USER'S GUIDE FOR SQOPT 5.3: A FORTRAN PACKAGE FOR LARGE-SCALE LINEAR AND QUADRATIC 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

DRAFT, October 1997

Abstract

SQOPT is a set of Fortran subroutines for minimizing a convex quadratic function subject to both equality and inequality constraints. (SQOPT may also be used for linear programming and for finding a feasible point for a set of linear equalities and inequalities.) The method of SQOPT is of the two-phase, active-set type, and is related to the method used in the package QPOPT (Gill, Murray and Saunders [3]). The method used is most efficient when many constraints or bounds are active at the solution.

SQOPT is primarily intended for (but is not restricted to) large linear and quadratic problems with sparse matrices—i.e., matrices with sufficiently many zero elements to justify storing them implicitly.

SQOPT is part of the SNOPT package for large-scale nonlinearly constrained optimization. SQOPT uses stable numerical methods throughout and includes a reliable basis package (for maintaining sparse LU factors of the basis matrix), a practical anti-degeneracy procedure, and optional automatic scaling of the constraints.

The source code for SQOPT is suitable for any machine with a Fortran compiler. SQOPT may be called from a driver program (typically in Fortran, C or MATLAB). SQOPT can also be used as a stand-alone package, reading data in the MPS format used by commercial mathematical programming systems.

Keywords: large-scale linear programming, large-scale quadratic programming, convex quadratic programming, sparse linear constraints, Fortran software.

pgill@ucsd.edu
http://sdna3.ucsd.edu/~peg

walter@sol-walter.stanford.edu
http://www.stanford.edu/~walter/

mike@sol-michael.stanford.edu
http://www.stanford.edu/~saunders/

Contents

1. Introduction

SQOPT is a collection of Fortran 77 subroutines for solving the large-scale *linear or quadratic programming problem*, which is assumed to be stated in following form:

LCQP	$\underset{x}{\text{minimize}} \quad q(x)$
	subject to $\quad l \leq \begin{pmatrix} x \\ Ax \end{pmatrix} \leq u,$

where l and u are constant lower and upper bounds, and A is a sparse matrix and $q(x)$ is a linear or quadratic function objective function that may be specified in a variety of ways, depending upon the particular problem being solved. The optional parameter `maximize` may be used to specify a problem in which q is maximized instead of minimized.

Upper and lower bounds are 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 $l_j = u_j$. 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 $+\infty$.

The possible forms for the function $q(x)$ are summarized in Table 1. The most general form for $q(x)$ is

$$q(x) = f + \sum_{j=1}^{n} c_j x_j + \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} x_i H_{ij} x_j = f + c^T x + \tfrac{1}{2} x^T H x,$$

where f is a constant, c is a constant n vector and H is a constant symmetric $n \times n$ matrix with elements $\{H_{ij}\}$. In this form, q is a quadratic function of x and Problem LCQP is known as a *quadratic program* (QP). SQOPT is suitable for all *convex* quadratic programs. The defining feature of a convex QP is that the matrix H must be *positive semidefinite*—i.e., it must satisfy $x^T H x \geq 0$ for all x. If not, $q(x)$ is nonconvex and SQOPT will terminate with the error indicator `inform` $= 4$.

Table 1: Choices for the objective function $q(x)$.

Problem type	Objective function q	Hessian matrix
Quadratic Programming (QP)	$f + c^T x + \frac{1}{2} x^T H x$	Symmetric positive semidefinite
Linear Programming (LP)	$f + c^T x$	$H = 0$
Feasible Point (FP)	Not Applicable	$f = 0$, $c = 0$, $H = 0$

If $H = 0$, then $q(x) = f + c^T x$ and the problem is known as a *linear program* (LP). In this case, rather than defining an H with zero elements, you can define H to have no columns (see the parameter `ncolH` for subroutine `sqopt`).

If $H = 0$, $f = 0$, and $c = 0$, there is no objective function and the problem is a *feasible point problem* (FP), which is equivalent to finding a point that satisfies the constraints on x. In the situation where no feasible point exists, several options are available for finding a point that minimizes the constraint violations (see the optional parameter `Elastic option`).

SQOPT is suitable for large LPs and QPs in which the matrix A is *sparse*—i.e., when there are sufficiently many zero elements in A to justify storing them implicitly. The matrix A is input to SQOPT by means of the parameters `a(*)`, `ha(*)`, and `ka(*)` (see §3). This allows the user to specify the pattern of nonzero elements in A.

SQOPT exploits structure or sparsity in H by requiring H to be defined *implicitly* in a subroutine that computes the product Hx for any given vector x. In many cases, the product Hx can be computed very efficiently for any given x—e.g., H may be a sparse matrix, or a sum of matrices of rank-one. An example of such a subroutine is included with the sample program for the SQOPT package.

There is considerable flexibility allowed in the definition of $q(x)$ in Table 1. The vector c defining the linear term c^Tx can be input in three ways: as a sparse row of A; as an explicit dense vector c; or as both a sparse row *and* an explicit vector (in which case, c^Tx will be the sum of two linear terms). When stored in A, c is the `iObj`-th row of A, which is known as the *objective row*. The objective row must always be a *free* row of A in the sense that its lower and upper bounds must be $-\infty$ and $+\infty$. Storing c as part of A is recommended if c is a sparse vector. Storing c as an explicit vector is recommended for a sequence of problems, each with a different objective (see parameters `c` and `lenc` of subroutine `sqopt`).

1.1. Subroutines

SQOPT is accessed via the following routines:

`sqInit` (§5.2) Must be called before any other SQOPT routines.

`sqSpec` (§5.3) May be called to input a SPECS file (a list of run-time options).

`sqset`, `sqseti`, `sqsetr` (§5.4) May be called to specify a single option.

`sqgetc`, `sqgeti`, `sqgetr` (§5.5) May be called to obtain an option's current value.

`qpHx` (§4) Called by `sqopt`. Must be supplied by the user to define the matrix-vector product Hx for given vectors x. For FP and LP, you can either provide your own "empty" `qpHx` or use the dummy routine `nullHx` provided with the SQOPT distribution.

`sqopt` (§3) The main solver.

`sqMem` (In distribution file `sn12sqzz.f`) Computes the size of the workspace arrays `cw`, `iw`, `rw` required for given problem dimensions. Intended for Fortran 90 drivers that reallocate workspace if necessary.

The user routine `qpHx` has a fixed parameter list but may have any convenient name. It is passed to `sqopt` as a parameter.

The SQOPT routines are intended to be re-entrant (as long as the Fortran compiler allocates local variables dynamically). Hence they may be used in a parallel or multi-thread environment. They may also be called recursively.

1.2. Files

SQOPT reads or creates the following files:

SPECS file. A list of run-time options, input by `sqSpec`.

PRINT file. A detailed iteration log, error messages, and optionally the printed solution.

SUMMARY file. A brief iteration log, error messages, and the final solution status. Intended for screen output in an interactive environment.

SOLUTION file. A separate copy of the printed solution.

BASIS files. To allow restarts.

You must define unit numbers for the `specs`, `print` and `summary` files by specifying appropriate parameters for `sqInit` and `sqSpec`. For a more detailed description of the files that can be created by SQOPT, see §9.

2. Brief description of the method

Here we briefly describe some features of the algorithm used in SQOPT and introduce some terminology used in the description of the subroutine and its arguments. For further details, see §8.

2.1. Formulation of the problem

The upper and lower bounds on the m components of Ax are said to define the *general constraints* of the problem. Internally SQOPT converts the general constraints to equalities by introducing a set of *slack variables* s, where $s = (s_1, s_2, \ldots, s_m)^T$. For example, the linear constraint $5 \leq 2x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \leq s_1 \leq +\infty$. Problem LCQP can therefore be rewritten in the following equivalent form

$$\underset{x,s}{\text{minimize}} \quad q(x) \qquad \text{subject to} \qquad Ax - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u.$$

Since the slack variables s are subject to the same upper and lower bounds as the components of Ax, they allow us to think of the bounds on Ax and x as simply bounds on the combined vector (x, s). (In order to indicate their special role in problem QP, the original variables x are sometimes known as "column variables", and the slack variables s are known as "row variables")

Each LP or QP is solved using an *active-set* method. This is an iterative procedure with two phases: a *phase 1* (sometimes called the *feasibility phase*), in which the sum of infeasibilities is minimized to find a feasible point; and a *phase 2* (or *optimality phase*), in which q is minimized by constructing a sequence of iterations that lies within the feasible region.

Phase 1 involves solving a linear program of the form

$$\begin{array}{ll} \text{Phase 1} & \underset{x,v,w}{\text{minimize}} \quad \displaystyle\sum_{j=1}^{n+m} (v_j + w_j) \\ & \text{subject to} \quad Ax - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} - v + w \leq u, \quad v \geq 0, \quad w \geq 0, \end{array}$$

which is equivalent to minimizing the sum of the constraint violations. If the constraints are feasible (i.e., at least one feasible point exists), eventually a point will be found at which both v and w are zero. The associated value of (x, s) satisfies the original constraints and is used as the starting point for the phase 2 iterations for minimizing q.

If the constraints are infeasible (i.e., $v \neq 0$ or $w \neq 0$ at the end of phase 1), no solution exists for Problem LCQP and the user has the option of either terminating or continuing in so-called *elastic mode* (see the discussion of the optional parameter `Elastic option`). In elastic mode, a "relaxed" or "perturbed" problem is solved in which $q(x)$ is minimized while allowing some of the bounds to become "elastic"—i.e., to change from their specified values. Variables subject to elastic bounds are known as *elastic variables*. An elastic variable is free to violate one or both of its original upper or lower bounds. The user is able to assign which bounds will become elastic if elastic mode is ever started—see the parameter `helast` of subroutine `sqopt`.

To make the relaxed problem meaningful, SQOPT minimizes q while (in some sense) finding the "smallest" violation of the elastic variables. In the situation where all the

variables are elastic, the relaxed problem has the form

$$\text{Phase2}(\gamma) \qquad \underset{x,v,w}{\text{minimize}} \quad q(x) + \gamma \sum_{j=1}^{n+m} (v_j + w_j)$$
$$\text{subject to} \quad Ax - s = 0, \quad l \le \begin{pmatrix} x \\ s \end{pmatrix} - v + w \le u, \quad v \ge 0, \quad w \ge 0,$$

where γ is a nonnegative parameter known as the *elastic weight*, and $q(x) + \gamma \sum_j (v_j + w_j)$ is called the *composite objective*. In the more general situation where only a subset of the bounds are elastic, the v's and w's for the non-elastic bounds are fixed at zero.

The `elastic weight` can be chosen to make the composite objective behave like either the the original objective $q(x)$ or the sum of infeasibilities. If $\gamma = 0$, SQOPT will attempt to minimize q subject to the (true) upper and lower bounds on the nonelastic variables (and declare the problem infeasible if the nonelastic variables cannot be made feasible). At the other extreme, choosing γ sufficiently large, will have the effect of minimizing the sum of the violations of the elastic variables subject to the original constraints on the non-elastic variables. Choosing a large value of the elastic weight is useful for defining a "least-infeasible" point for an infeasible problem.

In phase 1 and elastic mode, all calculations involving v and w are done implicitly in the sense that an elastic variable x_j is allowed to violate its lower bound (say) and an explicit value of v can be recovered as $v_j = l_j - x_j$.

2.2. The main iteration

A constraint is said to be *active* or *binding* at x if the associated component of either x or Ax is equal to one of its upper or lower bounds. Since an active constraint in Ax has its associated slack variable at a bound, we can neatly describe the status of both simple and general upper and lower bounds in terms of the status of the variables (x, s). A variable is said to be *nonbasic* if it is temporarily fixed at its upper or lower bound. It follows that regarding a general constraint as being *active* is equivalent to thinking of its associated slack as being *nonbasic*.

At each iteration of an active-set method, the constraints $Ax - s = 0$ are (conceptually) partitioned into the form $Bx_B + Sx_S + Nx_N = 0$, where x_N comprises the nonbasic components of (x, s) and the *basis matrix* B is square and nonsingular. The elements of x_B and x_S are called the *basic* and *superbasic* variables respectively; with x_N they are a permutation of the elements of x and s. At a QP solution, the basic and superbasic variables will lie somewhere between their bounds, while the nonbasic variables will be equal to one of their upper or lower bounds. At each iteration, x_S is regarded as a set of independent variables that are free to move in any desired direction, namely one that will improve the value of the QP objective (or sum of infeasibilities). The basis variables are then adjusted in order to ensure that that (x, s) continues to satisfy $Ax - s = 0$. The number of superbasic variables (n_S say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, n_S is a measure of *how nonlinear* the problem is. In particular, n_S will always be zero for FP and LP problems.

If it appears that no improvement can be made with the current definition of B, S and N, a nonbasic variable is selected to be added to S, and the process is repeated with the value of n_S increased by one. At all stages, if a basic or superbasic variables encounters one of its bounds, the variable is made nonbasic and the value of n_S is decreased by one.

Associated with each of the m equality constraints $Ax - s = 0$ is a *dual variable* π_i. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j (also known as a

reduced cost). The reduced gradients for the variables x are the quantities $g - A^T\pi$, where g is the gradient of the QP objective; and the reduced gradients for the slacks s are the dual variables π. The QP subproblem is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for all superbasic variables. In practice, an *approximate* QP solution is found by slightly relaxing these conditions on d_j (see the `Optimality tolerance` described in §5.6).

The process of computing and comparing reduced gradients is known as *pricing* (a term first introduced in the context of the simplex method for linear programming). To "price" a nonbasic variable x_j means that the reduced gradient d_j associated with the relevant active upper or lower bound on x_j is computed via the formula $d_j = g_j - a_j^T\pi$, where a_j is the jth column of $(A \quad -I)$. (The variable selected by the price, and its corresponding value of d_j (i.e., its reduced gradient) are printed in the columns marked `+SBS` and `dj` in the `Print file` output.) If A has significantly more columns than rows (i.e., $n \gg m$), pricing can be computationally expensive. In this case, a strategy known as *partial pricing* can be used to compute and compare only a subset of the d_j's.

References

[1] R. Fourer, *Solving staircase linear programs by the simplex method. 1: Inversion*, Math. Prog., 23 (1982), pp. 274–313.

[2] P. E. Gill and W. Murray, *Numerically stable methods for quadratic programming*, Math. Prog., 14 (1978), pp. 349–372.

[3] P. E. Gill, W. Murray, and M. A. Saunders, *User's guide for QPOPT 1.0: a Fortran package for quadratic programming*, Report SOL 95-4, Department of Operations Research, Stanford University, Stanford, CA, 1995.

[4] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *Maintaining LU factors of a general sparse matrix*, Linear Algebra and its Applications, 88/89 (1987), pp. 239–270.

[5] ———, *A practical anti-cycling procedure for linearly constrained optimization*, Math. Prog., 45 (1989), pp. 437–474.

[6] ———, *Inertia-controlling methods for general quadratic programming*, SIAM Review, 33 (1991), pp. 1–36.

[7] J. A. J. Hall and K. I. M. McKinnon, *The simplest examples where the simplex method cycles and conditions where EXPAND fails to prevent cycling*, Tech. Rep. MS 96-010, Department of Mathematics and Statistics, University of Edinburgh, 1996.

3. Specification of subroutine sqopt

Problem QP is solved by a call to subroutine `sqopt`, whose parameters are defined here. Note that most machines use `double precision` declarations as shown, but some machines use `real`. The same applies to the user routine `qpHx`.

```
      subroutine sqopt ( start, qpHx, m,
     $                   n, lena, nName, lenc, ncolH,
     $                   iObj, ObjAdd, Prob,
     $                   a, ha, ka, bl, bu, c, Names,
     $                   helast, hs, xs, pi, rc,
     $                   inform, mincw, miniw, minrw,
     $                   nS, nInf, sInf, Obj,
     $                   cu, lencu, iu, leniu, ru, lenru,
     $                   cw, lencw, iw, leniw, rw, lenrw )

      external            qpHx
      character*(*)       start
      character*8         Prob
      character*8         Names(nName)
      integer             m, n, lena, nName, lenc, ncolH
      integer             iObj, nS, nInf
      integer             inform, mincw, miniw, minrw
      integer             ha(lena), helast(n+m), hs(n+m)
      integer             ka(n+1)
      double precision    ObjAdd, sInf, Obj
      double precision    a(lena), bl(n+m), bu(n+m), xs(n+m)
      double precision    c(*)
      double precision    pi(m), rc(n+m)

      integer             lencu, lencw, leniu, lenru, leniw, lenrw
      character*8         cu(lencu), cw(lencw)
      integer             iu(leniu), iw(leniw)
      double precision    ru(lenru), rw(lenrw)
```

On entry:

`start` is a character string that specifies how a starting basis (and certain other items) are to be obtained.

- `'Cold'` requests that the CRASH procedure be used to choose an initial basis, unless a basis file is provided via `OLD BASIS`, `INSERT` or `LOAD` in the Specs file.
- `'Basis file'` is the same as `start = 'Cold'` but is more meaningful when a basis file is given.
- `'Warm'` means that a basis is already defined in `hs` (probably from an earlier call).

`m` is m, the number of general inequalities (`m` > 0). This is the number of rows in the constraint matrix A.

Note that A must have at least one row. If your problem has no constraints, or only upper and lower bounds on the variables, then you must include a dummy row with

sufficiently wide upper and lower bounds. See the discussion of the parameters a, ha and ka below.

n is the number of variables, excluding slacks (n > 0). This is the number of columns in A.

lena is the number of nonzero entries in A (lena > 0).

nName is the number of column and row names provided in the character array Names. If nName = 1, there are *no* names. Generic names will be used in the printed solution. Otherwise, nName = $n + m$ and all names must be provided.

lenc is the number of elements in the constant objective vector c (lenc $\geq$ 0).

ncolH is the number of leading nonzero columns of the QP Hessian (ncolH $\geq$ 0).

If ncolH = 0, there is no quadratic term, and the problem is an FP or LP problem. In this case you must provide a dummy subroutine qpHx or use the subroutine nullHx provided in the SQOPT distribution.

If ncolH > 0, you must provide your own version of qpHx to compute the matrix-vector product Hx.

iObj says which row (if any) of A is a free row containing a linear objective vector c ($0 \leq$ iObj $\leq m$). If there is no such vector, iObj = 0.

ObjAdd is the constant f added to the objective for printing purposes. Typically ObjAdd = 0.0d+0.

Prob is an 8-character name for the problem. Prob is used in the printed solution and in some routines that output BASIS files. A blank name may be used.

a(lena), ha(lena), ka(n+1) define the nonzero elements of the constraint matrix A. The nonzeros are stored column-wise. A pair of values (a(k),ha(k)) contains a matrix element and its corresponding row index, and the array ka(*) is a set of pointers to the beginning of each column of A within a(*) and ha(*). Thus for $j = 1 : n$, the entries of column j are held in a($k : l$) and their corresponding row indices are in ha($k : l$), where k = ka(j) and l = ka($j + 1$) $- 1$,

Note: every element of a(*) must be assigned a value in the calling program.

1. It is *essential* that ka(1) = 1 and ka($n + 1$) = lena + 1.
2. The row indices ha(k) for a column may be in any order.
3. If lenc > 0, the first lenc columns of a and ha belong to variables corresponding to the constant objective term c.
4. If the problem has a quadratic objective, the first ncolH columns of a and ha belong to variables corresponding to the nonzero block of the QP Hessian. Subroutine qpHx knows about these variables.
5. If your problem has no constraints, or just bounds on the variables, you may include a dummy "free" row with a single (zero) element by setting a(1) = 0.0, ha(1) = 1, ka(1) = 1, and ka(j) = 2 for $j = 2 : n + 1$. This row is made "free" by setting its bounds to be bl($n + 1$) = $-$bigbnd and bu($n + 1$) = bigbnd, where bigbnd is typically 1.0e+20 (see next paragraph).

bl(n+m) contains the lower bounds on the variables and slacks (x, s).

The first n entries of bl, bu, hs and xs refer to the variables x. The last m entries refer to the slacks s.

To specify a non-existent lower bound ($l_j = -\infty$), set bl(j) $\leq$ -BigBnd, where BigBnd is the Infinite Bound, whose default value is 10^{20}.

To fix the jth variable (say $x_j = \beta$, where $|\beta| <$ BigBnd), set bl(j) = bu(j) = β.

To make the jth constraint an *equality* constraint (say $s_j = \beta$, where $|\beta| <$ BigBnd), set bl($n+j$) = bu($n+j$) = β.

bu(n+m) contains the upper bounds on (x, s). 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.

c(lenc) contains the explicit objective vector (if any). If the problem is of type FP, or if lenc = 0, then c is not referenced. (In that case, c may be dimensioned (1), or it could be any convenient array.)

Names(nName) sometimes contains 8-character names for the variables and constraints. If nName = 1, Names is not used. The printed solution will use generic names for the columns and row. If nName = $n+m$, Names(j) should contain the 8-character name of the jth variable ($j = 1 : n+m$). If $j = n+i$, the jth variable is the ith row.

helast(n+m) defines which variables are to be treated as being elastic in elastic mode. The allowed values of helast are helast(j) = 0, 1, 2, 3, which have the following meaning:

helast(j)	Status in elastic mode
0	variable j is non-elastic and cannot be infeasible
1	variable j can violate its lower bound
2	variable j can violate its upper bound
3	variable j can violate either its lower or upper bound

helast need not be assigned if Elasticmode = 0.

hs(n+m) sometimes contains a set of initial states for each variable x, or for each variable and slack (x, s). See the following discussion of xs.

xs(n+m) sometimes contains a set of initial values for x or (x, s).

1. If start = 'Cold' or 'Basis file', and a BASIS file of some sort is to be input (an OLD BASIS file, INSERT file or LOAD file), then hs and xs need not be set at all.
2. Otherwise, hs($1 : n$) and xs($1 : n$) must be defined for a Cold start. If nothing special is known about the problem, or if there is no wish to provide special information, you may set hs(j) = 0, xs(j) = 0.0 for all $j = 1 : n$. All variables will be eligible for the initial basis.

   Less trivially, to say that the optimal value of variable j will probably be equal to one of its bounds, set hs(j) = 4 and xs(j) = bl(j) or hs(j) = 5 and xs(j) = bu(j) as appropriate.
3. For Cold starts with no basis file, a CRASH procedure is used to select an initial basis. The initial basis matrix will be triangular (ignoring certain small entries in each column). The values hs(j) = $0, 1, 2, 3, 4, 5$ have the following meaning:

hs(j)	State of variable j during CRASH
$\{0, 1, 3\}$	Eligible for the basis. 3 is given preference
$\{2, 4, 5\}$	Ignored

After CRASH, columns for which hs(j) = 2 are made superbasic. Other entries not selected for the basis are made nonbasic at the value xs(j) if bl(j) $\leq$ xs(j) $\leq$ bu(j), or at the value bl(j) or bu(j) closest to xs(j). See the description of hs below (on exit).

4. For Warm starts, all of hs($1 : n+m$) must be 0, 1, 2 or 3 (probably from some previous call) and all of xs($1 : n+m$) must have values.

nS need not be specified for Cold starts, but should retain its value from a previous call when a Warm start is used.

qpHx is the name of the subroutine that defines the product of H and a given vector x when solving a quadratic program. A valid subroutine name must always be provided. If you are solving a quadratic problem, your qpHx must multiply your specific H by a given vector x. (This is the only way that SQOPT accesses the matrix H in the objective function.) For a detailed description of qpHx, see §4.

For problems of type FP and LP, qpHx is never called by sqopt, and you can either provide your own empty qpHx or use the dummy routine nullHx provided with the SQOPT distribution.

cu(lencu), iu(leniu), ru(lenru) are 8-character, integer and real arrays of user workspace. They may be used to pass data or workspace to your function routine qpHx (which has the same parameters). They are not touched by sqopt.

If the function routine doesn't reference these parameters, you may use any arrays of the appropriate type, such as cw, iw, rw (see next paragraph). Alternatively, you should use the latter arrays if qpHx needs to access sqopt's workspace.

cw(lencw), iw(leniw), rw(lenrw) are 8-character, integer and real arrays of workspace for sqopt.

lencw, leniw, lenrw must all be at least 500. In general, lencw = 500 is appropriate but leniw and lenrw should be as large as possible because it is uncertain how much storage will be needed for the basis factors. As an estimate, leniw should be about $10(m+n)$ or larger, and lenrw should be about $20(m+n)$ or larger.

Appropriate values may be obtained from a preliminary run with lencw = leniw = lenrw = 500. If Print level is positive, the required amounts of workspace are printed before sqopt terminates with inform = 42, 43 or 44. The values are returned in mincw, miniw and minrw.

On exit:

hs gives the state of the final xs. The elements of hs have the following meaning:

hs(j)	State of variable j	Usual value of xs(j)
0	nonbasic	bl(j)
1	nonbasic	bu(j)
2	superbasic	Between bl(j) and bu(j)
3	basic	ditto

Basic and superbasic variables may be outside their bounds by as much as the value of the optional parameter Feasibility tolerance. Note that if scaling is specified, the Feasibility tolerance applies to the variables of the *scaled* problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled. Check the "Primal infeasibility" printed after the EXIT message.

Very occasionally some nonbasic variables may be outside their bounds by as much as the `Feasibility tolerance`, and there may be some nonbasics for which `xs`(j) lies strictly between its bounds.

If `nInf` > 0, some basic and superbasic variables may be outside their bounds by an arbitrary amount (bounded by `sInf` if scaling was not used).

`xs(n+m)` is the final variables and slacks (x, s).

`pi(m)` contains the dual variables π (a set of Lagrange multipliers (shadow prices) for the general constraints).

`rc(n+m)` is a vector of reduced costs, $g-(\ A\ \ -I\ )^T\pi$, where g is the gradient of the objective if `xs` is feasible (or the gradient of the Phase-1 objective otherwise). The last m entries are π.

`inform` reports the result of the call to `sqopt`. Possible values are:

0	Optimal solution found. (The reduced gradients are optimal, and `xs` satisfies the constraints to the accuracy requested.
1	The problem is infeasible.
2	The problem is unbounded (or badly scaled).
3	Too many iterations.
4	The QP Hessian H appears to be indefinite (the QP is non-convex).
5	The `Superbasics limit` is too small.
6	A weak solution has been found—i.e., the solution is not unique.
10	Numerical error in trying to satisfy the constraints $Ax - s = 0$. The basis is very ill-conditioned.
20	Not enough storage for the basis factorization.
21	Error in basis package.
22	The basis is singular after several attempts to factorize it (and add slacks where necessary).
30	An OLD BASIS file had dimensions that did not match the current problem.
32	System error. Wrong number of basic variables.
42	Not enough 8-character workspace to solve the problem.
43	Not enough integer workspace to solve the problem.
44	Not enough real workspace to solve the problem.

`mincw, miniw, minrw` say how much character, integer and real storage is needed to solve the problem. If SQOPT terminates because of insufficient storage (`inform` = 42, 43 or 44), these values may be used to define better values of `lencw`, `leniw` or `lenrw`.

If `inform` = 42, the work array `cw`(`lencw`) was too small. `sqopt` may be called again with `lencw` suitably larger than `mincw`.

If `inform` = 43 or 44, the work arrays `iw`(`leniw`) or `rw`(`lenrw`) are too small. `sqopt` may be called again with `leniw` or `lenrw` suitably larger than `miniw` or `minrw`. (The bigger the better, since it is not certain how much storage the basis factors need.)

`nS` is the final number of superbasics.

`nInf` is the number of infeasibilities.

`sInf` is the sum of infeasibilities.

`Obj` is the value of the objective function. If `nInf` $= 0$, `Obj` includes the quadratic objective if any. If `nInf` > 0, `Obj` is just the linear objective if any.

4. User-Supplied Subroutines

For QP problems, you must provide a subroutine that defines products of the form Hx for given vectors x. This is the only way in which SQOPT accesses the the matrix H in the objective function. Your subroutine is input via the parameter `qpHx` (an external subroutine).

(For FP and LP problems, `qpHx` is never called by SQOPT, and hence you can either provide your own dummy `qpHx` or use the "empty" routine `nullHx` provided in the SQOPT distribution.

Your version of `qpHx` should function as follows.

```
 subroutine qpHx  ( ncolH, x, Hx, nState,
$                   cu, lencu, iu, leniu, ru, lenru )

 integer            ncolH, nState
 double precision   x(ncolH), Hx(ncolH)

 integer            lencu, leniu, lenru
 character*8        cu(lencu)
 integer            iu(leniu)
 real               ru(lenru)
```

On entry:

`ncolH` is the same as the input parameter of `sqopt` ($0 \leq$ `ncolH` $\leq$ `n`). `ncolH` must not be altered within `qpHx`. Similarly for the parameters `iu`, `leniu`, `ru` and `lenru`.

If some of the variables enter the objective function linearly, then H will have zero rows and columns. In this case, it is most efficient to order the variables so that the nonlinear variables appear first. For example, if y contains the nonlinear variables of x, and $x = (y, z)$, then

$$Hx = \begin{pmatrix} H_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} H_1 y \\ 0 \end{pmatrix}.$$

`x` contains a vector x such that the product Hx should be returned in `Hx`.

`nState` allows you to save computation time if certain data must be read or calculated only once.

If `nState` $= 0$, there is nothing special about the current call to `qpHx`.

If `nState` $= 1$, SQOPT is calling your subroutine for the first time. Some data may need to be input or computed and saved in local or `common` storage.

If `nState` ≥ 2, SQOPT is calling your subroutine for the *last* time. You may wish to perform some additional computation on the final solution.

In general, the last call is made with `nState` $= 2+$`ierror`, where `ierr` indicates the status of the final solution. In particular, if `nState` $= 2$, the current `x` is *optimal*; if `nState` $= 3$, the problem appears to be infeasible; if `nState` $= 4$, the problem appears to be unbounded; and if `nState` $= 5$, the iterations limit was reached.

`cu(lencu)`, `iu(leniu)`, `ru(lenru)` are character, integer and real arrays that can be used to pass user-defined auxiliary information into `qpHx`. The arrays `cu`, `iu` and `ru` are not touched by `sqopt` and can be used to retain information between calls of `qpHx`.

In certain applications, the objective may depend on the values of certain internal SQOPT variables stored in the arrays `cw`, `iw` and `rw`. In this case, `sqopt` should be called with `cw`, `iw` and `rw` as actual arguments for `cu`, `iu` and `ru`, thereby making `cw`, `iw` and `rw` accessible to `qpHx`.

If you require user work space in this situation, elements `501:maxcw`, `501:maxrw` and `501:maxiw` of `cw`, `rw` and `iw` are set aside for this purpose. (See the definition of the optional parameters `User character workspace`, `User real workspace` and `User integer workspace` in §5.6.

If you do not require workspace to be passed into `qpHx`, the `sqopt` work arrays `cw`, `iw` and `rw` can be used for `cu`, `iu` and `ru`.

On exit:

`Hx` should contain the product Hx for the vector stored in `x`.

5. The SPECS file

The performance of SQOPT is controlled by a number of parameters or "options". Each option has a default value that should be appropriate for most problems. (The defaults are given in the next section.) For special situations it is possible to specify non-standard values for some or all of the options, using data in the following general form:

```
Begin SQOPT options
   Iterations limit             500
   Feasibility tolerance  1.0e-7
   Scale all variables
End SQOPT options
```

We shall call such data a SPECS file, since it specifies various options. The file starts with the keyword `Begin` and ends with `End`. The file is in free format. Each line specifies a single option, using one or more items as follows:

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

The items may be entered in upper or lower case or a mixture of both. Some of the keywords have synonyms, and certain abbreviations are allowed, as long as there is no ambiguity. Blank lines and comments may be used to improve readability. A comment begins with an asterisk (`*`), which may appear anywhere on a line. All subsequent characters on the line are ignored.

Although most options take default values, some of them must be specified; for example, the number of nonlinear variables if there are any.

It may be useful to include a comment on the first (`Begin`) line of the file. This line is echoed to the SUMMARY file, and appears on the screen in an interactive environment.

Most of the options described in the next section should be left at their default values for any given model. If experimentation is necessary, we recommend changing just one option at a time.

5.1. SPECS File Checklist and Defaults

The following example SPECS file shows all valid *keywords* and their *default values*. The keywords are grouped according to the function they perform.

Some of the default values depend on ϵ, the relative precision of the machine being used. The values given here correspond to double-precision arithmetic on most current machines ($\epsilon \approx 2.22 \times 10^{-16}$). Similar values would apply to any machine having about 15 decimal digits of precision.

```
BEGIN checklist of SPECS file parameters and their default values
* Printing
  Print level                   1       * 1-line iteration log
  Print   file                 15       *
  Summary file                  6       * typically the screen
  Print   frequency             1       * iterations log on PRINT file
  Summary frequency             1       * iterations log on SUMMARY file
  Solution                      Yes     * on the PRINT file
* Suppress options listing              * default: options are listed

* Convergence Tolerances
  Feasibility tolerance         1.0e-6  * for satisfying the simple bounds
  Optimality  tolerance         1.0e-6  * target value for reduced gradients

* Scaling
  Scale option                  2       * All constraints and variables
  Scale tolerance               0.9     *
* Scale Print                           * default: scales are not printed

* Other Tolerances
  Crash tolerance               0.1     *
  LU factor tolerance           10.0    * limits size of multipliers in L
  LU update tolerance           10.0    * the same during updates
  LU singularity tolerance      2.0e-6  *
  Pivot tolerance               3.7e-11 * eps^(2/3)

* LP/QP problems
  Crash option                  0       * all slack initial basis
  Elastic weight                1.0     * used only during elastic mode
  Iterations limit              10000   * or m if that is more
  Minimize                              * (opposite of Maximize)
  Partial price                 1       * 10 for large LPs
  Superbasics limit             500     * or n1 + 1 if that is less
  Reduced Hessian dimension     500     * or Superbasics limit
  Unbounded step size           1.0e+18 *

* Infeasible problems
  Elastic weight                100     * used only during elastic mode
  Elastic mode                    1     * use elastic mode when infeasible
  Elastic Objective               2     * infinite weight on the elastics

* Frequencies
  Check frequency               60      * test row residuals ||Ax - s||
  Expand frequency              10000   * for anti-cycling procedure
  Factorization frequency       100     *
  Save frequency                100     * save basis map

* BASIS files
  OLD BASIS file                0       * input basis map
  NEW BASIS file                0       * output basis map
  BACKUP BASIS file             0       * output basis map
  INSERT file                   0       * input in industry format
  PUNCH file                    0       * output INSERT data
  LOAD file                     0       * input names and values
  DUMP file                     0       * output LOAD data
  SOLUTION file                 0       * different from printed solution
```

```
* Partitions of cw, iw, rw
  Total character workspace      lencw   *
  Total integer   workspace      leniw   *
  Total real      workspace      lenrw   *
  User  character workspace      500     *
  User  integer   workspace      500     *
  User  real      workspace      500     *
End of SPECS file checklist
```

5.2. Subroutine sqInit

Subroutine `sqInit` must be called before any other SQOPT routines. It defines the PRINT and SUMMARY files, prints a title on both files, and sets all user options to be undefined. (`sqopt` will later check the options and set undefined ones to default values.)

```
      subroutine sqInit( iPrint, iSumm,
     $                   cw, lencw, iw, leniw, rw, lenrw )

      integer            iPrint, iSumm
      integer            lencw, leniw, lenrw
      character*8        cw(lencw)
      integer            iw(leniw)
      double precision   rw(lenrw)
```

On entry:

`iPrint` defines a unit number for the PRINT file. Typically `iPrint` = 9.

On some systems, the file may need to be opened before `sqInit` is called. If `iPrint` ≤ 0, there will be no PRINT file output.

`iSumm` defines a unit number for the SUMMARY file. Typically `iSumm` = 6. (In an interactive environment, this usually denotes the screen.)

On some systems, the file may need to be opened before `sqInit` is called. If `iSumm` $\leq$ 0, there will be no SUMMARY file output.

`cw(lencw), iw(leniw), rw(lenrw)` must be the same arrays that are passed to `sqopt` and other routines. They must all have length 500 or more.

On exit:

Some elements of `cw`, `iw`, `rw` are given values to indicate that most optional parameters are undefined.

5.3. Subroutine sqSpec

Subroutine `sqSpec` may be called to input a SPECS file (to specify options for a subsequent call of `sqopt`).

```
      subroutine sqSpec( iSpecs, inform,
     $                   cw, lencw, iw, leniw, rw, lenrw )

      integer            iSpecs, inform
      integer            lencw, leniw, lenrw
      character*8        cw(lencw)
      integer            iw(leniw)
      double precision   rw(lenrw)
```

On entry:

`iSpecs` is a unit number for the SPECS file (`iSpecs` > 0). Typically `iSpecs` $= 4$.

On some systems, the file may need to be opened before `sqSpec` is called.

On exit:

`cw(lencw), iw(leniw), rw(lenrw)` contain the specified options.

`inform` is 0 if the SPECS file was successfully read. Otherwise, it returns the number of errors encountered.

5.4. Subroutines sqset, sqseti, sqsetr

These routines specify a single option that might otherwise be defined in one line of a SPECS file.

```
      subroutine sqset ( buffer,         iPrint, iSumm, inform,
     $                   cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqseti( buffer, ivalue, iPrint, iSumm, inform,
     $                   cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqsetr( buffer, rvalue, iPrint, iSumm, inform,
     $                   cw, lencw, iw, leniw, rw, lenrw )

      character*(*)      buffer
      integer            ivalue, iPrint, iSumm, inform
      double precision   rvalue
      integer            lencw, leniw, lenrw
      character*8        cw(lencw)
      integer            iw(leniw)
      double precision   rw(lenrw)
```

On entry:

`buffer` is a string to be decoded. Use `sqset` if the string contains all relevant data. For example, if the value 1000 is known at compile time, say

```
call sqset ( 'Iterations 1000',    iPrint, iSumm, inform, ... )
```

Restriction: `len(buffer)` ≤ 72 (`sqset`) or ≤ 55 (`sqseti` and `sqsetr`).

`ivalue` is an integer value associated with the keyword in `buffer`. Use `sqseti` if it is convenient to define the value at run time. For example, the following allows the iterations limit to be computed:

```
itnlim = 1000
if (m .gt. 500) itnlim = 8000
call sqseti( 'Iterations', itnlim, iPrint, iSumm, inform, ... )
```

`rvalue` is a real value associated with the keyword in `buffer`. The following illustrates how the LU stability tolerance could be defined at run time:

```
factol = 100.0d+0
if ( illcon ) factol = 5.0d+0
call sqsetr( 'LU factor tol', factol, iPrint, iSumm, inform, ...)
```

`iPrint` is a file number for printing each line of data, along with any error messages. `iPrint` $= 0$ suppresses this output.

`iSumm` is a file number for printing any error messages. `iSumm` $= 0$ suppresses this output.

`inform` should be 0 for the first call to the `sqset` routines.

On exit:

`inform` is the number of errors encountered so far.

`cw(lencw), iw(leniw), rw(lenrw)` record the specified option.

5.5. Subroutines sqgetc, sqgeti, sqgetr

These routines obtain the current value of a single option.

```
      subroutine sqgetc( buffer, cvalue, inform,
     $                   cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqgeti( buffer, ivalue, inform,
     $                   cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqgetr( buffer, rvalue, inform,
     $                   cw, lencw, iw, leniw, rw, lenrw )

      character*(*)      buffer
      character*8        cvalue
      integer            ivalue, inform
      double precision   rvalue
      integer            lencw, leniw, lenrw
      character*8        cw(lencw)
      integer            iw(leniw)
      double precision   rw(lenrw)
```

On entry:

`buffer` is a string to be decoded. Restriction: `len(buffer)` ≤ 72.

`inform` should be 0 for the first call to the `sqget` routines.

On exit:

`cvalue` is a string associated with the keyword in `buffer`. Use `sqgetc` to obtain the names ...
For example, if ..., say

```
call sqgetc( 'Bounds',    inform, ... )
```

`ivalue` is an integer value associated with the keyword in `buffer`. Example:

```
call sqgeti( 'Iterations limit', itnlim, inform, ... )
```

`rvalue` is a real value associated with the keyword in `buffer`. Example:

```
call sqgetr( 'LU factor tol', factol, inform, ...)
```

`inform` is the number of errors encountered so far.

`cw(lencw), iw(leniw), rw(lenrw)` contain the required option value.

5.6. Description of the optional parameters

The following is an alphabetical list of the options that may appear in the SPECS file, and a description of their effect.

`Backup Basis file` i Default = 0

This is intended as a safeguard against losing the results of a long run. Suppose that a NEW BASIS file is being saved every 100 iterations, and that SQOPT is about to save such a basis at iteration 2000. It is conceivable that the run may be interrupted during the next few milliseconds (in the middle of the save). In this case the basis file will be corrupted and the run will have been essentially wasted.

To eliminate this risk, both a NEW BASIS file and a BACKUP BASIS file may be specified. The following would be suitable for the above example:

```
OLD BASIS file      11     (or 0)
BACKUP BASIS file   11
NEW BASIS file      12
Save frequency     100
```

The current basis will then be saved every 100 iterations, first on file 12 and then immediately on file 11. If the run is interrupted at iteration 2000 during the save on file 12, there will still be a usable basis on file 11 (corresponding to iteration 1900).

Note that a NEW BASIS will be saved at the end of a run if it terminates normally, but there is no need for a further BACKUP BASIS. In the above example, if an optimum solution is found at iteration 2050 (or if the iteration limit is 2050), the final basis on file 12 will correspond to iteration 2050, but the last basis saved on file 11 will be the one for iteration 2000.

`Check frequency` k Default = 60

Every kth iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general constraints. The constraints are of the form $Ax - s = 0$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = s - Ax$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

`Check frequency 1` is useful for debugging purposes, but otherwise this option should not be needed.

`Crash option` i Default = 0
`Crash tolerance` r Default = 0.1

Except on restarts, a CRASH procedure is used to select an initial basis from certain rows and columns of the constraint matrix $(A \quad -I)$. The `Crash option` i determines which rows and columns of A are eligible initially, and how many times CRASH is called. Columns of $-I$ are used to pad the basis where necessary.

i	*Meaning*
0	The initial basis contains only slack variables: $B = I$.
1	CRASH is called once, looking for a triangular basis in all rows and columns of the matrix A.

2 CRASH is called once, looking for a triangular basis in linear rows.

3 CRASH is called twice. The two calls treat *linear equalities* and *linear inequalities* separately.

If $i \geq 1$, certain slacks on inequality rows are selected for the basis first. (If $i \geq 2$, numerical values are used to exclude slacks that are close to a bound.) CRASH then makes several passes through the columns of A, searching for a basis matrix that is essentially triangular. A column is assigned to "pivot" on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For remaining unassigned rows, slack variables are inserted to complete the basis.

The `Crash tolerance` r allows the starting procedure CRASH to ignore certain "small" nonzeros in each column of A. If $a_{\max}$ is the largest element in column j, other nonzeros a_{ij} in the column are ignored if $|a_{ij}| \leq a_{\max} \times r$. (To be meaningful, r should be in the range $0 \leq r < 1$.)

When $r > 0.0$, the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first m columns of A form the matrix shown under `LU factor tolerance`; i.e., a tridiagonal matrix with entries -1, 4, -1. To help CRASH choose all m columns for the initial basis, we would specify `Crash tolerance` r for some value of $r > 1/4$.

`Dump File` i Default = 0

If $i > 0$, the last solution obtained will be output to the file with unit number i in the format described in §9.3. The file will usually have been output previously as a `LOAD` file.

`Elastic mode` i Default = 1

This parameter determines if (and when) elastic mode is to be started. Three elastic modes are available as follows:

i	*Meaning*
0	Elastic mode is never invoked. SQOPT will terminate as soon as infeasibility is detected. There may be other points with significantly smaller sums of infeasibilities.
1	Elastic mode is invoked only if the constraints are found to be infeasible (the default). If the constraints are infeasible, continue in elastic mode with the composite objective determined by the values of `Elastic objective` and `Elastic weight`.
2	The iterations start and remain in elastic mode. This option allows you to minimize the composite objective function directly without first performing phase-1 iterations. The success of this option will depend critically on your choice of `Elastic weight`. If `Elastic weight` is sufficiently large and the constraints are feasible, the minimizer of the composite objective and the solution of the original problem are identical. However, if the `Elastic weight` is not sufficiently large, the minimizer of the composite function may be infeasible, even though a feasible point for the constraints may exist.

`Elastic objective` *i* Default = 2

This option determines the form of the composite objective. Three types of composite objectives are available.

i	*Meaning*
0	Include only the true objective $q(x)$ in the composite objective. This option sets $\gamma = 0$ in the composite objective and will allow SQOPT to ignore the elastic bounds and find a solution that minimzes q subject to the nonelastic constraints. This option is useful if there are some "soft" constraints that you would like to ignore if the constraints are infeasible.
1	Use a composite objective defined with γ determined by the value of `Elastic weight`. This value is intended to be used in conjunction with `Elastic mode` = 2.
2	Include only the elastic variables in the composite objective. The elastics are weighted by $\gamma = 1$. This choice minimizes the violations of the elastic variable at the expense of possibly increasing the true objective. This option can be used to find a point that minimizes the sum of the violations of a subset of constraints determined by the parameter `helast`.

`Elastic weight` *r* Default = 1.0

This keyword defines the value of γ in the composite objective.

- At each iteration of elastic mode, the composite objective is defined to be

$$\text{minimize} \quad \sigma q(x) + r(\text{sum of infeasibilities}),$$

  where $\sigma = 1$ for `Minimize`, $\sigma = -1$ for `Maximize`, and q is the current objective value.

- Note that the effect of r is *not* disabled once a feasible iterate is obtained.

`Expand frequency` *i* Default = 10000

This option is part of the EXPAND anti-cycling procedure [5] designed to make progress even on highly degenerate problems.

The strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the `Feasibility tolerance` is δ. Over a period of i iterations, the tolerance actually used by SQOPT increases from 0.5δ to δ (in steps of $0.5\delta/i$).

Increasing i helps reduce the number of slightly infeasible nonbasic basic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see `Pivot tolerance`).

`Factorization Frequency` *k* Default = 100 (LP) or 50 (QP)

At most k basis changes will occur between factorizations of the basis matrix.

- With linear programs, the basis factors are usually updated every iteration. The default k is reasonable for typical problems. Higher values up to k = `100` (say) may be more efficient on problems that are extremely sparse and well scaled.

- When the objective function is quadratic, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the `Check frequency`) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of k updates is reached.

`Feasibility tolerance` t Default = `1.0E-6`

A *feasible problem* is one in which all variables satisfy their upper and lower bounds to within the absolute tolerance t. (This includes slack variables. Hence, the general constraints are also satisfied to within t.)

- SQOPT attempts to find a feasible point for the non-elastic constraints before optimizing the objective. If the sum of the infeasibilities of these constraints cannot be reduced to zero, the problem is declared INFEASIBLE. If `sInf` is quite small, it may be appropriate to raise t by a factor of 10 or 100. Otherwise, some error in the data should be suspected.
- *Note*: if `sInf` is not small and you have not asked SQOPT to minimize the violations of the elastic variables (i.e., you have not specified `Elastic objective` = 2, there may be other points that have a *significantly smaller sum of infeasibilities*. SQOPT will not attempt to find the solution that minimizes the sum unless `Elastic objective` = 2.
- If `scale` is used, feasibility is defined in terms of the scaled problem (since it is then more likely to be meaningful).

`Insert File` f Default = 0

If $f > 0$, this references a file containing basis information in the format of §9.2.

- The file will usually have been output previously as a `punch` file.
- The file will not be accessed if an `old basis` file is specified.

`Infinite Bound Size` r Default = `1.0E+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 $r \leq 0$, the default value is used.

`Iterations Limit` k Default = $3 * m$

This is the maximum number of iterations of the simplex method or the QP reduced-gradient algorithm allowed.

- `Itns` is an alternative keyword.
- $k = 0$ is valid. Both feasibility and optimality are checked.

`Load File` f Default = 0

If $f > 0$, this references a file containing basis information in the format of §9.3.

- The file will usually have been output previously as a `DUMP` file.
- The file will not be accessed if an `OLD BASIS` file or an `INSERT` file is specified.

`Log Frequency` k Default = 1
see `Print Frequency`

`LU factor tolerance` r_1 Default = 100.0
`LU update tolerance` r_2 Default = 10.0

These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy $r_1, r_2 \geq 1.0$. The matrix L is a product of matrices of the form

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix},$$

where the multipliers μ satisfy $|\mu| \leq r_i$. Smaller values of r_i favor stability, while larger values favor sparsity. The default values usually strike a good compromise.

- For large and relatively dense problems, $r_1 = 10.0$ or 5.0 (say) may give a useful improvement in stability without impairing sparsity to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to reduce r_1 and/or r_2 in order to achieve stability. For example, if the columns of A include a submatrix of the form

$$\begin{pmatrix} 4 & -1 & & & & \\ -1 & 4 & -1 & & & \\ & -1 & 4 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 4 & -1 \\ & & & & -1 & 4 \end{pmatrix},$$

  one should set both r_1 and r_2 to values in the range $1.0 \leq r_i < 4.0$.

`LU singularity tolerance` r_3 Default = $\epsilon^{2/3}$

This tolerance should satisfy $r_3 \leq \epsilon^{1/4} \approx 10^{-4}$. It helps guard against ill-conditioned basis matrices. When the LU factors of B are computed directly (not updated), the diagonal elements of U are tested as follows: if $|U_{jj}| \leq r_3$ or $|U_{jj}| < r_3 \max_i |U_{ij}|$, the j-th column of the basis is replaced by the corresponding slack variable. (Replacements are rare because the LU updating method is stable. They are most likely to occur during the first factorization.)

`LU density tolerance` r_1 Default = 0.6
`LU singularity tolerance` r_2 Default = $\epsilon^{2/3} \approx 10^{-11}$

The density tolerance r_1 is used during LU factorization of the basis matrix. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds r_1,

the Markowitz strategy for choosing pivots is altered to reduce the time spent searching for each remaining pivot. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.

The singularity tolerance r_2 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \leq r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the jth column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)

`Minimize` Default
`Maximize`

This specifies the required direction of optimization. It applies to both linear and quadratic terms in the objective.

`New Basis File` f Default = 0

If $f > 0$, a basis map will be saved on file f every kth iteration, where k is the `Save frequency`.

- The first card of the file will contain the word `PROCEEDING` if the run is still in progress.
- If $f > 0$, a basis map will also be saved at the end of a run, with some other word indicating the final solution status.

`Old Basis File` f Default = 0

If $f > 0$, the starting point will be obtained from this file in the format of §9.1.

- The file will usually have been output previously as a NEW BASIS file.
- The file will not be acceptable if the number of rows or columns in the problem has been altered.

`Optimality tolerance` t Default = `1.0e-6`

This is used to judge the size of the reduced gradients $d_j = g_j - \pi^T a_j$, where g_j is the jth component of the gradient, a_j is the associated column of the constraint matrix $tmatA{-}I$, and π is the set of dual variables.

- By construction, the reduced gradients for basic variables are always zero. The problem will be declared optimal if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy

$$d_j/\|\pi\| \geq -t \quad \text{or} \quad d_j/\|\pi\| \leq t$$

  respectively, and if $|d_j|/\|\pi\| \leq t$ for superbasic variables.

- In the above tests, $\|\pi\|$ is a measure of the size of the dual variables. It is included to make the tests independent of a scale factor on the objective function.
- The quantity $\|\pi\|$ actually used is defined by

$$\|\pi\| = \max\{\sigma/\sqrt{m}, 1\}, \quad \text{where} \quad \sigma = \sum_{i=1}^{m} |\pi_i|,$$

  so that only *large* scale factors are allowed for.

- If the objective is scaled down to be very *small*, the optimality test reduces to comparing d_j against $0.01t$.

`Partial Price` i Default = 10 (LP) or 1 (QP)

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each "pricing" operation (when a nonbasic variable is selected to become superbasic).

- When $i = 1$, all columns of the constraint matrix $(A \quad -I)$ are searched.
- Otherwise, A and I are partitioned to give i roughly equal segments A_j, I_j ($j = 1$ to i). If the previous pricing search was successful on A_j, I_j, the next search begins on the segments A_{j+1}, I_{j+1}. (All subscripts here are modulo i.)
- If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments A_{j+2}, I_{j+2}, and so on.
- `Partial price` t (or $t/2$ or $t/3$) may be appropriate for time-stage models having t time periods.

`Pivot Tolerance` r Default = $\epsilon^{2/3}$

Broadly speaking, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular.

- When x changes to $x+\alpha p$ for some search direction p, a "ratio test" is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.
- For linear problems, elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r.
- It is common for two or more variables to reach a bound at essentially the same time. In such cases, the `Feasibility tolerance` (say t) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of t should therefore not be specified.
- To a lesser extent, the `Expand frequency` (say f) also provides some freedom to maximize the pivot element. Excessively *large* values of f should therefore not be specified.

`Print frequency` k Default = 1

One line of the iteration log will be printed every kth iteration. A value such as $k =$ `10` is suggested for those interested only in the final solution.

`Print level` k Default = 1

This controls the amount of printing produced by SQOPT as follows.

0 No output except error messages. If you want to suppress all output, set `Print file` = 0.

≥ 1 The set of selected options (including workspace limits), problem statistics, summary of the scaling procedure, information about the initial basis resulting from a CRASH or a BASIS file. A single line of output each iteration (controlled by `Print frequency`), and the exit condition with a summary of the final solution.

≥ 10 Basis factorization statistics.

`Punch file` f Default = 0

If $f > 0$, the final solution obtained will be output to file f in the format described in §9.2. For linear programs, this format is compatible with various commercial systems.

`Save frequency` k Default = 100

If a NEW BASIS file has been specified, a basis map describing the current solution will be saved on the appropriate file every kth iteration. A BACKUP BASIS file will also be saved if specified.

`Scale option` i Default = 2 (LP) or 1 (QP)
`Scale tolerance` r Default = 0.9
`Scale Print`

Three scale options are available as follows:

i	*Meaning*
0	No scaling. This is recommended if it is known that x and the constraint matrix never have very large elements (say, larger than 1000).
1	The constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer [1]). This will sometimes improve the performance of the solution procedures.
2	The constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side b or the solution x is large. This takes into account columns of $(A \quad -I)$ that are fixed or have positive lower bounds or negative upper bounds.

`Scale tolerance` affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_j = \max_i |a_{ij}| / \min_i |a_{ij}| \qquad (a_{ij} \neq 0).$$

If $\max_j \rho_j$ is less than r times its previous value, another scaling pass is performed to adjust the row and column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A. At most 10 passes are made.

`Scale Print` causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $\bar{a}_{ij} = a_{ij}c(j)/r(i)$, and the scaled bounds on the variables and slacks are $\bar{l}_j = l_j/c(j)$, $\bar{u}_j = u_j/c(j)$, where $c(j) \equiv r(j-n)$ if $j > n$.

`Solution`	`Yes`	
`Solution`	`No`	
`Solution`	`If Optimal, Infeasible, or Unbounded`	
`Solution File`	f	Default = 0

The first four options determine whether the final solution obtained is to be output to the PRINT file. The `FILE` option operates independently; if $f > 0$, the final solution will be output to file f (whether optimal or not).

- For the `YES`, `IF OPTIMAL`, and `IF ERROR` options, floating-point numbers are printed in `F16.5` format, and "infinite" bounds are denoted by the word `NONE`.
- For the `FILE` option, all numbers are printed in `1pe16.6` format, including "infinite" bounds which will have magnitude `1.000000E+20`.
- To see more significant digits in the printed solution, it will sometimes be useful to make f refer to the system PRINT file.

`Summary file`	f	Default = 6
`Summary frequency`	k	Default = 100

If $f > 0$, a brief log will be output to file f, including one line of information every kth iteration. In an interactive environment, it is useful to direct this output to the terminal, to allow a run to be monitored on-line. (If something looks wrong, the run can be manually terminated.) Further details are given in §6.7.

`Superbasics limit`	i	Default = $\min\{500, n_1 + 1\}$

This places a limit on the storage allocated for superbasic variables. Ideally, i should be set slightly larger than the "number of degrees of freedom" expected at an optimal solution.

For linear programs, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is no more than m, the number of general constraints.) The default value of i is therefore 1.

For quadratic problems, the number of degrees of freedom is often called the "number of independent variables".

- Normally, i need not be greater than `ncolH` + 1, where `ncolH` is the number of leading nonzero columns of H.
- For many problems, i may be considerably smaller than `ncolH`. This will save storage if `ncolH` is very large.
- This parameter also sets the `Reduced Hessian dimension`, unless the latter is specified explicitly (and conversely).

`Suppress Parameters`

Normally SQOPT prints the SPECS file as it is being read, and then prints a complete list of the available keywords and their final values. The `Suppress Parameters` option tells SQOPT not to print the full list.

`Total real workspace`	`maxrw`	Default = `lenrw`
`Total integer workspace`	`maxiw`	Default = `leniw`
`Total character workspace`	`maxcw`	Default = `lencw`
`User real workspace`	`maxru`	Default = 500
`User integer workspace`	`maxiu`	Default = 500
`User character workspace`	`maxcu`	Default = 500

These options may be used to confine SQOPT to certain parts of its workspace arrays `cw`, `iw`, `rw`. (The arrays are defined by the last six parameters of `sqopt`.)

The `Total ...` options place an *upper* limit on `sqopt`'s workspace. They may be useful on machines with virtual memory. For example, some systems allow a very large array `rw(lenrw)` to be declared at compile time with no overhead in saving the resulting object code. At run time, when various problems of different size are to be solved, it may be sensible to restrict SQOPT to the lower end of `rw` in order to reduce paging activity slightly. (However, SQOPT accesses storage contiguously wherever possible, so the benefit may be slight. In general it is far better to have too much storage than not enough.)

If `sqopt`'s "user" parameters `ru`, `lenru` happen to be the same as `rw`, `lenrw`, the nonlinear function routines will be free to use `ru`($maxrw + 1 : lenru$) for their own purpose. Similarly for the other work arrays.

The `User ...` options place a *lower* limit on `sqopt`'s workspace (not counting the first 500 elements). Again, if `sqopt`'s parameters `ru`, `lenru` happen to be the same as `rw`, `lenrw`, the function routines will be free to use `ru`($501 : maxru$) for their own purpose. Similarly for the other work arrays.

`Unbounded Step Size` $\alpha_{\max}$ Default = `1.0E+18`

This parameter is intended to detect unboundedness in a quadratic problem. (It may or may not achieve that purpose!) During a line search, q is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. if α exceeds $\alpha_{\max}$, iterations are terminated with the exit message `problem is unbounded`.

Note that unboundedness in x is best avoided by placing finite upper and lower bounds on the variables.

6. Output

The following information is output to the PRINT file during the solution of each problem referred to in the SPECS file.

- A listing of the relevant part of the SPECS file.
- A listing of the parameters that were or could have been set in the SPECS file.
- An estimate of the amount of working storage needed, compared to how much is available.
- Some statistics about the problem.
- The amount of storage available for the LU factorization of the basis matrix.
- A summary of the scaling procedure, if `Scale` was specified.
- Notes about the initial basis resulting from a CRASH procedure or a BASIS file.
- The iteration log.
- Basis factorization statistics.
- The EXIT condition and some statistics about the solution obtained.
- The printed solution, if requested.

The last four items are described in the following sections. Further brief output may be directed to the SUMMARY file, as discussed in §6.7.

6.1. The iteration Log

If `Print level` > 0, one line of information is output to the PRINT file every kth iteration, where k is the specified `Print frequency` (default $k =$ `1`). A heading is printed before the first such line following a basis factorization. The heading contains the items described below. In this description, a PRICE operation is defined to be the process by which one or more nonbasic variables are selected to become superbasic (in addition to those already in the superbasic set). The variable selected will be denoted by `jq`. If the problem is purely linear, variable `jq` will usually become basic immediately (unless it should happen to reach its opposite bound and return to the nonbasic set).

If `Partial price` is in effect, variable `jq` is selected from $A_{\texttt{pp}}$ or $I_{\texttt{pp}}$, the `pp`th segments of the constraint matrix $(A \quad -I)$.

Label	*Description*
`Itn`	The current iteration number.
`pp`	The Partial Price indicator. The variable selected by the last PRICE operation came from the `pp`th partition of A and $-I$. `pp` is set to zero when the basis is refactored.
`dj`	This is `dj`, the reduced cost (or reduced gradient) of the variable `jq` selected by PRICE at the start of the present iteration. Algebraically, `dj` is $d_j = g_j - \pi^T a_j$ for $j =$ `jq`, where g_j is the gradient of the current objective function, π is the vector of dual variables, and a_j is the jth column of the constraint matrix $(A \quad -I)$. Note that `dj` is the norm of the reduced-gradient vector at the start of the iteration, just after the PRICE operation.

`+SBS` The variable `jq` selected by PRICE to be added to the superbasic set.

`-SBS` The variable chosen to leave the set of superbasics. It has become basic if the entry under `-B` is nonzero; otherwise it has become nonbasic.

`-BS` The variable removed from the basis (if any) to become nonbasic.

`-B` The variable removed from the basis (if any) to swap with a slack variable made superbasic by the latest PRICE. The swap is done to ensure that there are no superbasic slacks.

`Step` The step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. If a variable is made superbasic during the current iteration (i.e., `+SBS` is positive), `Step` will be the step to the nearest bound. During phase 2, the step can be greater than one only if the reduced Hessian is not positive definite.

`Pivot` If column a_q replaces the rth column of the basis B, `Pivot` is the rth element of a vector y satisfying $By = a_q$. Wherever possible, `Step` is chosen to avoid extremely small values of `Pivot` (since they cause the basis to be nearly singular). In rare cases, it may be necessary to increase the `Pivot tolerance` to exclude very small elements of y from consideration during the computation of `Step`.

`L` The number of nonzeros representing the basis factor L. Immediately after a basis factorization $B = LU$, this is `lenL`, the number of subdiagonal elements in the columns of a lower triangular matrix. Further nonzeros are added to `L` when various columns of B are later replaced. (Thus, `L` increases monotonically.)

`U` The number of nonzeros in the basis factor U. Immediately after a basis factorization, this is `lenU`, the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix. As columns of B are replaced, the matrix U is maintained explicitly (in sparse form). The value of `U` may fluctuate up or down; in general it will tend to increase.

`ncp` The number of compressions required to recover storage in the data structure for U. This includes the number of compressions needed during the previous basis factorization. Normally `ncp` should increase very slowly. If not, the amount of integer and real workspace available to SQOPT should be increased by a significant amount. As a suggestion, the work arrays `iw(*)` and `rw(*)` should be extended by `L + U` elements.

`nInf` The number of infeasibilities *before* the present iteration. This number will not increase unless the iterations are in elastic mode.

`Sinf,Objective` If `nInf` > 0, this is `sInf`, the sum of infeasibilities before the present iteration. (It will usually decrease at each nonzero `Step`, but if `nInf` decreases by 2 or more, `sInf` may occasionally increase. However, in elastic mode, it will decrease monotonically.)

Otherwise, it is the value of the current objective function *after* the present iteration.

Note: If `Elastic mode` $= 2$, the heading is `Composite Obj`.

The following items are printed if the problem is a QP or if the superbasic set is non-empty (i.e., if the current solution is nonbasic).

Label *Description*

`Norm rg` This quantity is `rg`, the norm of the reduced-gradient vector at the start of the iteration. (It is the Euclidean norm of the vector with elements d_j for variables j in the superbasic set. During phase 2 this norm will be approximately zero after a unit step.

`nS` The current number of superbasic variables.

`Cond Hz` An estimate of the condition number of the reduced Hessian. It is the square of the ratio of the largest and smallest diagonals of the upper triangular matrix R. This constitutes a lower bound on the condition number of the reduced Hessian $R^T R$.

To guard against high values of `cond Hz`, attention should be given to the scaling of the variables and the constraints.

6.2. Basis Factorization Statistics

When `Print Level` ≥ 20 and `Print file` > 0, the following lines of intermediate printout (< 120 characters) are produced on the unit number specified by `Print file` whenever the matrix B or $B_S = (\,B \;\; S\,)^T$ is factorized. Gaussian elimination is used to compute an LU factorization of B or B_S, where PLP^T is a lower triangular matrix and PUQ is an upper triangular matrix for some permutation matrices P and Q. This factorization is stabilized in the manner described under `LU factor tolerance` in §5.6.

Label — *Description*

`Factorize` The number of factorizations since the start of the run.

`Demand` A code giving the reason for the present factorization.

Code	*Meaning*
0	First LU factorization.
1	Number of updates reached the value of the optional parameter `Factorization Frequency`.
2	Excessive non-zeros in updated factors.
7	Not enough storage to update factors.
10	Row residuals too large (see the description for `Check Frequency`).
11	Ill-conditioning has caused inconsistent results.

`Iteration` The current iteration number.

`Infeas` The number of infeasibilities at the *start* of the previous iteration.

`Objective` If `Infeas` > 0, this is the sum of infeasibilities at the start of the previous iteration.

If `Infeas` $= 0$, this is the value of the objective function *after* the previous iteration.

`Nonlinear` The number of nonlinear variables in the current basis B. (not printed if B_S is factorized).

`Linear` The number of linear variables in B. (not printed if B_S is factorized).

`Slacks` The number of slack variables in B. (not printed if B_S is factorized).

`Elems` The number of nonzero matrix elements in B. (not printed if B_S is factorized).

`Density` The percentage nonzero density of B, $100 \times$ `Elems`$/($`m`$\times$`m`$)$, where `m` is the number of rows in the problem (`m` = `Linear` + `Slacks`).

`Comprssns` The number of times the data structure holding the partially factored matrix needed to be compressed, to recover unused storage. Ideally this number should be zero. If it is more than 3 or 4, the amount of workspace available to SQOPT should be increased for efficiency.

`Merit` The average Markowitz merit count for the elements chosen to be the diagonals of PUQ. Each merit count is defined to be $(c-1)(r-1)$ where c and r are the number of nonzeros in the column and row containing the element at the time it is selected to be the next diagonal. `Merit` is the average of `m` such quantities. It gives an indication of how much work was required to preserve sparsity during the factorization.

`lenL` The number of nonzeros in L. On most machines, each nonzero is represented by one eight-byte `REAL` and two two-byte `integer` data types.

`lenU` The number of nonzeros in U. The storage required for each nonzero is the same as for the nonzeros of L.

`Increase` The percentage increase in the number of nonzeros in L and U relative to the number of nonzeros in B; i.e., $100 \times$ (`lenL` + `lenU` $-$ `Elems`)/`Elems`.

`m` is the number of rows in the problem. Note that `m` = `Ut` + `Lt` + `bp`.

`Ut` is the number of triangular rows of B at the top of U.

`d1` is the number of columns remaining when the density of the basis matrix being factorized reached 0.3.

`Lmax` The maximum subdiagonal element in the columns of L. This will be no larger than the `LU factor tolerance`.

`Bmax` The maximum nonzero element in B.

`Umax` The maximum nonzero element in U, excluding elements of B that remain in U unaltered. (For example, if a slack variable is in the basis, the corresponding row of B will become a row of U without alteration. Elements in such rows will not contribute to `Umax`. If the basis is strictly triangular, *none* of the elements of B will contribute, and `Umax` will be zero.)

Ideally, `Umax` should not be substantially larger than `Bmax`. If it is several orders of magnitude larger, it may be advisable to reduce the `LU factor tolerance` to some value nearer 1.0. (The default value is 10.0.)

`Umax` is not printed if B_S is factorized.

`Umin` The smallest *diagonal* element of PUQ in absolute magnitude.

`Growth` The ratio `Umax`/`Bmax`, which should not be too large (see above).

As long as `Lmax` is not large (say 10.0 or less), the ratio
max{`Bmax`, `Umax`} / `Umin` gives an estimate of the condition number of B. If this number is extremely large, the basis is nearly singular and some numerical difficulties could conceivably occur. (However, an effort is made to avoid near-singularity by using slacks to replace columns of B that would have made `Umin` extremely small. Messages are issued to this effect, and the modified basis is refactored.)

`Lt` is the number of triangular columns of B at the beginning of L.

`bp` is the size of the "bump" or block to be factorized nontrivially after the triangular rows and columns have been removed.

`d2` is the number of columns remaining when the density of the basis matrix being factorized reached 0.6.

6.3. Crash statistics

When `Print Level` ≥ 20 and `Print file` > 0, the following CRASH statistics (< 120 characters) are produced on the unit number specified by `Print file` whenever `Start` = 'Cold' (see §5.6). They refer to the number of columns selected by the CRASH procedure during each of several passes through A, whilst searching for a triangular basis matrix.

Label	*Description*
`Slacks`	is the number of slacks selected initially.
`Free cols`	is the number of free columns in the basis.
`Preferred`	is the number of "preferred" columns in the basis (i.e., `hs`$(j) = 3$ for some $j \leq n$).
`Unit`	is the number of unit columns in the basis.
`Double`	is the number of double columns in the basis.
`Triangle`	is the number of triangular columns in the basis.
`Pad`	is the number of slacks used to pad the basis.

6.4. EXIT conditions

For each problem in the SPECS file, a message of the form `EXIT -- message` is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

A number is associated with each message below. It is the final value assigned to the integer variable `inform`.

> The following messages arise when the SPECS file is found to contain no further problems.

`-2. EXIT -- input error. SQOPT encountered end-of-file or an endrun card before finding a specs file on unit nn`

The SPECS file may not be properly assigned. Its unit number `nn` is defined at compile time in subroutine `sqInit`, and normally it is the system card input stream.

Otherwise, the SPECS file may be empty, or cards containing the keywords `Skip` or `Endrun` may imply that all problems should be ignored (see §5).

`-1. ENDRUN`

This message is printed at the end of a run if SQOPT terminates of its own accord. Otherwise, the operating system will have intervened for one of many possible reasons (excess time, missing file, arithmetic error in the user routine, etc.).

> The following messages arise when optimization terminates gracefully. A solution exists, any of the BASIS files may be saved, and the solution will be printed and/or saved on the SOLUTION file if requested.

`0. EXIT -- optimal solution found`
The final point seems to be a unique solution of LCQP. This means that `x` is *feasible* (it satisfies the constraints to the accuracy requested by the `Feasibility tolerance`), the reduced gradient is negligible, the reduced costs are optimal, and R is nonsingular.

The input data for `sqopt` should always be checked (even if `sqopt` terminates with the value `inform` = 0!).

`1. EXIT -- the problem is infeasible`
Feasibility is measured with respect to the upper and lower bounds on the variables. The message tells us that among all the points satisfying the general constraints $Ax - s = 0$, there is apparently no point that satisfies the bounds on x and s. Violations as small as the `Feasibility tolerance` are ignored, but at least one component of x or s violates a bound by more than the tolerance.

Note: Although the objective function is the sum of infeasibilities (when `nInf` > 0), this sum will usually not have been *minimized* when SQOPT recognizes the situation and exits. There may exist other points that have a significantly lower sum of infeasibilities.

`2. EXIT -- the problem is unbounded (or badly scaled)`
For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. A message prior to the EXIT message will give the index of the nonbasic variable. Consider adding an upper or lower bound to the variable. Also, examine the constraints that have nonzeros in the associated column, to see if they have been formulated as intended.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the `Scale` option.

`3. EXIT -- iteration limit exceeded`
The `Iterations limit` was exceeded before the required solution could be found. Check the iteration log to be sure that progress was being made. If so, restart the run using a basis file that was saved (or *should* have been saved!) at the end of the run.

`4. EXIT -- QP Hessian appears to be indefinite`
The problem appears to be nonconvex and cannot be solved using this version of SQOPT. The matrix H cannot be positive semidefinite, i.e., there must exist a vector y such that $y^T H y < 0$.

You should check that `qpHx` is coded correctly and that all relevant components of `Hx` are assigned their correct values.

`5. EXIT -- the superbasics limit is too small: nnn`
The problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a PRICE operation is necessary to continue, but there are already `nnn` superbasics (and no room for any more).

In general, raise the `Superbasics limit` s by a reasonable amount, bearing in mind the storage needed for the reduced Hessian. (The `Hessian dimension` h will also increase to s unless specified otherwise, and the associated storage will be about $\frac{1}{2}s^2$ words.) In extreme cases you may have to set $h < s$ to conserve storage, but beware that the rate of convergence will probably fall off severely.

`6. EXIT -- weak solution found`
The final point is a *weak minimizer*. (The objective value is a global optimum, but it may be achieved by an infinite set of points x.)

This exit will occur when (i) the problem is feasible, (ii) the reduced gradient is negligible, (iii) the Lagrange multipliers are optimal, and (iv) the reduced Hessian is singular or there

are some very small multipliers. This exit cannot occur if H is positive definite (i.e., $q(x)$ is strictly convex).

`10. EXIT -- cannot satisfy the general constraints`
An LU factorization of the basis has just been obtained and used to recompute the basic variables x_B, given the present values of the superbasic and nonbasic variables. A single step of "iterative refinement" has also been applied to increase the accuracy of x_B. However, a row check has revealed that the resulting solution does not satisfy the current constraints $Ax - s = 0$ sufficiently well.

This probably means that the current basis is very ill-conditioned. Request the `Scale` option if there are any linear constraints and variables.

For certain highly structured basis matrices (notably those with band structure), a systematic growth may occur in the factor U. Consult the description of `Umax`, `Umin` and `Growth` in §6.2, and set the `LU factor tolerance` to 2.0 (or possibly even smaller, but not less than 1.0).

> If the following exits occur during the *first* basis factorization, the basic variables x_B will have certain default values that may not be particularly meaningful, and the dual vector π will be zero. BASIS files will be saved if requested, but certain values in the printed solution will not be meaningful. The problem will be terminated.

`20. EXIT -- not enough integer/real storage for the basis factors`
The main integer or real storage array `iw(*)` or `rw(*)` is apparently not large enough for this problem. The routine declaring `rw` should be recompiled with a larger dimension for `rw` or `iw`. The new value should also be assigned to `leniw` or `lenz`.

In some cases it may be sufficient to increase the specified `Workspace (user)`, if it is currently less than `Workspace (total)`.

An estimate of the additional storage required is given in messages preceding the exit.

`21. EXIT -- error in basis package`
A preceding message will describe the error in more detail. One such message says that the current basis has more than one element in row i and column j. This could be caused by a corresponding error in the input parameters `a(*)`, `ha(*)`, and `ka(*)`.

`22. EXIT -- singular basis after nnn factorization attempts`
This exit is highly unlikely to occur. The first factorization attempt will have found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix PUQ were respectively zero or smaller than a certain tolerance.) The associated variables are replaced by slacks and the modified basis is refactorized. The ensuing singularity must mean that the problem is badly scaled, or the `LU factor tolerance` is too high.

> If the following messages arise, either an OLD BASIS file could not be loaded properly, or some fatal system error has occurred. New BASIS files cannot be saved, and there is no solution to print. The problem is abandoned.

`30. EXIT -- the basis file dimensions do not match this problem`
On the first line of the OLD BASIS file, the dimensions labelled `m` and `n` are different from those associated with the problem that has just been defined. You have probably loaded a file that belongs to some other problem.

Remember, if you have added rows or columns to `a(*)`, `ha(*)` and `ka(*)`, you will have to alter `m` and `n` *and* the map beginning on the third line (a hazardous operation). It may be easier to restart with a PUNCH or DUMP file from the earlier version of the problem.

`31. EXIT -- the basis file state vector does not match this problem`
For some reason, the OLD BASIS file is incompatible with the present problem, or is not consistent within itself. The number of basic entries in the state vector (i.e., the number of 3's in the map) is not the same as `m` on the first card, or else some of the 2's in the map did not have a corresponding "$j \quad x_j$" entry following the map.

`32. EXIT -- system error. Wrong no. of basic variables: nnn`
This exit should never happen. If it does, something is seriously awry in the SQOPT source code. Perhaps the single- and double-precision files have been mixed up.

The following messages arise if additional storage is needed to allow optimization to begin. The problem is abandoned.

`42. EXIT -- not enough 8-character storage to start solving the problem`
The main character storage array `cw(*)` is not large enough.

`43. EXIT -- not enough integer storage to start solving the problem`
The main integer storage array `iw(*)` is not large enough to provide workspace for the optimization procedure. See the advice given for Exit 20.

`44. EXIT -- not enough real storage to start solving the problem`
The main real storage array `rw(*)` is not large enough to provide workspace for the optimization procedure. Be sure that the `Superbasics limit` and `Hessian dimension` are not unreasonably large. Otherwise, see the advice given for Exit 43.

6.5. Solution Output

At the end of a run, the final solution will be output to the PRINT file in accordance with the `Solution` keyword. Some header information appears first to identify the problem and the final state of the optimization procedure. A ROWS section and a COLUMNS section then follow, giving one line of information for each row and column. The format used is similar to that seen in commercial systems, though there is no rigid industry standard.

The ROWS section

The general constraints take the form $l \leq Ax \leq u$. The ith constraint is therefore of the form

$$\alpha \leq a^T x \leq \beta.$$

Internally, the constraints take the form $Ax - s = 0$, where s is the set of slack variables (which happen to satisfy the bounds $l \leq s \leq u$). For the ith constraint it is the slack variable s_i that is directly available, and it is sometimes convenient to refer to its state. To reduce clutter, a "·" is printed for any numerical value that is exactly zero.

Label — *Description*

`Number` The value $n + i$. This is the internal number used to refer to the ith slack in the iteration log.

`Row` The name of the ith row.

`State` The state of the ith row relative to the bounds α and β. The various states possible are as follows.

- `LL` The row is at its lower limit, α.
- `UL` The row is at its upper limit, β.
- `EQ` The lower and upper limit are the same, $\alpha = \beta$.
- `BS` The constraint is not binding. s_i is basic.
- `SBS` The constraint is not binding. s_i is superbasic.

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

- `A` *Alternative optimum possible.* The slack is nonbasic, but its reduced gradient is essentially zero. This means that if the slack were allowed to start moving away from its bound, there would be no change in the value of the objective function. The values of the basic and superbasic 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 dual variables *might* also change.
- `D` *Degenerate.* The slack is basic or superbasic, but it is equal to (or very close to) one of its bounds.
- `I` *Infeasible.* The slack is basic or superbasic and it is currently violating one of its bounds by more than the `Feasibility tolerance`.
- `N` *Not precisely optimal.* The slack is nonbasic or superbasic. If the `Optimality tolerance` were tightened by a factor of 10 (e.g., if it were reduced from 10^{-5} to 10^{-6}), the solution would not be declared optimal because the reduced gradient for the slack would not be considered negligible. (If a loose tolerance

has been used, or if the run was terminated before optimality, this key might be helpful in deciding whether or not to restart the run.)

Note: If `Scale` is specified, the tests for assigning the `A`, `D`, `I`, `N` keys are made on the scaled problem, since the keys are then more likely to be correct.

`Activity` The row value; i.e., the value of $a^T x$.

`Slack activity` The amount by which the row differs from its nearest bound. (For free rows, it is taken to be minus the `Activity`.)

`Lower limit` α, the lower bound on the row.

`Upper limit` β, the upper bound on the row.

`Dual activity` The value of the dual variable π_i, often called the shadow price (or simplex multiplier) for the ith constraint. The full vector π always satisfies $B^T\pi = g_B$, where B is the current basis matrix and g_B contains the associated gradients for the current objective function.

`I` The constraint number, i.

The COLUMNS section

Here we talk about the "column variables" x. For convenience we let the jth component of x be the variable x_j and assume that it satisfies the bounds $\alpha \leq x_j \leq \beta$. A "." is printed for any numerical value that is exactly zero.

`Label` *Description*

`Number` The column number, j. This is the internal number used to refer to x_j in the iteration log.

`Column` The name of x_j.

`State` The state of x_j relative to the bounds α and β. The various states possible are as follows.

- `LL` x_j is nonbasic at its lower limit, α.
- `UL` x_j is nonbasic at its upper limit, β.
- `EQ` x_j is nonbasic and fixed at the value $\alpha = \beta$.
- `FR` x_j is nonbasic and currently zero, even though it is free to take any value. (Its bounds are $\alpha = -\infty$, $\beta = +\infty$. Such variables are normally basic.)
- `BS` x_j is basic.
- `SBS` x_j is superbasic.

A key is sometimes printed before the `State` to give some additional information about the state of x_j. The possible keys are `A`, `D`, `I` and `N`. They have the same meaning as described above (for the ROWS section of the solution), but the words "the slack" should be replaced by "x_j".

`Activity` The value of the variable x_j.

`Obj Gradient` g_j, the jth component of the linear and quadratic objective function $q(x) + c^T x$. (We define $g_j = 0$ if the current solution is infeasible.)

`Lower limit` α, the lower bound on x_j.

`Upper limit` β, the upper bound on x_j.

`Reduced gradnt` The reduced gradient $d_j = g_j - \pi^T a_j$, where a_j is the jth column of the constraint matrix (or the jth column of the Jacobian at the start of the final major iteration).

`M+J` The value $m + j$.

An example of the printed solution is given in §6. Infinite `Upper` and `Lower limits` are output as the word `None`. Other real values are output with format `f16.5`. The maximum record length is 111 characters, including the first (carriage-control) character.

Note: If two problems are the same except that one minimizes $q(x)$ and the other maximizes $-q(x)$, their solutions will be the same but the signs of the dual variables π_i and the reduced gradients d_j will be reversed.

6.6. The SOLUTION file

If a positive SOLUTION file is specified, the information contained in a printed solution may also be output to the relevant file (which may be the PRINT file if so desired). Infinite `Upper` and `Lower limits` appear as $\pm 10^{20}$ rather than `None`. Other real values are output with format `1pe16.6`. Again, the maximum record length is 111 characters, including what would be the carriage-control character if the file were printed.

A SOLUTION file is intended to be read from disk by a self-contained program that extracts and saves certain values as required for possible further computation. Typically the first 14 records would be ignored. Each subsequent record may be read using

```
format(i8, 2x, 2a4, 1x, a1, 1x, a3, 5e16.6, i7)
```

adapted to suit the occasion. The end of the ROWS section is marked by a record that starts with a `1` and is otherwise blank. If this and the next 4 records are skipped, the COLUMNS section can then be read under the same format. (There should be no need to use any `BACKSPACE` statements.)

6.7. The SUMMARY file

If `Summary file` f is specified with $f > 0$, certain brief information will be output to file f. When SQOPT is run interactively, file f will usually be the terminal. For batch jobs a disk file should be used, to retain a concise log of each run if desired. (A SUMMARY file is more easily perused than the associated PRINT file).

A SUMMARY file (like the PRINT file) is not rewound after a problem has been processed. It can therefore accumulate a log for every problem in the SPECS file, if each specifies the same file. The maximum record length is 72 characters, including a carriage-control character in column 1.

The following information is included:

1. The `Begin` card from the SPECS file.
2. The basis file loaded, if any.
3. The status of the solution after each basis factorization (whether feasible; the objective value; the number of function calls so far).
4. The same information every kth iteration, where k is the specified `Summary frequency` (default $k =$ `100`).
5. Warnings and error messages.
6. The exit condition and a summary of the final solution.

Item 4 is preceded by a blank line, but item 5 is not.

All items are illustrated below, where we give the SUMMARY file for the first problem in the example program (`Summary frequency = 1`).

```
==============================
S Q O P T  5.3       (Oct  97)
==============================

Begin sqmain (Example program for sqopt)


Scale option  2,    Partial price   1


------------------------------------------------------------------
Itn      0: Phase 1A -- making the linear equality rows feasible

  Itn       dj    Step nInf    SumInf       Objective
    0  0.0E+00 0.0E+00    1 8.868E+01  0.00000000E+00
    1  0.0E+00 3.3E+01    0 0.000E+00  0.00000000E+00
Itn      1: Feasible linear equality rows
Itn      1: Phase 1B -- making all linear rows feasible

  Itn       dj    Step nInf    SumInf       Objective Norm rg   nS
    1  0.0E+00 0.0E+00    2 5.317E+01  0.00000000E+00 3.4E+00    3
    2  0.0E+00 0.0E+00    2 5.317E+01  0.00000000E+00 4.6E-01    2
    3  0.0E+00 4.7E+02    1 2.896E+01  0.00000000E+00 4.8E-02    1
    4  0.0E+00 9.2E+02    1 2.681E+01  0.00000000E+00 0.0E+00    0


This is problem  sqmain.   ncolH =   5

    5  6.4E-02 6.5E+03    0 0.000E+00 -1.46750000E+06 0.0E+00    0
Itn      5: Feasible linear rows
    6 -4.1E+03 2.1E-01    0 0.000E+00 -1.78368567E+06 0.0E+00    0
    7  1.4E+03 1.0E+00    0 0.000E+00 -1.98453602E+06 1.4E-12    1
    8 -6.3E+02 9.8E-01    0 0.000E+00 -2.04366386E+06 1.3E+01    1
    9  0.0E+00 1.0E+00    0 0.000E+00 -2.04366504E+06 1.1E-12    1
EXIT -- optimal solution found

Problem name                 sqdat1..
No. of iterations                   9   Objective value     -2.0436650381E+06
No. of Hessian products             8   Linear objective     0.0000000000E+00
                                        Quadratic objective -2.0436650381E+06
No. of superbasics                  1   No. of basic nonlinears             2
No. of degenerate steps             1   Percentage                      11.11
Norm of xs  (scaled)          3.5E+03   Norm of pi  (scaled)          8.9E+03
Norm of xs                    1.6E+03   Norm of pi                    1.1E+04
Max Prim inf(scaled)        0 0.0E+00   Max Dual inf(scaled)        6 2.0E-12
Max Primal infeas           0 0.0E+00   Max Dual infeas             3 9.6E-13

Solution printed on file   9
```

7. Example problem

8. Algorithmic Details

SQOPT is based on an inertia-controlling method that maintains a Cholesky factorization of the reduced Hessian (see below). The method follows Gill and Murray [2] and is described in [6]. Here we briefly summarize the main features of the method. Where possible, explicit reference is made to items listed in the printed output, and to the names of the relevant optional parameters.

8.1. Overview

SQOPT's method has a *feasibility phase* (also known as *phase 1*), in which a feasible point is found by minimizing the sum of infeasibilities, and an *optimality phase* (or *phase 2*), in which the quadratic objective is minimized within the feasible region. The computations in both phases are performed by the same subroutines, with the change of phase being characterized by the objective changing from the sum of infeasibiilities (the printed quantity `sInf`) to the quadratic objective (the printed quantity `Objective`).

In general, an iterative process is required to solve a quadratic program. Given an iterate (x, s) in both the original variables x and the slack variables s, a new iterate $(\bar{x}, \bar{s})$ is defined by

$$\begin{pmatrix} \bar{x} \\ \bar{s} \end{pmatrix} = \begin{pmatrix} x \\ s \end{pmatrix} + \alpha p, \tag{8.1}$$

where the *step length* α is a non-negative scalar, and p is called the *search direction*. (For simplicity, we shall consider a typical iteration and avoid reference to the index of the iteration.) Once an iterate is feasible (i.e., satisfies the constraints), all subsequent iterates remain feasible.

8.2. Definition of the working set

At each iterate (x, s), a *working set* of constraints is defined to be a linearly independent subset of the constraints that are satisfied "exactly" (to within the value of the `Feasibility tolerance`). The working set is the current prediction of the constraints that hold with equality at a solution of the LP or QP. Let m_w denote the number of constraints in the working set (including bounds), and let W denote the associated $m_w \times (n+m)$ *working-set matrix* consisting of the m_w gradients of the working-set constraints.

The search direction is defined so that constraints in the working set remain *unaltered* for any value of the step length. It follows that p must satisfy the identity $Wp = 0$. This characterization allows p to be computed using any $n \times n_z$ full-rank matrix Z that spans the null space of W. (Thus, $n_z = n - m_w$ and $WZ = 0$.) The null-space matrix Z is defined from a sparse LU factorization of part of W; see (8.2)–(8.3) below). The direction p will satisfy $Wp = 0$ if $p = Zp_z$ for any n_z-vector p_z.

The working set contains the constraints $Ax - s = 0$ and a subset of the upper and lower bounds on the variables (x, s). Since the gradient of a bound constraint $x_j \geq l_j$ or $x_j \leq u_j$ is a vector of all zeros except for ± 1 in position j, it follows that the working-set matrix contains the rows of $(A \quad -I)$ and the unit rows associated with the upper and lower bounds in the working set.

The working-set matrix W can be represented in terms of a certain column partition of the matrix $(A \quad -I)$. As in §2 we partition the constraints $Ax - s = 0$ so that

$$Bx_B + Sx_S + Nx_N = 0,$$

where B is a square non-singular basis and x_B, x_S and x_N are the basic, superbasic and nonbasic variables respectively. The nonbasic variables are equal to their upper or lower

bounds at (x, s), and the superbasic variables are independent variables that are chosen to improve the value of the current objective. The number of superbasic variables is n_S, which is printed as the quantity `nS`. Given values of x_N and x_S, the basic variables x_B are adjusted so that (x, s) satisfies $Bx_B + Sx_S + Nx_N = 0$.

If P is a permutation such that $(A \quad -I)P = (B \quad S \quad N)$, then the working-set matrix W satisfies

$$WP = \begin{pmatrix} B & S & N \\ 0 & 0 & I_N \end{pmatrix}, \tag{8.2}$$

where I_N is the identity matrix with the same number of columns as N.

The null-space matrix Z is defined from a sparse LU factorization of part of W. In particular, Z is maintained in "reduced-gradient" form, using the package LUSOL [4] to maintain sparse LU factors of the basis matrix B that alters as the working set W changes. Given the permutation P, the null-space basis is given by

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}. \tag{8.3}$$

This matrix is used only as an operator, i.e., it is never computed explicitly. Products of the form Zv and Z^Tg are obtained by solving with B or B^T. This choice of Z implies that n_Z, the number of "degrees of freedom" at (x, s), is the same as n_S, the number of superbasic variables.

Let g_Z and H_Z denote the *reduced gradient* and *reduced Hessian*:

$$g_Z = Z^Tg \qquad \text{and} \qquad H_Z = Z^THZ, \tag{8.4}$$

where g is the objective gradient at (x, s). Roughly speaking, g_Z and H_Z describe the first and second derivatives of an n_S-dimensional *unconstrained* problem for the calculation of p_Z. (The quantity `Cond Hz` printed in the summary-file output is a condition estimator of H_Z.)

At each iteration, an upper-triangular factor R is available such that $H_Z = R^TR$. Normally, R is computed from $R^TR = Z^THZ$ at the start of phase 2 and is then updated as the QP working set changes. For efficiency the dimension of R should not be excessive (say, $n_S \leq 1000$). This is guaranteed if the number of nonlinear variables is "moderate".

If the QP contains linear variables, H is positive semi-definite and R may be singular with at least one zero diagonal. However, an inertia-controlling active-set strategy is used to ensure that only the last diagonal of R can be zero. (See [6] for discussion of a similar strategy for indefinite quadratic programming.)

If the initial R is singular, enough variables are fixed at their current value to give a nonsingular R. This is equivalent to including temporary bound constraints in the working set. Thereafter, R can become singular only when a constraint is deleted from the working set (in which case no further constraints are deleted until R becomes nonsingular).

8.3. The main iteration

If the reduced gradient is zero, (x, s) is a constrained stationary point on the working set. During phase 1, the reduced gradient will usually be zero only at a vertex (although it may be zero elsewhere in the presence of constraint dependencies). During phase 2, a zero reduced gradient implies that x minimizes the quadratic objective when the constraints in the working set are treated as equalities. At a constrained stationary point, Lagrange multipliers λ are defined from the equations $W^T\lambda = g(x)$. A Lagrange multiplier λ_j corresponding

to an inequality constraint in the working set is said to be *optimal* if $\lambda_j \leq \sigma$ when the associated constraint is at its *upper bound*, or if $\lambda_j \geq -\sigma$ when the associated constraint is at its *lower bound*, where σ depends on the `Optimality tolerance`. If a multiplier is non-optimal, the objective function (either the true objective or the sum of infeasibilities) can be reduced by continuing the minimization with the corresponding constraint excluded from the working set (this step is sometimes referred to as "deleting" a constraint from the working set). If optimal multipliers occur during the feasibility phase but the sum of infeasibilities is not zero, there is no feasible point.

The special form (8.2) of the working set allows the multiplier vector λ, the solution of $W^T\lambda = g$, to be written in terms of the vector

$$d = \begin{pmatrix} g \\ 0 \end{pmatrix} - \begin{pmatrix} A & -I \end{pmatrix}^T \pi = \begin{pmatrix} g - A^T\pi \\ \pi \end{pmatrix}, \tag{8.5}$$

where π satisfies the equations $B^T\pi = g_B$, and g_B denotes the basic components of g. The components of π are the Lagrange multipliers λ_j associated with the equality constraints $Ax - s = 0$. The vector d_N of nonbasic components of d consists of the Lagrange multipliers λ_j associated with the upper and lower bound constraints in the working set. The vector d_S of superbasic components of d is the reduced gradient g_Z (8.4). The vector d_B of basic components of d is zero, by construction. (The Euclidean norm of d_S, and the final values of d_S, g and π are the quantities `norm rg`, `Reduced Gradnt`, `Obj Gradient` and `Dual Activity` in the PRINT file output.)

If the reduced gradient is not zero, Lagrange multipliers need not be computed and the search direction is given by $p = Z_Z p_Z$, where p_Z is defined below. The step length is chosen to maintain feasibility with respect to the satisfied constraints.

There are two possible choices for p_Z, depending on whether or not H_Z is singular. If H_Z is nonsingular, R is nonsingular and p_Z is computed from the equations,

$$R^T R p_Z = -g_Z, \tag{8.6}$$

where g_Z is the reduced gradient at x. In this case, $(x, s)+p$ is the minimizer of the objective function subject to the working-set constraints being treated as equalities. If $(x, s) + p$ is feasible, α is defined to be one. In this case, the reduced gradient at $(\bar{x}, \bar{s})$ will be zero, and Lagrange multipliers are computed at the next iteration. Otherwise, α is set to α_N, the step to the boundary of the "nearest" constraint along p. This constraint is added to the working set at the next iteration.

If H_Z is singular, then R must also be singular, and an inertia-controlling strategy is used to ensure that only the last diagonal element of R is zero. (See [6] for discussion of a similar strategy for indefinite quadratic programming.) In this case, p_Z satisfies

$$p_Z^T H_Z p_Z = 0 \quad \text{and} \quad g_Z^T p_Z \leq 0,$$

which allows the objective function to be reduced by any step of the form $(x, s)+\alpha p$, $\alpha > 0$. The vector $p = Zp_Z$ is a direction of unbounded descent for the QP in the sense that the QP objective is linear and decreases without bound along p. If no finite step of the form $(x, s) + \alpha p$ $(\alpha > 0)$ reaches a constraint not in the working set, the QP is unbounded and SQOPT terminates at (x, s) and declares the problem to be unbounded. Otherwise, α is defined as the maximum feasible step along p and a constraint active at $(x, s) + \alpha p$ is added to the working set for the next iteration.

8.4. Miscellaneous

If the basis matrix is not chosen carefully, the condition of the null-space matrix Z (8.3) could be arbitrarily high. To guard against this, SQOPT implements a "basis repair" feature

in the following way. LUSOL is used to compute the rectangular factorization

$$(B \quad S)^T = LU, \tag{8.7}$$

returning just the permutation P that makes PLP^T unit lower triangular. The pivot tolerance is set to require $|PLP^T|_{ij} \leq 2$, and the permutation is used to define P in (8.2). It can be shown that $\|Z\|$ is likely to be little more than 1. Hence, Z should be well-conditioned *regardless of the condition of* W.

This feature is applied at the beginning of the optimality phase if a potential B-S ordering is known.

The EXPAND procedure (see Gill *et al.* [5]) is used to reduce the possibility of cycling at a point where the active constraints are nearly linearly dependent. Although there is no absolute guarantee that cycling will not occur, the probability of cycling is extremely small (see Hall and McKinnon [7]). The main feature of expand is that the feasibility tolerance is increased at every iteration, perhaps at the expense of violating the bounds on (x, s) by a simple amount.

Suppose that the value of `Feasibility tolerance` is δ. Over a period of K iterations (where K is the value of the optional parameter `Expand frequency`, the feasibility tolerance used in SQOPT (i.e., the working feasibility tolerance) increases from $\frac{1}{2}\delta$ to δ in steps of $\frac{1}{2}\delta/K$.

At certain stages, the following "resetting procedure" is used to remove small constraint infeasibilities. First, all nonbasic variables are moved exactly onto their bounds. A count is kept of the number of non-trivial adjustments made. If the count is nonzero, the basic variables are recomputed. Finally, the working feasibility tolerance is reinitialized to $\frac{1}{2}\delta$.

If a problem requires more than K iterations, the resetting procedure is invoked and a new cycle of iterations is started. (The decision to resume phase 1 or phase 2 is based on comparing any constraint infeasibilities with δ.)

The resetting procedure is also invoked if when SQOPT reaches an apparently optimal, infeasible, or unbounded solution, unless this situation has already occurred twice. If any non-trivial adjustments are made, iterations are continued.

The EXPAND procedure not only allows a positive step to be taken at every iteration, but also provides a potential *choice* of constraint to be added to the working set. All constraints at a distance α ($\alpha \leq \alpha_N$) along p from the current point are then viewed as acceptable candidates for inclusion in the working set. The constraint whose normal makes the biggest angle with the search direction is added to the working set. This strategy helps keep the the basis matrix B well-conditioned.

9. BASIS Files

For non-trivial problems, it is advisable to save a BASIS file at the end of a run, in order to restart the run if necessary, or to provide a good starting point for some closely related problem.

Three formats are available for saving basis descriptions. They are invoked by SPECS lines of the following form:

```
NEW BASIS FILE   10
BACKUP   FILE    11      (same as NEW BASIS but on a different file)
PUNCH    FILE    20
DUMP     FILE    30
```

The file numbers may be whatever is convenient, or zero for files that are not wanted.

NEW BASIS and BACKUP files are saved every kth iteration, in that order, where k is the `Save frequency`.

NEW BASIS, PUNCH and DUMP files are saved at the end of a run, in that order. They may be re-loaded at the start of a subsequent run by specifying SPECS lines of the following form respectively:

```
OLD BASIS FILE      10
INSERT    FILE      20
LOAD      FILE      30
```

Only one such file will actually be loaded. If more than one positive file number is specified, the order of precedence is as shown. If no BASIS files are specified, one of the `Crash options` takes effect.

Figures 1–3 illustrate the data formats used for BASIS files. 80-character fixed-length records are suitable in all cases. (36-character records would be adequate for PUNCH and DUMP files.) The files shown correspond to the optimal solution for the SQOPT example program, described in §7. Selected column numbers are included to define significant data fields. The problem has 10 nonlinear constraints, 10 linear constraints, and 30 variables.

9.1. NEW and OLD BASIS Files

We sometimes call these files *basis maps*. They contain the most compact representation of the state of each variable. They are intended for restarting the solution of a problem at a point that was reached by an earlier run on the *same problem* or a related problem with the *same dimensions*. (Perhaps the `Iterations limit` was previously too small, or some other objective row is to be used.)

As illustrated in Figure 1, the following information is recorded in a NEW BASIS file.

1. A line containing the problem name, the iteration number when the file was created, the status of the solution (`OPTIMAL SOLN`, `INFEASIBLE`, `UNBOUNDED`, `EXCESS ITNS`, `ERROR CONDN`, or `PROCEEDING`), the number of infeasibilities, and the current objective value (or the sum of infeasibilities).

2. A line containing the `OBJECTIVE`, `RHS`, `RANGES` and `BOUNDS` names, `M` = the number of rows in the constraint matrix, `N` = the number of columns in the constraint matrix, and `SB` = the number of superbasic variables.

3. A set of $(\texttt{N}+\texttt{M}-1)/80+1$ lines indicating the state of the `N` column variables and the `M` slack variables in that order. One character `HS(`j`)` is recorded for each $j = 1, 2, \ldots,$ `N+M` as follows, written with `FORMAT(80I1)`.

HS(*j*)	*State of the j-th variable*
0	Nonbasic at lower bound
1	Nonbasic at upper bound
2	Superbasic
3	Basic

If variable j is *fixed* (lower bound = upper bound), then HS(j) may be 0 or 1. The same is true if variable j is *free* (infinite bounds) and still nonbasic, although free variables will almost always be basic.

4. A set of lines of the form

$$j \qquad\qquad x_j$$

written with `format(i8, 1pe24.14)` and terminated by an entry with $j = 0$, where j denotes the jth variable and x_j is a real value. The jth variable is either the jth column or the $(j - \mathtt{N})$th slack, if $j > \mathtt{N}$. Typically, HS(j) = 2 (superbasic). When nonlinear constraints are present, this list of superbasic variables is extended to include all basic nonlinear variables. The Jacobian matrix can then be reconstructed exactly for a restart.

Loading a NEW BASIS *file*

A file that has been saved as an OLD BASIS file may be input at the beginning of a later run as a NEW BASIS file. The following notes are relevant:

1. The first line is input and printed but otherwise not used.
2. The values labelled M and N on the second line must agree with those for the problem that has just been defined. The value labelled sb is input and printed but is not used.
3. The next set of lines must contain exactly m values hs(j) = 3, denoting the basic variables.
4. The list of j and x_j values must include an entry for every variable whose state is hs(j) = 2 (the superbasic variables).
5. Further j and x_j values may be included, in any order.
6. For any j in this list, if hs(j) = 3 (basic), the value x_j will be recorded for nonlinear variables, but the variable will remain basic.
7. If hs(j) $\neq$ 3, variable j will be initialized at the value x_j and its state will be reset to 2 (superbasic). If the number of superbasic variables has already reached the `Superbasics limit`, then variable j will be made nonbasic at the bound nearest to x_j (or at zero if it is a free variable).

```
sqdat2..  ITN       0    Optimal Soln  NINF      0      OBJ  -2.043665038075E+06
OBJ=         RHS=          RNG=          BND=          M=       8 N=       7 SB=     1
033023303133003
       5    4.33461578293999E+02
       0
```

Figure 1: Format of NEW and OLD BASIS files

9.2. PUNCH and INSERT Files

These files provide compatibility with commercial mathematical programming systems. The PUNCH file from a previous run may be used as an INSERT file for a later run on the same problem. It may also be possible to modify the INSERT file and/or problem and still obtain a useful advanced basis.

The standard MPS format has been slightly generalized to allow the saving and reloading of nonbasic solutions. It is illustrated in Figure 2. Apart from the first and last line, each entry has the following form:

Columns	2–3	5–12	15–22	25–36
Contents	*Key*	*Name1*	*Name2*	*Value*

The various keys are best defined in terms of the action they cause on input. It is assumed that the basis is initially set to be the full set of slack variables, and that column variables are initially at their smallest bound in absolute magnitude.

Key	*Action to be taken during* INSERT
`XL`	Make variable *Name1* basic and slack *Name2* nonbasic at its lower bound.
`XU`	Make variable *Name1* basic and slack *Name2* nonbasic at its upper bound.
`LL`	Make variable *Name1* nonbasic at its lower bound.
`UL`	Make variable *Name1* nonbasic at its upper bound.
`SB`	Make variable *Name1* superbasic at the specified *Value*.

Note that *Name1* may be a column name or a row name, but (on `XL` and `XU` lines) *Name2* must be a row name. In all cases, row names indicate the associated slack variable, and if *Name1* is a nonlinear variable then its *Value* is recorded for possible use in defining the initial Jacobian matrix.

The key `SB` is an addition to the standard MPS format to allow for nonbasic solutions.

Notes on PUNCH Data

1. Variables are output in natural order. For example, on the first `XL` or `XU` line, *Name1* will be the first basic column and *Name2* will be the first row whose slack is not basic. (The slack could be nonbasic or superbasic.)
2. `LL` lines are *not* output for nonbasic variables if the corresponding lower bound value is zero.
3. Superbasic slacks are output last.
4. PUNCH and INSERT files deal with the status and values of *slack variables*. This is in contrast to the printed solution and the SOLUTION file, which deal with *rows*.

Notes on INSERT Data

1. Before an INSERT file is read, column variables are made nonbasic at their smallest bound in absolute magnitude, and the slack variables are made basic.
2. Preferably an INSERT file should be an unmodified PUNCH file from an earlier run on the same problem. If some rows have been added to the problem, the INSERT file need not be altered. (The slacks for the new rows will be in the basis.)
3. Entries will be ignored if *Name1* is already basic or superbasic. `XL` and `XU` lines will be ignored if *Name2* is not basic.

4. SB lines may be added before the ENDATA line, to specify additional superbasic columns or slacks.

5. An SB line will not alter the status of *Name1* if the SUPERBASICS LIMIT has been reached. However, the associated *Value* will be retained if *Name1* is a Jacobian variable.

9.3. DUMP and LOAD Files

These files are similar to PUNCH and INSERT files, but they record solution information in a manner that is more direct and more easily modified. In particular, no distinction is made between columns and slacks. Apart from the first and last line, each entry has the form

Columns	2–3	5–12	25–36
Contents	*Key*	*Name*	*Value*

as illustrated in Figure 3. The keys LL, UL, BS and SB mean Lower Limit, Upper Limit, Basic and Superbasic respectively.

Notes on DUMP Data

1. A line is output for every variable, columns followed by slacks.

2. Nonbasic free variables will be output with either LL or UL keys and with *Value* zero.

Notes on LOAD Data

1. Before a LOAD file is read, all columns and slacks are made nonbasic at their smallest bound in absolute magnitude. The basis is initially empty.

2. Each LL, UL or BS line causes *Name* to adopt the specified status. The associated *Value* will be retained if *Name* is a Jacobian variable.

3. An SB line causes *Name* to become superbasic at the specified *Value*.

4. An entry will be ignored if *Name* is already basic or superbasic. (Thus, only the first BS or SB line takes effect for any given *Name*.)

5. An SB line will not alter the status of *Name* if the Superbasics limit has been reached, but the associated *Value* will be retained if *Name* is a Jacobian variable.

6. (*Partial basis*) Let M be the number of rows in the problem. If fewer than M variables are specified to be basic, a tentative basis list will be constructed by adding the requisite number of slacks, starting from the first row and taking those that were not previously specified to be basic or superbasic. (If the resulting basis proves to be singular, the basis factorization routine will replace a number of basic variables by other slacks.) The starting point obtained in this way will not necessarily be "good".

7. (*Too many basics*) If M variables have already been specified as basic, any further BS keys will be treated as though they were SB. This feature may be useful for combining solutions to smaller problems.

```
NAME          sqdat2..  PUNCH/INSERT
 XL ...100    ....01     3.89064E+02
 XU ...101    ....02     6.19233E+02
 LL ...102    ....03     1.00000E+02
 SB ...105    ....04     4.33462E+02
 XL ...107    ....05     3.00048E+02
 XL ...111    ....06     1.58194E+02
ENDATA
```

Figure 2: Format of PUNCH/INSERT files

```
NAME          sqdat2..     DUMP/LOAD
 LL ...100                 0.00000E+00
 BS ...101                 3.89064E+02
 BS ...102                 6.19233E+02
 LL ...103                 1.00000E+02
 SB ...104                 4.33462E+02
 BS ...105                 3.00048E+02
 BS ...106                 1.58194E+02
 LL ...107                 2.00000E+03
 BS ...108                 4.83627E+01
 UL ...109                 1.00000E+02
 BS ...110                 3.24241E+01
 BS ...111                 1.60065E+01
 LL ...112                 1.50000E+03
 LL ...113                 2.50000E+02
 BS ...114                -2.90022E+06
ENDATA
```

Figure 3: Format of DUMP/LOAD files

9.4. Restarting Modified Problems

Sections 9.1–9.3 document three distinct starting methods (OLD BASIS, INSERT and LOAD files), which may be preferable to any of the cold start (CRASH) options. The best choice depends on the extent to which a problem has been modified, and whether it is more convenient to specify variables by number or by name. The following notes offer some rules of thumb.

Protection

In general there is no danger of specifying infinite values. For example, if a variable is specified to be nonbasic at an upper bound that happens to be $+\infty$, it will be made nonbasic at its lower bound. Conversely if its lower bound is $-\infty$. If the variable is *free* (both bounds infinite), it will be made nonbasic at value zero. No warning message will be issued.

Default Status

If the status of a variable is not explicitly given, it will initially be nonbasic at the bound that is smallest in absolute magnitude. Ties are broken in favor of lower bounds, and free variables will again take the value zero.

Restarting with Different Bounds

Suppose that a problem is to be restarted after the bounds on some variable `X` have been altered. Any of the basis files may be used, but the starting point obtained depends on the status of `X` at the time the basis is saved.

If `X` is basic or superbasic, the starting point will be the same as before (all other things being equal). The value of `X` may lie outside its new set of bounds, but there will be minimal loss of feasibility or optimality for the problem as a whole.

If `X` was previously *fixed*, it is likely to be nonbasic at its *lower* bound (which happens to be the same as its upper bound). Increasing its upper bound will not affect the solution.

In contrast, if `X` is nonbasic at its *upper* bound and if that bound is altered, the starting values for an arbitrary number of basic variables could be changed (since they will be recomputed from the nonbasic and superbasic variables). This may not be of great consequence,

but sometimes it may be worthwhile to retain the old solution precisely. To do this, one must make `X` superbasic at the original bound value.

For example, if `x` is nonbasic at an upper bound of 5.0 (which has now been changed), one should insert a line of the form

```
j                    5.0
```

near the end of an OLD BASIS file, or the line

```
SB X                 5.0
```

near the end of an INSERT or LOAD file. Note that the SPECS file must specify a `Superbasics limit` at least as large as the number of variables involved, even for purely linear problems.

Sequences of Problems

Whenever practical, a series of related problems should be ordered so that the *most tightly constrained* cases are solved first. Their solutions will often provide feasible starting points for subsequent relaxed problems, as long the above precautions are taken.