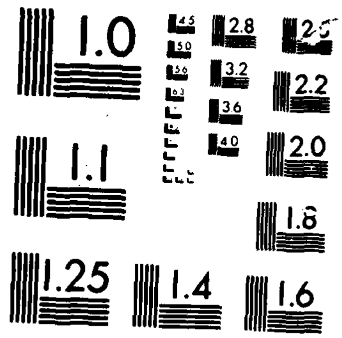
AD-A169 115 USER'S GUIDE FOR NPSOL (VERSION 4.0): A FORTRAN PACKAGE FOR NONLINEAR PROGRAMMING(U) STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB P E GILL ET AL. JAN 86 SOL-86-2 1/1

UNCLASSIFIED ARO-21592.8-MA N00014-85-K-0343 F/G 9/2 NL



MICROCOP CHART

ARO 21592.8-MA

2

Systems
Optimization
Laboratory

AD-A169 115

User's Guide for NPSOL (Version 4.0)†:
A Fortran Package for Nonlinear Programming

by

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 86-2

January 1986

(Replaces SOL 84-7)

Department of Operations Research
Stanford University
Stanford, CA 94305

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

User's Guide for NPSOL (Version 4.0)†: A Fortran Package for Nonlinear Programming

by

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 86-2

January 1986

(Replaces SOL 84-7)

†NPSOL is available from the Office of Technology Licensing, 350 Cambridge Avenue, Suite 250, Palo Alto, California 94306, USA.

Research and reproduction of this report were partially supported by the U.S. Department of Energy Contract DE-AA03-76SF00326, PA# DE-AT03-76ER72018; the National Science Foundation Grants DCR-8413211 and ECS-8312142; Office of Naval Research Contract N00014-85-K-0343; and Army Research Office Contract DAAG29-84-K-0156.

User's Guide for NPSOL (Version 4.0)†: a Fortran Package for Nonlinear Programming

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright
Systems Optimization Laboratory
Department of Operations Research
Stanford University
Stanford, California 94305

January 1986

ABSTRACT

This report forms the user's guide for Version 4.0 of NPSOL, a set of Fortran subroutines designed to minimize a smooth function subject to constraints, which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints. (NPSOL may also be used for unconstrained, bound-constrained and linearly constrained optimization.) The user must provide subroutines that define the objective and constraint functions and (optionally) their gradients. All matrices are treated as dense, and hence NPSOL is not intended for large sparse problems.

NPSOL uses a sequential quadratic programming (SQP) algorithm, in which the search direction is the solution of a quadratic programming (QP) subproblem. The algorithm treats bounds, linear constraints and nonlinear constraints separately. The Hessian of each QP subproblem is a positive-definite quasi-Newton approximation to the Hessian of the Lagrangian function. The steplength at each iteration is required to produce a sufficient decrease in an augmented Lagrangian merit function. Each QP subproblem is solved using a quadratic programming package with several features that improve the efficiency of an SQP algorithm.

---

† NPSOL is available from the Stanford Office of Technology Licensing, 350 Cambridge Avenue, Suite 250, Palo Alto, California 94306, USA.

The material contained in this report is based upon research supported by the U.S. Department of Energy Contract DE-AA03-76SF00326, PA No. DE-AS03-76ER72018; National Science Foundation Grants DCR-8413211 and ECS-8312142; the Office of Naval Research Contract N00014-85-K-0343; and the U.S. Army Research Office Contract DAAG29-84-K-0156.

CONTENTS

1. PURPOSE

NPSOL is a collection of Fortran 77 subroutines designed to solve the *nonlinear programming problem*: the minimization of a smooth nonlinear function subject to a set of constraints on the variables. The problem is assumed to be stated in the following form:

$$\text{NP} \qquad \underset{x \in \Re^n}{\text{minimize}} \quad F(x)$$

$$\text{subject to} \quad \ell \le \begin{Bmatrix} x \\ A_L x \\ c(x) \end{Bmatrix} \le u,$$

where $F(x)$ (the *objective function*) is a nonlinear function, A_L is an $m_L \times n$ constant matrix of general constraints, and $c(x)$ is an m_N-vector of nonlinear constraint functions. (The matrix A_L and the vector $c(x)$ may be empty.) The objective function F and the constraint functions are assumed to be *smooth*, i.e., at least twice-continuously differentiable. (The method of NPSOL will usually solve NP if there are only isolated discontinuities away from the solution).

Note that *upper and lower bounds are specified for all the variables and for all the constraints.* This form allows full generality in specifying other types of constraints. In particular, the i-th constraint may be defined as an *equality* by setting $\ell_i = u_i$. If certain bounds are not present, the associated elements of ℓ or u can be set to special values that will be treated as $-\infty$ or $+\infty$.

If there are no nonlinear constraints in NP and F is linear or quadratic, the QPSOL or LSSOL packages (Gill *et al.*, 1984a, 1986a) will generally be more efficient than NPSOL. If the problem is large and sparse, the MINOS package (Murtagh and Saunders, 1982, 1983) should be used, since NPSOL treats all matrices as dense.

The user must supply an initial estimate of the solution to NP, and subroutines that define $F(x)$, $c(x)$, and as many first partial derivatives as possible; unspecified derivatives are approximated by finite-differences.

NPSOL is based on subroutines from Version 1.0 of the LSSOL constrained linear least-squares package; the documentation of LSSOL (Gill *et al.*, 1986a) should be consulted in conjunction with this report. NPSOL contains approximately 15,000 lines of ANSI (1977) Standard Fortran, of which 47% are comments.

2. DESCRIPTION OF THE ALGORITHM

Here we briefly summarize the main features of the method of NPSOL. Where possible, explicit reference is made to the names of variables that are parameters of subroutine NPSOL or appear in the printed output.

At a solution of NP, some of the constraints will be *active*, i.e., satisfied exactly. An active simple bound constraint implies that the corresponding variable is *fixed* at its bound, and hence the variables are partitioned into *fixed* and *free* variables. Let C denote the $m \times n$ matrix of gradients of the active general linear and nonlinear constraints. The number of fixed variables will be denoted by n_{FX}, with n_{FR} ($n_{\text{FR}} = n - n_{\text{FX}}$) the number of free variables. The subscripts "FX" and "FR" on a vector or matrix will denote the vector or matrix composed of the components corresponding to fixed or free variables.

A point x is a *first-order Kuhn-Tucker point* for NP (see, e.g., Powell, 1974) if the following conditions hold:

(i) x is feasible;

(ii) there exist vectors ξ and λ (the *Lagrange multiplier vectors for the bound and general constraints*) such that

$$g = C^T\lambda + \xi, \tag{1}$$

where g is the gradient of F evaluated at x, and $\xi_j = 0$ if the j-th variable is free.

(iii) The Lagrange multiplier corresponding to an inequality constraint active at its lower bound must be non-negative, and non-positive for an inequality constraint active at its upper bound.

Let Z denote a matrix whose columns form a basis for the set of vectors orthogonal to the rows of C_{FR}; i.e., $C_{\text{FR}}Z = 0$. An equivalent statement of the condition (1) in terms of Z is

$$Z^T g_{\text{FR}} = 0.$$

The vector $Z^T g_{\text{FR}}$ is termed the *projected gradient* of F at x. Certain additional conditions must be satisfied in order for a first-order Kuhn-Tucker point to be a solution of NP (see, e.g., Powell, 1974).

The method of NPSOL 4.0 is a sequential quadratic programming (SQP) method. For an overview of SQP methods, see, for example, Fletcher (1981), Gill, Murray and Wright (1981) and Powell (1983).

The basic structure of NPSOL involves *major* and *minor* iterations. The major iterations generate a sequence of iterates $\{x_k\}$ that converge to x^*, a first-order Kuhn-Tucker point of NP. At a typical major iteration, the new iterate $\bar{x}$ is defined by

$$\bar{x} = x + \alpha p, \tag{2}$$

where x is the current iterate, the non-negative scalar α is the *step length*, and p is the *search direction*. (For simplicity, we shall always consider a typical iteration and avoid reference to the index of the iteration.) Also associated with each major iteration are estimates of the Lagrange multipliers and a prediction of the active set.

The search direction p in (2) is the solution of a quadratic programming subproblem of the form

$$\begin{array}{ll} \underset{p}{\text{minimize}} & g^Tp + \tfrac{1}{2}p^THp \\ \text{subject to} & \bar{l} \le \begin{Bmatrix} p \\ A_Lp \\ A_Np \end{Bmatrix} \le \bar{u}, \end{array} \tag{3}$$

where g is the gradient of F at x, the matrix H is a positive-definite quasi-Newton approximation to the Hessian of the Lagrangian function (see Section 2.3), and A_N is the Jacobian matrix of c evaluated at x. (Finite-difference estimates may be used for g and A_N; see the optional parameter "`Derivative Level`" in Section 5.2.) Let ℓ in NP be partitioned into three sections: ℓ_B, ℓ_L and ℓ_N, corresponding to the bound, linear and nonlinear constraints. The vector $\bar{\ell}$ in (3) is similarly partitioned, and is defined as

$$\bar{\ell}_B = \ell_B - x, \quad \bar{\ell}_L = \ell_L - A_L x, \quad \text{and} \quad \bar{\ell}_N = \ell_N - c,$$

where c is the vector of nonlinear constraints evaluated at x. The vector $\bar{u}$ is defined in an analogous fashion.

The estimated Lagrange multipliers at each major iteration are the Lagrange multipliers from the subproblem (3) (and similarly for the predicted active set). (The numbers of bounds, general linear and nonlinear constraints in the QP active set are the quantities "`Bnd`", "`Lin`" and "`Nln`" in the printed output of NPSOL.) In NPSOL, (3) is solved using subroutines from Version 1.0 of the LSSOL package (Gill *et al.*, 1986a). Since solving a quadratic program is itself an iterative procedure, the *minor* iterations of NPSOL are the iterations of LSSOL. (More details about solving the subproblem are given in Section 2.1.)

Certain matrices associated with the QP subproblem are relevant in the major iterations. Let the subscripts "`FX`" and "`FR`" refer to the *predicted* fixed and free variables, and let C denote the $m \times n$ matrix of gradients of the general linear and nonlinear constraints in the predicted active set. First, we have available the TQ factorization of C_{FR}:

$$C_{FR} Q_{FR} = \begin{pmatrix} 0 & T \end{pmatrix}, \tag{4}$$

where T is a nonsingular $m \times m$ reverse-triangular matrix (i.e., $t_{ij} = 0$ if $i + j < m$), and the non-singular $n_{FR} \times n_{FR}$ matrix Q_{FR} is the product of orthogonal transformations (see Gill *et al.*, 1984a). Second, we have the upper-triangular Cholesky factor R of the *transformed and re-ordered* Hessian matrix

$$R^T R = H_Q \equiv Q^T \tilde{H} Q, \tag{5}$$

where $\tilde{H}$ is the Hessian H with rows and columns permuted so that the free variables are first, and Q is the $n \times n$ matrix

$$Q = \begin{pmatrix} Q_{FR} & \\ & I_{FX} \end{pmatrix}, \tag{6}$$

with I_{FX} the identity matrix of order n_{FX}. If the columns of Q_{FR} are partitioned so that

$$Q_{FR} = \begin{pmatrix} Z & Y \end{pmatrix},$$

the n_Z ($n_Z \equiv n_{FR} - m$) columns of Z form a basis for the null space of C_{FR}. The matrix Z is used to compute the projected gradient $Z^T g_{FR}$ at the current iterate. (The values "`Nz`", "`Norm Gf`", and "`Norm Gz`" printed by NPSOL give n_Z and the norms of g_{FR} and $Z^T g_{FR}$.)

A theoretical characteristic of SQP methods is that the predicted active set from the QP subproblem (3) is identical to the correct active set in a neighborhood of x^*. In NPSOL, this feature is exploited by using the QP active set from the previous iteration as a prediction of the active set for the next QP subproblem, which leads in practice to optimality of the subproblems in only one iteration as the solution is approached. Separate treatment of bound and linear constraints in NPSOL also saves computation in factorizing C_{FR} and H_Q.

Once p has been computed, the major iteration proceeds by determining a steplength α that produces a "sufficient decrease" in an augmented Lagrangian *merit function* (see Section 2.2). Finally, the approximation to the transformed Hessian matrix H_Q is updated using a modified BFGS quasi-Newton update (see Section 2.3) to incorporate new curvature information obtained in the move from x to $\bar{x}$.

On entry to NPSOL, an iterative procedure from the LSSOL package is executed, starting with the user-provided initial point, to find a point that is feasible with respect to the bounds and linear constraints (using the tolerance specified by "`Linear Feasibility Tolerance`"; see Section 5.2). If no feasible point exists for the bound and linear constraints, NP has no solution and NPSOL terminates. Otherwise, the problem functions will thereafter be evaluated only at points that are feasible with respect to the bounds and linear constraints. The only exception involves variables whose bounds differ by an amount comparable to the finite-difference interval (see the discussion of "`Difference Interval`" in Section 5.2). In contrast to the bounds and linear constraints, it must be emphasized that *the nonlinear constraints will not generally be satisfied until an optimal point is reached.*

Facilities are provided to check whether the user-provided gradients appear to be correct (see the optional parameter "`Verify`" in Section 5.2). In general, the check is provided at the first point that is feasible with respect to the linear constraints and bounds. However, the user may request that the check be performed at the initial point.

In summary, the method of NPSOL first determines a point that satisfies the bound and linear constraints. Thereafter, each iteration includes: (a) the solution of a quadratic programming subproblem; (b) a linesearch with an augmented Lagrangian merit function; and (c) a quasi-Newton update of the approximate Hessian of the Lagrangian function. These three procedures are described in more detail in the next three subsections.

2.1. Solution of the quadratic programming subproblem

The search direction p is obtained by solving (3) using subroutines from the LSSOL package (Gill *et al.*, 1986a), which was specifically designed to be used within an SQP algorithm for nonlinear programming.

The method of LSSOL is a two-phase (primal) quadratic programming method. The two phases of the method are: finding an initial feasible point by minimizing the sum of infeasibilities (the *feasibility phase*), and minimizing the quadratic objective function within the feasible region (the *optimality phase*). The computations in both phases are performed by the same subroutines. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the quadratic objective function.

In general, a quadratic program must be solved by iteration. Let p denote the current estimate of the solution of (3); the new iterate $\bar{p}$ is defined by

$$\bar{p} = p + \sigma d, \tag{7}$$

where, as in (2), σ is a non-negative step length and d is a search direction.

At the beginning of each iteration of LSSOL, a *working set* is defined of constraints (general and bound) that are satisfied exactly. The vector d is then constructed so that the values of constraints in the working set remain *unaltered* for any move along d. For a bound constraint in the working set, this property is achieved by setting the corresponding component of d to zero, i.e., by fixing the variable at its bound. As before, the subscripts "FX" and "FR" denote selection of the components associated with the fixed and free variables.

Let C denote the submatrix of rows of

$$\begin{pmatrix} A_L \\ A_N \end{pmatrix}$$

corresponding to general constraints in the working set. The general constraints in the working set will remain unaltered if

$$C_{FR} d_{FR} = 0, \tag{8}$$

which is equivalent to defining d_{FR} as

$$d_{FR} = Z d_Z \tag{9}$$

for some vector d_Z, where Z is the matrix associated with the TQ factorization (4) of C_{FR}.

The definition of d_Z in (9) depends on whether the current p is feasible. If not, d_Z is zero except for a component γ in the j-th position, where j and γ are chosen so that the sum of infeasibilities is decreasing along d. (For further details, see Gill *et al.*, 1986a.) In the feasible case, d_Z satisfies the equations

$$R_Z^T R_Z d_Z = -Z^T q_{FR}, \tag{10}$$

where R_Z is the Cholesky factor of $Z^T H_{FR} Z$ and q is the gradient of the quadratic objective function ($q = g + Hp$). (The vector $Z^T q_{FR}$ is the projected gradient of the QP.) With (10), $p + d$ is the minimizer of the quadratic objective function subject to treating the constraints in the working set as equalities.

If the QP projected gradient is zero, the current point is a constrained stationary point in the subspace defined by the working set. During the feasibility phase, the projected gradient will usually be zero only at a vertex (although it may vanish at non-vertices in the presence of constraint dependencies). During the optimality phase, a zero projected gradient implies that p minimizes the quadratic objective function when the constraints in the working set are treated as equalities. In either case, Lagrange multipliers are computed. Given a positive constant δ of the order of the machine precision, the Lagrange multiplier μ_j corresponding to an inequality constraint in the working set at its upper bound is said to be *optimal* if $\mu_j \le \delta$ when the j-th constraint is at its upper bound, or if $\mu_j \ge -\delta$ when the associated constraint is at its lower bound. If any multiplier is non-optimal, the current objective function (either the true objective or the sum of infeasibilities) can be reduced by deleting the corresponding constraint from the working set.

If optimal multipliers occur during the feasibility phase and the sum of infeasibilities is nonzero, no feasible point exists. The QP algorithm will then continue iterating to determine the minimum sum of infeasibilities. At this point, the Lagrange multiplier μ_j will satisfy $-(1+\delta) \le \mu_j \le \delta$ for an inequality constraint at its upper bound, and $-\delta \le \mu_j \le 1 + \delta$ for an inequality at its lower bound. The Lagrange multiplier for an equality constraint will satisfy $|\mu_j| \le 1 + \delta$.

The choice of step length σ in the QP iteration (7) is based on remaining feasible with respect to the satisfied constraints. During the optimality phase, if $p + d$ is feasible, σ will be taken as unity. (In this case, the projected gradient at $\bar{p}$ will be zero.) Otherwise, σ is set to σ_M, the step to the "nearest" constraint, which is added to the working set at the next iteration.

Each change in the working set leads to a simple change to C_{FR}: if the status of a general constraint changes, a *row* of C_{FR} is altered; if a bound constraint enters or leaves the working set, a *column* of C_{FR} changes. Explicit representations are recurred of the matrices T, Q_{FR} and R, and of the vectors $Q^T q$ and $Q^T g$.

2.2. The merit function

After computing the search direction as described in Section 2.1, each major iteration proceeds by determining a steplength α in (2) that produces a "sufficient decrease" in the augmented Lagrangian merit function

$$\mathcal{L}(x,\lambda,s) = F(x) - \sum_i \lambda_i\big(c_i(x) - s_i\big) + \frac{1}{2}\sum_i \rho_i\big(c_i(x) - s_i\big)^2, \tag{11}$$

where x, λ and s vary during the linesearch. The summation terms in (11) involve only the *nonlinear* constraints. The vector λ is an estimate of the Lagrange multipliers for the nonlinear constraints of NP. The non-negative *slack variables* $\{s_i\}$ allow nonlinear inequality constraints to be treated without introducing discontinuities. The solution of the QP subproblem (3) provides a vector triple that serves as a direction of search for the three sets of variables. The non-negative vector ρ of *penalty parameters* is initialized to zero at the beginning of the first major iteration. Thereafter, selected components are increased whenever necessary to ensure descent for the merit function. Thus, the sequence of norms of ρ (the printed quantity "`Penalty`"; see Section 6) is generally non-decreasing, although each ρ_i may be reduced a limited number of times.

The merit function (11) and its global convergence properties are described in Gill *et al.* (1986b).

2.3. The quasi-Newton update

The matrix H in (3) is a *positive-definite quasi-Newton* approximation to the Hessian of the Lagrangian function. (For a review of quasi-Newton methods, see Dennis and Schnabel, 1983.) At the end of each major iteration, a new Hessian approximation $\bar{H}$ is defined as a rank-two modification of H. In NPSOL, the BFGS quasi-Newton update is used:

$$\bar{H} = H - \frac{1}{s^THs}Hss^TH + \frac{1}{y^Ts}yy^T, \tag{12}$$

where $s = \bar{x} - x$ (the change in x).

In NPSOL, H is required to be positive definite. If H is positive definite, $\bar{H}$ as defined by (12) will be positive definite if and only if y^Ts is positive (see, e.g., Dennis and Moré, 1977). Ideally, y in (12) would be taken as y_L, the change in gradient of the Lagrangian function

$$y_L = \bar{g} - \bar{A}_N^T\mu_N - g + A_N^T\mu_N, \tag{13}$$

where μ_N denotes the QP multipliers asociated with the nonlinear constraints of the original problem. If y_L^Ts is not sufficiently positive, an attempt is made to perform the update with a vector y of the form

$$y = y_L + \sum_{i=1}^{m_N}\omega_i\big(a_i(\bar{x})c_i(\bar{x}) - a_i(x)c_i(x)\big),$$

where $\omega_i \geq 0$. If no such vector can be found, the update is performed with a scaled y_L; in this case, "`M`" is printed to indicate that the update was modified.

Rather than modifying H itself, the Cholesky factor of the *transformed Hessian* H_Q (4) is updated, where Q is the matrix from (3) associated with the active set of the QP subproblem. The update (12) is equivalent to the following update to H_Q:

$$\bar{H}_Q = H_Q - \frac{1}{s_Q^TH_Qs_Q}H_Qs_Qs_Q^TH_Q + \frac{1}{y_Q^Ts_Q}y_Qy_Q^T, \tag{14}$$

where $y_Q = Q^Ty$, and $s_Q = Q^Ts$. This update may be expressed as a *rank-one* update to R (see Dennis and Schnabel, 1981).

Full details concerning the Hessian update are given in Gill *et al.* (1986c).

3. SPECIFICATION OF SUBROUTINE NPSOL

The formal specification of NPSOL is the following:

```
      SUBROUTINE NPSOL ( N, NCLIN, NCNLN, NROWA, NROWJ, NROWR,
                         A, BL, BU,
                         CONFUN, OBJFUN,
                         INFORM, ITER, ISTATE,
                         C, CJAC, CLAMDA, OBJF, GRAD, R, X,
                         IW, LENIW, W, LENW )
      INTEGER            N, NCLIN, NCNLN,
                         NROWA, NROWJ, NROWR, INFORM, ITER, LENIW, LENW
      INTEGER            ISTATE(N+NCLIN+NCNLN), IW(LENIW)
      REAL               OBJF
      REAL               A(NROWA,*), BL(N+NCLIN+NCNLN), BU(N+NCLIN+NCNLN),
                         C(*), CJAC(NROWJ,*), CLAMDA(N+NCLIN+NCNLN), GRAD(N),
                         R(NROWR,*), X(N), W(LENW)
      EXTERNAL           CONFUN, OBJFUN
```

Note: Here and elsewhere, the specification of a parameter as REAL should be interpreted as *working precision*, which may be DOUBLE PRECISION in some installations.

3.1. Formal parameters

N (**Input**) The number of variables in the problem, i.e., the dimension of X. (N must be positive.)

NCLIN (**Input**) The number of general linear constraints in the problem. (NCLIN may be zero.)

NCNLN (**Input**) The number of nonlinear constraints in the problem. (NCNLN may be zero.)

NROWA (**Input**) The declared row dimension of the array A. NROWA must be at least 1 and at least NCLIN.

NROWJ (**Input**) The declared row dimension of the array CJAC. NROWJ must be at least 1 and at least NCNLN.

NROWR (**Input**) The declared row dimension of the array R. NROWR must be at least N.

A (**Input**) A real array of declared dimension (NROWA,*), where the second dimension must be at least N. A contains the matrix A_L of general linear constraints in the problem specification NP (Section 1). The i-th row of A, $i = 1$ to NCLIN, contains the coefficients of the i-th general *linear* constraint. If NCLIN is zero, A is not accessed and may be dimensioned (1,1).

BL (**Input**) A real array of dimension at least N + NCLIN + NCNLN that contains the lower bounds for all the constraints, in the following order (which is also observed for BU, CLAMDA and ISTATE). The first N elements of BL contain the lower bounds on the variables. If NCLIN > 0, the next NCLIN elements of BL contain the lower bounds for the general linear constraints. If NCNLN > 0, the next NCNLN elements of BL contain the lower bounds for the nonlinear constraints. In order for the problem specification to be meaningful, it is required that BL(j) $\leq$ BU(j) for all j. To specify a non-existent lower bound (i.e., $\ell_j = -\infty$), the value used must satisfy BL(j) $\leq$ -BIGBND, where BIGBND is the value of the optional parameter `Infinite Bound`, whose default value is 10^{10} (see Section 5.2). To specify the j-th constraint as an *equality*, the user must set BL(j) = BU(j) = β, say, where $|\beta|$ < BIGBND.

BU **(Input)** A real array of dimension at least N + NCLIN + NCNLN that contains the upper bounds for all the constraints, in the same order described above for BL. To specify a non-existent upper bound (i.e., $u_j = \infty$), the value used must satisfy BU(j) $\geq$ BIGBND.

CONFUN **(User-defined subroutine)** The name of a subroutine that calculates the vector $c(x)$ of nonlinear constraint functions and (optionally) its Jacobian for a specified n-vector x. CONFUN must be declared as EXTERNAL in the routine that calls NPSOL. For a detailed description of CONFUN, see Section 4.2.

OBJFUN **(User-defined subroutine)** The name of a subroutine that calculates the objective function $F(x)$ and (optionally) its gradient for a specified n-vector x. OBJFUN must be declared as EXTERNAL in the routine that calls NPSOL. For a detailed description of OBJFUN, see Section 4.1.

INFORM **(Output)** An integer that indicates the result of NPSOL. (A short description of INFORM is printed if `Major Print Level` > 0.) The possible values of INFORM are:

INFORM	**Meaning**
< 0	The user has set MODE to this negative value in CONFUN or OBJFUN (see Section 4).
0	The iterates have converged to a point X that satisfies the first-order Kuhn-Tucker conditions to the accuracy requested by the optional parameter `Optimality Tolerance` (see Section 5.2), i.e., the projected gradient and active constraint residuals are negligible at X.
1	The final iterate X satisfies the first-order Kuhn-Tucker conditions to the accuracy requested, but the sequence of iterates has not yet converged. NPSOL was terminated because no further improvement could be made in the merit function.
2	No feasible point could be found for the linear constraints and bounds. The problem has no feasible solution. See Section 7 for further comments.
3	No feasible point could be found for the nonlinear constraints. The problem may have no feasible solution. See Section 7 for further comments.
4	The limiting number of iterations (determined by the optional parameter `Major Iteration Limit`: see Section 5.2) has been reached.
6	X does not satisfy the first-order Kuhn-Tucker conditions, and no improved point for the merit function could be found during the final line search.
7	The user-provided derivatives of the objective function and/or nonlinear constraints appear to be incorrect.
9	An input parameter is invalid.

ITER **(Output)** The number of major iterations performed.

ISTATE **(Input)** An integer array of dimension at least N + NCLIN + NCNLN. ISTATE need not be initialized if NPSOL is called with a `Cold Start` (the default option; see Section 5.2). The ordering of ISTATE is the same as that described above for BL, i.e., the first N components of ISTATE refer to the upper and lower bounds on the variables, components N + 1 through N + NCLIN refer to the upper and lower bounds on $A_L x$, and components N + NCLIN + 1 through N + NCLIN + NCNLN refer to the upper and lower bounds on $c(x)$. When a `Warm Start` option is chosen, the components of ISTATE corresponding to the bounds and linear constraints define the initial working set for the procedure that finds a feasible point for the linear constraints and bounds. The active set at the conclusion of this procedure and the components of ISTATE corresponding to nonlinear constraints then define the initial working set for the first QP subproblem. Possible values for ISTATE(j) are

ISTATE(j)	**Meaning**
0	The corresponding constraint is not in the initial QP working set.
1	This inequality constraint should be in the working set at its lower bound.
2	This inequality constraint should be in the working set at its upper bound.
3	This equality constraint should be in the initial working set. This value must not be specified unless BL(j) = BU(j). The values 1, 2 or 3 all have the same effect when BL(j) = BU(j).

Other values of ISTATE are also acceptable. In particular, if NPSOL has been called previously with the same values of N, NCLIN and NCNLN, ISTATE already contains satisfactory values. If necessary, NPSOL will override the user's specification of ISTATE, so that a poor choice will not cause the algorithm to fail.

(Output) If NPSOL exits with INFORM = 0 or 1, the values in the array ISTATE correspond to the active set of the final QP subproblem, and are a prediction of the status of the constraints at the solution of the problem. Otherwise, ISTATE indicates the composition of the QP working set at the final iterate. The significance of each possible value of ISTATE(j) is as follows:

ISTATE(j)	**Meaning**
−2	This constraint violates its lower bound by more than the feasibility tolerance (see the optional parameters `Linear Feasibility Tolerance` and `Nonlinear Feasibility Tolerance` in Section 5.2). This value can occur only when no feasible point can be found for a QP subproblem.
−1	This constraint violates its upper bound by more than the appropriate feasibility tolerance (see the optional parameters `Linear Feasibility Tolerance` and `Nonlinear Feasibility Tolerance` in Section 5.2). This value can occur only when no feasible point can be found for a QP subproblem.
0	The constraint is satisfied to within the feasibility tolerance, but is not in the working set.
1	This inequality constraint is included in the QP working set at its lower bound.
2	This inequality constraint is included in the QP working set at its upper bound.

3 This constraint is included in the QP working set as an equality. This value of ISTATE can occur only when BL(j) = BU(j).

C **(Output)** A real array of dimension at least NCNLN. If NCNLN = 0, C is not accessed, and may then be declared to be of dimension (1), or the actual parameter may be any convenient array. If NCNLN > 0, C contains the values of the nonlinear constraint functions c_i, $i = 1$ to NCNLN, at the final iterate.

CJAC **(Input)** A real array of dimension (NROWJ,*), where the second dimension must be at least N. If NCNLN = 0, CJAC is not accessed, and may then be declared to be of d ιension (1,1), or the actual parameter may be any convenient array.

In general, CJAC need not be initialized before the call to NPSOL. However, if Derivative Level = 3, the user may optionally set the constant elements of CJAC (see Section 4.3). Such constant elements need not be re-assigned on subsequent calls to CONFUN.

(Output) If NCNLN > 0, CJAC contains the Jacobian matrix of the nonlinear constraint functions at the final iterate, i.e., CJAC(i,j) contains the partial derivative of the i-th constraint function with respect to the j-th variable, $i = 1$ to NCNLN, $j = 1$ to N. (See the discussion of CJAC under CONFUN in Section 4.2.)

CLAMDA **(Input)** A real array of dimension at least N + NCLIN + NCNLN. CLAMDA need not be initialized if NPSOL is called with the (default) Cold Start option. With the Warm Start option, CLAMDA must contain a multiplier estimate for each *nonlinear* constraint with a sign that matches the status of the constraint specified by the ISTATE array (as above). The ordering of CLAMDA is the same as that given above for BL. If the j-th constraint is defined as "inactive" by the initial value of the ISTATE array, CLAMDA(j) should be zero; if the j-th constraint is an inequality active at its lower bound, CLAMDA(j) should be non-negative; if the j-th constraint is an inequality active at its upper bound, CLAMDA(j) should be non-positive.

(Output) CLAMDA gives the QP multipliers from the last QP subproblem. CLAMDA(j) should be non-negative if ISTATE(j) = 1 and non-positive if ISTATE(j) = 2.

OBJF **(Output)** The value of the objective function $F(x)$ at the final iterate.

OBJGRD **(Output)** A real array of dimension at least N that contains the objective gradient (or its finite-difference approximation) at the final iterate.

R **(Input)** A real array of declared dimension (NROWR,*), where the second dimension must be at least N. R need not be initialized if NPSOL is called with a Cold Start option (the default), and will be taken as the identity. With a Warm Start, R must contain the upper-triangular Cholesky factor of the initial approximation of the Hessian of the Lagrangian function, *with the variables in the natural order*. Elements not in the upper-triangular part of R are assumed to be zero and need not be assigned.

(Output) If Hessian = No (the default; see Section 5.2), R contains the upper-triangular Cholesky factor of $Q^T\tilde{H}Q$, an estimate of the transformed and re-ordered Hessian of the Lagrangian at X (see (5) in Section 2). If Hessian = Yes, R contains the upper-triangular Cholesky factor of H, the approximate (untransformed) Hessian of the Lagrangian, with the variables in the natural order.

X (Input) A real array of dimension at least N. X must contain an initial estimate of the solution.

(Output) X contains the final estimate of the solution.

3.2. Workspace parameters

IW (Input) An integer array of dimension LENIW that provides integer workspace for NPSOL.

LENIW (Input) The dimension of IW. LENIW must be at least 3 N + NCLIN + 2 NCNLN.

W (Input) A real array of dimension LENW that provides real workspace for NPSOL.

LENW (Input) The dimension of W. If there are no general linear constraints and no nonlinear constraints (i.e., NCLIN = 0 and NCNLN = 0), LENW must be at least 20 N. If there are no nonlinear constraints (i.e., NCNLN = 0), LENW must be at least $2\,N^2 + 20\,N + 11\,NCLIN$. Otherwise, LENW must be at least $2\,N^2 + N*NCLIN + 2\,N*NCNLN + 20\,N + 11\,NCLIN + 21\,NCNLN$.

If Major Print Level > 0, the required amounts of workspace are printed. As an alternative to computing LENIW and LENW from the formulas given above, the user may prefer to obtain appropriate values from the output of a preliminary run with a positive value of Major Print Level and LENIW and LENW set to 1. (NPSOL will then terminate with INFORM = 9.)

4. USER-SUPPLIED SUBROUTINES

The user must provide subroutines that define the objective function and nonlinear constraints. The objective function is defined by subroutine OBJFUN, and the nonlinear constraints are defined by subroutine CONFUN. *On every call*, these subroutines must return appropriate values of the objective and nonlinear constraints in OBJF and C. The user should also provide the available partial derivatives. Any unspecified derivatives are approximated by finite differences; see Section 5.2 for a discussion of the optional parameter **Derivative Level**. Just before either OBJFUN or CONFUN is called, each element of the current gradient array OBJGRD or CJAC is initialized to a special value. On exit, any element that retains the given value is estimated by finite differences.

For maximum reliability, it is preferable for the user to provide *all* partial derivatives (see Chapter 8 of Gill, Murray and Wright, 1981, for a detailed discussion). If all gradients cannot be provided, it is similarly advisable to provide as many as possible. While developing the subroutines OBJFUN and CONFUN, the **Verify** parameter (see Section 5.2) should be used to check the calculation of any known gradients.

4.1. Subroutine OBJFUN

This subroutine must calculate the objective function $F(x)$ and (optionally) the gradient $g(x)$. The specification of OBJFUN is

```
SUBROUTINE OBJFUN( MODE, N, X, OBJF, OBJGRD, NSTATE )
INTEGER            MODE, N, NSTATE
REAL               OBJF
REAL               X(N), OBJGRD(N)
```

Parameters:

MODE — **(Input)** This parameter is set by NPSOL to indicate the values that must be assigned during each call of OBJFUN. MODE will always have the value 2 if all components of the objective gradient are specified by the user, i.e., if **Derivative Level** is either 1 or 3 (see Section 5.2). If some gradient elements are unspecified, NPSOL will call OBJFUN with MODE = 0, 1 or 2.

If MODE = 2, compute OBJF and the available components of OBJGRD.

If MODE = 1, compute all available components of OBJGRD; OBJF is not required.

If MODE = 0, only OBJF needs to be computed; OBJGRD is ignored.

(Output) If for some reason you wish to terminate the solution of the current problem, set MODE to a negative value, e.g., −1.

N — **(Input)** The number of variables, i.e., the dimension of X. The actual parameter N will always be the same Fortran variable as that input to NPSOL, and *must not be altered by* OBJFUN.

X — **(Input)** An array of dimension at least N containing the values of the variables x for which F must be evaluated. *The array* X *must not be altered by* OBJFUN.

OBJF — **(Output)** The computed value of the objective function $F(x)$.

OBJGRD — **(Output)** The available components of the gradient vector $g(x)$, i.e., OBJGRD(j) contains the partial derivative $\partial F/\partial x_j$.

NSTATE — **(Input)** If NSTATE = 1, NPSOL is calling OBJFUN for the first time. This parameter setting allows the user to save computation time if certain data must be read or calculated only once. If there are nonlinear constraints, the first call to CONFUN will occur before the first call to OBJFUN.

4.2. Subroutine CONFUN

This subroutine must compute the nonlinear constraint functions $c(x)$ and (optionally) their gradients. (A dummy subroutine CONFUN must be provided if all constraints are linear.) The i-th row of the Jacobian matrix CJAC is the vector $\nabla c_i \equiv (\partial c_i/\partial x_1, \partial c_i/\partial x_2, \ldots, \partial c_i/\partial x_n)^T$. The specification of CONFUN is

```
SUBROUTINE CONFUN( MODE, NCNLN, N, NROWJ,
                   NEEDC, X, C, CJAC, NSTATE )
INTEGER            MODE, NCNLN, N, NROWJ
INTEGER            NEEDC(*)
REAL               X(N), C(*), CJAC(NROWJ,*)
```

Parameters:

MODE (Input) This parameter is set by NPSOL to indicate the values that must be assigned during each call of CONFUN. MODE will always have the value 2 if all elements of the Jacobian are available, i.e., if **Derivative Level** is either 2 or 3 (see Section 5.2). If some elements of CJAC are unspecified, NPSOL will call CONFUN with MODE = 0, 1, or 2:

If MODE = 2, only the elements of C corresponding to positive values of NEEDC need to be set (and similarly for the available components of the rows of CJAC).

If MODE = 1, the available components of the rows of CJAC corresponding to positive values in NEEDC must be set. Other rows of CJAC and the array C will be ignored.

If MODE = 0, the components of C corresponding to positive values in NEEDC must be set. Other components and the array CJAC are ignored.

(Output) If for some reason you wish to terminate the solution of the current problem, set MODE to a negative value, e.g., −1.

NCNLN (Input) The number of nonlinear constraints, i.e., the dimension of C. The actual parameter NCNLN is the same Fortran variable as that input to NPSOL, and *must not be altered by* CONFUN.

N (Input) The number of variables, i.e., the dimension of X. The actual parameter N is the same Fortran variable as that input to NPSOL, and *must not be altered by* CONFUN.

NROWJ (Input) The leading dimension of the array CJAC. NROWJ must be at least 1 and at least NCNLN.

NEEDC (Input) An array that specifies the indices of the elements of C or CJAC that *must be* evaluated by CONFUN. NEEDC need not be checked if the user always provides all values, since the unneeded values are ignored.

X (Input) An array of dimension at least N containing the values of the variables X for which the constraints must be evaluated. X *must not be altered by* CONFUN.

C (Output) An array of dimension at least NCNLN that contains the appropriate values of the nonlinear constraints. If NEEDC(i) > 0 and MODE = 0 or 2, the value of the i-th constraint at X must be stored in C(i). (The other components of C are ignored.)

CJAC (Output) A real array of declared dimension (NROWJ,*), where the second dimension must be at least N, containing the appropriate elements of the Jacobian matrix evaluated at X. (See the discussion of MODE and CJAC above.)

The parameter NSTATE has the same meaning as for OBJFUN.

4.3. Constant Jacobian elements

If all constraint gradients (Jacobian elements) are known (i.e., Derivative Level = 2 or 3; see Section 5.2), any *constant* elements may be assigned to CJAC one time only at the start of the optimization. An element of CJAC that is not subsequently assigned in CONFUN will retain its initial value throughout. Constant elements may be loaded into CJAC either *before* the call to NPSOL or during the the first call to CONFUN (signalled by the value NSTATE = 1). The ability to preload constants is useful when many Jacobian elements are identically zero, in which case CJAC may be initialized to zero and non-zero elements may be reset by CONFUN.

Note that constant *nonzero* elements do affect the values of the constraints. Thus, if CJAC(i, j) is set to a constant value, it need not be reset in subsequent calls to CONFUN, but the value CJAC(i, j)*X(j) must nonetheless be added to C(i).

It must be emphasized that, if Derivative Level < 2, unassigned elements of CJAC are *not* treated as constant; they are estimated by finite differences, at non-trivial expense. If the user does not supply a value for Difference Interval (see Section 5.2), an interval for each component of x is computed automatically at the start of the optimization. The automatic procedure can usually identify constant elements of CJAC, which are then computed once only by finite differences.

5. OPTIONAL INPUT PARAMETERS

Several optional parameters in NPSOL define choices in the problem specification or the algorithm logic. In order to reduce the number of formal parameters of NPSOL, these optional parameters have associated *default values* (see Section 5.2) that are appropriate for most problems. Therefore, the user needs to specify only those optional parameters whose values are to be different from their default values. The remainder of this section can be skipped by users who wish to use the default values for *all* optional parameters. A complete list of optional parameters and their default values is given in Section 5.3.

Each optional parameter is defined by a single character string of up to 72 characters, including one or more *items*. The items associated with a given option must be separated by spaces or equal signs (=). Alphabetic characters may be upper or lower case. The string

```
Print level = 5
```

is an example of an optional parameter.

For each option, the string contains the following items.

1. The *keyword* (required for all options).
2. A *phrase* (one or two words) that qualifies the keyword (only for some options).
3. A *number* that specifies either an `INTEGER` or a `REAL` value (only for some options). Such numbers may be up to 16 contiguous characters in Fortran 77's `I`, `F`, `E` or `D` formats, terminated by a space.

Blank strings and comments are ignored and may be used to improve readability. A *comment* begins with an asterisk (*) and all subsequent characters are ignored. If the string is not a comment and is not recognized, a warning message is printed on the specified output device (see Section 8.5). Synonyms are recognised for some of the keywords, and abbreviations may be used.

The following are examples of valid option strings for NPSOL:

```
NOLIST
warm start
COLD START
Verify Constraint gradients
Start OBJECTIVE check at variable 9
Stop constraint check at variable = 20 * The '=' is optional
Linear Feasibility tolerance    1.0E-8 * for IBM in double precision.
CRASH TOLERANCE = .002
* This string will be completely ignored.
Hessian Yes
Iteration limit 100
```

5.1. Specification of the optional parameters

Optional parameters may be specified in two ways, as follows.

- **Using subroutine `NPFILE` and an external file**

The subroutine `NPFILE` provided with the NPSOL package will read options from an external *options file*, and should be called *before* a call to NPSOL. Each line of the options file defines a single optional parameter. The file must begin with `Begin` and end with `End`. (An options file consisting only of these two lines corresponds to supplying no options.)

The specification of NPFILE is

```
SUBROUTINE NPFILE( IOPTNS, INFORM )
INTEGER            IOPTNS, INFORM
```

IOPTNS must be the unit number of the options file, in the range [0,99], and is unchanged on exit from NPFILE. INFORM need not be set on entry. On return, INFORM will be 0 if the file is a valid options file and IOPTNS is in the correct range. INFORM will be set to 1 if IOPTNS is out of range, and will be set to 2 if the file does not begin with **Begin** or end with **End**.

An example of a valid options file is

```
Begin
   Print level = 5
   Verify Objective Gradients
End
```

The call

```
CALL NPFILE( 5, INFORM )
```

will read an options file on unit 5.

- **Using subroutine NPOPTN**

The second method of setting the optional parameters is through a series of calls to the subroutine NPOPTN provided with the NPSOL package. The specification of NPOPTN is

```
SUBROUTINE NPOPTN( STRING )
CHARACTER*(*)      STRING
```

STRING must be a single valid option string (see above), and will be unchanged on exit. NPOPTN must be called once for every optional parameter to be set. An example of a call to NPOPTN is

```
CALL NPOPTN( 'Print level = 5' )
```

- **Use of the Nolist and Defaults option**

In general, each user-specified optional parameter is printed as it is read or defined. By using the special parameter Nolist, the user may suppress this printing for a given call of NPSOL. To take effect, Nolist must be the first parameter specified in the options file; for example

```
Begin
   Nolist
   Verify Objective Gradients
End
```

Alternatively, the first call to NPOPTN, before or after a call to NPSOL, must be

```
CALL NPOPTN( 'Nolist' ).
```

All parameters not specified by the user are automatically set to their default values. Any optional parameters that are set by the user are not altered by NPSOL, and hence changes to the options are *cumulative*. For example, calling `NPOPTN( 'Print level = 5' )` sets the print level to 5 for all subsequent calls to NPSOL until it is reset by the user. The only exception to this rule is permitted by the special optional parameter **Defaults**, whose effect is to reset *all* optional parameters to their default values (see Section 5.3). For example, in the following situation

```
      CALL NPSOL ( ...  )
C
      CALL NPOPTN( 'Print level 5' )
      CALL NPOPTN( 'Iteration limit = 100' )
      CALL NPSOL ( ...  )
C
      CALL NPOPTN( 'Defaults' )
      CALL NPSOL ( ...  )
```

the first and last runs of NPSOL will occur with the default parameter settings. However, in the second run, the print level and iteration limit are altered.

5.2. Description of the optional parameters

The following list (in alphabetical order) gives the valid options. For each option, we give the keyword, any essential optional qualifiers, the default value, and the definition. The minimum valid abbreviation of each keyword is underlined. If no characters of an optional qualifier are underlined, the qualifer may be omitted. The letter a denotes a phrase (character string) that qualifies an option. The letters i and r denote INTEGER and REAL values required with certain options. The *number* ϵ *is a generic notation* for machine precision, and ϵ_R denotes the relative precision of the objective function (the optional parameter **Function Precision**; see below).

Central Difference Interval r Default values are computed

If the algorithm switches to central differences because the forward-difference approximation is not sufficiently accurate, the value of r is used as the difference interval for every component of x. The use of finite-differences is discussed further below under the optional parameter **Difference Interval**.

Cold Start Default = **Cold Start**
Warm Start

This option controls the specification of the initial working set in both the procedure for finding a feasible point for the linear constraints and bounds, and in the first QP subproblem thereafter. With a **Cold Start**, the first working set is chosen by NPSOL based on the values of the variables and constraints at the initial point. Broadly speaking, the initial working set will include equality constraints and bounds or inequality constraints that violate or "nearly" satisfy their bounds (within **Crash Tolerance**; see below). With a **Warm Start**, the user must set the ISTATE array and define CLAMDA and R as discussed in Section 3. ISTATE values associated with bounds and linear constraints determine the initial working set of the procedure to find a feasible point with respect to the bounds and linear constraints. ISTATE values associated with nonlinear constraints determine the initial working set of the first QP subproblem after such a feasible point has been found. NPSOL will override the user's specification of ISTATE *if* necessary, so that a poor choice of the working set will not cause a fatal error. A warm start will be advantageous if a good estimate of the initial working set is available—for example, when NPSOL is called repeatedly to solve related problems.

Crash Tolerance r Default = .01

This value is used in conjunction with the optional parameter **Cold start** (the default value). When making a cold start, the QP algorithm in NPSOL must select an initial working set. When $r \geq 0$, the initial working set will include (if possible) bounds or general inequality constraints that lie within r of their bounds. In particular, a constraint of the form $a_j^T x \geq l$ will be included in the initial working set if $|a_j^T x - l| \leq r(1 + |l|)$. If $r < 0$ or $r > 1$, the default value is used.

Derivative Level i Default = 3

This parameter indicates which derivatives are provided by the user in subroutines OBJFUN and CONFUN. The possible choices for i are the following.

i	**Meaning**
3	All objective and constraint gradients are provided by the user.
2	All of the Jacobian is provided, but some components of the objective gradient are not specified by the user.
1	All elements of the objective gradient are known, but some elements of the Jacobian matrix are not specified by the user.
0	Some elements of both the objective gradient and the Jacobian matrix are not specified by the user.

The value $i = 3$ should be used whenever possible, since NPSOL is more reliable and will usually be more efficient when all derivatives are exact.

If $i = 0$ or 2, NPSOL will *estimate* the unspecified components of the objective gradient, using finite differences. The computation of finite-difference approximations usually increases the total run-time, since a call to OBJFUN is required for each unspecified element. Furthermore, less accuracy can be attained in the solution (see Chapter 8 of Gill, Murray and Wright, 1981, for a discussion of limiting accuracy).

If $i = 0$ or 1, NPSOL will approximate unspecified elements of the Jacobian. One call to CONFUN is needed for each *variable* for which partial derivatives are not available. For example, if the Jacobian has the form

$$\begin{pmatrix} * & * & * & * \\ * & ? & ? & * \\ * & * & ? & * \\ * & * & * & * \end{pmatrix}$$

where "*" indicates an element provided by the user and "?" indicates an unspecified element, NPSOL will call CONFUN twice: once to estimate the missing element in column 2, and again to estimate the two missing elements in column 3. (Since columns 1 and 4 are known, they require no calls to CONFUN.)

At times, central differences are used rather than forward differences, in which case twice as many calls to OBJFUN and CONFUN are needed. (The switch to central differences is not under the user's control.)

Difference Interval r Default values are computed

This option defines an interval used to estimate gradients by finite differences in the following circumstances:

1. For verifying the objective and/or constraint gradients (see the description of **Verify**, below).

2. For estimating unspecified elements of the objective gradient or the Jacobian matrix.

In general, a derivative with respect to the j-th variable is approximated using the interval δ_j, where $\delta_j = r(1 + |\hat{x}_j|)$, with $\hat{x}$ the first point feasible with respect to the bounds and linear constraints. If the functions are well scaled, the resulting derivative approximation should be accurate to $O(r)$. See Gill, Murray and Wright (1981) for a discussion of the accuracy in finite-difference approximations.

If a difference interval is not specified by the user, a finite-difference interval will be computed automatically for each variable by a procedure that requires up to six calls of CONFUN and OBJFUN for each component. This option is recommended if the function is badly scaled or the user wishes to have NPSOL determine constant elements in the objective and constraint gradients (see the descriptions of CONFUN and OBJFUN in Section 4).

Feasibility Tolerance r Default $= \sqrt{\epsilon}$

The scalar r defines the maximum acceptable *absolute* violations in linear and nonlinear constraints at a "feasible" point; i.e., a constraint is considered satisfied if its violation does not exceed r. If $r \le 0$, the default value is used. Using this keyword sets both optional parameters **Linear Feasibility Tolerance** and **Nonlinear Feasibility Tolerance** to r. (Additional details are given below under the descriptions of these parameters.)

Function Precision r Default $= \epsilon^{0.9}$

This parameter defines ϵ_R, which is intended to be a measure of the accuracy with which the problem functions F and c can be computed. The value of ϵ_R should reflect the relative precision of $1+|F(x)|$; i.e., ϵ_R acts as a relative precision when $|F|$ is large, and as an absolute precision when $|F|$ is small. For example, if $F(x)$ is typically of order 1000 and the first six significant digits are known to be correct, an appropriate value for ϵ_R would be 1.0E-6. In contrast, if $F(x)$ is typically of order 10^{-4} and the first six significant digits are known to be correct, an appropriate value for ϵ_R would be 1.0E-10. The choice of ϵ_R can be quite complicated for badly scaled problems; see Chapter 8 of Gill, Murray and Wright (1981) for a discussion of scaling techniques. The default value is appropriate for most simple functions that are computed with full accuracy. However, when the accuracy of the computed function values is known to be significantly worse than full precision, the value of ϵ_R should be large enough so that NPSOL will not attempt to distinguish between function values that differ by less than the error inherent in the calculation.

Hessian **No** **Default = No**
Hessian **Yes**

This option controls the contents of the upper-triangular matrix R (see Section 3). NPSOL works exclusively with the *transformed and re-ordered* Hessian H_Q (5), and hence extra computation is required to form the Hessian itself. If **Hessian = No**, R contains the Cholesky factor of the transformed and re-ordered Hessian. If **Hessian = Yes**, the Cholesky factor of the approximate Hessian itself is formed and stored in R. The user should select **Hessian = Yes** if a warm start will be used for the next call to NPSOL.

Infinite Bound Size r Default $= 10^{10}$

If $r > 0$, r defines the "infinite" bound BIGBND in the definition of the problem constraints. Any upper bound greater than or equal to BIGBND will be regarded as plus infinity (and similarly for a lower bound less than or equal to −BIGBND). If $r \le 0$, the default value is used.

Infinite Step Size r Default $= \max(\text{BIGBND}, 10^{10})$

If $r > 0$, r specifies the magnitude of the change in variables that is treated as a step to an unbounded solution. If the change in x during an iteration would exceed the value of **Infinite**

Step, the objective function is considered to be unbounded below in the feasible region. If $r \leq 0$, the default value is used.

`Iteration Limit` i Default $= \max(50, 3(n + m_L) + 10m_N)$

See `Major Iteration Limit` below.

`Linear Feasibility Tolerance` r_1 Default $= \sqrt{\epsilon}$
`Nonlinear Feasibility Tolerance` r_2 Default $= \sqrt{\epsilon}$

The scalars r_1 and r_2 define the maximum acceptable *absolute* violations in linear and nonlinear constraints at a "feasible" point; i.e., a linear constraint is considered satisfied if its violation does not exceed r_1, and similarly for a nonlinear constraint and r_2. The default values are used if r_1 or r_2 is non-positive.

On entry to NPSOL, an iterative procedure is executed in order to find a point that satisfies the linear constraint and bounds on the variables to within the tolerance r_1. All subsequent iterates will satisfy the linear constraints to within the same tolerance (unless r_1 is comparable to the finite-difference interval).

For nonlinear constraints, the feasibility tolerance r_2 defines the largest constraint violation that is acceptable at an optimal point. Since nonlinear constraints are generally not satisfied until the final iterate, the value of `Nonlinear Feasibility Tolerance` acts as a partial *termination criterion* for the iterative sequence generated by NPSOL (see the discussion of `Optimality Tolerance`).

These tolerances should reflect the precision of the corresponding constraints. For example, if the variables and the coefficients in the linear constraints are of order unity, and the latter are correct to about 6 decimal digits, it would be appropriate to specify r_1 as 10^{-6}.

`Linesearch Tolerance` r Default = `0.9`

The value r ($0 \leq r < 1$) controls the accuracy with which the step α taken during each iteration approximates a minimum of the merit function along the search direction (the smaller the value of r, the more accurate the linesearch). The default value $r =$ `0.9` requests an inaccurate search, and is appropriate for most problems, particularly those with any nonlinear constraints.

If there are no nonlinear constraints, a more accurate search may be appropriate when it is desirable to reduce the number of major iterations—for example, if the objective function is cheap to evaluate, or if a substantial number of gradients are unspecified.

`Major Iteration Limit` i Default $= \max(50, 3(n + m_L) + 10m_N)$
`Iteration Limit`
`Iters`
`Itns`

The value of i specifies the maximum number of major iterations allowed before termination. Setting $i = 0$ and `Major Print Level` > 0 means that the workspace needed will be computed and printed, but no iterations will be performed.

`Major Print Level` i Default = `10`
`Print Level`

The value of i controls the amount of printout produced by the major iterations of NPSOL. (See also `Minor Print Level`, below). The levels of printing available are indicated below.

i	**Output**
0	No output.
1	The final solution only.
5	One line of output for each major iteration (no printout of the final solution).
≥ 10	The final solution and one line of output for each iteration.
≥ 20	At each major iteration, the objective function, the Euclidean norm of the nonlinear constraint violations, the values of the nonlinear constraints (the array c), the values of the linear constraints (the array $A_L x$), and the current values of the variables (the array x).
≥ 30	At each major iteration, the diagonal elements of the matrix T associated with the TQ factorization (4) of the QP working set, and the diagonal elements of R, the triangular factor of the transformed and re-ordered Hessian (5).

`Minor Iteration Limit` i Default $= \max(50, 3(n + m_L + m_N))$

The value of i specifies the maximum number of iterations for the optimality phase of each QP subproblem.

`Minor Print Level` i Default = 0

The value of i controls the amount of printout produced by the minor iterations of NPSOL, i.e., the iterations of the quadratic programming algorithm. (See also `Major Print Level`, above.) The following levels of printing are available.

i	**Output**
0	No output.
1	The final QP solution.
5	One line of output for each minor iteration (no printout of the final QP solution).
≥ 10	The final QP solution and one brief line of output for each minor iteration.
≥ 20	At each minor iteration, the current estimates of the QP multipliers, the current estimate of the QP search direction, the QP constraint values, and the status of each QP constraint.
≥ 30	At each minor iteration, the diagonal elements of the matrix T associated with the TQ factorization (4) of the QP working set, and the diagonal elements of the Cholesky factor R of the transformed Hessian (5).

`Nonlinear Feasibility Tolerance` r Default $= \sqrt{\epsilon}$

See `Linear Feasibility Tolerance`, above.

`Optimality Tolerance` r Default $= \epsilon_R^{0.8}$

The parameter r ($\epsilon_R \leq r \leq 1$) specifies the accuracy to which the user wishes the final iterate to approximate a solution of the problem. Broadly speaking, r indicates the number of correct figures desired in the objective function at the solution. For example, if r is 10^{-6} and NPSOL terminates successfully, the final value of F should have approximately six correct figures.

NPSOL will terminate successfully if the iterative sequence of x-values is judged to have converged and the final point satisfies the first-order Kuhn-Tucker conditions (see Section 2). The sequence of iterates is considered to have converged at x if

$$\alpha\|p\| \leq \sqrt{r}(1 + \|x\|), \tag{15a}$$

where p is the search direction and α the step length from (2). An iterate is considered to satisfy the first-order conditions for a minimum if

$$\|Z^T g_{FR}\| \leq \sqrt{r}\big(1 + \max(1 + |F(x)|, \|g_{FR}\|)\big) \tag{15b}$$

and

$$|res_j| \leq ftol \quad \text{for all } j, \tag{15c}$$

where $Z^T g_{FR}$ is the projected gradient (see Section 2), g_{FR} is the gradient of $F(x)$ with respect to the free variables, res_j is the violation of the j-th active nonlinear constraint, and *ftol* is the `Nonlinear Feasibility Tolerance`.

`Start Objective Check At Variable`	k	Default = 1
`Start Constraint Check At Variable`	k	Default = 1
`Stop Objective Check At Variable`	l	Default = n
`Stop Constraint Check At Variable`	l	Default = n

These keywords take effect only if `Verify level` > 0 (see below). They may be used to control the verification of gradient elements computed by subroutines `OBJFUN` and `CONFUN`. For example, if the first 30 components of the objective gradient appeared to be correct in an earlier run, so that only component 31 remains questionable, it is reasonable to specify `Start Objective Check At Column 31`. If the first 30 variables appear linearly in the objective, so that the corresponding gradient elements are constant, the above choice would also be appropriate.

`Verify Level`	i	Default = 0

```
Verify                        No
Verify Level                  -1

Verify Level                  0

Verify Objective Gradients
Verify Level                  1

Verify Constraint Gradients
Verify Level                  2

Verify
Verify                        Yes
Verify Gradients
Verify Level                  3
```

These keywords refer to finite-difference checks on the gradient elements computed by the user-provided subroutines `OBJFUN` and `CONFUN`. (Unspecified gradient components are not checked.) It is possible to specify `Verify Levels 0–3` in several ways, as indicated above. For example, the nonlinear objective gradient (if any) will be verified if either `Verify Objective` or `Verify Level`

1 is specified. Similarly, the objective and the constraint gradients will be verified if **Verify Yes** or **Verify Level 3** or **Verify** is specified.

If $0 \le i \le 3$, gradients will be verified at the first point that satisfies the linear constraints and bounds. If $i = 0$, only a "cheap" test will be performed, requiring one call to OBJFUN and one call to CONFUN. If $1 \le i \le 3$, a more reliable (but more expensive) check will be made on individual gradient components, within the ranges specified by the **Start** and **Stop** keywords described above. A result of the form "OK" or "BAD?" is printed by NPSOL to indicate whether or not each component appears to be correct.

If $10 \le i \le 13$, the action is the same as for $i - 10$, except that it will take place at the user-specified initial value of x.

We suggest that **Verify Level 3** be specified whenever a new function routine is being developed.

5.3. Optional parameter checklist and default values

For easy reference, the following sample NPOPTN list shows all valid keywords and their default values. The default options **Function Precision**, **Linear Feasibility Tolerance**, **Nonlinear Feasibility Tolerance** and **Optimality Tolerance** depend upon ϵ, the relative precision of the machine being used. The values given here correspond to double precision arithmetic on IBM 360 and 370 systems and their successors ($\epsilon \approx 2.22 \times 10^{-16}$). Similar values would apply to any machine having about 16 decimal digits of precision.

* List of optional parameters.			
Central Difference Interval	?	*	Computed automatically
Cold Start		*	
Crash Tolerance	.01	*	
Derivative Level	3	*	
Difference Interval	?	*	Computed automatically
Function Precision	8.2E-15	*	$\epsilon^{0.9}$
Hessian	No	*	
Infinite Bound	1.0E+10	*	Plus infinity
Infinite Step	1.0E+10	*	
Linear Feasibility Tolerance	1.5E-8	*	$\sqrt{\epsilon}$
Linesearch Tolerance	0.9	*	
Major Iteration Limit	50	*	or $3(n + m_L) + 10m_N$
Major Print Level	10	*	
Minor Iteration Limit	50	*	or $3(n + m_L + m_N)$
Minor Print Level	0	*	
Nonlinear Feasibility Tolerance	1.5E-8	*	$\sqrt{\epsilon}$
Optimality Tolerance	5.4E-12	*	$\epsilon^{0.8}$
Start Objective Check	1	*	
Start Constraint Check	1	*	
Stop Objective Check	?	*	n
Stop Constraint Check	?	*	n
Verify Level	0	*	Cheap test

6. DESCRIPTION OF THE PRINTED OUTPUT

The level of printed output from NPSOL is controlled by the user (see the descriptions of **Major Print Level** and **Minor Print Level** in Section 5.2). If **Minor Print Level** > 0, output is obtained from the subroutines that solve the QP subproblem. For a detailed description of this information the reader should refer to the user's guide for LSSOL (Gill *et al.*, 1986a).

When **Major Print Level** ≥ 5, the following line of output is produced at every major iteration of NPSOL. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

Itn is the iteration count.

ItQP is the sum of the iterations required by the feasibility and optimality phases of the QP subproblem. Generally, ItQP will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 2).

Note that ItQP may be greater than the **Minor Iteration Limit** if some iterations are required for the feasibility phase.

Step is the step α taken along the computed search direction. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

Nfun is the cumulative number of evaluations of the objective function needed for the linesearch. Evaluations needed for the estimation of the gradients by finite differences are not included. Nfun is printed as a guide to the amount of work required for the linesearch.

Merit is the value of the augmented Lagrangian merit function (11) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section 2.2). As the solution is approached, Merit will converge to the value of the objective function at the solution.

If the QP subproblem does not have a feasible point (signified by "I" at the end of the current output line), the merit function is a large multiple of the constraint violations, weighted by the penalty parameters. During a sequence of major iterations with infeasible subproblems, the sequence of Merit values will decrease monotonically until either a feasible subproblem is obtained or NPSOL terminates with INFORM $= 3$ (no feasible point could be found for the nonlinear constraints).

If no nonlinear constraints are present (i.e., NCNLN $= 0$), this entry contains Objective, the value of the objective function $F(x)$. The objective function will decrease monotonically to its optimal value when there are no nonlinear constraints.

Bnd is the number of simple bound constraints in the predicted active set.

Lin is the number of general linear constraints in the predicted active set.

Nln is the number of nonlinear constraints in the predicted active set (not printed if NCNLN is zero).

Nz is the number of columns of Z (see Section 2.1). The value of Nz is the number of variables minus the number of constraints in the predicted active set; i.e., Nz $=$ N $-$ (Bnd $+$ Lin $+$ Nln).

Norm Gf is the Euclidean norm of g_{FR}, the gradient of the objective function with respect to the free variables, i.e., variables not currently held at a bound.

Norm Gz is $\|Z^T g_{FR}\|$, the Euclidean norm of the projected gradient (see Section 2.1). Norm Gz will be approximately zero in the neighborhood of a solution.

Cond H is a lower bound on the condition number of the Hessian approximation H.

Cond Hz is a lower bound on the condition number of the projected Hessian approximation H_Z ($H_Z = Z^T H_{FR} Z = R_Z^T R_Z$; see (5) and (10) in Section 2). The larger this number, the more difficult the problem.

Cond T is a lower bound on the condition number of the matrix of predicted active constraints.

Norm C is the Euclidean norm of the residuals of constraints that are violated or in the predicted active set (not printed if NCNLN is zero). Norm C will be approximately zero in the neighborhood of a solution.

Penalty is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if NCNLN is zero).

Conv is a three-letter indication of the status of the three convergence tests (15a)-(15c) defined in the description of the optional parameter Optimality Tolerance in Section 5. Each letter is "T" if the test is satisfied, and "F" otherwise. The three tests indicate whether: (a) the sequence of iterates has converged; (b) the projected gradient (Norm Gz) is sufficiently small; and (c) the norm of the residuals of constraints in the predicted active set (Norm C) is small enough.

If any of these indicators is "F" when NPSOL terminates with INFORM = 0, the user should check the solution carefully.

M is printed if the quasi-Newton update was modified to ensure that the Hessian approximation is positive-definite (see Section 2.3).

I is printed if the QP subproblem has no feasible point.

C is printed if central differences were used to compute the unspecified objective and constraint gradients. If the value of Step is zero, the switch to central differences was made because no lower point could be found in the linesearch. (In this case, the QP subproblem is re-solved with the central-difference gradient and Jacobian.) If the value of Step is non-zero, central differences were computed because Norm Gz and Norm C imply that X is close to a Kuhn-Tucker point.

When Major Print Level = 1 or Major Print Level $\geq$ 10, the summary printout at the end of execution of NPSOL includes a listing of the status of every variable and constraint. Note that default names are assigned to all variables and constraints.

The following describes the printout for each variable.

Variable gives the name (VARBL) and index j (j = 1 to N) of the variable.

State gives the state of the variable in the predicted active set (FR if neither bound is in the active set, EQ if a fixed variable, LL if on its lower bound, UL if on its upper bound). If the variable is predicted to lie outside its upper or lower bound by more than the feasibility tolerance, State will be "++" or "--" respectively. (The latter situation can occur only when there is no feasible point for the bounds and linear constraints.)

`Value`	is the value of the variable at the final iteration.
`Lower bound`	is the lower bound specified for the variable. ("`None`" indicates that BL(j) $\leq$ −BIGBND.)
`Upper bound`	is the upper bound specified for the variable. ("`None`" indicates that BU(j) $\geq$ BIGBND.)
`Lagr multiplier`	is the value of the Lagrange multiplier for the associated bound constraint. This will be zero if `State` is FR. If X is optimal, the multiplier should be non-negative if `State` is LL, and non-positive if `State` is UL.
`Residual`	is the difference between the variable "`Value`" and the nearer of its bounds BL(j) and BU(j).

The printout for general constraints is the same as for variables, except for the following:

`Linear constr`	is the name (LNCON) and index i ($i = 1$ to NCLIN) of a linear constraint.
`Nonlnr constr`	is the name (NLCON) and index i ($i = 1$ to NCNLN) of a nonlinear constraint.

7. INTERPRETATION OF THE RESULTS

The input data for NPSOL should always be checked (even if NPSOL terminates with the value INFORM = 0!). Two common sources of error are uninitialized variables and incorrect gradients, which may cause underflow or overflow on some machines. The user should check that all components of A, BL, BU and X are defined on entry to NPSOL, and that OBJFUN and CONFUN compute all relevant components of OBJGRD, C and CJAC.

In the following, we list the different ways in which NPSOL is terminated and discuss what further action may be necessary.

Termination	Discussion and Recommended Action
Underflow	A single underflow will always occur if machine constants are computed automatically (as in the distributed version of NPSOL; see Section 8). Other floating-point underflows may occur occasionally, but can usually be ignored.
Overflow	If the printed output before the overflow error contains a warning about serious ill-conditioning in the working set when adding the j-th constraint, it may be possible to avoid the difficulty by increasing the magnitude of the optional parameter `Linear Feasibility Tolerance` or `Nonlinear Feasibility Tolerance`, and rerunning the program. If the message recurs even after this change, the offending linearly dependent constraint (with index "j") must be removed from the problem. If overflow occurs in one of the user-supplied routines (e.g., if the nonlinear functions involve exponentials or singularities), it may help to specify tighter bounds for some of the variables (i.e., reduce the gap between appropriate ℓ_j and u_j). If overflow continues to occur for no apparent reason, contact the authors at Stanford University.
INFORM = 0	The iterates have converged to a point X that satisfies the first-order Kuhn-Tucker conditions to the accuracy requested by the optional parameter `Optimality tolerance` (see Section 5.2), i.e., the projected gradient and active constraint residuals are negligible at X. The user should check whether the following four conditions are satisfied: *(i)* the final value of `Norm Gz` is significantly less than that at the starting point; *(ii)* during the final major iterations, the values of `Step` and `ItQP` are both one; *(iii)* the last few values of both `Norm Gz` and `Norm C` become small at a fast linear rate; and *(iv)* `Cond Hz` is small. If all these conditions hold, X is almost certainly a local minimum of NP. (See Section 9 for a specific example.)
INFORM = 1	The point X satisfies the Kuhn-Tucker conditions to the accuracy requested, but the sequence of iterates has not yet converged. NPSOL was terminated because no further improvement could be made in the merit function. This value of INFORM may occur in several circumstances. The most common situation is that the user asks for a solution with accuracy that is not attainable with the given precision of the problem (as specified by `Function Precision`; see Section 5.2). This condition will also occur if, by chance, an iterate is an "exact" Kuhn-Tucker point, but the change in the variables was significant at the previous iteration. (This situation often happens when minimizing very simple functions, such as quadratics.) If the four conditions listed above for INFORM = 0 are satisfied, X is likely to be a solution of NP *regardless of the value of* INFORM.

INFORM = 2 NPSOL has terminated without finding a feasible point for the linear constraints and bounds, which means that no feasible point exists for the given value of **Linear Feasibility Tolerance**. The user should check that there are no constraint redundancies. If the data for the constraints are accurate only to an absolute precision σ, the user should ensure that the value of the optional parameter **Linear Feasibility Tolerance** is *greater* than σ. For example, if all elements of A are of order unity and are accurate to only three decimal places, **Linear Feasibility Tolerance** should be at least 10^{-3}.

INFORM = 3 There has been a sequence of QP subproblems for which no feasible point could be found (indicated by "I" at the end of each terse line of output). This behavior will occur if there is no feasible point for the nonlinear constraints. (However, there is no general test that can determine whether a feasible point exists for a set of nonlinear constraints.) If the infeasible subproblems occur from the very first major iteration, it is highly likely that no feasible point exists. If infeasibilities occur when earlier subproblems have been feasible, small constraint inconsistencies may be present. The user should check the validity of constraints with negative values of ISTATE. If the user is convinced that a feasible point *does* exist, NPSOL should be restarted at a different starting point.

INFORM = 4 If the algorithm appears to be making progress, **Major Iteration Limit** may be too small. If so, increase its value and rerun NPSOL (possibly using the **Warm Start** option). If the algorithm seems to be "bogged down", the user should check for incorrect gradients or ill-conditioning as described below under INFORM = 6.

Note that ill-conditioning in the working set is sometimes resolved automatically by the algorithm, in which case performing additional iterations may be helpful. However, ill-conditioning in the Hessian approximation tends to persist once it has begun, so that allowing additional iterations without altering R is usually inadvisable. If the quasi-Newton update of the Hessian approximation was modified during the latter iterations (i.e., an "M" occurs at the end of each terse line), it may be worthwhile to try a warm start at the final point as suggested above.

INFORM = 6 A sufficient decrease in the merit function could not be attained during the final linesearch. This sometimes occurs because an overly stringent accuracy has been requested, i.e., **Optimality Tolerance** is too small. In this case the user should apply the four tests described under INFORM = 0 above to determine whether or not the final solution is acceptable (see Gill, Murray and Wright, 1981, for a discussion of the attainable accuracy).

If many iterations have occurred in which essentially no progress has been made, or NPSOL has failed completely to move from the initial point, subroutines OBJFUN or CONFUN may be incorrect. The user should refer to the comments below under INFORM = 7 and check the gradients using the **Verify** parameter. Unfortunately, there may be small errors in the objective and constraint gradients that cannot be detected by the verification process. Finite-difference approximations to first derivatives are catastrophically affected by even small inaccuracies. An indication of this situation is a dramatic alteration in the iterates if the finite-difference interval is altered. One might also suspect this type of error if a switch is made to central differences even when Norm Gz and Norm C are large.

Another possibility is that the search direction has become inaccurate because of ill-conditioning in the Hessian approximation or the matrix of constraints in the working set; either form of ill-conditioning tends to be reflected in large values of `ItQP` (the number of iterations required to solve each QP subproblem).

If the condition estimate of the projected Hessian (`Cond Hz`) is extremely large, it may be worthwhile to rerun NPSOL from the final point with the `Warm Start` option. In this situation, `ISTATE` should be left unaltered and `R` should be reset to the identity matrix.

If the matrix of constraints in the working set is ill-conditioned (i.e., `Cond T` is extremely large), it may be helpful to run NPSOL with a relaxed value of the `Feasibility Tolerance`. (Constraint dependencies are often indicated by wide variations in size in the diagonal elements of the matrix T, whose diagonals will be printed for `Major Print Level` ≥ 30.)

`INFORM` = 7 Large errors were found in the derivatives of the objective function and/or nonlinear constraints. This value of `INFORM` will occur if the verification process indicated that at least one gradient or Jacobian component had *no* correct figures. The user should refer to the printed output to determine which elements are suspected to be in error.

As a first step, the user should check that the code for the objective and constraint values is correct—for example, by computing the function at a point where the correct value is known. However, care should be taken that the chosen point fully tests the evaluation of the function. It is remarkable how often the values $x = 0$ or $x = 1$ are used to test function evaluation procedures, and how often the special properties of these numbers make the test meaningless.

Special care should be used in this test if computation of the objective function involves subsidiary data communicated in `COMMON` storage. Although the first evaluation of the function may be correct, subsequent calculations may be in error because some of the subsidiary data has accidentally been overwritten.

Errors in programming the function may be quite subtle in that the function value is "almost" correct. For example, the function may not be accurate to full precision because of the inaccurate calculation of a subsidiary quantity, or the limited accuracy of data upon which the function depends. A common error on machines where numerical calculations are usually performed in double precision is to include even one single-precision constant in the calculation of the function; since some compilers do not convert such constants to double precision, half the correct figures may be lost by such a seemingly trivial error.

`INFORM` = 9 An input parameter is invalid. The user should refer to the printed output to determine which parameter must be re-defined.

8. IMPLEMENTATION INFORMATION

8.1. Format of the distribution tape

The source code and example program for NPSOL are distributed on a magnetic tape containing 12 files. The tape characteristics are described in a document accompanying the tape; normally they are 9 track, 1600 bpi, unlabeled, ASCII, 80-character records (card images), 4800-character blocks.

The following is a list of the files and a summary of their contents. For reference purposes we give a name to each file. However, the names will not be recorded on unlabeled tapes. The MACH, LSCODE and NPCODE files are composed of several smaller source files described in Section 8.3.

File	*Name*	*Type*	*Cards*†	*Description*
1.	DPMACH	FORTRAN	450	Double-precision source file 1: MCSUBS
2.	DPLSCODE	FORTRAN	8250	Double-precision source files 2–5: BLAS, ..., OPSUBS
3.	DPNPCODE	FORTRAN	6880	Double-precision source files 6–8: CHSUBS, ..., SRSUBS
4.	DPLSMAIN	FORTRAN	260	Double-precision source file LSMAIN
5.	DPNPMAIN	FORTRAN	500	Double-precision source file NPMAIN
6.	LSMAIN	DATA	6	Options file for LSMAIN
7.	NPMAIN	DATA	14	Options file for NPMAIN
8.	SPMACH	FORTRAN	450	Single-precision source file 1
9.	SPLSCODE	FORTRAN	8250	Single-precision source files 2–5
10.	SPNPCODE	FORTRAN	6880	Single-precision source files 6–8
11.	SPLSMAIN	FORTRAN	260	Single-precision version of file 4
12.	SPNPMAIN	FORTRAN	500	Single-precision version of file 5

† Approximate figure.

One MACH, one LSCODE and one NPCODE file should be selected for any given installation. DPMACH, DPLSCODE and DPNPCODE are intended for machines that generally require double precision computation. Examples include IBM Systems 360, 370, 3033, 3081, etc.; Amdahl 470, Facom, Fujitsu, Hitachi, and other systems analogous to IBM; DEC VAX systems; Data General MV/8000; ICL 2900 series; recent PRIME systems; DEC Systems 10 and 20; Honeywell systems; and the Univac 1100 series.

SPMACH, SPLSCODE and SPNPCODE are intended for machines for which single precision is suitably accurate for numerical computation. Examples include the Burroughs 6700 and 7700 series; the CDC 6000 and 7000 series and their Cyber counterparts; and the Cray-1 and Cray-2.

8.2. Installation procedure

1. Obtain the appropriate MACH, LSCODE and NPCODE files from the tape.
2. If necessary, edit the subroutine MCHPAR according to Section 8.5.
3. Decide whether or not to split the LSCODE and NPCODE tape files into source files BLAS through SRSUBS as suggested in Section 8.3.
4. Compile all the routines that were originally in the LSCODE and NPCODE files together with those from MACH. Run them in conjunction with the main program NPMAIN from either file 5 or file 12. Check the output against that in Section 9.

8.3. Source files

NPSOL has been written in ANSI (1977) Fortran and tested on an IBM 3081K computer using the IBM Fortran 77 compiler VS Fortran. Certain unavoidable machine dependencies are confined to the routine MCHPAR.

The source code is divided into 8 logical parts. For ease of handling, these are combined into the MACH, LSCODE and NPCODE files on the distribution tape, but for subsequent maintenance we recommend that 8 separate files be kept. In the description below we suggest a name for each file and summarize its purpose. We then list the names of the Fortran subroutines and functions involved. The naming convention should minimize the risk of a clash with user-written routines.

File 1. MCSUBS *Computation of machine-dependent constants.*

MCHPAR MCEPS MCENV1 MCENV2 MCSTOR

File 2. BLAS *Basic Linear Algebra Subprograms* (a subset).

DASUM DAXPY DCOPY DDOT DNRM2 DSWAP DSCAL IDAMAX

These routines are functionally similar to members of the BLAS package (Lawson *et al.*, 1979). If possible they should be replaced by authentic BLAS routines. (Versions may exist that have been tuned to your particular machine.)

DGEMV DGER1

These routines are functionally similar to members of the Level 2 BLAS packages (Dongarra *et al.*, 1985).

DCOND DDIV DDSCL DLOAD DNORM DSSQ DSWAP ICOPY IDRANK ILOAD

These are additional utility routines that could be tuned to your machine. DLOAD is used the most frequently, to load a vector with a constant value.

DROT3 DROT3G DGEAPQ DGEQR DGEQRP DGRFG

These linear algebra routines are used to compute and update various matrix factorizations in NPSOL.

File 3. CMSUBS *General utility routines.*

CMALF CMALF1 CMCHK CMFEAS CMPRT CMQMUL CMRSOL CMRSWP CMR1MD CMTSOL

File 4. LSSUBS *Least-squares routines.*

LSADD LSADDS LSBNDS LSCHOL LSCORE LSCRSH LSDEL LSDFLT LSFEAS LSFILE LSGETP LSGSET LSKEY LSLOC LSMOVE LSMULS LSOPTN LSPRT LSSETX LSSOL

File 5. OPSUBS *Option string handling routines.*

OPFILE OPLOOK OPNUM OPSCAN OPTOKN OPUPPR

File 6. CHSUBS *Derivative checking routines.*

CHCORE CHFD CHKGRD CHKJAC

File 7. NPSUBS *Nonlinear optimization routines.*

NPCHKD	NPCORE	NPCRSH	NPDFLT	NPFEAS	NPFILE	NPGQ	NPIQP
NPKEY	NPLOC	NPMRT	NPOPTN	NPPRT	NPSETX	NPSRCH	NPUPDT
NPSOL							

File 8. SRSUBS *Linesearch routines.*

SRCHQ SRCHC

8.4. Common blocks

Certain Fortran COMMON blocks are used in the NPSOL source code to communicate between subroutines. Their names are listed below.

CMDEBG	LSDEBG	NPDEBG	LSPAR1	LSPAR2	NPPAR1	NPPAR2	SOL1CM
SOL3CM	SOL4CM	SOL5CM	SOL6CM	SOLMCH	SOL1NP	SOL4NP	SOL5NP
SOL6NP	SOL7NP	SOL1LS	SOL3LS	SOL1SV			

8.5. Machine-dependent subroutines

The routine MCHPAR in the MACH file may require modification to suit a particular machine or a non-standard application.

At the beginning of NPSOL, MCHPAR is called to assign the machine-dependent constants and the standard input and output unit numbers. These parameters are stored in the array WMACH(15) in the labeled COMMON block SOLMCH, and are defined as follows.

WMACH(1)	is NBASE, the base of floating-point arithmetic.
WMACH(2)	is NDIGIT, the number of NBASE digits of precision.
WMACH(3)	is EPS, the floating-point precision.
WMACH(4)	is RTEPS, the square root of EPSMCH.
WMACH(5)	is RMIN, the smallest positive floating-point number.
WMACH(6)	is RTMIN, the square root of RMIN.
WMACH(7)	is RMAX, the largest positive floating-point number.
WMACH(8)	is RTMAX, the square root of RMAX.
WMACH(10)	is NIN, the file number for the input stream.
WMACH(11)	is NOUT, the file number for the output stream.

Within routine MCHPAR, the machine constants are set in one of two ways, depending upon the value of the logical variable HDWIRE, which is set in-line.

If HDWIRE is .FALSE. (the value set for the distributed copy of MCHPAR), the machine constants are computed *automatically* for the machine being used. If HDWIRE is .TRUE., machine constants appropriate for the IBM 360/370 Series are assigned directly to the elements of WMACH.

Before selecting the method of assigning the machine constants, you should note the following. The computation of the machine constants will always generate a single arithmetic underflow, and hence some appropriate remedial action may need to be taken if your machine traps underflow.

If you wish to implement the in-line assignment of machine constants for a machine other than one from the IBM 360/370 Series, MCHPAR must be modified as follows.

1. Change the in-line assignment of HDWIRE from .FALSE. to .TRUE..

2. Set the values of WMACH appropriate for the machine and precision being used. The values of NBASE, NDIGIT, EPSMCH, RMIN and RMAX for several machines are given in the following table, for both single and double precision; RTEPS, RTMIN and RTMAX may be computed using Fortran statements. The values NIN and NOUT depend on the machine installation.
   For each precision, we give two values for EPSMCH, RMIN and RMAX. The first value is a Fortran decimal approximation of the exact quantity; use of this value in MCHPAR should cause no difficulty except in extreme circumstances. The second value is the exact mathematical representation.

Table of machine-dependent parameters

	IBM 360/370 Single	CDC 6000/7000 Single	DEC 10/20 Single	Univac 1100 Single	DEC Vax Single
NBASE	16	2	2	2	2
NDIGIT	6	48	27	27	24
EPS	9.54E-7 16^{-5}	7.11E-15 2^{-47}	7.46E-9 2^{-27}	1.50E-8 2^{-26}	1.20E-7 2^{-23}
RMIN	1.0E-78 16^{-65}	1.0E-293 2^{-975}	1.0E-38 2^{-129}	1.0E-38 2^{-129}	1.0E-38 2^{-128}
RMAX	1.0E+75 $16^{63}(1-16^{-6})$	1.0E+322 $2^{1070}(1-2^{-48})$	1.0E+38 $2^{127}(1-2^{-27})$	1.0E+38 $2^{127}(1-2^{-27})$	1.0E+38 $2^{127}(1-2^{-24})$

	IBM 360/370 Double	CDC 6000/7000 Double	DEC 10/20 Double	Univac 1100 Double	DEC Vax Double
NBASE	16	2	2	2	2
NDIGIT	14	96	62	61	56
EPS	2.22D-16 16^{-13}	2.53D-29 2^{-95}	2.17D-19 2^{-62}	8.68D-19 2^{-60}	2.78D-17 2^{-55}
RMIN	1.0D-78 16^{-65}	1.0D-293 2^{-975}	1.0D-38 2^{-129}	1.0D-308 2^{-1025}	1.0D-38 2^{-128}
RMAX	1.0D+75 $16^{63}(1-16^{-14})$	1.0D+322 $2^{1070}(1-2^{-96})$	1.0D+38 $2^{127}(1-2^{-62})$	1.0D+307 $2^{1023}(1-2^{-61})$	1.0D+38 $2^{127}(1-2^{-56})$

9. EXAMPLE PROBLEM

This section describes one version of the so-called "hexagon" problem (a different formulation is given as Problem 108 in Hock and Schittkowski, 1981). The problem is to determine the hexagon of maximum area such that no two of its vertices are more than one unit apart (the solution is *not* a regular hexagon). The corresponding sample main program and output from NPSOL are given in the Appendix.

All constraint types are included (bounds, linear, nonlinear), and the Hessian of the Lagrangian function is not positive definite at the solution. The problem has nine variables, non-infinite bounds on seven of the variables, four general linear constraints, and fourteen nonlinear constraints.

The objective function is

$$F(x) = -x_2x_6 + x_1x_7 - x_3x_7 - x_5x_8 + x_4x_9 + x_3x_8.$$

The bounds on the variables are

$$x_1 \ge 0, \quad -1 \le x_3 \le 1, \quad x_5 \ge 0, \quad x_6 \ge 0, \quad x_7 \ge 0, \quad x_8 \le 0, \quad \text{and} \quad x_9 \le 0.$$

Thus,

$$\ell_B = (\ 0,\ -\infty,\ -1,\ -\infty,\ 0,\ 0,\ 0,\ -\infty,\ -\infty)^T$$

$$u_B = (\ \infty,\ \infty,\ 1,\ \infty,\ \infty,\ \infty,\ \infty,\ 0,\ 0)^T.$$

The general linear constraints are

$$x_2 - x_1 \ge 0, \quad x_3 - x_2 \ge 0, \quad x_3 - x_4 \ge 0, \quad \text{and} \quad x_4 - x_5 \ge 0.$$

Hence,

$$\ell_L = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad A_L = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad u_L = \begin{pmatrix} \infty \\ \infty \\ \infty \\ \infty \end{pmatrix}.$$

The nonlinear constraints are all of the form $c_i(x) \le 1$, for $i = 1, \ldots, 14$; hence, all components of ℓ_N are $-\infty$, and all components of u_N are 1. The fourteen functions $\{c_i(x)\}$ are

$$\begin{aligned}
c_1(x) &= x_1^2 + x_6^2, & c_2(x) &= (x_2 - x_1)^2 + (x_7 - x_6)^2, \\
c_3(x) &= (x_3 - x_1)^2 + x_6^2, & c_4(x) &= (x_1 - x_4)^2 + (x_6 - x_8)^2, \\
c_5(x) &= (x_1 - x_5)^2 + (x_6 - x_9)^2, & c_6(x) &= x_2^2 + x_7^2, \\
c_7(x) &= (x_3 - x_2)^2 + x_7^2, & c_8(x) &= (x_4 - x_2)^2 + (x_8 - x_7)^2, \\
c_9(x) &= (x_2 - x_5)^2 + (x_7 - x_9)^2, & c_{10}(x) &= (x_4 - x_3)^2 + x_8^2, \\
c_{11}(x) &= (x_5 - x_3)^2 + x_9^2, & c_{12}(x) &= x_4^2 + x_8^2, \\
c_{13}(x) &= (x_4 - x_5)^2 + (x_9 - x_8)^2, & c_{14}(x) &= x_5^2 + x_9^2.
\end{aligned}$$

An optimal solution (to five figures) is

$$x^* = (\ .060947,\ .59765,\ 1.0,\ .59765,\ .060947,\ .34377,\ .5,\ -.5,\ -.34377\ )^T,$$

and $F(x^*) = -1.34996$. (The optimal objective function is unique, but is achieved for other values of x.) Five nonlinear constraints and one simple bound are active at x^*. The sample solution output is given later in this section, following the sample main program and problem definition.

Two calls are made to NPSOL in order to demonstrate some of its features. For the first call, the starting point is:

$$x_0 = (\ .1,\ .125,\ .666666,\ .142857,\ .111111,\ .2,\ .25,\ -.2,\ -.25\ )^T.$$

All objective and constraint derivatives are specified in the user-provided subroutines `OBJFN1` and `CONFN1`, i.e., the default option **`Derivative Level = 3`** is used.

On completion of the first call to NPSOL, the optimal variables are perturbed to produce the initial point for a second run in which the problem functions are defined by the subroutines `OBJFN2` and `CONFN2`. To illustrate one of the finite-difference options in NPSOL, these routines are programmed so that the first six components of the objective gradient and the constant elements of the Jacobian matrix are not specified; hence, the option **`Derivative Level = 0`** is chosen. During computation of the finite-difference intervals, the constant Jacobian elements are identified and set, and NPSOL automatically increases the derivative level to 2.

The second call to NPSOL illustrates the use of the **`Warm Start`** option to utilize the final active set, nonlinear multipliers and approximate Hessian from the first run. Note that **`Hessian = Yes`** was specified for the first run so that the array `R` would contain the Cholesky factor of the approximate Hessian of the Lagrangian.

The two calls to NPSOL illustrate the alternative methods of assigning default parameters. For the first run, the parameters are read from the options file `NPMAIN DATA` supplied on the distribution tape. In the second run, the parameters are modified using calls to subroutine `NPOPTN`. (There is no special significance in the order of these assignments; an options file may just as easily be used to modify parameters set by `NPOPTN`.)

The results are typical of those obtained from NPSOL when solving well behaved (non-trivial) nonlinear problems. The approximate Hessian and working set remain relatively well-conditioned. Similarly, the penalty parameters remain small and approximately constant. The numerical results illustrate much of the theoretically predicted behavior of a quasi-Newton SQP method. As x approaches the solution, only one minor iteration is performed per major iteration, and the "`Norm Gz`" and "`Norm C`" columns exhibit the fast linear convergence rate mentioned in Sections 6 and 7. Note that the constraint violations converge earlier than the projected gradient. The final values of the projected gradient norm and constraint norm reflect the limiting accuracy of the two quantities. It is possible to achieve almost full precision in the constraint norm but only half precision in the projected gradient norm. Note that the final accuracy in the nonlinear constraints is considerably better than the feasibility tolerance, because the constraint violations are being refined during the last few iterations while the algorithm is working to reduce the projected gradient norm. In this problem, the constraint values and Lagrange multipliers at the solution are "well balanced", i.e., all the multipliers are approximately the same order of magnitude. This behavior is typical of a well-scaled problem.

10. REFERENCES

Dennis, J. E., Jr. and Moré, J. J. (1977). Quasi-Newton methods, motivation and theory, *SIAM Review* **19**, pp. 46–89.

Dennis, J. E., Jr. and Schnabel, R. B. (1981). "A new derivation of symmetric positive definite secant updates", in *Nonlinear Programming 4* (O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds.), pp. 167–199, Academic Press, London and New York.

Dennis, J. E., Jr. and Schnabel, R. B. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Dongarra, J. J., Du Croz, J. J., Hammarling, S. J. and Hanson, R. J. (1985). A proposal for an extended set of Fortran basic linear algebra subprograms, *SIGNUM Newsletter* 20, 1, 2–18.

Fletcher, R. (1981). *Practical Methods of Optimization, Volume 2, Constrained Optimization*, John Wiley and Sons, New York and Toronto.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1984a). User's guide for SOL/QPSOL Version 3.2, Report SOL 84-5, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1984b). Procedures for optimization problems with a mixture of bounds and general linear constraints, ACM *Transactions on Mathematical Software* **10**, pp. 282–298.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1986a). User's guide for LSSOL (Version 1.0), Report SOL 86-1, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1986b). Properties of an augmented Lagrangian merit function for inequality constraints, SOL Report (to appear), Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1986c). A sequential quadratic programming method for nonlinear optimization, SOL Report (to appear), Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W. and Wright, M. H. (1981). *Practical Optimization*, Academic Press, London and New York.

Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems 187, Springer-Verlag, Berlin and New York.

Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. (1979). Basic linear algebra subprograms for Fortran usage, *ACM Transactions on Mathematical Software* **5**, pp. 308–325.

Murtagh, B. A. and Saunders, M. A. (1982). A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints, *Math. Prog. Study* **16**, pp. 84–118.

Murtagh, B. A. and Saunders, M. A. (1983). MINOS 5.0 User's Guide, Report SOL 83-20, Department of Operations Research, Stanford University, California.

Powell, M. J. D. (1974). "Introduction to constrained optimization", in *Numerical Methods for Constrained Optimization* (P. E. Gill and W. Murray, eds.), pp. 1–28, Academic Press, London and New York.

Powell, M. J. D. (1983). "Variable metric methods for constrained optimization", in *Mathematical Programming: The State of the Art*, (A. Bachem, M. Grötschel and B. Korte, eds.), pp. 288–311, Springer-Verlag, Berlin, Heidelberg, New York and Tokyo.

APPENDIX. SAMPLE PROGRAM AND OUTPUT

```
*********************************************************************
*     FILE NPMAIN FORTRAN
*
*     Sample program for NPSOL Version 4.0  February 1986.
*********************************************************************

      IMPLICIT           DOUBLE PRECISION(A-H,O-Z)

*     Set the declared array dimensions.
*     NROWA  = the declared row dimension of  A.
*     NROWJ  = the declared row dimension of  CJAC.
*     NROWR  = the declared row dimension of  R.
*     MAXN   = maximum no. of variables allowed for.
*     MAXBND = maximum no. of variables + linear & nonlinear constrnts.
*     LIWORK = the length of the integer work array.
*     LWORK  = the length of the double precision work array.

      PARAMETER         (NROWA  =  5, NROWJ  =  20, NROWR =   10,
     $                   MAXN   =  9, LIWORK =  70, LWORK = 1000,
     $                   MAXBND =  MAXN + NROWA + NROWJ)

      INTEGER            ISTATE(MAXBND)
      INTEGER            IWORK(LIWORK)
      DOUBLE PRECISION   A(NROWA,MAXN)
      DOUBLE PRECISION   BL(MAXBND), BU(MAXBND)
      DOUBLE PRECISION   C(NROWJ), CJAC(NROWJ,MAXN), CLAMDA(MAXBND)
      DOUBLE PRECISION   OBJGRD(MAXN), R(NROWR,MAXN), X(MAXN)
      DOUBLE PRECISION   WORK(LWORK)
      EXTERNAL           OBJFN1, OBJFN2, CONFN1, CONFN2

      PARAMETER         (ZERO = 0.0, ONE = 1.0)

*     Set the actual problem dimensions.
*     N      = the number of variables.
*     NCLIN  = the number of general linear constraints (may be 0).
*     NCNLN  = the number of nonlinear constraints (may be 0).

      N      = 9
      NCLIN  = 4
      NCNLN  = 14
      NBND   = N + NCLIN + NCNLN

*     ---------------------------------------------------------------
*     Assign file numbers and the data arrays.
*     NOUT   = the unit number for printing.
*     IOPTNS = the unit number for reading the options file.
*     Bounds  .ge.    BIGBND  will be treated as plus  infinity.
*     Bounds  .le.  - BIGBND  will be treated as minus infinity.
*     A      = the linear constraint matrix.
*     BL     = the lower bounds on  x,  a'x  and  c(x).
*     BU     = the upper bounds on  x,  a'x  and  c(x).
*     X      = the initial estimate of the solution.
*     ---------------------------------------------------------------
      NOUT   = 6
      IOPTNS = 5
```

```
      BIGBND = 1.0D+15

*     Set the matrix  A.

      DO 40 J = 1, N
         DO 30 I = 1, NCLIN
            A(I,J) = ZERO
   30    CONTINUE
   40 CONTINUE
      A(1,1) = -ONE
      A(1,2) =  ONE
      A(2,2) = -ONE
      A(2,3) =  ONE
      A(3,3) =  ONE
      A(3,4) = -ONE
      A(4,4) =  ONE
      A(4,5) = -ONE

*     Set the bounds.

      DO 50 J  =  1, NBND
         BL(J) = -BIGBND
         BU(J) =  BIGBND
   50 CONTINUE
      BL(1)  =  ZERO
      BL(3)  = -ONE
      BL(5)  =  ZERO
      BL(6)  =  ZERO
      BL(7)  =  ZERO

      BU(3)  =  ONE
      BU(8)  =  ZERO
      BU(9)  =  ZERO

*     Set lower bounds of zero for all four linear constraints.

      DO 60 J  =  N+1,  N+NCLIN
         BL(J) =  ZERO
   60 CONTINUE

*     Set upper bounds of one for all 14 nonlinear constraints.

      DO 70 J  =  N + NCLIN + 1,  NBND
         BU(J) =  ONE
   70 CONTINUE

*     Set the initial estimate of  X.

      X(1)   =  .1
      X(2)   =  .125
      X(3)   =  .666666
      X(4)   =  .142857
      X(5)   =  .111111
      X(6)   =  .2
      X(7)   =  .25
```

```
      X(8)   = -.2
      X(9)   = -.25


*     ---------------------------------------------------------------
*     Read the options file.
*     ---------------------------------------------------------------

      CALL NPFILE( IOPTNS, INFORM )
      IF (INFORM .NE. 0) THEN
         WRITE (NOUT, 3000) INFORM
         STOP
      END IF

*     ---------------------------------------------------------------
*     Solve the problem.
*     ---------------------------------------------------------------

      CALL NPSOL ( N, NCLIN, NCNLN, NROWA, NROWJ, NROWR,
     $             A, BL, BU,
     $             CONFN1, OBJFN1,
     $             INFORM, ITER, ISTATE,
     $             C, CJAC, CLAMDA, OBJF, OBJGRD, R, X,
     $             IWORK, LIWORK, WORK, LWORK )

      IF (INFORM .GT. 0) GO TO 900

*     ---------------------------------------------------------------
*     The following is for illustrative purposes only.
*     A second run solves the same problem,  but defines the objective
*     and constraints via the subroutines OBJFN2 and CONFN2.  Some
*     objective derivatives and the constant Jacobian elements are not
*     supplied.
*     We do a warm start using
*              ISTATE    (the working set)
*              CLAMDA    (the Lagrange multipliers)
*              R         (the Hessian approximation)
*     from the previous run, but with a slightly perturbed starting
*     point.  The previous option file must have specified
*              Hessian    Yes
*     for R to be a useful approximation.
*     ---------------------------------------------------------------

      DO 100 J = 1, N
         X(J)  = X(J) + 0.01
  100 CONTINUE

*     The previous parameters are retained and updated.

      CALL NPOPTN( '   Derivative level             0')
      CALL NPOPTN( '   Verify                      No')
      CALL NPOPTN( '   Warm Start')
      CALL NPOPTN( '   Major iterations            20')

      CALL NPOPTN( '   Major print level           10')
```

```
      CALL NPSOL ( N, NCLIN, NCNLN, NROWA, NROWJ, NROWR,
     $             A, BL, BU,
     $             CONFN2, OBJFN2,
     $             INFORM, ITER, ISTATE,
     $             C, CJAC, CLAMDA, OBJF, OBJGRD, R, X,
     $             IWORK, LIWORK, WORK, LWORK )

      IF (INFORM .GT. 0) GO TO 900
      STOP

*     -----------
*     Error exit.
*     -----------

  900 WRITE (NOUT, 3010) INFORM
      STOP

 3000 FORMAT(/ ' NPFILE terminated with  INFORM =', I3)
 3010 FORMAT(/ ' NPSOL  terminated with  INFORM =', I3)

*     End of the example program for NPSOL.

      END
*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      SUBROUTINE OBJFN1( MODE, N, X, OBJF, OBJGRD, NSTATE )
      IMPLICIT           DOUBLE PRECISION(A-H,O-Z)
      DOUBLE PRECISION   X(N), OBJGRD(N)

*-----------------------------------------------------------------------
*     OBJFN1  computes the value and first derivatives of the nonlinear
*     objective function.
*-----------------------------------------------------------------------
      OBJF   = - X(2)*X(6) + X(1)*X(7) - X(3)*X(7) - X(5)*X(8)
     $         + X(4)*X(9) + X(3)*X(8)

      OBJGRD(1) =   X(7)
      OBJGRD(2) = - X(6)
      OBJGRD(3) = - X(7) + X(8)
      OBJGRD(4) =   X(9)
      OBJGRD(5) = - X(8)
      OBJGRD(6) = - X(2)
      OBJGRD(7) = - X(3) + X(1)
      OBJGRD(8) = - X(5) + X(3)
      OBJGRD(9) =   X(4)

      RETURN

*     End of  OBJFN1.

      END
*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      SUBROUTINE CONFN1( MODE, NCNLN, N, NROWJ,
```

```
     $                      NEEDC, X, C, CJAC, NSTATE )

      IMPLICIT              DOUBLE PRECISION(A-H,O-Z)
      INTEGER               NEEDC(*)
      DOUBLE PRECISION      X(N), C(*), CJAC(NROWJ,*)

*-----------------------------------------------------------------------
*     CONFN1  computes the values and first derivatives of the nonlinear
*     constraints.
*
*     The zero elements of Jacobian matrix are set only once.  This
*     occurs during the first call to CONFN1  (NSTATE = 1).
*-----------------------------------------------------------------------
      PARAMETER            (ZERO = 0.0, TWO = 2.0)

      IF (NSTATE .EQ. 1) THEN

*        First call to CONFN1.  Set all Jacobian elements to zero.
*        N.B.  This will only work with `Derivative Level = 3'.

         DO 120 J = 1, N
            DO 110 I = 1, NCNLN
               CJAC(I,J) = ZERO
  110       CONTINUE
  120    CONTINUE

      END IF

      IF (NEEDC(1) .GT. 0) THEN
         C(1)       =   X(1)**2  +  X(6)**2
         CJAC(1,1)  =   TWO*X(1)
         CJAC(1,6)  =   TWO*X(6)
      END IF

      IF (NEEDC(2) .GT. 0) THEN
         C(2)       =   (X(2) - X(1))**2  +  (X(7) - X(6))**2
         CJAC(2,1)  = - TWO*(X(2) - X(1))
         CJAC(2,2)  =   TWO*(X(2) - X(1))
         CJAC(2,6)  = - TWO*(X(7) - X(6))
         CJAC(2,7)  =   TWO*(X(7) - X(6))
      END IF

      IF (NEEDC(3) .GT. 0) THEN
         C(3)       =   (X(3) - X(1))**2  +  X(6)**2
         CJAC(3,1)  = - TWO*(X(3) - X(1))
         CJAC(3,3)  =   TWO*(X(3) - X(1))
         CJAC(3,6)  =   TWO*X(6)
      END IF

      IF (NEEDC(4) .GT. 0) THEN
         C(4)       =   (X(1) - X(4))**2  +  (X(6) - X(8))**2
         CJAC(4,1)  =   TWO*(X(1) - X(4))
         CJAC(4,4)  = - TWO*(X(1) - X(4))
         CJAC(4,6)  =   TWO*(X(6) - X(8))
         CJAC(4,8)  = - TWO*(X(6) - X(8))
```

```
      END IF

      IF (NEEDC(5) .GT. 0) THEN
         C(5)       =   (X(1) - X(5))**2  +  (X(6) - X(9))**2
         CJAC(5,1)  =   TWO*(X(1) - X(5))
         CJAC(5,5)  = - TWO*(X(1) - X(5))
         CJAC(5,6)  =   TWO*(X(6) - X(9))
         CJAC(5,9)  = - TWO*(X(6) - X(9))
      END IF

      IF (NEEDC(6) .GT. 0) THEN
         C(6)       =   X(2)**2  +  X(7)**2
         CJAC(6,2)  =   TWO*X(2)
         CJAC(6,7)  =   TWO*X(7)
      END IF

      IF (NEEDC(7) .GT. 0) THEN
         C(7)       =   (X(3) - X(2))**2  +  X(7)**2
         CJAC(7,2)  = - TWO*(X(3) - X(2))
         CJAC(7,3)  =   TWO*(X(3) - X(2))
         CJAC(7,7)  =   TWO*X(7)
      END IF

      IF (NEEDC(8) .GT. 0) THEN
         C(8)       =   (X(4) - X(2))**2  +  (X(8) - X(7))**2
         CJAC(8,2)  = - TWO*(X(4) - X(2))
         CJAC(8,4)  =   TWO*(X(4) - X(2))
         CJAC(8,7)  = - TWO*(X(8) - X(7))
         CJAC(8,8)  =   TWO*(X(8) - X(7))
      END IF

      IF (NEEDC(9) .GT. 0) THEN
         C(9)       =   (X(2) - X(5))**2  +  (X(7) - X(9))**2
         CJAC(9,2)  =   TWO*(X(2) - X(5))
         CJAC(9,5)  = - TWO*(X(2) - X(5))
         CJAC(9,7)  =   TWO*(X(7) - X(9))
         CJAC(9,9)  = - TWO*(X(7) - X(9))
      END IF

      IF (NEEDC(10) .GT. 0) THEN
         C(10)      =   (X(4) - X(3))**2  +  X(8)**2
         CJAC(10,3) = - TWO*(X(4) - X(3))
         CJAC(10,4) =   TWO*(X(4) - X(3))
         CJAC(10,8) =   TWO*X(8)
      END IF

      IF (NEEDC(11) .GT. 0) THEN
         C(11)      =   (X(5) - X(3))**2  +  X(9)**2
         CJAC(11,3) = - TWO*(X(5) - X(3))
         CJAC(11,5) =   TWO*(X(5) - X(3))
         CJAC(11,9) =   TWO*X(9)
      END IF

      IF (NEEDC(12) .GT. 0) THEN
         C(12)      =   X(4)**2  +  X(8)**2
```

```
            CJAC(12,4) =   TWO*X(4)
            CJAC(12,8) =   TWO*X(8)
         END IF

         IF (NEEDC(13) .GT. 0) THEN
            C(13)      =   (X(4) - X(5))**2  +  (X(9) - X(8))**2
            CJAC(13,4) =   TWO*(X(4) - X(5))
            CJAC(13,5) = - TWO*(X(4) - X(5))
            CJAC(13,8) = - TWO*(X(9) - X(8))
            CJAC(13,9) =   TWO*(X(9) - X(8))
         END IF

         IF (NEEDC(14) .GT. 0) THEN
            C(14)      =   X(5)**2  +  X(9)**2
            CJAC(14,5) =   TWO*X(5)
            CJAC(14,9) =   TWO*X(9)
         END IF

         RETURN

*     End of  CONFN1.

         END
*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

         SUBROUTINE OBJFN2( MODE, N, X, OBJF, OBJGRD, NSTATE )
         IMPLICIT            DOUBLE PRECISION(A-H,O-Z)
         DOUBLE PRECISION    X(N), OBJGRD(N)

*----------------------------------------------------------------------
*     OBJFN2  computes the value and some first derivatives of the
*     nonlinear objective function.
*----------------------------------------------------------------------

         OBJF   = - X(2)*X(6) + X(1)*X(7) - X(3)*X(7) - X(5)*X(8)
        $          + X(4)*X(9) + X(3)*X(8)

         OBJGRD(3) = - X(7) + X(8)
         OBJGRD(7) = - X(3) + X(1)
         OBJGRD(8) = - X(5) + X(3)

         RETURN

*     End of  OBJFN2.

         END
*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

         SUBROUTINE CONFN2( MODE, NCNLN, N, NROWJ,
        $                   NEEDC, X, C, CJAC, NSTATE )

         IMPLICIT            DOUBLE PRECISION(A-H,O-Z)
         INTEGER             NEEDC(*)
         DOUBLE PRECISION    X(N), C(*), CJAC(NROWJ,*)

```

```
*-----------------------------------------------------------------------
*     CONFN2  computes the values and the non-constant derivatives of
*     the nonlinear constraints.
*-----------------------------------------------------------------------
      PARAMETER         (TWO = 2.0)

      IF (NEEDC(1) .GT. 0) THEN
         C(1)       =   X(1)**2  +  X(6)**2
         CJAC(1,1)  =   TWO*X(1)
         CJAC(1,6)  =   TWO*X(6)
      END IF

      IF (NEEDC(2) .GT. 0) THEN
         C(2)       =   (X(2) - X(1))**2  +  (X(7) - X(6))**2
         CJAC(2,1)  = - TWO*(X(2) - X(1))
         CJAC(2,2)  =   TWO*(X(2) - X(1))
         CJAC(2,6)  = - TWO*(X(7) - X(6))
         CJAC(2,7)  =   TWO*(X(7) - X(6))
      END IF

      IF (NEEDC(3) .GT. 0) THEN
         C(3)       =   (X(3) - X(1))**2  +  X(6)**2
         CJAC(3,1)  = - TWO*(X(3) - X(1))
         CJAC(3,3)  =   TWO*(X(3) - X(1))
         CJAC(3,6)  =   TWO*X(6)
      END IF

      IF (NEEDC(4) .GT. 0) THEN
         C(4)       =   (X(1) - X(4))**2  +  (X(6) - X(8))**2
         CJAC(4,1)  =   TWO*(X(1) - X(4))
         CJAC(4,4)  = - TWO*(X(1) - X(4))
         CJAC(4,6)  =   TWO*(X(6) - X(8))
         CJAC(4,8)  = - TWO*(X(6) - X(8))
      END IF

      IF (NEEDC(5) .GT. 0) THEN
         C(5)       =   (X(1) - X(5))**2  +  (X(6) - X(9))**2
         CJAC(5,1)  =   TWO*(X(1) - X(5))
         CJAC(5,5)  = - TWO*(X(1) - X(5))
         CJAC(5,6)  =   TWO*(X(6) - X(9))
         CJAC(5,9)  = - TWO*(X(6) - X(9))
      END IF

      IF (NEEDC(6) .GT. 0) THEN
         C(6)       =   X(2)**2  +  X(7)**2
         CJAC(6,2)  =   TWO*X(2)
         CJAC(6,7)  =   TWO*X(7)
      END IF

      IF (NEEDC(7) .GT. 0) THEN
         C(7)       =   (X(3) - X(2))**2  +  X(7)**2
         CJAC(7,2)  = - TWO*(X(3) - X(2))
         CJAC(7,3)  =   TWO*(X(3) - X(2))
         CJAC(7,7)  =   TWO*X(7)
      END IF
```

```

      IF (NEEDC(8) .GT. 0) THEN
         C(8)         =   (X(4) - X(2))**2  +  (X(8) - X(7))**2
         CJAC(8,2)    = - TWO*(X(4) - X(2))
         CJAC(8,4)    =   TWO*(X(4) - X(2))
         CJAC(8,7)    = - TWO*(X(8) - X(7))
         CJAC(8,8)    =   TWO*(X(8) - X(7))
      END IF

      IF (NEEDC(9) .GT. 0) THEN
         C(9)         =   (X(2) - X(5))**2  +  (X(7) - X(9))**2
         CJAC(9,2)    =   TWO*(X(2) - X(5))
         CJAC(9,5)    = - TWO*(X(2) - X(5))
         CJAC(9,7)    =   TWO*(X(7) - X(9))
         CJAC(9,9)    = - TWO*(X(7) - X(9))
      END IF

      IF (NEEDC(10) .GT. 0) THEN
         C(10)        =   (X(4) - X(3))**2  +  X(8)**2
         CJAC(10,3) = - TWO*(X(4) - X(3))
         CJAC(10,4) =   TWO*(X(4) - X(3))
         CJAC(10,8) =   TWO*X(8)
      END IF

      IF (NEEDC(11) .GT. 0) THEN
         C(11)        =   (X(5) - X(3))**2  +  X(9)**2
         CJAC(11,3) = - TWO*(X(5) - X(3))
         CJAC(11,5) =   TWO*(X(5) - X(3))
         CJAC(11,9) =   TWO*X(9)
      END IF

      IF (NEEDC(12) .GT. 0) THEN
         C(12)        =   X(4)**2  +  X(8)**2
         CJAC(12,4) =   TWO*X(4)
         CJAC(12,8) =   TWO*X(8)
      END IF

      IF (NEEDC(13) .GT. 0) THEN
         C(13)        =   (X(4) - X(5))**2  +  (X(9) - X(8))**2
         CJAC(13,4) =   TWO*(X(4) - X(5))
         CJAC(13,5) = - TWO*(X(4) - X(5))
         CJAC(13,8) = - TWO*(X(9) - X(8))
         CJAC(13,9) =   TWO*(X(9) - X(8))
      END IF

      IF (NEEDC(14) .GT. 0) THEN
         C(14)        =   X(5)**2  +  X(9)**2
         CJAC(14,5) =   TWO*X(5)
         CJAC(14,9) =   TWO*X(9)
      END IF

      RETURN

*     End of  CONFN2.

      END
```

```
OPTIONS file
------------

     BEGIN  Options for NPSOL 4.0 Sample problem.

        Verify Level                          3
        Major iterations limit               50
        Major print level                     5

        Start constraint check at column      1
        Stop  constraint check at column      2
        Start objective  check at column      7
        Stop  objective  check at column      9

        Hessian                             Yes   *  Ready for the next run.

     End


                        SOL/NPSOL  ---  Version 4.0  Feb 1986
                        =====================================

Parameters
----------

Linear constraints.....        4      Linear feasibility.....  1.49E-08      COLD start.............
Variables..............        9      Infinite bound size....  1.00E+10      Crash tolerance........   1.00E-02
                                      Infinite step size.....  1.00E+10

Nonlinear constraints..       14      Optimality tolerance...  5.36E-12      Function precision.....   8.16E-15
Nonlinear Jacobian vars        9      Nonlinear feasibility..  1.49E-08
Nonlinear objectiv vars        9      Linesearch tolerance...  9.00E-01
EPS (machine precision)  2.22E-16     Derivative level.......         3      Verify level...........          3

Major iterations limit.       50      Major print level......         5
Minor iterations limit.       81      Minor print level......         0

Workspace provided is     IW(    70),  W(  1000).
To solve problem we need  IW(    59),  W(   968).


Verification of the constraint gradients.
-----------------------------------------

The Jacobian seems to be ok.

The largest relative error was     9.98E-09   in constraint     2


Column     X(J)     DX(J)    Row     Jacobian Value      Difference Approxn  Itns
```

```
     1  1.00E-01  1.31E-07   1    2.00000048E-01    2.00000048E-01    OK      1
                  1.28E-07   2   -4.99999523E-02   -4.99999523E-02    OK      1
                  1.49E-07   3   -1.13333189E+00   -1.13333189E+00    OK      1
                  1.38E-07   4   -8.57139826E-02   -8.57139826E-02    OK      1
                  1.40E-07   5   -2.22219229E-02   -2.22219229E-02    OK      1

Column     X(J)      DX(J)    Row     Jacobian Value       Difference Approxn  Itns

     2  1.25E-01  1.28E-07   2    4.99999523E-02    4.99999523E-02    OK      1
                  1.33E-07   6    2.50000000E-01    2.50000000E-01    OK      1
                  1.49E-07   7   -1.08333194E+00   -1.08333194E+00    OK      1
                  1.40E-07   8   -3.57140303E-02   -3.57140303E-02    OK      1
                  1.43E-07   9    2.77780294E-02    2.77780294E-02    OK      1

    10  Jacobian elements out of the    10  set in cols     1  through     2  seem to be ok.

The largest relative error was    2.13E-11  in row    9,  column    2


Verification of the objective gradients.
----------------------------------------

The objective gradients seem to be ok.

Directional derivative of the objective    1.20539630E-01
Difference approximation                   1.20539630E-01

    J     X(J)     DX(J)           G(J)           Difference approxn  Itns

    7  2.50E-01  2.26E-06   -5.66665947E-01   -5.66665947E-01    OK      3
    8 -2.00E-01  2.17E-06    5.55554986E-01    5.55554986E-01    OK      3
    9 -2.50E-01  2.26E-06    1.42857015E-01    1.42857015E-01    OK      3

     3  Objective gradients out of the     3  set in cols     7  through     9  seem to be ok.

The largest relative error was    2.21E-11   in element     7


 Itn ItQP     Step  Nfun           Merit Bnd Lin Nln  Nz  Norm Gf  Norm Gz  Cond H Cond Hz  Cond T   Norm C  Penalty Conv
   0    5  0.0E+00     1 -3.134917E-01   3   0   1   5  8.8E-01  3.7E-01  1.E+00  1.E+00  1.E+00  8.8E-01  0.0E+00 F FF
   1    9  1.0E+00     2 -1.075027E+00   1   0   3   5  2.2E+00  1.5E+00  1.E+02  7.E+00  2.E+00  8.6E-01  1.3E+00 F FF
   2    4  1.0E+00     3 -1.268553E+00   1   0   4   4  1.7E+00  3.3E-01  9.E+00  1.E+00  2.E+00  1.3E-01  1.3E+00 F FF
   3    2  1.0E+00     4 -1.331667E+00   1   0   5   3  1.9E+00  2.5E-01  4.E+01  2.E+00  2.E+00  1.1E-01  1.3E+00 F FF
   4    1  1.0E+00     5 -1.349354E+00   1   0   5   3  1.8E+00  4.5E-02  3.E+01  1.E+00  2.E+00  1.4E-02  1.3E+00 F FF
   5    1  1.0E+00     6 -1.349874E+00   1   0   5   3  1.8E+00  6.7E-03  3.E+01  2.E+00  2.E+00  9.1E-04  1.3E+00 F FF
   6    1  1.0E+00     7 -1.349913E+00   1   0   5   3  1.8E+00  5.3E-03  3.E+01  2.E+00  2.E+00  5.7E-05  1.3E+00 F FF
   7    1  1.0E+00     8 -1.349963E+00   1   0   5   3  1.8E+00  1.2E-03  1.E+02  2.E+00  2.E+00  3.1E-04  6.8E+00 F FF
   8    1  1.0E+00     9 -1.349963E+00   1   0   5   3  1.8E+00  1.6E-04  1.E+02  3.E+00  2.E+00  9.0E-07  6.8E+00 F FF
   9    1  1.0E+00    10 -1.349963E+00   1   0   5   3  1.8E+00  5.4E-06  3.E+01  2.E+00  2.E+00  1.2E-08  6.8E+00 F TT
  10    1  1.0E+00    11 -1.349963E+00   1   0   5   3  1.8E+00  2.0E-07  4.E+01  2.E+00  2.E+00  6.4E-11  6.8E+00 F TT
  11    1  1.0E+00    12 -1.349963E+00   1   0   5   3  1.8E+00  1.1E-08  1.E+02  2.E+00  2.E+00  4.7E-14  6.8E+00 T TT

 Exit  NP phase.  INFORM =  0 MAJITS =      11   NFUN =     12   NGRAD =     12

 Exit NPSOL - Optimal solution found.
```

```
Final nonlinear objective value =   -1.349963


Calls to NPOPTN
---------------

        Derivative level              0
        Verify                       No
        Warm Start
        Major iterations             20
        Major print level            10



                     SOL/NPSOL  ---  Version 4.0  Feb 1986
                     =====================================


Parameters
----------

Linear constraints.....         4      Linear feasibility.....  1.49E-08        WARM start.............
Variables..............         9      Infinite bound size....  1.00E+10        Crash tolerance........  1.00E-02
                                       Infinite step size.....  1.00E+10

Nonlinear constraints..        14      Optimality tolerance...  5.36E-12        Function precision.....  8.16E-15
Nonlinear Jacobian vars         9      Nonlinear feasibility..  1.49E-08
Nonlinear objectiv vars         9      Linesearch tolerance...  9.00E-01
EPS (machine precision)  2.22E-16      Derivative level.......         0        Verify level...........        -1

Major iterations limit.        20      Major print level......        10
Minor iterations limit.        81      Minor print level......         0

Workspace provided is     IW(    70),  W(   1000).
To solve problem we need  IW(    59),  W(    968).


The user sets    44   out of   126   Jacobian elements.
Each iteration,     82   Jacobian elements will be estimated numerically.


The user sets     3   out of     9   objective gradient elements.
Each iteration,      6   gradient elements will be estimated numerically.


Computation of the finite-difference intervals
----------------------------------------------


    J        X(J)    Forward DX(J)    Central DX(J)       Error est.

    1   7.09E-02    1.935067E-06     1.935067E-05     1.979764E-08
    2   6.08E-01    2.904821E-06     2.904821E-05     1.318833E-08
    3   1.00E+00    3.613750E-07     3.613750E-06     0.000000E+00
    4   6.08E-01    2.904821E-06     2.904821E-05     1.318833E-08
    5   7.09E-02    1.935067E-06     1.935067E-05     1.979764E-08
    6   3.54E-01    2.446096E-06     2.446096E-05     1.566159E-08
    7   5.10E-01    2.728381E-07     2.728381E-06     0.000000E+00
```

```
    8 -4.90E-01     2.692244E-07     2.692244E-06     0.000000E+00
    9 -3.34E-01     2.409958E-06     2.409958E-05     1.589644E-08

   82  constant constraint gradient elements assigned.

    0  constant  objective gradient elements assigned.

All missing Jacobian elements are constants.   Derivative level increased to    2

 Itn ItQP     Step  Nfun          Merit Bnd Lin Nln  Nz  Norm Gf  Norm Gz  Cond H Cond Hz  Cond T   Norm C  Penalty Conv
   0    1  0.0E+00     1 -1.349188E+00   1   0   5   3  1.8E+00  1.5E-02  3.E+01  4.E+00  1.E+00  2.8E-02  2.2E+00 F FF
   1    1  1.0E+00     3 -1.349963E+00   1   0   5   3  1.8E+00  1.3E-03  1.E+02  7.E+00  1.E+00  3.0E-04  3.0E+02 F FFM
   2    1  1.0E+00     4 -1.349963E+00   1   0   5   3  1.8E+00  3.5E-04  6.E+01  6.E+00  2.E+00  7.8E-07  2.1E+01 F FF
   3    1  1.0E+00     5 -1.349963E+00   1   0   5   3  1.8E+00  2.0E-04  8.E+01  3.E+00  2.E+00  2.3E-08  7.7E+00 F FF
   4    1  1.0E+00     6 -1.349963E+00   1   0   5   3  1.8E+00  7.4E-06  9.E+01  3.E+00  2.E+00  3.9E-08  7.7E+00 F FF
   5    1  1.0E+00     7 -1.349963E+00   1   0   5   3  1.8E+00  5.9E-07  2.E+02  3.E+00  2.E+00  4.0E-11  7.7E+00 F TT
   6    1  1.0E+00     8 -1.349963E+00   1   0   5   3  1.8E+00  2.6E-09  6.E+01  2.E+00  1.E+00  2.0E-13  7.7E+00 T TT

Exit  NP phase.  INFORM =  0 MAJITS =     6   NFUN =     8   NGRAD =     7

Variable         State      Value       Lower bound      Upper bound  Lagr multiplier     Residual

VARBL  1           FR    0.6094665E-01  0.0000000E+00      None          0.0000000E+00      0.6095E-01
VARBL  2           FR    0.5976493          None           None          0.0000000E+00      0.1000E+16
VARBL  3           UL      1.000000     -1.000000        1.000000       -0.6875429          0.0000E+00
VARBL  4           FR    0.5976493          None           None          0.0000000E+00      0.1000E+16
VARBL  5           FR    0.6094665E-01  0.0000000E+00      None          0.0000000E+00      0.6095E-01
VARBL  6           FR    0.3437715      0.0000000E+00      None          0.0000000E+00      0.3438
VARBL  7           FR    0.5000000      0.0000000E+00      None          0.0000000E+00      0.5000
VARBL  8           FR   -0.5000000          None        0.0000000E+00    0.0000000E+00      0.5000
VARBL  9           FR   -0.3437715          None        0.0000000E+00    0.0000000E+00      0.3438

Linear constr    State      Value       Lower bound      Upper bound  Lagr multiplier     Residual

LNCON  1           FR    0.5367026      0.0000000E+00      None          0.0000000E+00      0.5367
LNCON  2           FR    0.4023507      0.0000000E+00      None          0.0000000E+00      0.4024
LNCON  3           FR    0.4023507      0.0000000E+00      None          0.0000000E+00      0.4024
LNCON  4           FR    0.5367027      0.0000000E+00      None          0.0000000E+00      0.5367

Nonlnr constr    State      Value       Lower bound      Upper bound  Lagr multiplier     Residual

NLCON  1           FR    0.1218933          None         1.000000        0.0000000E+00      0.8781
NLCON  2           FR    0.3124571          None         1.000000        0.0000000E+00      0.6875
NLCON  3           UL      1.000000         None         1.000000       -0.8318406E-01     -0.1652E-12
NLCON  4           UL      1.000000         None         1.000000       -0.3202625         -0.1104E-12
NLCON  5           FR    0.4727152          None         1.000000        0.0000000E+00      0.5273
NLCON  6           FR    0.6071847          None         1.000000        0.0000000E+00      0.3928
NLCON  7           FR    0.4118861          None         1.000000        0.0000000E+00      0.5881
NLCON  8           UL      1.000000         None         1.000000       -0.1992983          0.0000E+00
NLCON  9           UL      1.000000         None         1.000000       -0.3202625         -0.8882E-14
NLCON 10           FR    0.4118861          None         1.000000        0.0000000E+00      0.5881
NLCON 11           UL      1.000000         None         1.000000       -0.8318406E-01     -0.2665E-13
NLCON 12           FR    0.6071847          None         1.000000        0.0000000E+00      0.3928
NLCON 13           FR    0.3124571          None         1.000000        0.0000000E+00      0.6875
NLCON 14           FR    0.1218933          None         1.000000        0.[illegible]0000E+00      0.8781

 Exit NPSOL - Optimal solution found.

 Final nonlinear objective value =   -1.349963
```

INDEX

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ARO 21592.8-MA	2. GOVT ACCESSION NO.: AD-A169 115
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): User's Guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Philip E. Gill, Walter Murray, Michael A. Saunders and Margaret H. Wright	8. CONTRACT OR GRANT NUMBER(s): N00014-85-K-0343; DAAG29-84-K-0156
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Operations Research - SOL, Stanford University, Stanford, CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR-047-064
11. Office of Naval Research - Dept. of the Navy, 800 N. Quincy Street, Arlington, VA 22217	12. REPORT DATE: January 1986
	13. NUMBER OF PAGES: 54 pp.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): U.S. Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

optimization	mathematical software
nonlinear constraints	nonlinear programming
sequential quadratic programming method	finite-differences

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See next page.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT: USER'S GUIDE FOR NPSOL (VERSION 4.0): A FORTRAN PACKAGE FOR NONLINEAR PROGRAMMING by Philip E. Gill, Walter Murray, Michael A. Saunders and Margaret H. Wright.

This report forms the user's guide for Version 4.0 of NPSOL, a set of Fortran subroutines designed to minimize a smooth function subject to constraints, which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints. (NPSOL may also be used for unconstrained, bound-constrained and linearly constrained optimization.) The user must provide subroutines that define the objective and constraint functions and (optionally) their gradients. All matrices are treated as dense, and hence NPSOL is not intended for large sparse problems.

NPSOL uses a sequential quadratic programming (SQP) algorithm, in which the search direction is the solution of a quadratic programming (QP) subproblem. The algorithm treats bounds, linear constraints and nonlinear constraints separately. The Hessian of each QP subproblem is a positive-definite quasi-Newton approximation to the Hessian of the Lagrangian function. The steplength at each iteration is required to produce a sufficient decrease in an augmented Lagrangian merit function. Each QP subproblem is solved using a quadratic programming package with several features that improve the efficiency of an SQP algorithm.

END

DTIC

7-86