USER'S GUIDE FOR NPSOL 5.0: A FORTRAN PACKAGE FOR NONLINEAR PROGRAMMING

Philip E. GILL
Department of Mathematics
University of California, San Diego
La Jolla, California 92093-0112

Walter MURRAY and Michael A. SAUNDERS
Systems Optimization Laboratory
Department of EESOR
Stanford University
Stanford, California 94305-4023

Margaret H. Wright
Bell Laboratories
Lucent Technologies
Murray Hill
New Jersey 07974-0636

Technical Report SOL 86-1.* Revised July 30, 1998

Abstract

NPSOL is a set of Fortran subroutines for minimizing 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 provides subroutines to define the objective and constraint functions and (optionally) their first derivatives. NPSOL is not intended for large sparse problems, but there is no fixed limit on problem size.

NPSOL uses a sequential quadratic programming (SQP) algorithm, in which each search direction is the solution of a QP subproblem. Bounds, linear constraints and nonlinear constraints are treated separately. NPSOL requires relatively few evaluations of the problem functions. Hence it is especially effective if the objective or constraint functions are expensive to evaluate.

The source code for NPSOL is suitable for all scientific machines with a Fortran 77 compiler. This includes mainframes, workstations and PCs, preferably with 1MB or more of main storage. The source code, test problems and utilities are distributed on diskettes.

Keywords: Nonlinear programming, constrained optimization, nonlinear constraints, SQP methods, quasi-Newton updates, Fortran software.

pgill@ucsd.edu	walter@sol-walter.stanford.edu	mike@sol-michael.stanford.edu
http://sdna3.ucsd.edu/~peg	http://www.stanford.edu/~walter/	http://www.stanford.edu/~saunders/

*The material contained in this report is based upon research partially supported by the National Science Foundation Grants DDM-9204208 and DDM-9204547; US Department of Energy Contract DE-FG03-92ER25117; and the Office of Naval Research Grant N00014-90-J-1242.

Contents

1. Purpose

NPSOL is a collection of Fortran 77 subroutines for solving the *nonlinear programming problem*: minimize an objective function subject to a set of constraints. The problem is assumed to be stated in the form

$$\text{NP} \qquad \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x)$$

$$\text{subject to} \quad \ell \le r(x) \le u, \qquad r(x) \equiv \begin{pmatrix} x \\ Ax \\ c(x) \end{pmatrix},$$

where x is a set of variables, $f(x)$ is a nonlinear function, A is an $m_L \times n$ matrix, and $c(x)$ is an m_N-vector of nonlinear functions. (The matrix A and the vector $c(x)$ may be empty.) The functions $f(x)$ and $c(x)$ are assumed to be *smooth*, i.e., at least twice-continuously differentiable. (NPSOL will usually succeed if there are only isolated discontinuities away from the solution.)

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

Figure 1 illustrates the feasible region for the jth pair of constraints $\ell_j \le r_j(x) \le u_j$. The quantity δ is the feasibility tolerance, which can be set by the user (see §8.1). The constraints $\ell_j \le r_j \le u_j$ are considered "satisfied" if r_j lies in Regions 2, 3 or 4, and "inactive" if r_j lies in Region 3. The constraint $r_j \ge \ell_j$ is considered "active" in Region 2, and "violated" in Region 1. Similarly, $r_j \le u_j$ is active in Region 4, and violated in Region 5. For equality constraints ($\ell_j = u_j$), Regions 2 and 4 are the same and Region 3 is empty.

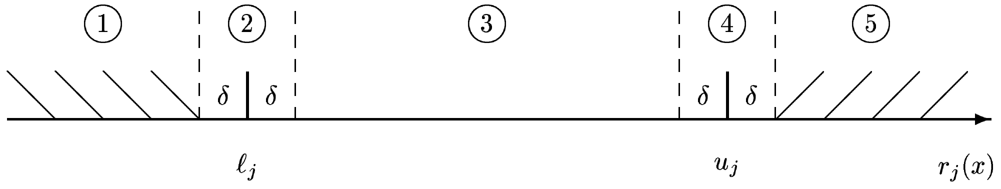

Figure 1: Illustration of the constraints $\ell_j \le r_j(x) \le u_j$. The bounds ℓ_j and u_j are considered "satisfied" if $r_j(x)$ lies in Regions 2, 3 or 4, where δ is the feasibility tolerance. The constraints $r_j(x) \ge \ell_j$ and $r_j(x) \le u_j$ are both considered "inactive" if $r_j(x)$ lies in Region 3.

The data A, ℓ, u, and an initial value of x are supplied as parameters to NPSOL. The functions $f(x)$, $c(x)$ are defined by subroutines, which should also define as many first partial derivatives as possible. Unspecified derivatives are approximated by finite differences.

NPSOL provides certain advanced features for the sophisticated user.

- Certain *optional parameters* can be set by the user (see §8.1). For example, the user can alter the convergence tolerances or control the amount of printed output. The default values of these parameters need not be set by the user.

- Not all derivatives need to be supplied. NPSOL will use finite-difference estimates of any derivatives that are not supplied by the user; see the optional parameter "`Derivative level`" in §8.1.

- Constant derivatives can be specified once and for all at the start of the minimization.
- Facilities are provided to check whether the user-provided gradients appear to be correct (see the optional parameter "`Verify`" in §8.1). 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.

If there are no nonlinear constraints in NP and f is linear or quadratic, the LSSOL package (Gill *et al.* , [GHM+86]) will generally be more efficient than NPSOL. If the problem is large and sparse, other packages such as MINOS (Murtagh and Saunders, [MS93]) may be more efficient, since NPSOL treats all matrices as dense. However, NPSOL requires relatively few evaluations of the problem functions. Hence it is especially effective if f and c (or their gradients) are expensive to evaluate.

NPSOL makes use of LSSOL, which may be called directly. The source code for NPSOL (including LSSOL) is approximately 23,000 lines of ANSI (1977) Standard Fortran, of which about 50% are comments. If there are n variables and m general constraints (linear or nonlinear), the total storage required is approximately $24n(n+m)$ Kbytes.

2. Brief 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. For more details, the interested reader is refered to §6. For an overview of SQP methods, see, for example, Fletcher [Fle81], Gill, Murray and Wright [GMW81] and Powell [Pow83].

Let $g(x)$ denote the gradient vector of first derivatives of the objective function: $g_j(x) = \partial f(x)/\partial x_j$. Similarly, let $J(x)$ denote the Jacobian matrix of first derivatives of $r(x)$, i.e., $J_{ij}(x) = \partial r_i(x)/\partial x_j$. A feasible point x satisfies the first-order conditions for optimality for NP if the following conditions hold:

1. there exists a vector of *Lagrange multipliers* λ such that the gradient of the Lagrangian $f(x) - \lambda^T r(x)$ is zero, i.e.,
$$g(x) = J(x)^T \lambda. \tag{2.1}$$
2. The Lagrange multiplier λ_j associated with the jth constraint satisfies $\lambda_j = 0$ if $\ell_j < r_j(x) < u_j$; $\lambda_j \geq 0$ if $\ell_j = r_j(x)$; $\lambda_j \leq 0$ if $r_j(x) = u_j$; and λ_j can have any value if $\ell_j = u_j$.

The method of NPSOL is a sequential quadratic programming (SQP) method. The basic structure of an SQP method involves *major* and *minor* iterations. The major iterations generate a sequence of iterates that is intended to converge to a point satisfying the first-order conditions for optimality. (For simplicity, we shall always consider a typical iteration and avoid reference to the index of the iteration.) Each new iterate $\bar{x}$ is defined by

$$\bar{x} = x + \alpha p, \tag{2.2}$$

where the *step length* α is a non-negative scalar, and p is called the *search direction*. The search direction is the solution of the quadratic program

$$\begin{array}{ll} \underset{p \in \mathbb{R}^n}{\text{minimize}} & f(x) + g(x)^T p + \tfrac{1}{2} p^T H p \\ \text{subject to} & \ell \leq r(x) + J(x) p \leq u, \end{array}$$

where H a positive-definite quasi-Newton approximation to the Hessian of the Lagrangian. In NPSOL, the QP subproblem is solved using subroutines from the LSSOL package (Gill *et al.* [GHM⁺86]). Since solving a quadratic program is itself an iterative procedure, the *minor* iterations of NPSOL are the iterations of LSSOL.

Once p has been computed, the major iteration proceeds by determining a step length α that produces a "sufficient decrease" in an augmented Lagrangian *merit function* that measures the quality of each iterate. Finally, the approximate Hessian H is updated 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. 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 the optional parameter "`Difference Interval`" in §8.1). 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.*

The working set

The *working set* is an important quantity for both the major and the minor iterations. The working set comprises a set of m_w constraints whose gradients are used to define the current search direction and Lagrange multipliers. As a solution is approached, the constraints in the working set become satisfied with equality. A working-set constraint can correspond to a simple-bound constraint, a linear constraint, or a linearized nonlinear constraint.

An important feature of constraints in the working set is that their gradients are linearly independent. The gradients form the rows of the *working-set matrix* W, an $m_w \times n$ full-rank submatrix of the Jacobian. Let the columns of the $n \times (n - n_w)$ matrix Z define a basis for the null space of W, so that $WZ = 0$. For any function with gradient g and Hessian H, the quantities $Z^T g$ and $Z^T H Z$ are known as the *reduced gradient* and *reduced Hessian* respectively. Under a suitable constraint regularity assumptions that usually hold in practice, the reduced gradient of the Lagrangian must be zero at a point satisfying the first-order optimality conditions. The values `Nz` and `Norm Gz` printed by NPSOL give n_z $(n_z \equiv n - m_w)$ and the norm of $Z^T g$.

During minor iterations, the working set is updated iteratively and is used to define a descent direction for the QP objective function. The final working set for the QP subproblem suggests a working set for the next constraint linearization. It may need to be altered to ensure an appropriate full rank W. The resulting independent working set is then used to start the next subproblem. In practice, this usually allows the subproblems to become optimal in only one iteration as a solution is approached. The numbers of bounds, general linear and nonlinear constraints in the final QP working set are the quantities `Bnd`, `Lin` and `Nln` in the printed output of NPSOL.)

3. Subroutine npsol

Problem NP is solved by a call to subroutine `npsol`, whose parameters are defined here. Several optional parameters in NPSOL define choices in the problem specification or the form of the algorithm. In order to reduce the number of formal parameters of NPSOL, these optional parameters have associated *default values* that are appropriate for most problems. The user may assign a value to an optional parameter by using a statement of the form "`parameter = value`", which can appear either in a user-supplied file of options, or as the argument of the subroutine `npoptn` supplied with NPSOL. For example, the tolerance δ that determines whether or not a constraint is feasible can be altered using a statement of the form `feasibility tolerance = 1.0e-5`. The user needs to specify only those optional parameters whose values are to be different from their default values. §8.1, which concerns the assignment of the default parameters, can be skipped by users who wish to use the default values for *all* optional parameters.

In the following specifications, we define $\texttt{nctotl} = \texttt{n} + \texttt{nclin} + \texttt{ncnln}$. Note that most machines use `double precision` declarations as shown, but some machines use `real`. The same applies to the user routines `funcon` and `funobj`.

```
      subroutine npsol ( n, nclin, ncnln, ldA, ldJ, ldR,
     $                   A, bl, bu,
     $                   funcon, funobj,
     $                   inform, iter, istate,
     $                   c, cJac, clamda, f, g, R, x,
     $                   iw, leniw, w, lenw )

      external           funcon, funobj
      integer            n, nclin, ncnln, ldA, ldJ, ldR
      integer            inform, iter, leniw, lenw
      integer            istate(n+nclin+ncnln)
      integer            iw(leniw)
      double precision   f
      double precision   A(ldA,*), bl(n+nclin+ncnln), bu(n+nclin+ncnln)
      double precision   c(*), cJac(ldJ,*), clamda(n+nclin+ncnln)
      double precision   g(n), R(ldR,*), x(n)
      double precision   w(lenw)
```

On entry:

`n` (> 0) is n, the number of variables in the problem.

`nclin` (≥ 0) is m_L, the number of general linear constraints.

`ncnln` (≥ 0) is m_N, the number of nonlinear constraints.

`ldA` (≥ 1 and $\geq$ `nclin`) is the row dimension of the array `A`.

`ldJ` (≥ 1 and $\geq$ `ncnln`) is the row dimension of the array `cJac`.

`ldR` ($\geq$ `n`) is the row dimension of the array `R`.

`A` is an array of dimension (`ldA`,k) for some $k \geq$ `n`. It contains the matrix A for the linear constraints. If `nclin` is zero, `A` is not referenced. (In that case, `A` may be dimensioned (`ldA,1`) with $\texttt{ldA} = 1$, or it could be any convenient array.)

`bl` is an array of dimension at least `nctotl` that contains ℓ, the lower bounds for $r(x)$ in problem NP. To specify a non-existent lower bound ($\ell_j = -\infty$), set $\texttt{bl}(j) \leq$

−`bigbnd`, where `bigbnd` is the `Infinite Bound`, whose default value is 10^{20}. To specify an *equality* constraint (say $r_j(x) = \beta$), set `bl`(j) = `bu`$(j) = \beta$, where $|\beta| <$ `bigbnd`.

`bu` is an array of dimension at least `nctotl` that contains u, the upper bounds for $r(x)$ in problem NP. To specify a non-existent upper bound ($u_j = \infty$), set `bu`$(j) \geq$ `bigbnd`. For the data to be meaningful, it is required that `bl`$(j) \leq$ `bu`(j) for all j.

`funcon` is 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. `funcon` must be declared as `external` in the routine that calls NPSOL. For a detailed description of `funcon`, see §7.2.

`funobj` is the name of a subroutine that calculates the objective function $f(x)$ and (optionally) its gradient for a specified n-vector x. `funobj` must be declared as `external` in the routine that calls NPSOL. For a detailed description of `funobj`, see §7.1.

`istate` is an integer array of dimension at least `nctotl`. It need not be initialized if NPSOL is called with a `Cold Start` (the default option).

For a `Warm start`, `istate` must be set. If NPSOL has just been called on a problem with the same dimensions, `istate` already contains valid values. Otherwise, `istate`(j) should indicate whether either of the constraints $r_j(x) \geq \ell_j$ or $r_j(x) \leq u_j$ is expected to be active at a solution of NP.

The ordering of `istate` is the same as for `bl`, `bu` and $r(x)$, i.e., the first `n` components of `istate` refer to the upper and lower bounds on the variables, the next `nclin` refer to the bounds on Ax, and the last `ncnln` refer to the bounds on $c(x)$. Possible values for `istate`(j) follow.

0	Neither $r_j(x) \geq \ell_j$ nor $r_j(x) \leq u_j$ is expected to be active.
1	$r_j(x) \geq \ell_j$ is expected to be active.
2	$r_j(x) \leq u_j$ is expected to be active.
3	This may be used if $\ell_j = u_j$. Normally an equality constraint $r_j(x) = \ell_j = u_j$ is active at a solution.

The values 1, 2 or 3 all have the same effect when `bl`(j) = `bu`(j). If necessary, NPSOL will override the user's specification of `istate`, so that a poor choice will not cause the algorithm to fail.

`cJac` is an array of dimension (`ldJ`,k) for some $k \geq$ `n`. If `ncnln` = 0, `cJac` is not referenced. (In that case, `cJac` may be dimensioned (`ldJ,1`) with `ldJ` = 1.)

In general, `cJac` need not be initialized before the call to NPSOL. However, if `Derivative level` is 3, any constant elements of `cJac` may be initialized. Such elements need not be reassigned on subsequent calls to `funcon` (see §7.3).

`clamda` is an array of dimension at least `nctotl`. It need not be initialized if NPSOL is called with a `Cold start` (the default).

The ordering of `clamda` is the same as for `bl`, `bu` and `istate`. For a `Warm start`, the components of `clamda` corresponding to nonlinear constraints must contain a multiplier estimate. The sign of each multiplier should match `istate` as follows. If the ith nonlinear constraint is defined as "inactive" via the initial value `istate`(j) = 0, j = `n` + `nclin` + i, then `clamda`(j) should be zero. If the constraint $r_j(x) \geq \ell_j$ is active (`istate`(j) = 1), `clamda`(j) should be non-negative, and if $r_j(x) \leq u_j$ is active (`istate`(j) = 2), `clamda`(j) should be non-positive.

If necessary, NPSOL will change `clamda` to match these rules.

`R` is an array of dimension `(ldR,`k`)` for some $k \geq$ `n`. `R` need not be initialized if NPSOL is called with a `Cold Start` (the default), and will be taken as the identity. For a `Warm Start`, `R` provides the upper-triangular Cholesky factor of the initial approximation of the Hessian of the Lagrangian. The subdiagonal elements of `R` need not be assigned.

`x` is an array of dimension at least `n`. It contains an initial estimate of the solution.

`iw` is an integer array of dimension `leniw` that provides integer workspace for NPSOL.

`leniw` is the dimension of `iw`. It must be at least $3\,\mathtt{n} + \mathtt{nclin} + 2\,\mathtt{ncnln}$.

`w` is an array of dimension `lenw` that provides real workspace for NPSOL.

`lenw` is the dimension of `w`. If there are no general linear constraints and no nonlinear constraints ($\mathtt{nclin} = 0$ and $\mathtt{ncnln} = 0$), `lenw` must be at least $20\,\mathtt{n}$. If there are no nonlinear constraints ($\mathtt{ncnln} = 0$), `lenw` must be at least $2\,\mathtt{n}^2 + 20\,\mathtt{n} + 11\,\mathtt{nclin}$. Otherwise, `lenw` must be at least $2\,\mathtt{n}^2 + \mathtt{n*nclin} + 2\,\mathtt{n*ncnln} + 20\,\mathtt{n} + 11\,\mathtt{nclin} + 21\,\mathtt{ncnln}$.

If `Major print level` is positive, the required amounts of workspace are printed. Thus, appropriate values may be obtained from a preliminary run with `Major print level` > 0 and `leniw` $=$ `lenw` $= 1$. (The values will be printed before NPSOL terminates with `inform` $= 9$.)

On exit:

`inform` reports the result of the call to NPSOL. (If `Major print level` > 0, a short description of `inform` is printed.) The possible values of `inform` follow.

< 0	Either `funcon` or `funobj` has set `mode` to this negative value (see §4).
0	The iterates have converged to a point `x` that satisfies the optimality conditions to the accuracy requested by the `Linear feasibility tolerance`, the `Nonlinear feasibility tolerance`, and the `Optimality tolerance`. That is, the active constraint residuals and the reduced gradient are negligible at `x`.
1	The final iterate `x` satisfies the optimality 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	The linear constraints and bounds could not be satisfied. The problem has no feasible solution. See §9.4 for further comments.
3	The nonlinear constraints could not be satisfied. The problem may have no feasible solution. See §9.4 for further comments.
4	The `Major iteration limit` was reached.
6	`x` does not satisfy the first-order optimality conditions to the required accuracy, and no improved point for the merit function could be found during the final linesearch.
7	The function derivatives returned by `funcon` or `funobj` appear to be incorrect.
9	An input parameter was invalid.

`iter` is the number of major iterations performed.

`istate` describes the status of the constraints $\ell \le r(x) \le u$ in problem NP. For the jth lower or upper bound, $j = 1$ to `nctotl`, the possible values of `istate`(j) are as follows (see Figure 1). δ is the appropriate feasibility tolerance.

-2	(Region 1) The lower bound is violated by more than δ.
-1	(Region 5) The upper bound is violated by more than δ.
0	(Region 3) Both bounds are satisfied by more than δ.
1	(Region 2) The lower bound is active (to within δ).
2	(Region 4) The upper bound is active (to within δ).
3	(Region 2 = Region 4) The bounds are equal and the equality constraint is satisfied (to within δ).

These values of `istate` are labeled in the printed solution according to the table in Figure 2.

Region	1	2	3	4	5	$2 \equiv 4$
`istate`(j)	-2	1	0	2	-1	3
Printed solution	`--`	`LL`	`FR`	`UL`	`++`	`EQ`

Figure 2: Labels used in the printed solution for the regions of Figure 1.

`c` is an 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` is nonzero, `c` contains the values of the nonlinear constraint functions c_i, $i = 1$ to `ncnln`, at the final iterate.

`cJac` contains the Jacobian matrix of the nonlinear constraints at the final iterate, i.e., `cJac`(i, j) contains the partial derivative of the ith constraint function with respect to the jth variable, $i = 1$ to `ncnln`, $j = 1$ to `n`. (See the discussion of `cJac` under `funcon` in §7.2.)

`clamda` contains the QP multipliers from the last QP subproblem. `clamda`(j) should be non-negative if `istate`$(j) = 1$ and non-positive if `istate`$(j) = 2$.

`f` is the value of the objective $f(x)$ at the final iterate.

`g` is an array of dimension at least `n` that contains the objective gradient (or its finite-difference approximation) at the final iterate.

`R` contains information about H, the Hessian of the Lagrangian. If `Hessian = Yes`, `R` is the upper-triangular Cholesky factor of an approximation to H, with the variables in natural order. If `Hessian = No` (the default), `R` contains the upper-triangular factor of $Q^T HQ$, an estimate of the transformed Hessian of the Lagrangian at `x` (see (6.10) in §6).

`x` contains the final estimate of the solution.

4. User-Supplied Subroutines

The user must provide subroutines that define the objective function and nonlinear constraints. The objective function is defined by subroutine `funobj`, and the nonlinear constraints are defined by subroutine `funcon`. *On every call*, these subroutines must return appropriate values of the objective and nonlinear constraints in `f` and `c`.

For maximum reliability, it is preferable for the user to provide *all* partial derivatives. If it is not possible to provide all gradients, see §7.

While developing the subroutines `funobj` and `funcon`, the `Verify` parameter (see §8.1) should be used to check the calculation of any known gradients.

4.1. Subroutine funobj

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

```
      subroutine funobj( mode, n, x, f, g, nstate )
      integer            mode, n, nstate
      double precision   f
      double precision   x(n), g(n)
```

On entry:

`mode` can be ignored if the default derivative level is being used.

`n` (> 0) is 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* `funobj`.

`x` is 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* `funobj`.

`nstate` can be ignored by the unsophisticated user.

On exit:

`mode` can be used to end the solution of the current problem. If `mode` is set to a negative value, `npsol` will be terminated.

`f` must contain the computed value of $f(x)$.

`g` must contain the components of the gradient vector $g(x)$, i.e., `g`(j) contains the partial derivative $\partial f/\partial x_j$.

4.2. Subroutine funcon

This subroutine must compute the nonlinear constraint functions $c(x)$ and (optionally) their gradients. (A dummy subroutine `funcon` must be provided if all constraints are linear.) The ith 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 `funcon` is

```
      subroutine funcon( mode, ncnln, n, ldJ,
     $                   needc, x, c, cJac, nstate )

      integer            mode, ncnln, n, ldJ, nstate
      integer            needc(*)
      double precision   x(n), c(*), cJac(ldJ,*)
```

On entry:

`mode`	can be ignored if the default derivative level is being used.
`ncnln`	is 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* `funcon`.
`n`	(> 0) is 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* `funcon`.
`ldJ`	(≥ 1 and $\geq$ `ncnln`) is the leading dimension of the array `cJac`.
`needc`	can be ignored by the unsophisticated user.
`x`	is 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* `funcon`.
`nstate`	has the same meaning as for `funobj`.

On exit:

`mode`	can be used to end the solution of the current problem. If `mode` is set to a negative value, `npsol` will be terminated.
`c`	is an array of dimension at least `ncnln` that contains the appropriate values of the nonlinear constraints. The value of the ith constraint at `x` must be stored in `c`(i).
`cJac`	is an array of declared dimension `(ldJ,`k`)`, where $k \geq$ `n`. It contains the elements of the Jacobian evaluated at `x`.

5. Printing the brief log

The default is for a line of information to be sent to the summary file at the end of each major iteration. (The amount of printed output from NPSOL can be controlled by the user. See the descriptions of `Major print level` and `Minor print level` in §8.1).

`Majr`	is the major iteration count.
`Minr`	is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, `Minr` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see §2). Note that `Minr` may be greater than the `Minor iteration limit` if some iterations are required for the feasibility phase.
`Step`	is the step taken along the search direction. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.
`Fun`	is the cumulative number of evaluations of the objective function needed for the line search. Evaluations needed for the estimation of the gradients by finite differences are not included. `Fun` is printed as a guide to the amount of work required for the line search.
`Merit`	is the value of the augmented Lagrangian merit function (6.5). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see §6.2). As the solution is approached, `Merit` will converge to the value of the objective 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 $f(x)$. In this case, the objective will decrease monotonically to its optimal value.
`Norm gZ`	is $\|Z^T g\|$, the Euclidean norm of the reduced gradient (see §6.1). `Norm gZ` will be approximately zero in the neighborhood of a solution.
`Violtn`	is the Euclidean (i.e., two-) norm of the residuals of constraints that are either violated or are in the working set. (This entry is not printed if `ncnln` is zero). `Violtn` will be approximately zero in the neighborhood of a solution.
`nZ`	is the number of columns of Z (see §6.1). The value of `nZ` is the number of variables less the number of constraints in the working set.
`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 (8.1) and 8.2a–b) defined in the description of the optional parameter `Optimality Tolerance` in §8.1. 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 reduced gradient (`Norm gZ`) is sufficiently small; and (c) the norm of the residuals of constraints in the working set is small enough.

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

`c`	is printed if central differences have been 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 line search. (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 `Violtn` imply that `x` is close to a point satisfying the first-order optimality conditions.
`l`	is printed if the line search has produced a relative change in x greater than the value defined by the optional parameter `Step limit`. If this output occurs frequently during later iterations of the run, `Step limit` should be set to a larger value.
`i`	is printed if the QP subproblem has no feasible point.
`m`	is printed if the quasi-Newton update has been modified to ensure that the Hessian approximation is positive-definite (see §6.3).
`r`	is printed if the approximate Hessian has been refactorized. If the diagonal condition estimator of R indicates that the approximate Hessian is badly conditioned, the approximate Hessian is refactorized using column interchanges. If necessary, R is modified so that its diagonal condition estimator is bounded.

6. Advanced Features: Further Details of the SQP Method

The brief description of §2 can be summarized as follows. 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 line search 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.

6.1. Solving the QP subproblem

Let x_0 denote the iterate at the start of a typical major iteration. The search direction p is the vector $\widehat{x} - x_0$, where $\widehat{x}$ is the solution of the quadratic program

$$\begin{array}{ll} \underset{x\in \mathbb{R}^n}{\text{minimize}} & f(x_0) + g(x_0)^T(x - x_0) + \frac{1}{2}(x - x_0)^T H(x - x_0) \\ \text{subject to} & \ell \le r(x_0) + J(x_0)(x - x_0) \le u, \end{array} \tag{6.1}$$

with H a positive-definite quasi-Newton approximation to the Hessian of the Lagrangian. This QP is solved using subroutines from the LSSOL package (Gill *et al.* [GHM+86]), 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. In phase 1 (the ***feasibility phase***), a feasible point for the constraints is found by minimizing the sum of infeasibilities. In phase 2 (the ***optimality phase***), the quadratic objective function is minimized within the feasible region. It is convenient, both conceptually and computationally, to regard the phase 1 and phase 2 iterates as part of a single sequence in which the objective function can change from the sum of infeasibilities to the quadratic objective. If x is any member of this sequence, the next iterate is

$$\bar{x} = x + \sigma d, \tag{6.2}$$

where σ is a non-negative step length and d is a search direction. At each x, the constraints in the working set are satisfied exactly and the direction d is constructed so that the values of constraints in the working set remain *unaltered* for any move along d. This implies that

$$Wd = 0, \quad \text{or, equivalently,} \quad d = Zd_z \quad \text{for some } d_z, \tag{6.3}$$

where W is the working-set matrix and Z is the matrix associated with the TQ factorization (6.8) of W.

The definition of d_z in (6.3) depends on whether the current x is feasible for the QP constraints. If not, d_z is the negative reduced gradient $-Z^Tg$, where g is the gradient of the sum of the infeasibilities at x. If x is feasible, d_z satisfies the equations

$$Z^THZd_z = -Z^Tg, \tag{6.4}$$

where Z^THZ and Z^Tg are the reduced Hessian and reduced gradient of the quadratic (6.1). With (6.4), $x + d$ is the minimizer of the quadratic objective subject to treating the constraints in the working set as equalities.

Whatever the definition of g, if Z^Tg is zero, the current point is a constrained stationary point in the subspace defined by the working set. In phase 1, the reduced gradient will usually be zero only at a vertex (although it may vanish at non-vertices in the presence of constraint dependencies). In phase 2, a zero reduced gradient implies that x minimizes the quadratic objective function when the constraints in the working set are treated as equalities. In either case, Lagrange multipliers are computed and the Lagrange multiplier

π_j is said to be *optimal* if $\pi_j \geq 0$. 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.

The choice of step length σ (6.2) is based on remaining feasible with respect to the satisfied constraints. In phase 2, if $x + d$ is feasible, σ is taken as one. (In this case, the reduced gradient at $x+d$ is zero.) Otherwise, σ is set to the step to the "nearest" constraint, which is added to the working set.

6.2. The merit function

After computing the search direction as described in §6.1, each major iteration proceeds by determining a step length α in (2.2) that produces a "sufficient decrease" in the augmented Lagrangian merit function

$$\mathcal{L}(x, \lambda, s) = f(x) - \sum_i \lambda_i \bigl(c_i(x) - s_i\bigr) + \frac{1}{2} \sum_i \rho_i \bigl(c_i(x) - s_i\bigr)^2, \tag{6.5}$$

where x, λ and s vary during the line search. The summation terms in (6.5) 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 (6.1) 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 §5) is generally non-decreasing, although each ρ_i may be reduced a limited number of times.

The merit function (6.5) and its global convergence properties are described in Gill *et al.* [GMSW92]).

6.3. The quasi-Newton update

The matrix H in (6.1) 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 [DS83].) 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:

$$\bar{H} = H - \frac{1}{s^T H s} H s s^T H + \frac{1}{y^T s} y y^T, \tag{6.6}$$

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 (6.6) will be positive definite if and only if $y^T s$ is positive (see, e.g., Dennis and Moré [DM77]). Ideally, y in (6.6) would be taken as y_L, the change in gradient of the Lagrangian function

$$y_L = g(\bar{x}) - J(\bar{x})^T \pi_N - g(x) + J(x)^T \pi_N, \tag{6.7}$$

where π_N denotes the QP multipliers associated with the nonlinear constraints of the original problem. If $y_L^T s$ is not sufficiently positive, an attempt is made to perform the update with a vector y of the form

$$y = y_L + J(\bar{x})^T \Omega c(\bar{x}) - J(x)^T \Omega c(x),$$

where Ω is a diagonal matrix with nonnegative entries. 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.

6.4. The transformed Hessian

NPSOL is sometimes known as a *transformed Hessian method* because a certain transformed Hessian is stored and updated instead of the Hessian itself. Given any working-set matrix W, the transformation for the Hessian is based on the TQ *factorization*

$$WQ = \begin{pmatrix} 0 & T \end{pmatrix}, \tag{6.8}$$

where T is a nonsingular $m_w \times m_w$ reverse-triangular matrix (i.e., $t_{ij} = 0$ if $i > j$), and the non-singular $n \times n$ matrix Q is the product of orthogonal transformations (see Gill *et al.* [GMSW84]). If the columns of Q are partitioned to match the zero and nonzero blocks of WQ, then

$$Q = \begin{pmatrix} Z & Y \end{pmatrix}, \tag{6.9}$$

and the n_Z ($n_Z \equiv n - m_w$) columns of Z form the requisite basis for the null space of W.

The nonsingular matrix Q defines a transformation of variables $x = Qx_Q$. For any function with gradient g and Hessian H, the gradient and Hessian with respect to the transformed variables are given by

$$g_Q = Q^T g, \quad \text{and} \quad H_Q = Q^T H Q,$$

which are known as the *transformed gradient* and *transformed Hessian* respectively. From (6.9), these derivatives may be partitioned as

$$g_Q = \begin{pmatrix} Z^T g \\ Y^T g \end{pmatrix} \quad \text{and} \quad H_Q = \begin{pmatrix} Z^T H Z & Z^T H Y \\ Y^T H Z & Y^T H Y \end{pmatrix}.$$

It follows that the reduced gradient and reduced Hessian are obtained as part of the transformed gradient and transformed Hessian. The upper-triangular Cholesky factor of H_Q satisfies

$$R^T R = Q^T H Q. \tag{6.10}$$

The form of Q (6.9) implies that the Cholesky factor of the reduced Hessian $Z^T H Z$ is simply the upper left corner of R.

During minor iterations, the matrices T, Q and R are updated to reflect changes in W. (In phase 2, the vector $Q^T g$ is also updated.) Since Q and R are known for each working set, the Hessian need not be stored explicitly.

Given *any* nonsingular matrix Q, the BFGS update to H implies the following update to $Q^T H Q$:

$$\bar{H}_Q = H_Q - \frac{1}{s_Q^T H_Q s_Q} H_Q s_Q s_Q^T H_Q + \frac{1}{y_Q^T s_Q} y_Q y_Q^T, \tag{6.11}$$

where $\bar{H}_Q = Q^T \bar{H} Q$, $H_Q = Q^T H Q$, $y_Q = Q^T y$ and $s_Q = Q^{-1} s$. This update may be expressed as a *rank-one* update to the Cholesky factor R of $Q^T H Q$ (see Goldfarb [Gol76], Dennis and Schnabel [DS81]).

6.5. Treatment of simple upper and lower bounds

NPSOL deals specially with bound constraints $\ell \le x \le u$. The presence of a bound constraint in the working set has the effect of fixing the corresponding component of the search direction to zero. Thus, the associated variable is *fixed*, and specification of the working set induces a partition of x into *fixed* and *free* variables. For some permutation P, the working-set matrix satisfies

$$WP = \begin{pmatrix} F & N \\ & I_N \end{pmatrix},$$

where $(\; F \quad N \;)$ is part of the matrix A, and I_N corresponds to some of the bounds. The matrices F and N contain the free and fixed columns of the general constraints in the working set. A TQ factorization $FQ_F = (\; 0 \quad T_F \;)$ of the smaller matrix F provides the required T and Q as follows:

$$Q = P \begin{pmatrix} Q_F & \\ & I_N \end{pmatrix}, \qquad T = \begin{pmatrix} T_F & N \\ & I_N \end{pmatrix}.$$

The matrix Q_F is implemented as a dense *orthogonal* matrix. Each change in the working set leads to a simple change to F: if the status of a general constraint changes, a *row* of F is altered; if a bound constraint enters or leaves the working set, a *column* of F changes. The matrices T_F, Q_F and R are held explicitly; together with the vectors Q^Tg, and Q^Tc. Products of plane rotations are used to update Q_F and T_F as the working set changes. The triangular factor R associated with the reduced Hessian is only updated during the optimality phase.

7. Advanced Features: User-Supplied Subroutines

The user must provide subroutines that define the objective function and nonlinear constraints. The objective function is defined by subroutine `funobj`, and the nonlinear constraints are defined by subroutine `funcon`. *On every call*, these subroutines must return appropriate values of the objective and nonlinear constraints in `f` and `c`. The user should also provide the available partial derivatives. Any unspecified derivatives are approximated by finite differences; see §8.1 for a discussion of the optional parameter `Derivative level`. Just before either `funobj` or `funcon` is called, each element of the current gradient array `g` 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 [GMW81], for a detailed discussion). If all gradients cannot be provided, it is similarly advisable to provide as many as possible. While developing the subroutines `funobj` and `funcon`, the `Verify` parameter (see §8.1) should be used to check the calculation of any known gradients.

7.1. Subroutine funobj

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

```
subroutine funobj( mode, n, x, f, g, nstate )
integer            mode, n, nstate
double precision   f
double precision   x(n), g(n)
```

On entry:

`mode` (≥ 0 and ≤ 2) is set by NPSOL to indicate which values are to be assigned during the call of `funobj`. `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. If some gradient elements are unspecified, NPSOL will call `funobj` with `mode` = 0, 1 or 2.

- If `mode` = 2, assign `f` and the known components of `g`.
- If `mode` = 1, assign all available components of `g`; `f` is not required.
- If `mode` = 0, only `f` needs to be assigned; `g` is ignored.

`n` (> 0) is 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* `funobj`.

`x` is 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* `funobj`.

`nstate` allows the user to save computation time if certain data must be read or calculated only once. If `nstate` = 1, NPSOL is calling `funobj` for the first time. If there are nonlinear constraints, the first call to `funcon` will occur before the first call to `funobj`.

On exit:

`mode` can be used to end the solution of the current problem. If `mode` is set to a negative value, NPSOL will be terminated.

f must contain the computed value of $f(x)$.

g must contain the assigned components of the gradient vector $g(x)$, i.e., g(j) contains the partial derivative $\partial f/\partial x_j$.

7.2. Subroutine funcon

This subroutine must compute the nonlinear constraint functions $c(x)$ and (optionally) their gradients. (A dummy subroutine `funcon` must be provided if all constraints are linear.) The ith 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 `funcon` is

```
      subroutine funcon( mode, ncnln, n, ldJ,
     $                   needc, x, c, cJac, nstate )

      integer            mode, ncnln, n, ldJ, nstate
      integer            needc(*)
      double precision   x(n), c(*), cJac(ldJ,*)
```

On entry:

`mode` is set by NPSOL to indicate the values that must be assigned during each call of `funcon`. `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 §8.1). If some elements of `cJac` are unspecified, NPSOL will call `funcon` 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.

`ncnln` is 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* `funcon`.

`n` (> 0) is 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* `funcon`.

`ldJ` (≥ 1 and $\geq$ `ncnln`) is the leading dimension of the array `cJac`.

`needc` is an array of dimension at least `ncnln` containing the indices of the elements of `c` or `cJac` that *must* be evaluated by `funcon`. `needc` can be ignored if every constraint is provided.

`x` is 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* `funcon`.

`nstate` has the same meaning as for `funobj`.

On exit:

`mode`	can be used to end the solution of the current problem. If `mode` is set to a negative value, NPSOL will be terminated.
`c`	is an array of dimension at least `ncnln` that contains the appropriate values of the nonlinear constraints. If `needc`(i) is nonzero and `mode` $= 0$ or 2, the value of the ith constraint at `x` must be stored in `c`(i). (The other components of `c` are ignored.)
`cJac`	is an array of declared dimension `(ldJ,`k`)`, where $k \geq$ `n`. It contains the appropriate elements of the Jacobian evaluated at `x`. (See the discussion of `mode` and `cJac` above.)

7.3. Constant Jacobian elements

If all constraint gradients (Jacobian elements) are known (i.e., `Derivative Level = 2` or `3`, 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 `funcon` 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 `funcon` (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 nonzero elements may be reset by `funcon`.

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 `funcon`, 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`, 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.

8. Advanced Features: Optional Input Parameters

This section can be skipped by users who wish to use the default values for *all* optional parameters.

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 `E`, `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. Synonyms are recognized 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
Feasibility tolerance      1.0E-8        *  for IEEE double precision
CRASH TOLERANCE   = .002
* This string will be completely ignored.
Hessian                  Yes
Iteration limit          100
```

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
```

The parameter `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 logical 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. 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.

8.1. 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 qualifier 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 the 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 = `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 §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. The user's specification of `istate` will be overridden 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 `funobj` and `funcon`. The possible choices for i are the following.

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 `funobj` is required for each unspecified element. Furthermore, less accuracy can be attained in the solution (see Chapter 8 of Gill, Murray and Wright [GMW81], for a discussion of limiting accuracy).

If i = `0` or `1`, NPSOL will approximate unspecified elements of the Jacobian. One call to `funcon` 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 `funcon` 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 `funcon`.)

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

`Difference interval` r Default = intervals 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 $\delta_j = r(1 + |\widehat{x}_j|)$, with $\widehat{x}$ 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 [GMW81] 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 `funcon` and `funobj` 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 `funcon` and `funobj` in §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 [GMW81] 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
`Hessian`	`Yes`	

This option controls the contents of the upper-triangular matrix `R` (see §3). NPSOL works exclusively with the factor of the *transformed* Hessian H_Q (6.10), 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 $= 10^{20}$

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 $= \max(bigbnd, 10^{20})$

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 $= \max(50, 3(n + m_L) + 10m_N)$
`Iters`
`Itns`

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}.

`Line search 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 line search). 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.

0 No output except error messages.

1 The final solution.

5 One line of output for each 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 (6.8) of the QP working set, and the diagonal elements of R, the triangular factor of the transformed Hessian (6.10).

`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.

0 No output except error messages.

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 (print file only).

≥ 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 (print file only).

≥ 30 At each minor iteration, the diagonal elements of the matrix T associated with the TQ factorization (6.8) of the QP working set, and the diagonal elements of the Cholesky factor R of the transformed Hessian (6.10) (print file only).

`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.

Successful termination will occur if the iterative sequence of x-values is judged to have converged and the final point satisfies the first-order optimality conditions (see §2). The sequence of iterates is considered to have converged at x if

$$\alpha\|p\| \leq \sqrt{r}(1 + \|x\|), \tag{8.1}$$

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

$$\|Z^T g\| \leq \sqrt{r}\big(1 + \max(1 + |f(x)|, \|g_{\text{FR}}\|)\big) \tag{8.2a}$$

$$|v_j| \leq \mathit{ftol} \quad \text{for all } j, \tag{8.2b}$$

where $Z^T g$ is the reduced gradient (see §2), g_{FR} is the gradient of $f(x)$ with respect to the free variables, v_j is the violation of the jth nonlinear constraint in the working set, and *ftol* is the value of the optional parameter `Nonlinear feasibility tolerance`.

`Print File` i Default $=$ `nout`

If $i > 0$ and `Print Level` > 0, a full log in 132-column format is sent to the file with logical unit number i. This option does not affect the printing of the optional parameters, which is done on the default printer `nout`. The option listing can be suppressed by including `Nolist` as the first option.

The option `Print file = 0` suppresses any printing not associated with the optional parameters, irrespective of the value of `Print level`.

Specifying both `Nolist` and `Print file = 0` suppresses all printing, *including error messages*.

If `Print file` and `Summary file` have the same unit number, the summary output takes precedence, i.e., only the summary output will appear will appear on unit i. For example, on UNIX systems with the default printer `nout` defined as the standard output, the default values of `Print file` and `Summary file` will result in the summary output appearing at the terminal. No other print files will be used.

`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 `funobj` and `funcon`. 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.

`Step limit` r Default = 2.0

If $r > 0$, r specifies the maximum change in variables at the first step of the line search.

In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The parameter r is therefore used to encourage evaluation of the problem functions at meaningful points. Given any major iterate x, the first point $\widetilde{x}$ at which f and c are evaluated during the line search is restricted so that

$$\|\widetilde{x} - x\|_2 \le r(1 + \|x\|_2).$$

The line search may go on and evaluate f and c at points further from x if this will result in a lower value of the merit function. In this case, the character "`l`" is printed at the end of the optional line of printed output. If "`l`" is printed for most of the iterations, r should be set to a larger value.

Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at wild values. The default value `Step limit = 2.0` should not affect progress on well-behaved functions, but values `0.1` or `0.01` may be helpful when rapidly varying functions are present. If a small value of `Step limit` is selected, a good starting point may be required. An important application is to the class of nonlinear least-squares problems. If $r \le 0$, the default value is used.

`Summary file` i Default = 6

If $i > 0$, a brief log will be output to file i. In an interactive environment, it is useful to direct this output to the terminal, to allow a run to be monitored on-line. If `Major print level` $>$ 0 a line of information is printed every major iteration. If `Minor print level` > 0 a line of information is printed every minor iteration.

If `Print file` and `Summary file` have the same unit number, the summary output takes precedence, i.e., only the summary output will appear will appear on unit i.

`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 `funobj` and `funcon`. (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 `funobj` and one call to `funcon`. 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.

8.2. 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 IEEE standard floating-point arithmetic in double precision ($\epsilon \approx 1.1 \times 10^{-16}$). Similar values would apply to any machine having about 16 decimal digits of precision.

```
BEGIN checklist of SPECS file parameters and their default values
* Printing
  Major print level             10      * 1-line major iteration log
  Minor print level              0      * no minor iteration log
  Print   file                nout      * set in subroutine mchpar
  Summary file                   6      * typically the screen
  Print   frequency              1      * minor iterations log on PRINT file
  Summary frequency              1      * minor iterations log on SUMMARY file

* Convergence Tolerances
  Feasibility tolerance           1.0e-8  * target nonlinear constraint violation
  Linear feasibility tolerance    1.0e-8  * linear constraint violation
  Nonlinear feasibility tolerance 1.0e-8  *
  Optimality  tolerance           1.0e-12 *

* Derivative checking
  Verify level                   0      * cheap check on gradients
  Start objective  check at col  1
  Stop  objective  check at col  n
  Start constraint check at col  1
  Stop  constraint check at col  n
```

```
* Other Tolerances
  Crash tolerance                0.01   *
  Line search tolerance          0.9    * smaller for more accurate search

* SQP method
  Cold start                            *
  Major iterations limit          50    * or 3(n + m_L) + 10m_N if that is more
  Minor iterations limit          50    * or 3(n + m_L + m_N) if that is more
  Step limit                     2.0    *
  Superbasics limit              500    * or n_1 + 1 if that is less
  Derivative level               3      * assumes all gradients are known
  Derivative linesearch                 *
  Function precision             1.0e-15 * ε^0.9 (almost full accuracy)
  Difference interval                   * Computed automatically
  Central difference interval           * Computed automatically
  Infinite step size             1.0e+20 *
  Infinite bound                 1.0e+20 *

* Hessian approximation
  Hessian}                       No     *  Lagrangian Hessian not saved
End of SPECS file checklist
```

9. Advanced Features: Printing Details of a Run

9.1. The full log of the major iterations

When `Major print level` ≥ 5, the following line of output is sent to the print file. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Majr`	is the major iteration count.
`Minr`	is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, `Minr` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see §6). Note that `Minr` may be greater than the `Minor iteration limit` if some iterations are required for the feasibility phase.
`Step`	is the step taken along the search direction. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.
`Fun`	is the cumulative number of evaluations of the objective function needed for the line search. Evaluations needed for the estimation of the gradients by finite differences are not included. `Fun` is printed as a guide to the amount of work required for the line search.
`Merit`	is the value of the augmented Lagrangian merit function (6.5). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see §6.2). As the solution is approached, `Merit` will converge to the value of the objective 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 $f(x)$. In this case, the objective will decrease monotonically to its optimal value.
`Norm gZ`	is $\|Z^Tg\|$, the Euclidean norm of the reduced gradient (see §6.1). `Norm gZ` will be approximately zero in the neighborhood of a solution.
`Violtn`	is the Euclidean (i.e., two-) norm of the residuals of constraints that are either violated or are in the working set. (This entry is not printed if `ncnln` is zero). `Violtn` will be approximately zero in the neighborhood of a solution.
`nZ`	is the number of columns of Z (see §6.1). The value of `nZ` is the number of variables less the number of constraints in the working set; i.e., `nZ` $=$ `n` $-$ (`Bnd` $+$ `Lin` $+$ `Nln`).
`Bnd`	is the number of simple bound constraints in the working set.
`Lin`	is the number of general linear constraints in the working set.
`Nln`	is the number of nonlinear constraints in the working set (not printed if `ncnln` is zero).
`Penalty`	is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if `ncnln` is zero).

`Cond Hz` is a lower bound on the condition number of the reduced Hessian approximation Z^THZ; see §2. The larger this number, the more difficult the problem.

`Cond T` is a lower bound on the condition number of the working set matrix.

`Conv` is a three-letter indication of the status of the three convergence tests (8.1) and 8.2a–b) defined in the description of the optional parameter `Optimality Tolerance` in §8.1. 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 reduced gradient (`Norm gZ`) is sufficiently small; and (c) the norm of the residuals of constraints in the working set is small enough.

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

`c` is printed if central differences have been 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 line search. (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 `Violtn` imply that `x` is close to a point satisfying the first-order optimality conditions.

`l` is printed if the line search has produced a relative change in x greater than the value defined by the optional parameter `Step limit`. If this output occurs frequently during later iterations of the run, `Step limit` should be set to a larger value.

`i` is printed if the QP subproblem has no feasible point.

`m` is printed if the quasi-Newton update has been modified to ensure that the Hessian approximation is positive-definite (see §6.3).

`r` is printed if the approximate Hessian has been refactorized. If the diagonal condition estimator of R indicates that the approximate Hessian is badly conditioned, the approximate Hessian is refactorized using column interchanges. If necessary, R is modified so that its diagonal condition estimator is bounded.

9.2. Printing the solution of NP

When `Major print level` $= 1$ or `Major print level` ≥ 10, the 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.

To aid interpretation of the printed results, we repeat the convention for numbering the constraints: indices 1 through `n` refer to the bounds on the variables, and indices $n + 1$ through `n` + `nclin` refer to the general constraints.

The following describes the printout for each variable.

`Variable` gives the state of the jth variable x_j. The various possible states are as follows (see Fig. 1). δ is the appropriate feasibility tolerance.

`State` gives the state of the jth variable x_j. The various possible states are as follows (see Fig. 1). δ is the appropriate feasibility tolerance.

`FR` The variable lies between its upper and lower bound.

`EQ` The variable is a fixed variable, with x_i equal to its upper and lower bound.

`LL`	The variable is active at its lower bound (to within δ).
`UL`	The variable is active at its upper bound (to within δ).
`TF`	The variable is temporarily fixed at its current value.
`--`	The lower bound is violated by more than δ.
`++`	The upper bound is violated by more than δ.

A key is sometimes printed before the `State` to give some additional information about the state of a variable.

`A`	*Alternative optimum possible.* The variable is active at one of its bounds, but its Lagrange multiplier is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change to the objective function. The values of the other free variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case, the values of the Lagrange multipliers might also change.
`D`	*Degenerate.* The variable is free, but it is equal to (or very close to) one of its bounds.
`I`	*Infeasible.* The variable is currently violating one of its bounds by more than δ.

`Value`	is the final value of the variable x_j.
`Lower bound`	is the lower bound specified for x_j. "None" indicates that `bl`$(j) \leq$ −`bigbnd`. A "·" is printed for any bound that is zero.
`Upper bound`	is the upper bound specified for x_j. "None" indicates that `bu`$(j) \geq$ `bigbnd`. A "·" is printed for any bound that is zero.
`Lagr multiplier`	is the Lagrange multiplier for the associated bound. 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`. A "·" is printed for any multiplier that is zero.
`Slack`	is the difference between the variable "`Value`" and the nearer of its (finite) bounds `bl`(j) and `bu`(j). A blank entry indicates that the associated variable is not bounded (i.e., `bl`$(j) \leq$ −`bigbnd` and `bu`$(j) \geq$ `bigbnd`).

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

`Linear constrnt`	is the name (`lncon`) and index i ($i = 1$ to `nclin`) of a linear constraint.
`Nonlin constrnt`	is the name (`nlcon`) and index i ($i = 1$ to `ncnln`) of a nonlinear constraint.

As in the variables section, a key is occasionally printed before the `State` to give some additional information about the state of the general constraints. The possible values are `A`, `D`, and `I`. They have the same meaning as described above (for the variables), except that the word "variable" must be replaced by "constraint". In this context, "movement off a constraint" can be interpreted as allowing the entry in the `slack` column to become positive.

9.3. Printing details of the minor iterations

If `Minor print level` > 0, output is obtained from the subroutines that solve the QP subproblem.

To aid interpretation of the printed results, we repeat the convention for numbering the constraints: indices 1 through `n` refer to the bounds on the variables, and indices $n+1$ through `n + nclin` refer to the general constraints. When the status of a constraint changes, the index of the constraint is printed, along with the designation "`L`" (lower bound), "`U`" (upper bound), "`E`" (equality), "`F`" (temporarily fixed variable) or "`A`" (artificial constraint). For a more detailed description of this information the reader should refer to the user's guide for LSSOL (Gill *et al.* . [GHM+86]).

When `Printl level` ≥ 5, the following line of output is produced at every iteration. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Itn`	is the minor iteration count.
`Jdel`	is the index of the constraint deleted from the working set. If `Jdel` is zero, no constraint was deleted.
`Jadd`	is the index of the constraint added to the working set. If `Jadd` is zero, no constraint was added.
`Step`	is the step taken along the computed QP search direction. If a constraint is added during the current minor iteration (i.e., `Jadd` is positive), `Step` will be the step to the nearest constraint.
`Ninf`	is the number of violated constraints (infeasibilities) in the QP subproblem. This number will be zero during the optimality phase.
`Sinf/Objective`	is the current value of the objective. If `x` is not feasible, `Sinf` gives a weighted sum of the magnitudes of constraint violations. If `x` is feasible, `Objective` is the value of the QP objective. The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true QP objective at the first feasible point. During the optimality phase, the value of the objective will be non-increasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the sign of the multipliers implies that no feasible point exists. Once optimal multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.
`Norm gZ`	is $\|Z_R^T g\|$, the Euclidean norm of the reduced gradient with respect to Z_R. During the optimality phase, this norm will be approximately zero after a unit step.
`Zr`	is the number of columns of Z_R (see §6). `Zr` is the dimension of the subspace in which the objective is currently being minimized. The value of `Zr` is the number of variables less the number of constraints in the working set; i.e., `Zr` = `n` − (`Bnd` + `Lin` + `Art`). The value of `Nz`, the number of columns of Z (see §6) can be calculated as `Nz` = `n` − (`Bnd` + `Lin`). A zero value of `Nz` implies that the current QP iterate lies at a vertex of the feasible region.

`Art`	is the number of artificial constraints in the working set, i.e., the number of columns of Z_A (see §6).
`Bnd`	is the number of simple bound constraints in the working set.
`Lin`	is the number of linear constraints and linearized constraints in the current QP working set.
`Cond T`	is an estimate of the condition number of the QP working set.
`Cond Rz`	is a lower bound on the condition number of the triangular factor R_Z (the Cholesky factor of the current reduced Hessian).

When `Print level` = 1 or `Print level` $\geq$ 10, the summary printout at the end of execution of `lssol` 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. An entry "`.`" indicates that the quantity to be printed is zero (zeros are suppressed in order to minimize clutter in the output).

`Variable` gives the name (`varbl`) and index j ($j = 1$ to `n`) of the variable.

`State` gives the state of the jth variable x_j. The various possible states are as follows (see Figure 1). δ is the appropriate feasibility tolerance.

`FR`	The variable lies between its upper and lower bound.
`EQ`	The variable is a fixed variable $x_i = \alpha = \beta$.
`LL`	The variable is active at its lower bound (to within δ).
`UL`	The variable is active at its upper bound (to within δ).
`TF`	The variable is temporarily fixed at its current value.
`--`	The lower bound is violated by more than δ.
`++`	The upper bound is violated by more than δ.

A key is sometimes printed before the `State` to give some additional information about the state of a variable.

`A`	*Alternative optimum possible.* The variable is active at one of its bounds, but its Lagrange multiplier is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change to the objective function. The values of the other free variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case, the values of the Lagrange multipliers might also change.
`D`	*Degenerate.* The variable is free, but it is equal to (or very close to) one of its bounds.
`I`	*Infeasible.* The variable is currently violating one of its bounds by more than δ.

`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`.
`Slack`	is the difference between the variable "`Value`" and the nearer of its bounds `bl`(j) and `bu`(j).

The meaning of the printout for general constraints is the same as that given above for variables, with "variable" replaced by "constraint", with the following change in the heading:

`Linear constr` is the name (`lncon`) and index i ($i = 1$ to `nclin`) of the constraint.

As in the variables section, a key is occasionally printed before the `State` to give some additional information about the state of the general constraints. The possible values are `A`, `D`, and `I`. They have the same meaning as described above (for the variables), except that the word "variable" must be replaced by "constraint". In this context, movement off a constraint can be interpreted as allowing the entry in the `slack` column to become positive.

9.4. Interpretation of the output

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 `funobj` and `funcon` compute all relevant components of `g`, `c` and `cJac`.

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

`Underflow` A single underflow will always occur if machine constants are computed automatically (as in the distributed version of NPSOL). 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 jth 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 reduce the value of the optional parameter `Step limit` or 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.

`inform` = 0 The iterates have converged to a point `x` that satisfies the first-order optimality conditions to the `Optimality tolerance`, i.e., the reduced 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 `Minr` are both one;

(iii) the last few values of both `Norm gZ` and `Violtn` become small at a fast linear rate;

(iv) and `Cond Hz` is small.

If all these conditions hold, `x` is almost certainly a local minimizer of NP. (See §10 for a specific example.)

`inform` = 1 The point `x` satisfies the first-order optimality 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`). This condition will also occur if, by chance, an iterate is an "exact" first-oder optimal 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 The linear constraints and bounds have not been satisfied. This means that either no feasible point exists for the given value of `Linear feasibility tolerance`, or no feasible point could be found in the number of iterations specified by `Minor iteration limit`. 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 a `Warm start`). 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 reset during the latter iterations (i.e., an "`r`" 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 line search. 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 [GMW81], 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 `funobj` or `funcon` 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 `Violtn` 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 `Minr` (the number of iterations required to solve each QP subproblem).

If the condition estimate of the reduced Hessian (`Cond Hz`) is extremely large, it may be worthwhile to rerun NPSOL from the final point with a `Warm start`. 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.

Gradient checking will be ineffective if the objective uses information computed by the constraints. The constraints are not necessarily computed prior to each function evaluation.

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.

10. Advanced Features: A Sample Problem

This section describes one version of the so-called “hexagon” problem (a different formulation is given as Problem 108 in Hock and Schittkowski [HS81]). 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 below.

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,

$$\ell_B = \begin{pmatrix} 0 \\ -\infty \\ -1 \\ -\infty \\ 0 \\ 0 \\ 0 \\ -\infty \\ -\infty \end{pmatrix} \quad \text{and} \quad u_B = \begin{pmatrix} \infty \\ \infty \\ 1 \\ \infty \\ \infty \\ \infty \\ \infty \\ 0 \\ 0 \end{pmatrix}.$$

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,

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

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 `fnobj1` and `fncon1`, 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 `fnobj2` and `fncon2`. 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.opt` supplied on the distribution diskette. 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 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 each major iteration, and the "`Norm gZ`" and "`Violtn`" columns exhibit the fast linear convergence rate mentioned in §5 and §9.4. Note that the constraint violations converge earlier than the reduced gradient. The final values of the reduced 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 reduced 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 reduced 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.

10.1. The brief log from the sample problem

```
                    NPSOL  ---  Version 5.0        May  1993
                    ========================================


 Majr Minr    Step  Fun  Merit function  Violtn Norm gZ   nZ Penalty Conv
    0    3 0.0E+00    1 -1.32869556E+00 5.0E-01 2.0E-01    3 3.7E-01 F FF
    1    1 1.0E+00    2 -1.34828028E+00 9.3E-02 1.1E-01    3 8.2E-01 F FF
    2    1 1.0E+00    3 -1.34936555E+00 3.8E-03 3.5E-02    3 8.2E-01 F FF
    3    1 1.0E+00    4 -1.34983423E+00 3.4E-04 1.5E-02    3 8.2E-01 F FF
    4    1 1.0E+00    5 -1.34996288E+00 3.1E-04 1.2E-04    3 1.3E+00 F FF


 Majr Minr    Step  Fun  Merit function  Violtn Norm gZ   nZ Penalty Conv
    5    1 1.0E+00    6 -1.34996289E+00 1.2E-07 1.3E-05    3 1.3E+00 F FT
    6    1 1.0E+00    7 -1.34996289E+00 2.5E-10 1.5E-06    3 1.3E+00 F TT
    7    1 1.0E+00    8 -1.34996289E+00 1.9E-12 5.8E-08    3 1.3E+00 T TT

 Exit NPSOL - Optimal solution found.

 Final nonlinear objective value =   -1.349963
```

10.2. Printed output from the sample problem

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

     BEGIN  Options for NPSOL 5.0 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


Calls to Option Routine
-----------------------

     Infinite Bound size =1.0d+21




                  NPSOL  ---  Version 5.0       May  1993
                  ========================================


Parameters
----------

Linear constraints.....         4     Linear feasibility.....  1.05E-08     cold start............
Variables.............          9     Infinite bound size....  1.00E+21     Crash tolerance........  1.00E-02
Step limit............   2.00E+00     Infinite step size.....  1.00E+21

Nonlinear constraints..        14     Optimality tolerance...  3.26E-12     Function precision.....  4.37E-15
Nonlinear Jacobian vars         9     Nonlinear feasibility..  1.05E-08     Unit round-off.........  1.11E-16
Nonlinear objectiv vars         9     Line search tolerance..  9.00E-01     Print file............          9
Derivative level.......         3     Verify level..........          3     Summary file..........          6

Major iterations limit.        50     Major print level......          5
Minor iterations limit.        81     Minor print level......          0
RUN loaded from file...         0     RUN to be saved on file         0     Save frequency.........        51

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



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

The constraint Jacobian seems to be ok.

The largest relative error was    7.33E-09  in row    2



Column    x(j)     dx(j)    Row    Jacobian Value      Difference Approxn  Itns

     1  1.00E-01  9.58E-08    1    2.00000000E-01    2.00000000E-01    OK     1
                  9.37E-08    2   -5.00000000E-02   -5.00000000E-02    OK     1
                  1.09E-07    3   -1.13333200E+00   -1.13333200E+00    OK     1
                  1.01E-07    4   -8.57140000E-02   -8.57140000E-02    OK     1
                  1.03E-07    5   -2.22220000E-02   -2.22220000E-02    OK     1
```

References

[DM77] J. E. Dennis Jr. and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19, 46–89, 1977.

[DS81] J. E. Dennis, Jr. and R. B. Schnabel. A new derivation of symmetric positive definite secant updates. In O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, editors, *Nonlinear Programming 4*, pages 167–199. Academic Press, London and New York, 1981.

[DS83] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[Fle81] R. Fletcher. *Practical Methods of Optimization.* Volume 2: Constrained Optimization. John Wiley and Sons, Chichester and New York, 1981.

[GHM+86] P. E. Gill, S. J. Hammarling, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for LSSOL (Version 1.0): a Fortran package for constrained linear least-squares and convex quadratic programming. Report SOL 86-1, Department of Operations Research, Stanford University, 1986.

[GMSW84] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Procedures for optimization problems with a mixture of bounds and general linear constraints. *ACM Transactions on Mathematical Software*, 10, 282–298, 1984.

[GMSW92] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Some theoretical properties of an augmented Lagrangian merit function. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 101–128. North Holland, North Holland, 1992.

[GMW81] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London and New York, 1981. ISBN 0-12-283952-8.

[Gol76] D. Goldfarb. Factorized variable metric methods for unconstrained optimization. *Mathematics of Computation*, 30, 796–811, 1976.

[HS81] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems 187. Springer Verlag, Berlin, Heidelberg and New York, 1981.

[MS93] B. A. Murtagh and M. A. Saunders. MINOS 5.4 User's Guide. Report SOL 83-20R, Department of Operations Research, Stanford University, 1993.

[Pow83] M. J. D. Powell. Variable metric methods for constrained optimization. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 288–311. Springer Verlag, London, Heidelberg, New York and Tokyo, 1983.