an excessive number of trial steps being tried in a poor search direction because of inaccuracies in the quasi-Newton Jacobian approximation.	10
MGLL	In effect when using the nonmonotonic step acceptance criterion of Grippo, Lampariello and Lucidi (1986), it tells how many previous values to compare to judge step acceptance. For function value, *f*, accept the step if: $$f(x_k + \lambda s^N) \le \max_{0 \le j \le m(k)} f(x_{k-j}) + \alpha \lambda \nabla f(x_k)^T s^N$$ $$m(0) = 0 \quad and \quad m(k) = min[m(k-1)+1, MGLL] \;, k \ge 1$$ A similar acceptance criterion can also be used for the natural objective if ACPTCR=12. If desired, the nonmonotonic step acceptance criterion can be delayed by setting the number of initial Armijo (strict descent) steps, NARMIJ, to be greater than 1.	10
MINQNS	Minimum number of iterations using quasi-Newton updated Jacobian between explicit Jacobian evaluations. This is aimed at minimizing function evaluations by not allowing finite-difference approximations of the Jacobian too often.	7
NARMIJ	Number of iterations, from the start in Newton-based methods and from the last explicit Jacobian evaluation in quasi-Newton methods, which require meeting the Armijo (1966) criterion (see below) for step acceptance (essentially strict descent). $$f(x_k + \lambda s^N) \le f(x_k) + \alpha \lambda \nabla f(x_k)^T s^N$$	1
NIEJEV	Number of initial explicit Jacobian evaluations, i.e., using an user-supplied analytical Jacobian or finite-difference approximation. This delays the start of quasi-Newton steps when the solution is more sensitive to the initial steps.	1
NJACCH	Number of iterations for which the analytical Jacobian is checked using forward differences if JACTYP=0. See FDTOLJ for a definition of an acceptable agreement.	1

OUTPUT	Signifies the level of output. This may be modified by variables MATSUP, SUPPRS and WRNSUP. OUTPUT=0 no output OUTPUT=1 final answer only OUTPUT=2 initial estimates and final answer OUTPUT=3 input data and iteration summaries OUTPUT=4 detailed summary of each step OUTPUT≥5 very detailed summary of each step	2
QNUPDM	Quasi-Newton update method: QNUPDM=0 unfactored Jacobian updated QNUPDM=1 factored Jacobian updated Mathematically these are equivalent although numerically they may give slightly different results. Use QNUPDM=1 as it is faster unless you want to see the unfactored updated Jacobian.	1
STOPCR	Stopping criterion, the following must be met: STOPCR=1 step size criterion only STOPCR=2 function value criterion only STOPCR=12 function value *or* step size criterion STOPCR=3 function value *and* step size criteria See also FTOL, NSTTOL and STPTOL for details of each.	12
SUPPRS	Suppresses detailed output for SUPPRS iterations; useful if failure is occurring after several iterations with N large.	0
TRMCOD	Termination code (this is output only): TRMCOD=0 problem not solved TRMCOD=1 step size criterion met TRMCOD=12 step size and function value criteria met when STOPCR=12 TRMCOD=2 function value stopping criterion met TRMCOD=3 step size and function value criteria met when STOPCR=3	-
TRUPDM	Trust region update method: TRUPDM=0 Powell's (1970) method (used in quasi-Newton methods only) TRUPDM=1 Dennis and Schnabel's (1983) method (can be used in quasi-Newton methods, must be used otherwise)	0
ALPHA	Factor $\in$ [0,1) in function value step acceptance criterion. See description of MGLL and NARMIJ.	1.0E-4
CONFAC	Factor $\in$ (0,1] used when forcing steps to stay within bounds; the trial step will only go this fraction of the distance to the first bound encountered in the search direction in line search methods or, in trust region methods, the reduced DELTA will be this fraction of the distance to the first bound.	.95

DELTA	Trust region size (updated internally). If negative on input NNES calculates the first trust region radius, DELTA, (see CAUCHY) or if positive, the first DELTA is the input value.	-1.0
DELFAC	Allows changing the factor by which trust region size, DELTA, is modified in trust region updating (it is the default value of 2.0 in both Powell's (1970) and Dennis and Schnabel's (1983) methods).	2.0
EPSMCH	Machine epsilon, ε_m, smallest number, x, such that: $$1.0 + x > 1.0$$	set in NNES
ETAFAC	Allows switching between dogleg and double dogleg method (see Dennis and Mei (1979) or Dennis and Schnabel (1983) or description in Section 3): ETAFAC = 1.0 single dogleg method ETAFAC $\in$ (0,1) double dogleg method	0.2
FDTOLJ	Tolerance in comparing the analytical Jacobian, JAC, to the forward finite-difference check, JACFD. The program stops if $$\left(\frac{\lvert JACFD_{ij}\rvert - \lvert JAC_{ij}\rvert}{max(\lvert JACFD_{ij}\rvert, \lvert JAC_{ij}\rvert, 1)}\right) \le FDTOLJ$$ NJACCH controls how many times the Jacobian is checked.	1.0E-6
FTOL	Stopping tolerance for function vector. This may be used alone or with step size stopping criteria; see NSTTOL, STPTOL and STOPCR to select. Choosing this criterion alone may avoid premature stopping at spots where the steps have become small. $$if: \ \max_i \left(SCALEF_i \cdot \lvert F_i\rvert\right) \le FTOL \quad stop$$	$\varepsilon_m^{1/3}$
LAM0	Initial line search relaxation factor $\in$ (0,1] - bounds first step; values less than 1.0 preclude quadratic convergence but may avoid wasted function evaluations in extremely nonlinear problems. If Deuflhard relaxation factor initialization is being used, then this value may be superseded; see DEUFLH. Values greater than 1.0 are not allowed.	1.0

MSTPF	Factor used in setting maximum step length, MAXSTP. $$MAXSTP = MSTPF \cdot max\,(\,\lVert D_x x_0 \rVert_2\,,\ \lVert D_x \rVert_2\,)$$ $$where \quad D_x = diag\,(SCALEX)$$ MAXSTP is used as a general bound on step size.	1000
NSTTOL	Stopping tolerance for Newton step. Stop if: $$\max_i \left(\frac{\lvert s_i^N \rvert}{max(\ \lvert x_{k_i} \rvert\ ,\ 1/SCALEX_i\ )} \right) \le NSTTOL\,(1 + \lVert D_x x_k \rVert)$$ $$where\ D_x = diag\,(SCALEX)$$ This stopping tolerance governs step size before the line search; see STPTOL for the step size control after the step reduction via line search or trust region updating.	$\varepsilon_m^{2/3}$
OMEGA	Factor $\in$ [0,1] in Lee and Lee (1989) update. $$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T P_k}{s_k^T P_k s_k} \quad , \quad B_0 = J\,(x_0)$$ $$P_{k+1} = P_k - \frac{P_k s_k s_k^T P_k}{\omega^{k+3} + s_k^T P_k s_k} \quad , \quad P_0 = I$$	0.1
RATIOF	Comparison factor $\in$ (0,1] in quasi-Newton methods. $$if\!:\ \frac{f_{new}}{f_{old}} > RATIOF \qquad then\!:\ NFAIL = NFAIL + 1$$ $$if\!:\ \frac{f_{new}}{f_{old}} < 0.01 \qquad then\!:\ NFAIL = NFAIL - 1$$ otherwise NFAIL remains unchanged. If NFAIL > MINQNS, NNES restarts at the best point, in terms of objective function value, found so far as if that point were an initial estimate.	0.7

SIGMA	Factor $\in$ (0,1) used for geometric line searches to decrease relaxation factor. $$\lambda_{new} = \sigma \lambda_{old}$$ See also GEOMS to choose type of line search.	0.5
STPTOL	Stopping tolerance for the step size. $$if: \max_i \left(\frac{\lvert x_{+_i} - x_{k_i} \rvert}{max(\lvert x_{+_i} \rvert , 1/SCALEX_i)} \right) \leq STPTOL \quad stop$$ This governs the step size after the line search or trust region updating; for a tolerance on the Newton step see NSTTOL. One or both may be disabled by setting the tolerance to 0.0. See also STOPCR for a designation of which stopping criteria are to be used.	$\varepsilon_m^{2/3}$
BOUNDL BOUNDU	Vectors containing lower and upper bounds for the components.	$\mp 10^{MAXEXP}$
SCALEF SCALEX	Vectors containing scaling factors for the functions and the components, respectively. These can be set permanently by the user or updated internally (see ITSCLF and ITSCLX).	1.0
HELP	Designates variables to be described using on-line help facility (see below).	NONE

Table 2 Description and Defaults for User-set Variables

2.1 Using NNES's Help Facility

A brief description of the variables in Table 2 may be obtained from the help facility in NNES. The default for the CHARACTER variable HELP is NONE. If a description of any particular variable is desired, set HELP='variable name', e.g., for a description of CAUCHY set HELP='CAUCHY' before the call to NNES. If the help facility is used the program will stop after printing a description of the variable(s).

If any family of variables is desired, set HELP to:

ALLLOG	all logical variables
ALLINT	all integer variables
ALLREA	all real variables
ALLMAT	all arrays (including vectors)

or, if the complete list is desired, set HELP to ALL. The variable name must be left justified, e.g., HELP='ALL' is correct while HELP='   ALL' fails. The help facility is designed to provide reminders to a user familiar with NNES; the descriptions are less detailed than those provided in the manual.

2.2 Making Changes to the Default Settings

If the default settings fail to provide the correct solution, then the following general rules may be useful to decide which of NNES's many options should be changed. This is followed by a series of suggestions for specific situations.

1. Generally the most effective way to find a difficult solution is to **try different starting points**. This will also confirm the uniqueness of a solution or, possibly, reveal multiple solutions. This shows the ultimate weakness of Newton-based nonlinear equations solvers; you must be "close" to the solution before you can find it.

2. **Changing to an undamped Newton's method** is unlikely to find solutions that the default method won't find, although there are some problems where a Newton step just happens to land at the correct spot. One strategy, however, is to let Newton's method run without bounds until a solution is found then adjust the infeasible solution components and run using default settings and bounds or switch to Absolute Newton's method and bounds. The **trust region method** is about as robust as the line search method. Because of different search directions it may find different solutions than the default but not usually.

3. **Bounds** usually greatly affect performance. There is a chance, however, that the program will get stuck at a boundary so it is best to leave bounds off initially then apply if necessary. If you use bounds, try not to use an initial estimate at the bound; leave the program room to move either way.

4. **Scaling** probably won't help if no solution is being found unless the problem is very badly scaled. It does affect performance however. User-selected scaling factors are generally the best. Look at the function values from a detailed output and judge which functions are contributing most to the objective function then set scaling factors, $SCALEF_i$, so that when multiplied by the function a result of $O(1)$ is found.

5. Switching to a **quasi-Newton method** will likely slow down the solution in terms of number of iterations and time, unless the functions are very complex, but decrease the number of function evaluations. However, it occasionally works better with bounded problems in that it has less tendency to get stuck at boundaries because the inaccuracies in the Jacobian result in search directions which don't pull toward the boundary as strongly.

6. The Lee and Lee (1989) **quasi-Newton method** is generally faster than Broyden's (1965) method although it does require extra $O(n^2)$ operations internally; the choice for OMEGA does affect performance. There is little difference in robustness between the two.

7. **RATIOF and MINQNS** can have an effect on performance in **quasi-Newton methods**; they control how often the Jacobian is evaluated explicitly. Try changing RATIOF by ±0.05's and MINQNS by ±1's. They usually won't affect whether you find a solution.

8. The parameter **NARMIJ** is usually best at 1. Forcing descent at each step (by setting NARMIJ to MAXIT) almost always will slow progress and usually not increase robustness. You will likely make slower, steadier progress by increasing NARMIJ.

9. The parameter **ACPTCR** is usually best at 12. In a sense, the aim is to accept as many steps as possible without diverging thus having two step acceptance criteria, i.e., *f* and the natural objective function, increases the percentage of longer, accepted steps.

10. If the program is stopping prematurely because of the step size criterion, try **changing STOPCR to 2** thus only allowing stopping on the basis of the max-norm of the functions nearing zero. The program will still stop if it gets stuck at a local minimum however it will not stop prematurely at "flat" spots in the objective function.

11. For well-behaved problems there is little advantage in calculating **analytical Jacobians** or using central differences over forward differences. Symbolic differentiation software can be useful in generating analytical derivatives if they are desired.

12. **OVERCH** may save problems that are just overflowing. Generally it just delays the inevitable as divergence has set in. The default damped Newton method is the least likely to overflow and the straight Newton's method the most likely. Use **OVERFL** in the function subroutine to prevent exceptions such as divide by zero, invalid arguments ... .

13. If you are getting stuck at boundaries try **reducing CONFAC** by 0.05's as a last resort.

14. **ALPHA** is usually fine at 1.0E-4. In problems with very steep objective functions, thus large directional derivatives, ALPHA can be reduced to near 0. Increasing ALPHA probably won't matter.

15. Within **line search methods**, in terms of solving/not solving the problem
    - switching GEOMS to FALSE usually has little effect
    - switching DEUFLH to FALSE usually has little effect
    - changing MAXNS and MAXQNS usually has little effect
    - changing SIGMA from 0.5 usually has little effect
    - changing LAM0 from 1.0 should only be used as a last resort as it slows performance dramatically. Try using MSTPF, bounds or checks within the function vector to avoid undefined regions.

    All of these parameters affect performance. SIGMA should not be changed dramatically from 0.5. As you near 0 or 1 performance degrades noticeably.

16. Within **trust region methods**, in terms of solving/not solving the problem
    - switching CAUCHY to TRUE usually has little effect
    - if you are having overflow problems at the first step try setting DELTA. Look at a detailed output to see what the DELTA calculated internally is then take some fraction of that
    - changing MAXNS and MAXQNS usually has little effect
    - DELFAC has a more pronounced effect when using Dennis and Schnabel's (1983) updating method. For difficult problems, or those with multiple solutions, changing DELFAC can affect what solution is found.
    - ETAFAC can be increased towards 1. This increases the influence of the steepest descent direction and thus is slightly more conservative.

17. If **progress is slow** in decreasing the objective function, check to see if the Jacobian is singular using detailed output; this may indicate an error in the equations.

To summarize, try changing starting points and methods first. Add constraints if necessary. The trust region method gives more flexibility. For line search methods try the minimization line search. Try increasing NARMIJ and changing ACPTCR to 1. Rarely, varying the parameters listed in 15 and 16 will make the difference. If you haven't found a solution yet, you're not likely to using NNES's parameters. A few suggestions follow which still might help.

2.3 Transforming Problems for Easier Solution

Transforming the problem can lead to easier solution. In general, the objective is to redefine arguments of logarithms... as variables, so constraints can be applied to these variables, and to transform highly nonlinear functions such as exponentials and high-order polynomials.

Consider these two examples from Shacham (1983) and (1986). The first is a one-dimensional problem which has an infeasible range of [.8,1]. The initial estimate is 0.5 and there are two solutions: 0.7574 (feasible) and 1.098 (infeasible).

$$F(x) = x/(1 - x) - 5\ln[0.4(1 - x)/(0.4 - 0.5x)] + 4.45977 = 0 \qquad x \le 1$$

This 1-dimensional problem can be transformed into the following 3-dimensional system:

$$x_1 = x$$
$$x_2 = 1 - x$$
$$x_3 = 0.5x - 0.4$$

$$F_1(\mathbf{x}) = x_1/x_2 - 5\ln(0.4x_2/x_3) + 4.45977 = 0$$
$$F_2(\mathbf{x}) = x_1 + x_2 - 1 = 0$$
$$F_3(\mathbf{x}) = 0.5x_1 + x_3 - 0.4 = 0$$

which has the limits: $x_1 \ge 0$; $x_2 > 0$ and $x_3 > 0$ which are continuous in these ranges and force the original x to be ≤ 1 thus implying its feasibility.

The second example involves redefining variables plus transforming the first equation by taking logarithms. The original version is

$$F_1(\mathbf{x}) = 120x_1 - 75k(1 - x_1) = 0$$
$$F_2(\mathbf{x}) = x_1(873 - x_2) - 11(x_2 - 300) = 0$$

where $\quad k = 0.12\exp[12581(x_2 - 298)/(298x_2)]$

while the modified problem, which no longer has the highly nonlinear exponential term, is

$$x_3 = (1 - x_1)/x_1$$

$$F_1(\mathbf{x}) = \ln(40/3x_3) - [12581(x_2 - 298)/(298x_2)] = 0$$
$$F_2(\mathbf{x}) = x_1(873 - x_2) - 11(x_2-300) = 0$$
$$F_3(\mathbf{x}) = (1 - x_1) - x_3x_1 = 0$$

which has the limit $x_3 > 0$. Notice that the second equation has remained unchanged. For the initial estimates of (0.5,320), NNES fails for the original formulation but using, for example, an initial estimate of 0.1 for x_3, converges quickly to the solution (0.96387, 346.16, 0.0374). The value of x_3 has no meaning but the first two entries are the correct solution to the original problem.

Another way to reparameterize is to change the variables. The most common transformations are to use logarithms and powers of the original variables as new variables. For example, use x^2 instead of x to ensure nonnegativity. Transforming in this manner may cause linear terms in other equations to become nonlinear so the trade-offs must be examined.

Consider equations of the following form with starting estimate (1,1):

$F_1(\mathbf{x}) = x_1x_2^{1/2} - 2x_1 - 3 = 0$
$F_2(\mathbf{x}) = (1 + x_2)x_1 + 3 = 0$

The square root in the first equation causes an error at the second step when using the default setting in NNES. This can be averted by using a lower bound of 0 for x_2 but one could also modify the problem by defining a new variable $x_3 = x_2^{1/2}$ to

$F_1(\mathbf{x}) = x_1x_3 - 2x_1 - 3 = 0$
$F_2(\mathbf{x}) = (1 + x_3^2)x_1 + 3 = 0$

which is easily solved to give the solution (-2.1708, 0.6180) so the solution of the original problem is (-2.1708, 0.3820).

2.4 Using Other Methods

Another possibility is to use the a form of the continuation method. NNES is not set up as a continuation method but a version can be developed using NNES as the core. The objective is to choose a continuation parameter, some variable in the problem which is causing the severe nonlinearity, and transform it in several stages from a level where the system is more readily solvable to its desired level. As an illustration, consider Hiebert's (1980) 2nd chemical engineering problem:

$F_1(\mathbf{x}) = x_1 + x_2 + x_4 - 0.001 = 0$
$F_2(\mathbf{x}) = x_5 + x_6 - 55.0 = 0$
$F_3(\mathbf{x}) = x_1 + x_2 + x_3 + 2x_5 + x_6 - 110.001 = 0$
$F_4(\mathbf{x}) = x_1 - 0.1x_2 = 0$
$F_5(\mathbf{x}) = x_1 - 10000.0x_3x_4 = 0$
$F_6(\mathbf{x}) = x_5 - 5.5\times10^{15}x_3x_6 = 0$

which is very badly scaled because of the 10^{15} in the last equation. One approach would be to replace the exponent 15 with a parameter, call it K, and solve a series of problems where the parameter varies from, say K=1 to K=15, using the results of the previous solution as the starting estimate with K increased by 1. This requires many more iterations overall for this problem, and this type of continuation does so in general, compared to being able to solve in one step.

If no solution has been found yet, it is unlikely that NNES will be able to find it. There are other methods available such as true continuation methods. Several are available through NetLib. General references are the books by Garcia and Zangwill (1981), Morgan (1987) and Allgower and Georg (1990). The last two have FORTRAN code included. Also, Shacham (1986) lists the FORTRAN source code for his simple continuation method.

Another developing area is interval methods which use a different approach than Newton- based methods; see Neumaier (1990) for a description of these methods. These methods do not require the closeness condition which defeats Newton's method; all that is required is an initial interval for each component in which the solution lies. Kearfott (1990) has made an interval bisection method, INTBIS, available through NetLib.

A third alternative are the ABS (Abaffy, Broyden and Spedicato) projection methods as advocated by Abaffy and Spedicato (1989).

At the time of writing, the solvers available from MINPACK, NAG and IMSL are based on the same principles as NNES so are unlikely to succeed where NNES fails.

3. General Description of NNES

Recall, NNES is intended to solve problems of the form:

$$\boldsymbol{F}(\boldsymbol{x})=\boldsymbol{0} \qquad \boldsymbol{F},\boldsymbol{x} \in \mathbb{R}^n \tag{1}$$

$$boundl_i \le x_i \le boundu_i \qquad i = 1,\ldots,n \tag{18}$$

given an initial estimate, $\mathbf{x}_0$. It encompasses four general approaches to solving these systems: Newton's method; Absolute Newton's method; Newton's method with line searches and a trust region method. As each is an extension of the prior method(s) they will be discussed in order. The actual implementation in NNES is more complex than that which will be discussed. The framework is the algorithm described in Dennis and Schnabel (1983). The major adaptations and changes are mentioned in the following text.

3.1 Newton's Method

Newton's method is based upon a truncated Taylor series:

$$\boldsymbol{F}(\boldsymbol{x}) \approx \boldsymbol{F}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\,(\boldsymbol{x} - \boldsymbol{x}_k) \tag{3}$$

where $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix ($\mathbf{J}_{ij}=\partial F_i/\partial x_j$). Setting the left hand side of equation (3) to $\mathbf{0}$ and defining the Newton step, $\mathbf{s}^N = \mathbf{x} - \mathbf{x}_k$, gives the following iterative sequence,

$$\boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{s}^N = -\boldsymbol{F}(\boldsymbol{x}_k) \tag{4}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}^N , \tag{5}$$

which is known as Newton's method. This is the iteration used if NEWTON is set to TRUE with the other default values intact. The major advantage of Newton's method is that it is locally quadratically convergent to a solution $\mathbf{x}_*$, i.e., for any norm,

$$\| \boldsymbol{x}_{k+1} - \boldsymbol{x}_* \| \le c\, \| \boldsymbol{x}_k - \boldsymbol{x}_* \|^2 . \tag{6}$$

The major drawback of Newton's method is that it is not globally convergent so, in practice, it fails too often to be a useful general method. The global methods discussed later try to reach a region where Newton steps may be taken to ultimately achieve quadratic convergence while taking precautions to prevent divergence.

The Jacobian matrix can be evaluated explicitly in this and all other methods in one of four ways: analytically using a user-supplied subroutine (the examples use JACOB); forward differences; backward differences and central differences. The evaluation method is controlled by JACTYP; forward differences are used for the default. Because finding and coding analytical Jacobians is a tedious process, there is the option of checking analytical derivatives by forward differences for "NJACCH" iterations. NNES checks for bounds when using divided differences.

Because it takes *n* additional function evaluations to update a dense Jacobian at every iteration using forward differences to approximate the derivatives, quasi-Newton update methods have been proposed to estimate the Jacobian by a matrix, **B**, using readily available information. Two such methods are available in NNES: the well-known Broyden's (1965) method and that of Lee and Lee (1989). The compromise is that it usually takes more iterations (in some cases many more) to achieve convergence. When the function evaluations are simple it is faster to recalculate the Jacobian at each iteration; this is the choice of the default setup. If a quasi-Newton method is desired, the choice of update method is governed by the parameter JUPDM. Quasi-Newton updates can also be used in the line search and trust region methods described later. The description of how NNES chooses to periodically update the Jacobian using finite differences is given in §3.4.4.

Broyden's method is a multi-dimensional analogue of the secant method for single equations. The update for the approximate Jacobian is

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T}{s_k^T s_k} \quad , \quad \text{where} \quad \begin{aligned} B_0 &= J_0 \\ s_k &= x_{k+1} - x_k \\ y_k &= F(x_{k+1}) - F(x_k) \ . \end{aligned} \tag{7}$$

Lee and Lee added another matrix, **P**, which also must be updated, and the so-called forgetting factor, ω, which is input as OMEGA,

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T P_k}{s_k^T P_k s_k} \quad , \quad \text{where} \quad \begin{aligned} B_0 &= J_0 \\ s_k &= x_{k+1} - x_k \\ y_k &= F(x_{k+1}) - F(x_k) \end{aligned} \tag{8}$$

$$P_{k+1} = P_k - \frac{P_k s_k s_k^T P_k}{\omega^{k+3} + s_k^T P_k s_k} \quad , \quad \text{where} \quad \begin{aligned} &P_0 = I \\ &\omega \ \text{is given} \ . \end{aligned} \tag{9}$$

The disadvantage of this method is that it requires storage of another $n \times n$ matrix. Performance is affected by the choice of ω.

The only information used in the updates is the current step, $\mathbf{s}_k$, and the difference of the function vectors, $\mathbf{F}(\mathbf{x}_{k+1})$ - $\mathbf{F}(\mathbf{x}_k)$. In NNES, a scale-invariant version of the above methods is implemented. Also, only the rows which make a significant change to the current **B** matrix are added. If the problem is suspected to be sensitive to the initial steps, the option is available to delay the start of quasi-Newton updates by setting the variable NIEJEV, the number of initial steps in which the Jacobian will be evaluated explicitly.

As presented above, the updated approximate Jacobian then has to be factored (using the QR decomposition in NNES) before the linear system, equation (4), with **B** replacing **J**, can be solved. Instead of updating the Jacobian and factoring, the default quasi-Newton update method, set by QNUPDM, is to update the QR factors directly. This is a time-saving step and usually leads to exactly the same solution as formulating the approximate Jacobian then factoring. Only in very sensitive problems, where numerical accuracy in the last few digits is important, will this choice make a difference. The other utility of using the unfactored update is that the user may want to see what is happening to **B** in case of failure, e.g., are some entries becoming extremely large or are sparsity patterns within the actual Jacobian being lost.

The other options for Newton's method are the stopping criteria. These are based on either step size or function value; the choice is determined by setting STOPCR. There are, in fact, two step size criteria. The first is governed by the constant NSTTOL and the second by STPTOL. In Newton's method they measure essentially the same thing, as step length is not modified by a line search or trust region, so only the first will be shown. The difference will be explained in §3.2. Occasionally, very small steps will be taken as the iterations pass through a *n*-dimensional flat spot; because NNES converges with respect to step size does not necessarily mean that a solution has been found. On the other hand, the step size criterion can save needless steps refining accuracy to greater than that needed.

The Newton step stopping criterion looks at the scaled max-norm of the Newton step calculated in equation (4).

$$If: \max_i \left(\frac{|s_i^N|}{max(\ |x_{k_i}|\ ,\ 1/SCALEX_i\ )} \right) \le NSTTOL(1 + \| D_x x_k \|)\ then\ stop \qquad \textbf{(10)}$$

$$where\ D_x = diag(SCALEX).$$

The function value stopping criterion is based on the max-norm of the scaled function vector. Scaling is not usually used in Newton's method, however this stopping criterion is used for all methods so scaling is included here.

$$If: \max_i \left(SCALEF_i \cdot |F_i| \right) \le FTOL\ \ then\ stop. \qquad \textbf{(11)}$$

The Absolute Newton's method is similar to Newton's method except that all the steps are forced to be inside a feasible region as shown in Figure 1 (p2). The options available are the same as for Newton's method, except that lower and/or upper bounds must be set using the vectors BOUNDL and BOUNDU. Though it is not particularly elegant, Morgan (1987) showed that this method can be effective for constrained problems. In simple terms, the iterations bounce around inside a user-defined box until, hopefully, a point is found at which the iterations begin to converge. In fact, the method in NNES is an extension of Morgan's method which has only lower bound of **0**, hence the name "Absolute".

The difficulty with Newton's method and Absolute Newton's method is that they are only local methods. Convergence proofs for Newton's methods, given for example in Dennis and Schnabel (1983) and Ortega and Rheinboldt (1970), describe the conditions under which Newton's method is guaranteed to converge. However, these conditions are so restrictive that they are of little use in practice. Newton's method will converge in cases where theses conditions are not met. There is also no practical way of checking to see if the criteria are met *a priori*.

Because of this limited range of convergence, so-called global methods have been developed which are designed to converge from any starting point although they do not necessarily find a correct (feasible) solution. They don't necessarily stop at a solution at all; they most stop at a local minimum of the objective function. The term "global" described here has a different connotation than that for global optimization; it certainly does not mean that there is any guarantee of finding a solution. Two global methods are available in NNES; a line search method described in §3.2 and a trust region method described in §3.3. The switch between these two fundamental methods is made using the parameter LINESR. The default is to use a line search method (LINESR is TRUE).

3.2 Line Search Method

The line search method expands on the strict Newton method. The point generated by equations (4) and (5), $\mathbf{x}_{k+1}$, is not necessarily accepted as a new point; it must meet some acceptance criterion. To do this, some measure of merit of the current point is required, i.e., is this point preferred versus the previous one (or, possibly, in the case of NNES, ones). A common method, which is available in NNES, is to define an objective function, *f*.

$$f = (1/2)\boldsymbol{F}(\boldsymbol{x}_k)^T\boldsymbol{F}(\boldsymbol{x}_k) \tag{12}$$

where the factor 1/2 is for convenience only. When *f* is at its global minimum, 0, a solution has been found, however, local minima of *f* can exist which are not solutions; this is the main downfall of global Newton-based methods. This objective function is used because it has the property that the Newton step generated using equation (4) is a descent step for *f*, i.e., the directional derivative, $\nabla f \cdot \mathbf{s}^N$, is less than zero. This allows the use of the Armijo (1966) criterion for step acceptance.

$$\begin{aligned} &\textit{If:}\ \ f(x_k + \lambda s^N) \le f(x_k) + \alpha\lambda\nabla f(x_k)^T s^N \ , \quad \textit{accept step} \\ &\textit{where}\ \begin{array}{l} \lambda \ \textit{is the relaxation factor} \in (0,1] \\ \alpha \ \textit{is a constant} \in (0,1]. \end{array} \end{aligned} \tag{13}$$

Since the Armijo constant, α, is usually very small when used in nonlinear equations (not necessarily so for other minimizations), this usually amounts to a strict decrease in the objective function being required at each step unless $|\nabla f \cdot \mathbf{s}^N|$ is very large. The Armijo constant is input as ALPHA. Since $\nabla f \cdot \mathbf{s}^N < 0$, there must be a range of the relaxation factor, λ, small enough so that the Armijo criterion can be met. To find an acceptable λ, a line search is conducted in which λ is decreased at each step by one of two mechanisms.

The simplest mechanism, the geometric line search, is activated by setting GEOMS to TRUE. After each failure, i.e., the objective function at the trial point is not an improvement versus the current objective function, λ is reduced by a factor, σ, until the Armijo criterion, equation (13), is satisfied, i.e.,

$$\lambda_{new} = \sigma\lambda_{old}. \tag{14}$$

Most commonly, the initial relaxation factor in any iteration, call it λ_0, is 1.0, corresponding to a full Newton step although this can be changed in NNES by setting LAM0 to a positive number less than 1.0. The standard reduction factor at each step of the line search is 2.0, (thus σ=0.5).

The other mechanism for reducing λ consists of finding the minimum of a quadratic in λ fit through $f(\mathbf{x}_k)$ and $f(\mathbf{x}_k+\lambda_0\mathbf{s}^N)$ using the directional derivative, $\nabla f \cdot \mathbf{s}^N$,

$$\lambda_{\min} = -\frac{\lambda_0^2\nabla f^T s^N}{2[f(x_k + \lambda_0 s^N) - f(x_k) - \lambda_0\nabla f^T s^N]}\ . \tag{15}$$

If $\lambda_{min} \in [.1\lambda_0,.5\lambda_0]$ then $\lambda_{new} = \lambda_{min}$. Otherwise λ_{new} is the nearest end-point. Similarly, for subsequent steps, the minimum of a cubic is found using the two most recent trial points, the current point and the directional derivative with the same restrictions applying. Details of this calculation can be found in

Dennis and Schnabel (1983). Usually this extra work is of no benefit, so NNES takes the geometric line search as its default.

NNES allows three important variations from this general setup: a nonmonotonic extension of the Armijo criterion as proposed by Grippo, Lampariello and Lucidi (1986) has been added; the initial relaxation factor in each line search, λ_0, can be estimated using a variation of the method given by Deuflhard (1974,1975) and a second objective function can be used. These three extensions are described below.

The nonmonotonic extension entails comparing the objective function at the trial point to the maximum of a user-selected number (MGLL in NNES) of previous trial points. Accept a step if

$$f(x_k + \lambda s^N) \leq \max_{0 \leq j \leq m(k)} f(x_{k-j}) + \alpha \lambda \nabla f(x_k)^T s^N \tag{16}$$

$$m(0) = 0 \quad and \quad 0 \leq m(k) \leq min[m(k-1)+1, MGLL] \;, k \geq 1 . \tag{33}$$

For example, if MGLL=10, the objective function at the first trial point is compared to that of the starting point, exactly as it would be in a monotonic method using Armijo. Once a successful first point has been found, the second point is established when its objective function is less than the maximum of the previous two (which must be the first). This pattern continues until you are comparing to the previous 10 objective functions. From this point on, the comparison is made to the maximum of the 10 most recent objective functions. The important consequence of this is that you need not descend at every step which implies that more longer steps are taken and thus, generally, quicker convergence is achieved.

NNES allows, through NARMIJ (number of Armijo steps), forcing any number of strict descent steps at the start of the problem before the nonmonotonic accumulation begins. If complete monotonicity is desired then NARMIJ can be set to MAXIT; this is the rule used by most Newton-based nonlinear equation solvers. The relevant objective function values are stored in the tracking vector FTRACK.

If a quasi-Newton method is being used, then the FTRACK vector is re-initialized whenever the Jacobian is updated explicitly. When the program restarts, it does so at the point with the smallest objective function, *f*, found so far, as if the new starting point were an initial estimate. NARMIJ then prescribes the number of Armijo steps since the last explicit Jacobian evaluation, i.e., the last restart. Details of how NNES chooses to restart are given in §3.4.4.

To establish the initial relaxation factor in any iteration, λ_0, NNES offers two choices. The simplest is to set it to LAM0. Usually this is 1.0 to allow full Newton steps to be taken. LAM0 may be set to positive values less than 1 to avoid numerical problems in extremely nonlinear problems but this strategy precludes quadratic convergence. There are other mechanisms such as boundaries, setting MAXSTP, switching to trust region methods and using OVERFL in the function vector which are better ways to avoid such problems.

The second method, devised by Deuflhard (1974, 1975), relies on calculating a vector, designated as $\bar{\mathbf{s}}$ in NNES, using the Jacobian from the previous point and the new function vector:
and then defining μ_k by

$$\overline{s_{k+1}} = J(x_k)^{-1} F(x_{k+1}) \tag{18}$$

$$\mu_k = \frac{\| s_{k-1}^N \|_2}{\| \overline{s_k} - s_k^N \|_2} \lambda_{k-1} \tag{19}$$

so, finally, λ_0 at iteration k is estimated by

$$\lambda_0 = \begin{cases} 1 & \text{if } \mu_k > 0.7 \\ \mu_k & \text{if } \mu_k \le 0.7 \end{cases} . \tag{20}$$

Since the original version of Deuflhard's initialization was designed for monotonically decreasing objective function values, NNES uses a more aggressive version of the same procedure because of its nonmonotonic searches:

$$\lambda_0 = \begin{cases} 1 & \text{if } \mu_k > 0.1 \\ 0.1 & \text{if } \mu_k \le 0.1 \end{cases} . \tag{21}$$

The choice of starting at at a user-supplied λ_0 or using Deuflhard's method is set by DEUFLH which has the default of TRUE; in general, Deuflhard's method is slightly faster.

Deuflhard (1974, 1975) also proposed a second step acceptance criterion, known as the *natural* objective function, based on $\overline{\mathbf{s}}$.

$$\textit{If}: \quad \| \overline{s_{k+1}} \|_2 \le \| s_k^N \|_2 \qquad \textit{accept step}. \tag{22}$$

This criterion can be used as well as the function value criterion, *f*, but not alone, by setting ACPTCR to 12.

The natural objective function also works in the nonmonotonic sense with the previous Newton step lengths being stored in STRACK. If ACPTCR is 12, then the number of previous function values compared for the nonmonotonic descent is the same for both criteria, i.e., MGLL is the same for both. This leads to some peculiar patterns in objective function value as a step accepted by step size only may have a huge *f* which is dutifully stored in FTRACK for comparisons with future *f*'s. The maximum value in STRACK is restricted, however, to be no greater than the initial Newton step length.

Finally, line search methods have an additional stopping criterion to equations (10) and (11) based on the step length <u>after</u> the line search,

$$\textit{if}: \quad \max_i \left(\frac{| x_{k+1_i} - x_{k_i} |}{\max(| x_{k+1_i} | , 1/SCALEX_i)} \right) \le STPTOL \quad \textit{then stop}. \tag{23}$$

If a step length stopping criterion is to be used, then the user must decide whether it is to be applied before the line search, after the line search or both, and thus set NSTTOL and STPTOL accordingly. A line search failure and a resulting halt in execution will occur if more than the allowed number of line search steps are taken; see §3.4.4 for details.

Other features, common to line search and trust region methods, are described in §3.4. Computationally, line search methods are simpler than trust region methods but they have less flexibility.

3.3 Trust Region Method

The trust region method attempts to combine the better global convergence properties of searches in the steepest descent direction with the local quadratic convergence of the Newton method. It is similar to the well-known Levenberg-Marquardt method for nonlinear least squares. The mechanism which is used in NNES is the hybrid (single dogleg) method of Powell (1970) which was extended by Dennis and Mei (1979) to the double dogleg method. A region is set, through DELTA, which represents a distance (radius) in *n*-space, in which some confidence is held that the quadratic model found by substituting equation (3) into the definition of the objective function,

$$f = 1/2\,F(x_k)^T F(x_k) + (J(x_k)^T F(x_k))^T s + 1/2\, s^T (J(x_k)^T J(x_k))\, s \tag{24}$$

is a good representation of the actual objective function. Two steps are established: the Newton step, $\mathbf{s}^N$, which goes to the minimum of the quadratic model and $\mathbf{s}^{CP}$ which goes to the minimum of the quadratic model in the direction of steepest descent of the objective function. This minimum is called the Cauchy point. It can be shown that $\|\mathbf{s}^{CP}\|_2 \le \|\mathbf{s}^N\|_2$, (thus the choice for the initial trust region radius, δ_0, via CAUCHY). Instead of conducting a line search in which the step length in the Newton direction is shortened, the trust region size, DELTA, is decreased until a satisfactory point is found.

Consider having found $\mathbf{s}^N$ at iteration k and given a δ_k. One of three situations can arise: $\mathbf{x}_k + \mathbf{s}^N$ is within the trust region; $\mathbf{x}_k + \mathbf{s}^{CP}$ is outside the trust region or, the case in which the single or double dogleg step must be used, $\|\mathbf{s}^{CP}\|_2 \le \delta_k \le \|\mathbf{s}^N\|_2$. For the single dogleg case the new point is at the intersection of the line from the Cauchy point to the Newton step with the border of the trust region. The double dogleg step is similar except that only a fraction of the Newton step is used. This case is illustrated in Figure 2.

The variable ETAFAC controls how far along the Newton step the second dogleg is. When ETAFAC is 1 the single dogleg is being used. As it approaches 0 the more influence the Newton step has in determining the trust region step. The default value 0.2 was suggested by Dennis and Mei (1979) and used by Dennis and Schnabel (1983). Note that an ETAFAC of zero corresponds to a minimum fraction, $\gamma\mathbf{s}^N$, of the Newton step being used, not to no contribution.

Two other situations can occur. If the Newton step is within the trust region then the Newton point is tested and the trust region size adjusted immediately to the length of the Newton step. If this point is a failure no line search is conducted; the trust region is decreased and a new point found. On the other hand, if the Cauchy point is outside the trust region, then the intersection of the steepest descent direction and the trust region becomes the trial point. As the trust region size decreases this final situation must occur thus is it is known that some positive step in the steepest descent direction can always be found which produces the required decrease in the objective function.

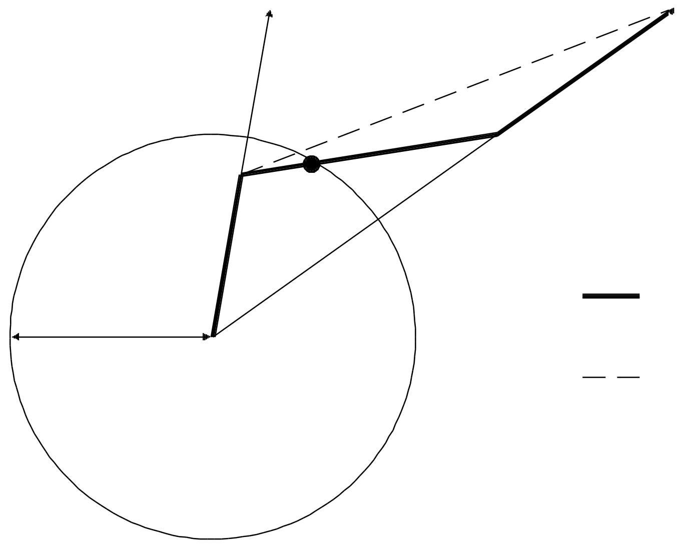

Figure 2 Schematic Representation of the Double Dogleg Step

The trial point, $\mathbf{x}_{k+1}$, is chosen at the first intersection with the trust region radius of a line drawn from the current point, $\mathbf{x}_k$, to the Cauchy point then to a point in the direction of the Newton step at a distance greater than $\gamma \|\mathbf{s}^N\|_2$ and finally to the end of the Newton step. If the Newton step is within the trust region radius then $\mathbf{x}_{k+1}^N$ is chosen as the trial point. As the trust region shrinks, the Cauchy point must eventually be outside the trust region so successive points in the steepest descent direction are tried.

The most complicated feature of trust region methods is updating δ. In NNES updating is governed by the objective function, *f*, even if the step is accepted by the step size criterion. If the first trial point is an improvement over the previous, is not the Newton point and there is close agreement between the model and the objective function then the trust region size is increased and a new trial point tested. There are two methods, selected via TRUPDM, to accomplish this: Powell's (1970) method and modified version that of Dennis and Schnabel (1983). Usually δ is multiplied DELFAC (default 2) if agreement is good but NNES uses a much more aggressive approach if a nonmontonic search is being conducted. When δ is increased it is taken to the maximum length possible, the length of the Newton step. If this point fails the trust region size works back down to a length no less than that of the original acceptable step.

If the original trial point is a success, but not an improvement over the previous f, then δ is reduced by DELFAC. If it is a failure then there are two options similar to those for line searches, controlled by GEOMS: reduce δ by a factor of DELFAC, or minimizing a quadratic, with the restriction $\delta_{new} \in [0.1\delta_{old}, 0.5\delta_{old}]$. NNES stops automatically if the step length becomes too small, i.e.,

$$if\colon\ \max_i \left(\frac{|s_i|}{max(\ |x_{+_i}|\ ,\ 1/SCALEX_i\ )} \right) \leq STPTOL \quad stop\,. \tag{25}$$

3.4 Other Features

3.4.1 Lower and Upper Bounds

Lower and upper bounds are used to impose two types of constraints: absolute for variables which must stay within a region for mathematical reasons such as logarithms and square roots and physical for those which represent physical quantities which have meaning only in certain regions such as mole fraction and temperature. These are handled the exactly same way in NNES. The distinction is made because it may not be necessary constrain physical variables. By allowing them to trespass into restricted regions the solution may be found more quickly. Constraints are handled in a similar way for both line search and trust region methods. The procedure is different from that described for the Absolute Newton's method.

Consider the line search case where the first trial point is the full Newton step. When a constraint is violated the first response is to bring the violating components back into the feasible region while leaving the nonviolating components alone thus departing from the Newton direction. For each violating component the ratio of the distance from the trial point to its boundary to the length of that component of the Newton step (called RATIO) is found. Instead of projecting onto the boundary or reflecting in the boundary, the violators are taken only the fraction CONFAC (default 0.95) of the way from the current point to the boundary if RATIO ≤ 0.5 or (1/2)CONFAC of the way if RATIO > 0.5. as illustrated in Figure 3. This creates a buffer zone near the constraints aimed at reducing the chances of getting stuck at a constraint.

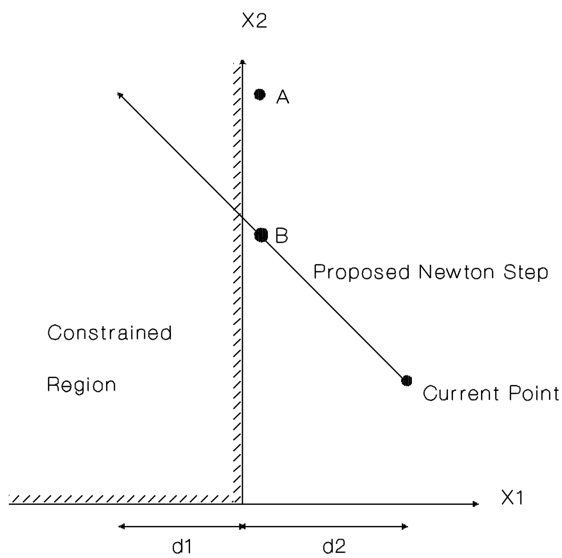

Figure 3 Handling Constraints When RATIO is Greater Than 0.5
RATIO = d1/(d1 + d2)

In the above example component x_1 violates a lower bound. The ratio d1/(d1 + d2) is less than the cutoff, 0.5, so the modified point, A, is set. The directional derivative to A is checked. If it is positive then A is disregarded and a line search begins at B. Otherwise A is checked first and if it is not acceptable then, again, the line search begins at B. Notice that A and B are not on the boundary; for component x_1, they are only 95% of the distance from the current point to the first violated constraint.

3.4.2 Scaling

Scaling can be used for both the variables and the functions. For variable scaling, the objective is to make the solution have components of comparable magnitude. The best way to do this is to have reasonably scaled equations but it can also be done internally via the vector SCALEX which represents a diagonal scaling matrix. Consider a simple case in which the units for x_1 are seconds where milliseconds would be more appropriate and the units for x_2 are millimeters where meters would be better. In this case the scaling matrix would be:

$$D_x = diag\{SCALEX\} = \begin{bmatrix} 10^3 & 0 \\ 0 & 10^{-3} \end{bmatrix} . \qquad (26)$$

Variable scaling does not change the Newton step but it affects the trust region step since the steepest descent direction is scale dependent. It also changes the condition number estimation; a well-scaled matrix is better conditioned and thus the likelihood of having to take remedial action for ill-conditioned Jacobians is reduced. The other main effect of variable scaling is on stopping criteria. Recall the stopping criterion for the Newton step size and note how it is dependent on scale:

$$if: \max_i \left(\frac{|s_i^N|}{max(\ |x_{k_i}|\ ,\ 1/SCALEX_i\ )} \right) \le NSTTOL(1 + \|D_x x_k\|) \ then\ stop \quad where\ D_x = diag(SCALEX) . \qquad (27)$$

Function scaling attempts to normalize the different magnitudes of the functions, F_i, so that one does not dominate in determining the objective function and thus control whether a step is accepted. Hiebert's (1980) 2nd chemical engineering problem is very badly scaled because of F_6. At an initial estimate of (1,...,1) it is 15 orders of magnitude greater than any other function thus a small scaling factor of, say, 10^{-8} in $SCALEF_6$ helps bring it in line with the others.

$F_1(\mathbf{x}) = x_1 + x_2 + x_4 - 0.001 = 0$
$F_2(\mathbf{x}) = x_5 + x_6 - 55.0 = 0$
$F_3(\mathbf{x}) = x_1 + x_2 + x_3 + 2x_5 + x_6 - 110.001 = 0$
$F_4(\mathbf{x}) = x_1 - 0.1x_2 = 0$
$F_5(\mathbf{x}) = x_1 - 10000.0x_3x_4 = 0$
$F_6(\mathbf{x}) = x_5 - 5.5\times10^{15}x_3x_6 = 0$

As with variable scaling, function scaling has no effect on the Newton step but affects the steepest descent direction.

Scaling factors can be set by the user or calculated internally. The adaptive scaling mechanism is to look at the rows of the Jacobian matrix for the components of SCALEF and to the columns for SCALEX and to choose the factor so that multiplying it by the corresponding row or column would leave the largest magnitude being 1. Adaptive scaling, controlled by ITSCLF and ITSCLX, in this manner is still questionable; often it has little effect, sometimes it improves performance and sometimes it is devastating. Thoughtful user selection usually works better; Morgan (1987) gave the algorithm and code for one method which is applicable only to polynomial systems.

When scaling is used, both the scaled and unscaled results are reported in the final output. The iteration by iteration output shows the unscaled components and function values as these are what are taken to and returned from the function vector subroutine respectively.

3.4.3 Using OVERFL in the Function Vector

The call to the function vector contains an extra logical variable, OVERFL, which can be used to prevent mathematical errors such as overflows, logarithms of negative numbers ... from halting execution. The user can set up their own check of any argument before the function is evaluated then set OVERFL to TRUE if a problem would arise and return. Both the line search and trust region methods will compensate but Newton's method will not work. In Absolute Newton's method, the bounds should prevent these potentially fatal excursions from happening.

3.4.4 Choosing When to Re-evaluate the Jacobian in Quasi-Newton Methods

The attraction of quasi-Newton methods is that n function evaluations are not required at every iteration to evaluate the Jacobian via finite differences. The problem is that the estimated Jacobian can become inaccurate, thus the need for periodic explicit Jacobian evaluation. In fact, the step generated using a quasi-Newton may not be a descent direction for the objective function f. NNES uses two criteria: success or failure of the line search and rate of progress of the objective function value to decide when to update explicitly.

Consider the line search method; the process is analogous for the trust region method. After the Jacobian is evaluated explicitly, i.e., through finite differences or user-supplied subroutine, a maximum of MAXNS (default 50) steps are allowed in the line search. If no improvement is found then the run stops. After a quasi-Newton update, MAXQNS (default 10) steps are allowed. The lesser number is an attempt to limit wasting function evaluations in a search direction which will not lead to significant improvement. If the line search fails, then the program returns to the best point so far, in terms of objective function f, and restarts as if it were an initial estimate.

If line searches aren't failing there is a mechanism based on improvement of the objective function for restarting. First, MINQNS establishes a minimum number of quasi-Newton steps which must be taken before the Jacobian is allowed to be explicitly re-evaluated. This can be overridden by failures in the line search mentioned above. At each iteration the new objective function value is compared to its predecessor:

$$\textit{if}: \quad \frac{f_{new}}{f_{old}} > RATIOF \qquad \textit{then}: \quad NFAIL = NFAIL + 1 \tag{28}$$

$$\textit{if}: \quad \frac{f_{new}}{f_{old}} < 0.01 \qquad \textit{then}: \quad NFAIL = NFAIL - 1 \tag{29}$$

If, at any time, NFAIL > MINQNS then, as above, the program restarts. The quasi-Newton method can be sensitive to both RATIOF and MINQNS. If n is small you may want to decrease MINQNS thus forcing more explicit Jacobians to be used. Unless the function evaluations are complicated, explicitly evaluating the Jacobian at each iteration using finite differences will probably be faster than using either quasi-Newton method; this is the default in NNES.

3.4.5 Handling Singular Jacobians

If the Jacobian becomes singular or if its condition number becomes too high, (> $\varepsilon_m^{-2/3}$) then, if BYPASS is FALSE, the remedial action described in Dennis and Schnabel (1983) is taken. If equation (3) is multiplied through by $\mathbf{J}^T$ then the following equation arises:
which can be perturbed on the left hand side by adding to the diagonal a fraction of the 1-norm of $\mathbf{J}^T\mathbf{J}$

$$J(x_k)^T J(x_k) s^N = -J(x_k) F(x_k) = -\nabla f(x_k) \quad (30)$$

and solving for $\mathbf{s}^N$ using the Cholesky decomposition:

$$(J(x_k)^T J(x_k) + \mu I) s^N = -J(x_k) F(x_k) = -\nabla f(x_k)\,. \quad (31)$$

This procedure saves the program from stopping at singular Jacobians but the search direction generated may not be near the Newton step if the problem is badly scaled. For this reason, one can bypass this procedure for ill-conditioned Jacobians by setting BYPASS to TRUE. The resulting Newton step may not be accurate but may be better than that found by the remedial action. In cases where ill-conditioned Jacobians are encountered setting scaling factors should be considered to reduce the condition number.

3.4.6 Overflow Checking using OVERCH

Switching OVERCH to TRUE forces checking at some key locations for potential overflows. If one is detected $\pm 10^{MAXEXP}$ is substituted where appropriate and a warning issued. This slows down computation somewhat and is intended to save those problems which would overflow only slightly. If divergence has set in this procedure will only delay the inevitable.

References

1. Abaffy, J. and Spedicato, E.; *ABS Projection Algorithms*; Wiley, New York, NY (1989)
2. Allgower, E.L. and Georg, K.; *Numerical Continuation Methods*; Springer-Verlag, New York, NY (1990)
3. Armijo, L.; "Minimization of functions having Lipschitz-continuous first partial derivatives"; *Pacific J. Math.,* **16**, 1-3 (1966)
4. Broyden, C. G.; "A class of methods for solving nonlinear simultaneous equations"; *Math. Comp.*, **19**, 577-593 (1965)
5. Dennis, J. E. and Mei, H. H. W.; "Two new unconstrained optimization algorithms which use function and gradient values"; *J. Optim. Theory Appl.*, **28**, 453-482 (1979)
6. Dennis, J. E. and Schnabel, R. B.; *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*; Prentice-Hall, Englewood Cliffs, NJ (1983)
7. Deuflhard, P.; "A modified Newton Method for the solution of ill-conditioned systems of nonlinear equations with applications to multiple shooting"; *Numer. Math.*, **22**, 289-315 (1974)
8. Deuflhard, P.; "A relaxation factor for the modified Newton method"; in *Optimization and Optimal Control*, Bulirsch, R., Oettli, W. and Stoer, J., eds., Lecture Notes in Math., **472**, 59-73, Springer, New York, NY (1975)
9. Garcia, C.B. and Zangwill, W.I.; *Pathways to Solutions, Fixed Points and Equilibria*; Prentice-Hall, Englewood Cliffs, NJ (1981)
10. Grippo, L., Lampariello, F. and Lucidi, S.; "A nonmonotone line search technique for Newton's method"; *SIAM J. Numer. Anal.*, **23**, 707-716 (1986)
11. Hiebert, K. L.; "A comparison of software which solves systems of nonlinear equations"; *Sandia Tech. Rep. SAND*80-0181, Sandia National Laboratories, Albequerque, New Mexico (1980)
12. Kearfott, R.B. and Novoa III, M.; "Algorithm 681 INTBIS, a portable interval Newton /bisection package"; *ACM Trans. Math. Soft.*, **16**, 152-157
13. Lee, J. and Lee, W. K.; "Least squares updates for quasi-Newton methods in process design and simulation"; *Comp. Chem. Eng.*, **13**, 717-723 (1989)
14. Neumaier, A.; *Interval Methods for Systems of Equations*; Cambridge University Press, New York, NY (1990)
15. Ortega, J.M. and Rheinboldt, W.C.; *Iterative Solutions for Nonlinear Equations in Several Variables*; Academic Press, New York, NY (1970)
16. Powell, M. J. D.; "A FORTRAN subroutine for solving systems of nonlinear algebraic equations"; in *Numerical Methods for Nonlinear Equations*, Rabinowitz, P., ed., Gordon Breach, New York, NY (1970)
17. Shacham, M.; "Recent developments in solution techniques for systems of non-linear equations"; Paper presented at the second International Conference on Foundations of Computer Aided Process Design, Snowmass, CO, 19-24 June (1983)
18. Shacham, M.; "Numerical solution of constrained nonlinear algebraic equations"; *Inter. J. Num. Methods in Eng.*, **23**, 1455-1481 (1986)

Appendix 1 Generic Driver

```
      PROGRAM NNESDR
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (  MGLL=     10,
   $               N=         ?,
   $               NUNIT=     ?)
      DOUBLE PRECISION
   $               JAC(N,N)  ,LAM0       ,MCHEPS     ,MSTPF    ,
   $               NSTTOL
      INTEGER      ACPTCR    ,OUTPUT     ,QNUPDM     ,STOPCR   ,
   $               SUPPRS    ,TRMCOD     ,TRUPDM
      DIMENSION    A(N,N)    ,BOUNDL(N)  ,BOUNDU(N)  ,DELF(N)  ,
   $               FSAVE(N)  ,FTRACK(0:MGLL-1)       ,FVEC(N)  ,
   $               FVECC(N)  ,H(N,N)     ,HHPI(N)    ,PLEE(N,N),
   $               RDIAG(N)  ,S(N)       ,SBAR(N)    ,SCALEF(N),
   $               SCALEX(N) ,SN(N)      ,SSDHAT(N)  ,
   $               STRACK(0:MGLL-1)      ,VHAT(N)    ,WV1(N)   ,
   $               WV2(N)    ,WV3(N)     ,WV4(N)     ,XC(N)    ,
   $               XPLUS(N)  ,XSAVE(N)
      LOGICAL      BYPASS    ,CAUCHY     ,DEUFLH     ,GEOMS    ,
   $               LINESR    ,MATSUP     ,NEWTON     ,OVERCH   ,
   $               WRNSUP
*********************************************************************
*                                                                   *
*       RENAME FUNCTION AND/OR JACOBIAN                             *
*                                                                   *
*********************************************************************
      EXTERNAL FCN,JACOB
      COMMON/NNES_1/MATSUP
      COMMON/NNES_2/WRNSUP
      COMMON/NNES_3/BYPASS
      COMMON/NNES_4/NFETOT
      OPEN(UNIT=NUNIT,FILE='NNESDR.OUT',STATUS='UNKNOWN')
*********************************************************************
*                                                                   *
*       INSERT INITIAL ESTIMATES                                    *
*                                                                   *
*********************************************************************
      CALL SETUP(ABSNEW ,CAUCHY ,DEUFLH ,GEOMS  ,LINESR ,
   $           NEWTON ,OVERCH ,ACPTCR ,ITSCLF ,ITSCLX ,
   $           JACTYP ,JUPDM  ,MAXEXP ,MAXIT  ,MAXNS  ,
   $           MAXQNS ,MINQNS ,N      ,NARMIJ ,NINITN ,
   $           NJACCH ,OUTPUT ,QNUPDM ,STOPCR ,SUPPRS ,
   $           TRUPDM ,ALPHA  ,CONFAC ,DELTA  ,DELFAC ,
   $           ETAFAC ,FDTOLJ ,FTOL   ,LAM0   ,MCHEPS ,
   $           MSTPF  ,NSTTOL ,OMEGA  ,RATIOF ,SIGMA  ,
   $           STPTOL ,BOUNDL ,BOUNDU ,SCALEF ,SCALEX  ,
   $           HELP   )
```

```
*******************************************************************
*                                                                 *
*       INSERT CHANGES TO DEFAULT SETTINGS                        *
*                                                                 *
*******************************************************************
        CALL NNES(ABSNEW ,CAUCHY ,DEUFLH ,GEOMS  ,LINESR ,
     $            NEWTON ,OVERCH ,ACPTCR ,ITSCLF ,ITSCLX ,
     $            JACTYP ,JUPDM  ,MAXEXP ,MAXIT  ,MAXNS  ,
     $            MAXQNS ,MGLL   ,MINQNS ,N      ,NARMIJ ,
     $            NIEJEV ,NJACCH ,NJETOT ,NUNIT  ,OUTPUT ,
     $            QNUPDM ,STOPCR ,SUPPRS ,TRMCOD ,TRUPDM ,
     $            ALPHA  ,CONFAC ,DELTA  ,DELFAC ,EPSMCH ,
     $            ETAFAC ,FCNNEW ,FDTOLJ ,FTOL   ,LAM0   ,
     $            MSTPF  ,NSTTOL ,OMEGA  ,RATIOF ,SIGMA  ,
     $            STPTOL ,A      ,BOUNDL ,BOUNDU ,DELF   ,
     $            FSAVE  ,FTRACK ,FVEC   ,FVECC  ,H      ,
     $            HHPI   ,JAC    ,PLEE   ,RDIAG  ,S      ,
     $            SBAR   ,SCALEF ,SCALEX ,SN     ,SSDHAT ,
     $            STRACK ,VHAT   ,WV1    ,WV2    ,WV3    ,
     $            WV4    ,XC     ,XPLUS  ,XSAVE  ,HELP   ,
     $            FCN    ,JACOB  )
        STOP
        END

        SUBROUTINE FCN(OVERFL,N,FVEC,XC)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DIMENSION FVEC(N),XC(N)
        COMMON/NNES_4/NFETOT
        LOGICAL OVERFL
        OVERFL=.FALSE.
        NFETOT=NFETOT+1
*******************************************************************
*                                                                 *
*       INSERT FUNCTIONS                                          *
*                                                                 *
*******************************************************************
        RETURN
        END
        SUBROUTINE JACOB(OVERFL,N,JAC,XC)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DOUBLE PRECISION JAC(N,N)
        DIMENSION XC(N)
        LOGICAL OVERFL
        OVERFL=.FALSE.
*******************************************************************
*                                                                 *
*       INSERT ANALYTICAL JACOBIAN IF DESIRED                     *
*                                                                 *
*******************************************************************
        RETURN
        END
```

Appendix 2 Rosenbrock's Banana

Calling Program

```
      PROGRAM EX1
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (  MGLL=     10,
     $             N=         2,
     $             NUNIT=    10)
      DOUBLE PRECISION
     $             JAC(N,N)   ,LAM0        ,MCHEPS      ,MSTPF     ,
     $             NSTTOL
      INTEGER      ACPTCR     ,OUTPUT      ,QNUPDM      ,STOPCR    ,
     $             SUPPRS     ,TRMCOD      ,TRUPDM
      DIMENSION    A(N,N)     ,BOUNDL(N)   ,BOUNDU(N)   ,DELF(N)   ,
     $             FSAVE(N)   ,FTRACK(0:MGLL-1)         ,FVEC(N)   ,
     $             FVECC(N)   ,H(N,N)      ,HHPI(N)     ,PLEE(N,N),
     $             RDIAG(N)   ,S(N)        ,SBAR(N)     ,SCALEF(N),
     $             SCALEX(N)  ,SN(N)       ,SSDHAT(N)   ,
     $             STRACK(0:MGLL-1)        ,VHAT(N)     ,WV1(N)    ,
     $             WV2(N)     ,WV3(N)      ,WV4(N)      ,XC(N)     ,
     $             XPLUS(N)   ,XSAVE(N)
      LOGICAL      BYPASS     ,CAUCHY      ,DEUFLH      ,GEOMS     ,
     $             LINESR     ,MATSUP      ,NEWTON      ,OVERCH    ,
     $             WRNSUP
      EXTERNAL FCN,JACOB
      COMMON/NNES_1/MATSUP
      COMMON/NNES_2/WRNSUP
      COMMON/NNES_3/BYPASS
      COMMON/NNES_4/NFETOT
      OPEN(UNIT=NUNIT,FILE='ROSENB.OUT',STATUS='UNKNOWN')
      XC(1)=-1.2D0
      XC(2)=1.0D0
      CALL SETUP(ABSNEW ,CAUCHY ,DEUFLH ,GEOMS  ,LINESR ,
     $           NEWTON ,OVERCH ,ACPTCR ,ITSCLF ,ITSCLX ,
     $           JACTYP ,JUPDM  ,MAXEXP ,MAXIT  ,MAXNS  ,
     $           MAXQNS ,MINQNS ,N      ,NARMIJ ,NINITN ,
     $           NJACCH ,OUTPUT ,QNUPDM ,STOPCR ,SUPPRS ,
     $           TRUPDM ,ALPHA  ,CONFAC ,DELTA  ,DELFAC ,
     $           ETAFAC ,FDTOLJ ,FTOL   ,LAM0   ,MCHEPS ,
     $           MSTPF  ,NSTTOL ,OMEGA  ,RATIOF ,SIGMA  ,
     $           STPTOL ,BOUNDL ,BOUNDU ,SCALEF ,SCALEX  ,
     $           HELP   )
      CALL NNES(ABSNEW ,CAUCHY ,DEUFLH ,GEOMS  ,LINESR ,
     $          NEWTON ,OVERCH ,ACPTCR ,ITSCLF ,ITSCLX ,
     $          JACTYP ,JUPDM  ,MAXEXP ,MAXIT  ,MAXNS  ,
     $          MAXQNS ,MGLL   ,MINQNS ,N      ,NARMIJ ,
     $          NIEJEV ,NJACCH ,NJETOT ,NUNIT  ,OUTPUT ,
     $          QNUPDM ,STOPCR ,SUPPRS ,TRMCOD ,TRUPDM ,
     $          ALPHA  ,CONFAC ,DELTA  ,DELFAC ,EPSMCH ,
     $          ETAFAC ,FCNNEW ,FDTOLJ ,FTOL   ,LAM0   ,
```

```
     $            MSTPF  ,NSTTOL ,OMEGA  ,RATIOF ,SIGMA  ,
     $            STPTOL ,A      ,BOUNDL ,BOUNDU ,DELF   ,
     $            FSAVE  ,FTRACK ,FVEC   ,FVECC  ,H      ,
     $            HHPI   ,JAC    ,PLEE   ,RDIAG  ,S      ,
     $            SBAR   ,SCALEF ,SCALEX ,SN     ,SSDHAT ,
     $            STRACK ,VHAT   ,WV1    ,WV2    ,WV3    ,
     $            WV4    ,XC     ,XPLUS  ,XSAVE  ,HELP   ,
     $            FCN    ,JACOB  )
        STOP
        END


        SUBROUTINE FCN(OVERFL,N,FVEC,XC)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DIMENSION FVEC(N),XC(N)
        COMMON/NNES_4/NFETOT
        LOGICAL OVERFL
        OVERFL=.FALSE.
        NFETOT=NFETOT+1
c
c       Rosenbrock Banana Function
c
c       f1 = 10(x2 - x1**2)
c       f2 = 1 - x1
c
c       start:  (-1.2,1)
c
c       sol'n:  (1,1)
c
        FVEC(1)=10.0D0*(XC(2)-XC(1)**2)
        FVEC(2)=1.0D0-XC(1)
        RETURN
        END
        SUBROUTINE JACOB(OVERFL,N,JAC,XC)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DOUBLE PRECISION JAC(N,N)
        DIMENSION XC(N)
        LOGICAL OVERFL
        OVERFL=.FALSE.
```

The analytical jacobian would look like

```
        JAC(1,1)=-20.0D0*XC(1)
        JAC(1,2)=10.0D0
        JAC(2,1)=-1.0D0
        JAC(2,2)=0.0D0

        RETURN
        END
```

Output

```
**********************************************************************
**********************************************************************
**********************************************************************
*                                                                    *
*                              NNES                                  *
*                                                                    *
*        NONMONOTONIC NONLINEAR EQUATION SOLVER  VERSION 1.05        *
*                                                                    *
*                  COPYRIGHT 1991, BY R.S. BAIN                      *
*                                                                    *
**********************************************************************
**********************************************************************
**********************************************************************




**********************************************************************
**********************************************************************
*                                                                    *
*    INITIAL ESTIMATES                 INITIAL FUNCTION VALUES       *
*                                                                    *
*  X(  1) =   -1.200E+00               F(  1) =   -4.400E+00         *
*  X(  2) =    1.000E+00               F(  2) =    2.200E+00         *
*                                                                    *
*  INITIAL OBJECTIVE FUNCTION VALUE =  1.210E+01                     *
*                                                                    *
**********************************************************************


**********************************************************************
*                                                                    *
*      CONVERGENCE REACHED; TERMINATION CODE: ...............     2  *
*                                                                    *
*                                                                    *
*         FINAL ESTIMATES                  FINAL FUNCTION VALUES     *
*                                                                    *
*      X(  1) =    1.00000E+00            F(  1) =    3.05532E-07    *
*      X(  2) =    1.00000E+00            F(  2) =    1.11022E-16    *
*                                                                    *
*      FINAL OBJECTIVE FUNCTION VALUE:   4.667E-14                   *
*                                                                    *
*                                                                    *
*      TOTAL NUMBER OF ITERATIONS: .........................     2   *
*      TOTAL NUMBER OF LINE SEARCH FUNCTION EVALUATIONS: ....    3   *
*      TOTAL NUMBER OF EXPLICIT JACOBIAN EVALUATIONS: .......    2   *
*      TOTAL NUMBER OF FUNCTION EVALUATIONS: ................    7   *
*                                                                    *
**********************************************************************
```