User's Guide for SQOPT Version 7: A Fortran Package for Large-Scale Linear and Quadratic Programming[*]

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

Walter MURRAY and Michael A. SAUNDERS
Systems Optimization Laboratory
Department of Management Science and Engineering
Stanford University, Stanford, CA 94305-4026

July 2005

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 [4]). SQOPT is most efficient if there are few superbasic variables (degrees of freedom) compared to the number of variables. A problem will have few degrees of freedom if only some of the variables appear in the quadratic term, or if the number of active constraints (including simple bounds) is nearly as large as the number of variables. However, unlike previous versions of SQOPT, there are no limits on the number of superbasic variables.

SQOPT is primarily intended for (but is not restricted to) large linear and quadratic problems with sparse constraint matrices. A quadratic term $\frac{1}{2}x^T Hx$ in the objective function is represented by a user subroutine that returns the product Hx for a given vector x.

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, optional automatic scaling of the constraints, and elastic bounds on any number of constraints and variables.

The source code for SQOPT is re-entrant and is suitable for any machine with a Fortran compiler or the `f2c` translator. 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://www.cam.ucsd.edu/~peg`
`walter@stanford.edu`	`http://www.stanford.edu/~walter`
`saunders@stanford.edu`	`http://www.stanford.edu/~saunders`

[*]Partially supported by National Science Foundation grants DMI-9204208, DMI-9500668, CCR-9988205, and CCR-0306662, and Office of Naval Research grants N00014-96-1-0274 and N00014-02-1-0076.

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 the form

$$\boxed{\text{LCQP} \qquad \begin{array}{ll} \underset{x}{\text{minimize}} & q(x) \\ \text{subject to} & l \le \begin{pmatrix} x \\ Ax \end{pmatrix} \le u, \end{array}}$$

where l and u are constant lower and upper bounds, A is a sparse matrix, and $q(x)$ is a linear or quadratic objective function that may be specified in a variety of ways, depending upon the problem being solved. An 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 called the *Hessian*, 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 H must be *positive semidefinite*—i.e., it must satisfy $x^T H x \ge 0$ for all x. If SQOPT encounters a negative $x^T H x$ for some x, it will terminate with the error indicator `INFO` $= 53$.

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

Problem type	Objective function q	Hessian matrix H
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 `Acol(*)`, `indA(*)`, and `locA(*)` (see Section 3.1). 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 dense vector x. For example, H may be a sparse matrix or a sum of matrices of low rank. The sample program provided with SQOPT includes such a subroutine.

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 row `iObj` of A and 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 with differing objectives (see parameters `cObj` and `lencObj` of subroutine `sqOpt`).

1.1. Implementation

SQOPT is implemented as a set of callable Fortran subroutines. The source code is compatible with all known Fortran 77, 90, and 95 compilers, and can be converted to C code by the `f2c` translator [1] included with the distribution.

All routines in SQOPT are intended to be re-entrant (as long as the compiler allocates local variables dynamically). Hence they may be used in a parallel or multi-threaded 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 section 6.

1.3. Overview of the package

SQOPT is normally accessed via a sequence of subroutine calls. For example, `sqOpt` may be invoked by the statements

```
call sqInit( iPrint, iSumm, ... )
call sqSpec( iSpecs, ...        )
call sqOpt ( Start, qpHx, m, ...)
```

where `sqSpec` reads a file of run-time options (if any). Also, individual run-time options may be "hard-wired" by calls to `sqSet`, `sqSeti`, and `sqSetr`.

Subroutine `sqInit` must be called before any other SQOPT routine. 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.)

1.4. Getting started

For a given value of n, suppose that we wish to find the n-vector x that is closest in Euclidean norm to a given vector x_0. The complication is that not only must x lie in the set

$$\mathcal{S} = \big\{x : \sum_{j=1}^{n} x_j = 1, \quad x \ge 0, \big\},$$

but also its components must be *nonincreasing*: $x_j \le x_{j+1}$. This problem may be written as a quadratic program

$$\begin{array}{ll} \underset{x_1,x_2}{\text{minimize}} & \frac{1}{2}\sum_{j=1}^{n} (x_j - (x_0)_j)^2 \\ \text{subject to} & x_j - x_{j+1} \le 0, \quad j = 1, 2, \ldots, n-1, \\ & \sum_{j=1}^{n} x_j = 1, \quad x \ge 0. \end{array} \tag{1.1}$$

The objective function to be minimized can be written in the form

$$\tfrac{1}{2}(x - x_0)^T(x - x_0) = \tfrac{1}{2}x_0^T x_0 - x_0^T x + \tfrac{1}{2}x^T x,$$

which is the quadratic $f + c^T x + \frac{1}{2}x^T H x$ with $f = \frac{1}{2}x_0^T x_0$, $c = -x_0$, and $H = I$.

The constraints $x_j \le x_{j+1}$ are written in the form $-\infty \le x_j - x_{j+1} \le 0$. These $n-1$ constraints, together with the restriction that the variables must sum to one, define n so-called "range constraints" of the form $l_A \le Ax \le u_A$. When $n = 3$,

$$l_A = \begin{pmatrix} -\infty \\ -\infty \\ 1 \end{pmatrix}, \quad A = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad \text{and} \quad u_A = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

These quantities define the *general constraints* of the problem. Similarly the nonnegativity constraints on the components of x may be written as n simple bounds $l_x \le x \le u_x$, where

$$l_x = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad \text{and} \quad u_x = \begin{pmatrix} +\infty \\ +\infty \\ +\infty \end{pmatrix}.$$

Internally `sqOpt` converts the general constraints to equalities by introducing a set of *slack variables* $s = (s_1, s_2, \ldots, s_m)^T$. For example, the first linear constraint $-\infty \le x_1 - x_2 \le 0$ is replaced by $x_1 - x_2 - s_1 = 0$ together with the bounded slack $-\infty \le s_1 \le 0$. Problem LCQP can therefore be rewritten in the following equivalent form:

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

The slack variables s are subject to the same bounds as the components of Ax. They allow us to think of the bounds on x and Ax as bounds on the combined vector (x, s).

Now we must provide `sqOpt` the following information:

1. *A subroutine* `qpHx` *that computes Hx, the product of H with a vector x.* For this simple example, H is the identity matrix and the `qpHx` output vector `Hx` is defined using the simple assignments:

```
Hx(1) =  x(1)
Hx(2) =  x(2)
Hx(3) =  x(3)
```

2. *The objective row* `cObj` *and constant term* `Objadd`. These quantities define f and c for the problem. For Example (1.1) above, `cObj` is the constant vector $-x_0$, and `ObjAdd` is the quantity $\frac{1}{2}x_0^Tx_0$.

   (SQOPT minimizes the quadratic $c^Tx + \frac{1}{2}x^THx$ and adds the constant f for printing purposes only.)

3. *The lower and upper bounds l and u on (x, s).* These vectors are input as arrays `bl` and `bu`, each of length at least $n + m$. The first n elements of `bl` and `bu` hold the bounds l_x and u_x:

```
infBnd  =  1.0d+20
bl(1)   =  0.0
bl(2)   =  0.0
bl(3)   =  0.0

bu(1)   =  infBnd
bu(2)   =  infBnd
bu(3)   =  infBnd
```

   where `infBnd` represents "infinity". It must be at least as large as the `Infinite Bound size` (default value 10^{20}).

   Elements $n + 1$ through $n + m$ of `bl` and `bu` hold the bounds l_A and u_A:

```
bl(n+1) = -infBnd
bl(n+2) = -infBnd
bl(n+3) =  1.0

bu(n+1) =  0.0
bu(n+2) =  0.0
bu(n+3) =  1.0
```

   Note that the third row, which simply sums the variables, must have equal bounds to make it an "equality" row.

4. *The nonzero elements of the matrix A.* These are stored by columns in the array `Acol`. The corresponding row numbers are stored in the parallel array `indA`. In our example, the matrix A has two nonzeros in each column, with

```
Acol = { -1.0  1.0  1.0 -1.0  1.0  1.0  1.0 }
indA = {  1    3    1    2    3    2    3   }
```

   One other integer array `locA` is needed to indicate where each column of A starts. In this case we have

```
locA = { 1  3  6  8 }
```

   Note that although A has 3 columns, `locA` has 4 elements. The element `locA(n+1)` is set to the number of nonzeros plus 1. Then for all j we may determine the number of nonzeros in the jth column using the expression `locA`$(j + 1) -$ `locA`(j).

   This scheme is easy to generalize to problems with arbitrary column dimension. The following code fragment defines the constraint data structure for Problem (1.1) with n variables and $m = n$ general constraints:

```
one  = 1.0d+0
neA  = 0                        ! Counts the nonzeros in A

do j = 1, n
   locA( j) =  neA + 1          ! Points to the start of column j
   if (j .gt. 1) then
      neA       =  neA + 1
      indA(neA) =  j   - 1
      Acol(neA) = -one
   endif
   if (j .lt. n) then
      neA       =  neA + 1
      indA(neA) =  j
      Acol(neA) =  one
   end if
   neA       =  neA + 1
   indA(neA) =  m
   Acol(neA) =  one
end do

locA(n+1) =  neA + 1
```

As a matter of good programming practice, we recommend the use of the counter `neA` to reference the elements of `Acol` and `indA`. It allows the code to be updated easily if new constraints or variables are added to the problem.

2. A brief description of quadratic programming

The method used in SQOPT follows Gill and Murray [3] and is described in [7]. Here we summarize the main features of the method and introduce some terminology used in the description of subroutine `sqOpt` and its arguments. Where possible, explicit reference is made to items listed in the printed output, and to the names of the relevant optional parameters.

2.1. Formulation of the problem

As mentioned in Section 1.4, Problem LCQP can be written in the equivalent form

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

where s is the vector of *slack variables*. The bounds on s are the bounds on Ax.

SQOPT solves LP or QP problems using an *active-set* method. This is an iterative procedure with two phases. *Phase 1* (sometimes called the *feasibility phase*) minimizes the sum of infeasibilities to find a feasible point (one that satisfies all constraints). *Phase 2* (the *optimality phase*) minimizes the objective $q(x)$ by constructing a sequence of iterates that are all feasible. 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 infeasibilities (the printed quantity `sInf`) to the quadratic objective (the printed quantity `Objective`).

Phase 1 solves a linear program of the form

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

using a sequence of points (x, s) that satisfy $Ax - s = 0$. This is equivalent to minimizing the sum of the constraint violations. If the constraints are feasible (i.e., at least one feasible point exists), a point will eventually be found at which both v and w are zero. The associated value of (x, s) satisfies the constraints in the original problem and is used as the starting point for the phase 2 iterations for minimizing $q(x)$.

2.2. Active-set methods

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*.

In an active-set method each iterate is feasible, which implies that the constraints $Ax - s = 0$ may be (conceptually) partitioned as $Bx_B + Sx_S + Nx_N = 0$, 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; together with x_N they form a permutation of the elements of x and s. At a QP solution, the basic and superbasic variables will lie on or somewhere between their bounds, while the nonbasic variables will be equal to one of their upper or lower bounds.

At a nonoptimal feasible point (x, s) we seek a search direction p such that $(x, s) + p$ is feasible for the same set of constraints yet gives an improved value of the QP objective (or sum of infeasibilities). If the new point is to be feasible it must hold that $p_N = 0$ and $Bp_B + Sp_S + Np_N = 0$, where p_B, p_S and p_N are the basic, superbasic and nonbasic components of p. These relations imply that once p_S is specified, p_B is uniquely determined from the system $Bp_B = -Sp_S$. It follows that the superbasic variables may be regarded as independent variables that are free to move in any desired direction, and that the values of the basic variables ensure $(x, s)+p$ satisfies $Ax - s = 0$. 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 need not be more than one for FP and LP problems.

The dependence of p on p_S may be expressed compactly as $p = Zp_S$, where Z is the so-called *null-space basis*

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix} \tag{2.1}$$

with P the permutation that arranges the columns of $\begin{pmatrix} A & -I \end{pmatrix}$ in the order $\begin{pmatrix} B & S & N \end{pmatrix}$. The minimization of $q(x)$ with respect to p_S now involves an unconstrained quadratic function with Hessian Z^THZ and constant vector Z^Tg, where g is the gradient of the objective. If Z^THZ is nonsingular, p_S is computed from the equations,

$$Z^THZp_S = -Z^Tg. \tag{2.2}$$

The matrix Z^THZ is known as the *reduced Hessian* and Z^Tg is the *reduced gradient*. The matrix Z 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. In particular, the sparse-matrix package LUSOL [5] is used to maintain the LU factors of the basis matrix B as the BSN partition changes.

The right-hand side vector Z^Tg can be written compactly in terms of the vector π of *dual variables* associated with the m equality constraints $Ax - s = 0$. 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 $g_x - A^T\pi$, where g_x is the x part of the objective gradient. The reduced gradients for the slacks s are the dual variables π. The vector Z^Tg is then $d_S = g_S - S^T\pi$, the vector of reduced gradients for the superbasic variables.

The optimality conditions for problem LCQP may also be written in terms of the vector of reduced costs. The current point 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 section 4.6).

If $d_S = 0$, no improvement can be made with the current BSN partition, and a nonbasic variable with non-optimal reduced cost is selected to be added to S. The iteration is then repeated with n_S increased by one. At all stages, if the step $(x, s) + p$ would cause a basic or superbasic variable to violate one of its bounds, a shorter step $(x, s) + \alpha p$ is taken, one of the variables is made nonbasic, and n_S is decreased by one.

The process of computing and testing reduced gradients is known as *pricing* (a term first introduced in the context of the simplex method for linear programming). To price the jth variable means that its reduced gradient is computed as $d_j = g_j - a_j^T\pi$, where a_j is the jth column of $(A \;\; -I)$. (In the PRINT file output, the nonbasic variable selected after pricing, along with its reduced gradient d_j, are printed in the columns marked `+SBS` and `dj`.) 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 test only a subset of the d_j's.

The vector d_B of basic components of d is zero by construction. The final value of $\|d_S\|_1$ is listed as `norm rg` after the `EXIT` message in the SUMMARY and PRINT files, and the final vectors π, g, and d are labeled `Dual Activity`, `Obj Gradient`, and `Reduced Gradnt` in the PRINT and SOLUTION files.

At each iteration, an upper-triangular matrix R is available such that $R^TR = Z^THZ$. Normally, R is computed from Z^THZ at the start of phase 2 and is then updated as the basic and nonbasic sets change. For efficiency the dimension of R should not be excessive (say, $n_S \le 1000$). 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. In this case, an inertia-controlling active-set strategy is used to ensure that only the last diagonal of R can be zero. (See [7] for discussion of a similar strategy for indefinite quadratic programming.)

2.3. Treatment of constraint infeasibilities

If the constraints are infeasible (i.e., $v \ne 0$ or $w \ne 0$ 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 $q(x)$ 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

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

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 original objective $q(x)$ or the sum of infeasibilities. If $\gamma = 0$, SQOPT will attempt to minimize $q(x)$ 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 $v_j = l_j - x_j$.

2.4. Degeneracy and the feasibility tolerance

For numerical reasons, SQOPT allows the variables (x, s) to stray outside their bounds by as much as a specified `Feasibility tolerance` δ (default value 10^{-6}). The EXPAND procedure of Gill *et al.* [6] takes advantage of δ to reduce the possibility of cycling at a point where the active constraints are nearly linearly dependent. Although there is no guarantee of preventing cycling, the probability is very small (see Hall and McKinnon [8]).

The main feature of EXPAND is that over a period of K iterations (where K is the specified `Expand frequency`), a "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 infeasibilities with δ.)

The resetting procedure is also invoked 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 allows a positive step to be taken at every iteration, and also provides a potential *choice* of constraint to be added to the working set. All constraints at a distance α ($\alpha \le \alpha_N$) 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.

2.5. Basis repair

If the basis matrix is not chosen carefully, the condition of the null-space matrix Z (2.1) could be arbitrarily high. (The quantity `Cond Hz` printed in the SUMMARY output is a condition estimator for Z^THZ.) To guard against this, SQOPT implements a "basis repair" feature in the following way. LUSOL is used to compute the rectangular factorization

$$\begin{pmatrix} B^T \\ S^T \end{pmatrix} = LU, \tag{2.3}$$

returning just the permutation P that makes PLP^T unit lower triangular. The stability tolerance is set to require $|L_{ij}| \le 2$, and the permutation is used to define P in (2.1). It can be shown that $\|Z\|$ is likely to be little more than 2. Since the smallest singular value of Z is at least 1, it means that Z should be well-conditioned *regardless of the condition of the constraints.*

This feature is applied at the beginning of the optimality phase if S has one or more columns.

3. Subroutines associated with `sqOpt`

The SQOPT package is accessed via the following routines:

`sqInit` (Section 3.3) must be called before any other SQOPT routines.

`sqSpec` (Section 4.3) may be called to input a SPECS file (a list of run-time options).

`sqSet`, `sqSeti`, `sqSetr` (Section 4.4) may be called to specify a single option.

`sqGet`, `sqGetc`, `sqGeti`, `sqGetr` (Section 4.5) may be called to obtain an option's current value.

`qpHx` (Section 3.2) is 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` (Section 3.1) is the main solver.

`sqMem` (Section 3.4) computes the size of the workspace arrays `cw`, `iw`, `rw` required for given problem dimensions. Intended for Fortran 90 and C 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-threaded environment. They may also be called recursively.

In the subroutine descriptions below, note that `double precision` declarations are suitable for most machines as shown, but some machines use `real`.

3.1. Subroutine sqOpt

Problem QP is solved by a call to subroutine `sqOpt`, whose parameters are defined here.

```
      subroutine sqOpt
     &   ( Start, qpHx, m,
     &     n, neA, nName, lencObj, ncolH,
     &     iObj, ObjAdd, Prob,
     &     Acol, indA, locA, bl, bu, cObj, Names,
     &     hElast, hs, x, pi, rc,
     &     INFO, mincw, miniw, minrw,
     &     nS, nInf, sInf, Obj,
     &     cu, lencu, iu, leniu, ru, lenru,
     &     cw, lencw, iw, leniw, rw, lenrw )

      external
     &     qpHx
      integer
     &     iObj, INFO, lencObj, lencu, leniu, lenru, lencw, leniw,
     &     lenrw, m, mincw, miniw, minrw, n, neA, nName, ncolH, nS,
     &     nInf, hElast(n+m), hs(n+m), indA(neA), iu(leniu), iw(leniw),
     &     locA(n+1)
      double precision
     &     Obj, ObjAdd, sInf, Acol(neA), bl(n+m), bu(n+m), cObj(*),
     &     pi(m), rc(n+m), x(n+m), ru(lenru), rw(lenrw)
      character*(*)
     &     Start
      character
     &     Prob*8, Names(nName)*8, cu(lencu)*8, cw(lencw)*8
```

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 via the argument `hs` (probably from an earlier call).
- `'Hot'` or `'Hot FHS'` means SQOPT should start with all three types of information available from an earlier call. Just one type may be requested as follows:

Start	Information held in work arrays
`'Hot F'`	Factors of the basis (LU)
`'Hot H'`	Factors of the reduced Hessian (Cholesky)
`'Hot S'`	Scale factors for the constraints and variables

Any combination of `F`, `H`, `S` may be specified, such as `'Hot FS'`.

`m` is m, the number of general inequalities ($\texttt{m} > 0$). 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 `Acol`, `indA` and `locA` below.

`n` is the number of variables, excluding slacks ($\texttt{n} > 0$). This is the number of columns in A.

`neA` is the number of nonzero entries in A ($\texttt{neA} > 0$).

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

`lencObj` is the number of elements in the constant objective vector c ($\texttt{lencObj} \geq 0$).

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

If $\texttt{ncolH} = 0$, 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 $\texttt{ncolH} > 0$, 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 ($0 \leq \texttt{iObj} \leq m$). If there is no such vector, $\texttt{iObj} = 0$.

`ObjAdd` is the constant f added to the objective for printing purposes. Typically `ObjAdd` is zero.

`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.

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

1. It is *essential* that $\texttt{locA}(1) = 1$ and $\texttt{locA}(n+1) = \texttt{neA} + 1$.
2. The row indices `indA(k)` for a column may be in any order.
3. If $\texttt{lencObj} > 0$, the first `lencObj` columns of `Acol` and `indA` belong to variables corresponding to the constant objective term c.
4. If the problem has a quadratic objective, the first `ncolH` columns of `Acol` and `indA` 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 $\texttt{Acol}(1) = \texttt{0.0}$, $\texttt{indA}(1) = 1$, $\texttt{locA}(1) = 1$, and $\texttt{locA}(j) = 2$ for $j = 2\!:\!n+1$. This row is made "free" by setting its bounds to be $\texttt{bl}(n+1) = -\texttt{infBnd}$ and $\texttt{bu}(n+1) = \texttt{infBnd}$, where `infBnd` 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 `x` 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`$(j) \le -$`infBnd`, where `infBnd` is the `Infinite Bound`, whose default value is 10^{20}.

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

To make the ith constraint an *equality* constraint (say $s_i = \beta$, where $|\beta| <$ `infBnd`), set `bl`$(n+i) =$ `bu`$(n+i) = \beta$.

`bu(n+m)` contains the upper bounds on (x, s). To specify a non-existent upper bound ($u_j = \infty$), set `bu`$(j) \ge$ `infBnd`. For the data to be meaningful, it is required that `bl`$(j) \le$ `bu`(j) for all j.

`cObj(lencObj)` sometimes contains the explicit objective vector c (if any). If the problem is of type FP, or if `lencObj` $= 0$, then `cObj` is not referenced. (In that case, `cObj` 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`, then `Names` is not used. The printed solution will use generic names for the columns and row. Otherwise, `nName` $= n + m$ and `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)` sometimes defines which variables are to be treated as being elastic in elastic mode. The values `hElast`$(j) = 0$, 1, 2, 3 have the following meaning:

`hElast`(j)	Status in elastic mode
0	variable j is non-elastic and cannot be infeasible
1	variable j may violate its lower bound
2	variable j may violate its upper bound
3	variable j may violate either of its bounds

`hElast` need not be assigned if `Elastic mode` $= 0$.

`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 the argument `x`.

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

1. If a basis file of some sort is to be input (`start` = `'Cold'` or `'Basis file'` and an OLD BASIS, INSERT, or LOAD file is specified in the SPECS file), then `hs` and `x` need not be set.
2. Otherwise, `hs`$(1:n)$ and `x`$(1:n)$ must be defined for a Cold start. If nothing special is known about the problem, or there is no wish to provide special information, you may set `hs`$(j) = 0$, `x`$(j) = 0.0$ for $j = 1 : n$. 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 `x`$(j) =$ `bl`(j) or `hs`$(j) = 5$ and `x`$(j) =$ `bu`(j) as appropriate.

   SQOPT then uses a CRASH procedure to select variables for the initial basis. The corresponding 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 x(j) if bl(j) ≤ x(j) ≤ bu(j), or at the value bl(j) or bu(j) closest to x(j). See the description of hs below (on exit).

3. For Warm and Hot starts, all of hs($1\!:\!n+m$) must be 0, 1, 2 or 3 (perhaps from some previous call) and all of x($1\!:\!n+m$) must have values. Use Warm rather than Cold if you wish to input the initial state of the slack variables.

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

qpHx is the name of the subroutine that defines the product of H with a given vector x when the problem is a quadratic program. This is the only way that SQOPT accesses the matrix H in the objective function. For a detailed description of qpHx, see Section 3.2.

For problems of type FP and LP, qpHx is never called by sqOpt. You may 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 qpHx doesn't reference these parameters, you may use any arrays of the appropriate type, such as cw, iw, rw (see next paragraph). 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. The integers 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 INFO = 82, 83, or 84. The values are returned in mincw, miniw, and minrw.

On exit:

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

hs(j)	State of variable j	Usual value of x(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 Feasibility tolerance (default value 10^{-6}). 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 `x`(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).

`x(n+m)` contains 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, $g - \begin{pmatrix} A^T \\ -I \end{pmatrix}\pi$. If `x` is feasible, g is the gradient of the objective. (The last m entries are g are zero, so the last m entries of `rc` are π.) Otherwise, g is the gradient of the Phase-1 objective.

`INFO` reports the result of the call to `sqOpt`. Here is a summary of possible values. Further details are in Section 5.4.

Finished successfully

1	optimality conditions satisfied
2	feasible point found
4	weak QP minimizer

The problem appears to be infeasible

11	infeasible linear constraints
12	infeasible linear equalities
14	infeasibilities minimized

The problem appears to be unbounded

21	unbounded objective

Resource limit error

31	iteration limit reached
33	the superbasics limit is too small

Terminated after numerical difficulties

42	singular basis
43	cannot satisfy the general constraints
44	ill-conditioned null-space basis

Error in the user-supplied functions

53	the QP Hessian is indefinite

User requested termination

73	terminated during call to `qpHx`
74	terminated from monitor routine

Insufficient storage allocated

81	work arrays must have at least 500 elements
82	not enough character storage
83	not enough integer storage
84	not enough real storage

Input arguments out of range

91	invalid input argument
92	basis file dimensions do not match this problem

System error

141	wrong number of basic variables
142	error in basis package

`mincw, miniw, minrw` say how much character, integer and real storage is needed to solve the problem. If `sqOpt` terminates because of insufficient storage (`INFO` = 82, 83, or 84), these values may be used to define better values of `lencw`, `leniw` or `lenrw`.

If `INFO` = 82, the work array `cw(lencw)` was too small. `sqOpt` may be called again with `lencw` = `mincw`.

If `INFO` = 83 or 84, 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 factorization needs.)

`nS` is the final number of superbasics.

`nInf` is the number of infeasibilities.

`sInf` is the sum of infeasibilities.

`Obj` is the final value of the explicit quadratic term. If `nInf` = 0, `Obj` is the explicit quadratic term (if any) defined from `cObj` and `qpHx`. If `nInf` > 0 and `cObj` is defined, `Obj` is the explicit linear term. Otherwise, `Obj` is zero.

Note that `Obj` does not include contributions from the constant term `ObjAdd` or the objective row, if there is one. The final value of the objective being optimized is `ObjAdd + x(n+iObj) + Obj`, where `iObj` is the index of the objective row in A.

3.2. Subroutine qpHx

For QP problems, you must provide a subroutine that defines products of the form Hx for given vectors x. This is the way `sqOpt` accesses the matrix H in the objective function. Your subroutine is input to `sqOpt` via the parameter `qpHx`, which must be declared `external` within the routine that calls `sqOpt`.

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.

```
      subroutine qpHx
     &   ( ncolH, x, Hx, Status,
     &     cu, lencu, iu, leniu, ru, lenru )

      integer
     &     lencu, leniu, lenru, ncolH, Status, iu(leniu)
      double precision
     &     x(ncolH), Hx(ncolH), ru(lenru)
      character
     &     cu(lencu)*8
```

On entry:

`ncolH` is the same as `sqOpt`'s input parameter ($0 \le$ `ncolH` $\le$ `n`). It must not be altered within `qpHx`. Similarly for the parameters `iu`, `leniu`, `ru`, `lenru`.

If some of the variables enter the objective function linearly, then H will have some 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 $x = (y, z)$ and only y enters the objective quadratically, then

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

In this case, `ncolH` should be the dimension of y and `qpHx` should compute $H_1 y$.

`x` contains a vector x such that the product Hx should be returned in `Hx`. If `ncolH` $<$ `n`, then "x" will be the vector y above.

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

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

If `Status` $= 1$, `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 `Status` ≥ 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 `Status` $= 2 +$ `INFO/10`, where `INFO` indicates the status of the final solution (see Section 5.4). In particular, if `Status` $= 2$, the current `x` is *optimal*; if `Status` $= 3$, the problem appears to be infeasible; if `Status` $= 4$, the problem appears to be unbounded; and if `Status` $= 5$, 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 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`, `rw`. In this case, `sqOpt` should be called with `cw`, `iw`, `rw` as actual arguments for `cu`, `iu`, `ru`, thereby making `cw`, `iw`, `rw` accessible to `qpHx`.

If you require user workspace in this situation, elements `501:maxcw`, `501:maxrw`, `501:maxiw` of `cw`, `rw`, `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 Section 4.6.)

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

On exit:

`Hx` should contain the product Hx for the vector stored in `x`. If `ncolH` $<$ `n`, it is really the product H_1y mentioned above.

3.3. Subroutine `sqInit`

Subroutine `sqInit` must be called before any other `sqOpt` routine. It defines the PRINT and SUMMARY files, prints a title on both files, and sets all user options to be undefined. (Each `sqOpt` interface 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, lencw, leniw, lenrw, iw(leniw)
      character
     &     cw(lencw)*8
      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` ≤ 0, there will be no SUMMARY file output.

`cw(lencw), iw(leniw), rw(lenrw)` must be the same arrays that are passed to other `sqOpt` 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.

3.4. Subroutine sqMem

This routine estimates the size of the workspace arrays `cw`, `iw`, `rw` required to solve an optimization problem of given dimensions. `sqMem` is not strictly needed in f77 because all workspace must be defined explicitly in the driver program at compile time. It is available for users wishing to allocate storage dynamically in f90 or C.

The actual storage required also depends on the values of `Reduced Hessian dimension` and `Superbasics limit`. If these options have not been set, default values are assumed. Ideally the correct values should be set *before* the call to `sqMem`.

```
      subroutine sqMem
     &   ( INFO, m, n, neA, lencObj, ncolH,
     &     mincw, miniw, minrw,
     &     cw, lencw, iw, leniw, rw, lenrw )
      integer
     &     INFO, m, n, neA, lencObj, ncolH, mincw, miniw, minrw,
     &     lencw, leniw, lenrw, iw(leniw)
      double precision
     &     rw(lenrw)
      character
     &     cw(lencw)*8
```

The arguments `m`, `n`, `neA`, `lencObj`, `ncolH` define the problem being solved and are identical to the arguments used in the call to `sqOpt` (see Section 3.1). For a sequence of problems, `sqMem` may be called once with overestimates of these quantities.

On entry:

`lencw, leniw, lenrw` must be of length at least 500.

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

On exit:

`INFO` reports the result of the call to `sqMem`. Here is a summary of possible values. Further details are given in Section 5.4.

Finished successfully

104 memory requirements estimated

Insufficient storage allocated

81 work arrays must have at least 500 elements

`mincw, miniw, minrw` estimate how much character, integer and real storage is needed to solve the problem.

To use `sqMem`, the first step is to allocate the work arrays. These may be temporary arrays `tmpcw`, `tmpiw`, `tmprw` (say) or the `sqOpt` arrays `cw`, `iw`, `rw`, which will be reallocated after the storage limits are known. Here we illustrate the use of `sqMem` using the same arrays for `sqMem` and `sqOpt`. Note that the `sqMem` arrays are used to store the optional parameters, and so any temporary arrays must be copied into the final `cw`, `iw`, `rw` arrays in order to retain the options.

The work arrays must have length at least 500, so we define

```
      ltmpcw = 500
      ltmpiw = 500
      ltmprw = 500
```

As with all sqOpt routines, sqInit *must* be called to initialize the optional parameters to their default values:

```
      call sqInit
     &   ( iPrint, iSumm, cw, ltmpcw, iw, ltmpiw, rw, ltmprw )
```

This installs ltmpcw, ltmpiw, ltmprw as the default internal upper limits on the sqOpt workspace (see the description of Total real workspace in Section 4.6). They are used to compute the boundaries of any user-defined workspace in cw, iw, or rw.

The next step is to call sqMem to obtain mincw, miniw, minrw as estimates of the storage needed by sqOpt:

```
      call sqMem
     &   ( INFO, m, n, neA, lencObj, ncolH,
     &     mincw, miniw, minrw,
     &     cw, ltmpcw, iw, ltmpiw, rw, ltmprw )
```

The output values of mincw, miniw, minrw may now be used to define the lengths of the sqOpt work arrays:

```
      lencw = mincw
      leniw = miniw
      lenrw = minrw
```

These values may be used in f90 or C to allocate the final work arrays for the problem.

One last step is needed before sqOpt is called. The current upper limits ltmpcw, ltmpiw, ltmprw must be replaced by the estimates mincw, miniw, minrw. This can be done using the option setting routine sqSeti as follows:

```
      Errors = 0                 ! Counts any errors while setting options
      iPrt   = 0                 ! Suppress print   output
      iSum   = 0                 ! Suppress summary output
      call sqSeti
     &   ( 'Total character workspace', lencw, iPrt, iSum, Errors,
     &     cw, ltmpcw, iw, ltmpiw, rw, ltmprw )
      call sqSeti
     &   ( 'Total integer   workspace', leniw, iPrt, iSum, Errors,
     &     cw, ltmpcw, iw, ltmpiw, rw, ltmprw )
      call sqSeti
     &   ( 'Total real      workspace', lenrw, iPrt, iSum, Errors,
     &     cw, ltmpcw, iw, ltmpiw, rw, ltmprw )
```

An alternative way is to call sqInit again with arguments lencw, leniw, lenrw:

```
      call sqInit
     &   ( iPrint, iSumm, cw, lencw, iw, leniw, rw, lenrw )
```

However, this has the twin effects of resetting all options to their default values and reprinting the sqOpt banner (unless iPrint = 0 and iSumm = 0 are set for the PRINT and SUMMARY files).

4. Optional parameters

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. This options may be defined in a file called a *SPECS file*, or may be defined in the calling program using a call to one of the option-setting routines `sqSet`, `sqSeti` and `sqSetr` (see Section 4.4). At any stage of the computation, the current value of an optional parameter may be examined by calling one of the routines `sqGet`, `sqGeti` and `sqGetr` (see Section 4.5).

4.1. The SPECS file

The specs file contains a list of option definitions, 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 call such data a SPECS file because 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.

4.2. 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
  System information               No       * prints more system information

* Problem specification
  Minimize                                  * (opposite of Maximize)
* Feasible point                            * (alternative to Max or Min)
  Infinite Bound size              1.0e+20 *

* 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      *
  Pivot tolerance                  3.7e-11 * $\epsilon^{\frac{2}{3}}$

* LP/QP problems
  QPSolver                       Cholesky   * default
  Crash option                     0        * all slack initial basis
  Iterations limit                 10000    * or $m$ if that is more
  Partial price                    1        * 10 for large LPs
  Superbasics limit                ncolH + 1
  Reduced Hessian dimension        1000     * or Superbasics limit if that is less
  Unbounded step size              1.0e+18 *
  Unbounded objective              1.0e+15 *

* 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

* LU options
  LU factor tolerance              10.0     * limits size of multipliers in $L$
  LU update tolerance              10.0     * the same during updates
  LU singularity tolerance         3.2e-11 *
  LU partial  pivoting                      * default pivot strategy
  LU rook     pivoting                    * use rook pivoting for the LU
  LU complete pivoting                    * use complete pivoting for the LU

* 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       *

* Miscellaneous
  Debug  level                  0         * for developers
  Timing level                  3         * prints cpu times
End of SPECS file checklist
```

4.3. Subroutine sqSpec

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

```
      subroutine sqSpec
     &   ( iSpecs, INFO, cw, lencw, iw, leniw, rw, lenrw )
      integer
     &     iSpecs, INFO, lencw, leniw, lenrw, iw(leniw)
      double precision
     &     rw(lenrw)
      character
     &     cw(lencw)*8
```

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.

`INFO` reports the result of the call to `sqSpec`. Here is a summary of possible values. Further details are given in Section 5.4.

finished successfully

101 OPTIONS file read

errors while reading OPTIONS file

131 no OPTIONS file specified

132 End-of-file encountered while reading OPTIONS file The OPTIONS file may be empty, or lines containing the keywords `Skip` or `Endrun` may imply that all problems should be ignored (see Section 4.1).

133 End-of-file encountered while looking for OPTIONS file. `sqOpt`encountered end-of-file or an `Endrun` line before finding a OPTIONS file. The OPTIONS file may not be properly assigned. Its unit number is defined at run time as an argument of subroutine `sqInit`.

134 `Endrun` found before any valid options

> 134 there were $i =$ `INFO` $- 134$ errors while reading the OPTIONS file

4.4. Subroutines sqSet, sqSeti, sqSetr

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

```
      subroutine sqSet
     &   ( buffer,         iPrint, iSumm, Errors,
     &                     cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqSeti
     &   ( buffer, ivalue, iPrint, iSumm, Errors,
     &                     cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqSetr
     &   ( buffer, rvalue, iPrint, iSumm, Errors,
     &                     cw, lencw, iw, leniw, rw, lenrw )

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

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, Errors, ... )
```

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, Errors, ... )
```

`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, Errors, ...)
```

`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.

`Errors` is the cumulative number of errors, so it should be 0 before the first call in a group of calls to option-setting routines.

On exit:

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

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

4.5. Subroutines sqGet, sqGetc, sqGeti, sqGetr

These routines obtain the current value of a single option or indicate if an option has been set.

```
      integer function sqGet
     &   ( buffer,         Errors, cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqGetc
     &   ( buffer, cvalue, Errors, cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqGeti
     &   ( buffer, ivalue, Errors, cw, lencw, iw, leniw, rw, lenrw )
      subroutine sqGetr
     &   ( buffer, rvalue, Errors, cw, lencw, iw, leniw, rw, lenrw )

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

On entry:

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

`Errors` is the cumulative number of errors, so it should be 0 before the first call in a group of calls to option-getting routines.

On exit:

`sqGet` is 1 if the option contained in buffer has been set, otherwise 0. Use `sqGet` to find if a particular optional parameter has been set. For example: if

```
i = sqGet( 'QPSolver Cholesky', Errors, ... )
```

then i will be 1 if `sqOpt` is using a Cholesky-based QP solver.

`cvalue` is a string associated with the keyword in `buffer`. Use `sqGetc` to obtain the names associated with an MPS file. For example, for the name of the bounds section use

```
call sqGetc( 'Bounds', MyBounds, Errors, ... )
```

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

```
call sqGeti( 'Iterations limit', itnlim, Errors, ... )
```

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

```
call sqGetr( 'LU factor tol', factol, Errors, ... )
```

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

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

4.6. 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 Section 6.3. 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` γ Default = 1.0

This keyword determines the initial weight γ associated with the composite objective in problem Phase2(γ) on Page 10.

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

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

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

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

`Expand frequency` i Default = 10000

This option is part of the EXPAND anti-cycling procedure [6] 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 $\frac{1}{2}\delta$ to δ (in steps of $\frac{1}{2}\delta/i$).

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).

`Feasible point`
see `Minimize`

`Hessian dimension` i Default = $\min\{1000, \texttt{nHcol} + 1\}$
see `Reduced Hessian dimension`

`Insert File` f Default = 0

If $f > 0$, this references a file containing basis information in the format of Section 6.2.

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

`Iterations Limit` k Default = $3 * \texttt{m}$

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 Section 6.3.

- 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 Partial  Pivoting
LU Rook     Pivoting
LU Complete Pivoting
```
Default

The LU factorization implements a Markowitz-type search for a pivot that locally minimizes the fill-in subject to a threshhold pivoting stability criterion. The default option is to use threshhold partial pivoting. The options `LU rook pivoting` and `LU complete pivoting` are more expensive than partial pivoting but are more stable and better at revealing rank.

`LU density     tolerance` r_1 Default = 0.6
`LU singularity tolerance` r_2 Default = $\epsilon^{2/3} \approx$ `3.2e-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 terminated. The remaining matrix is factored by a dense LU procedure. 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 $|U_{jj}| \le r_2$ 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.)

`Minimize` Default
`Maximize`
`Feasible point`

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

The keyword `Feasible point` means "Ignore the objective function" while finding a feasible point for the linear constraints. It can be used to check that the constraints are feasible without altering the call to `sqOpt`.

`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 line 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 Section 6.1.

- 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 $\begin{pmatrix} A & -I \end{pmatrix}$, 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

  $$d_j/\|\pi\| \ge -t \quad \text{or} \quad d_j/\|\pi\| \le t$$

  respectively, and if $|d_j|/\|\pi\| \le t$ 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 = $\epsilon^{2/3} \approx$ `3.7e-11`

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. 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 Section 6.2. For linear programs, this format is compatible with various commercial systems.

`QPSolver   Cholesky` Default
`QPSolver   CG`
`QPSolver   QN`

Specifies the active-set algorithm used to solve the quadratic program in Phase 2. `QPSolver Cholesky` holds the full Cholesky factor R of the reduced Hessian Z^THZ. As the QP iterations proceed, the dimension of R changes with the number of superbasic variables. If the number of superbasic variables needs to increase beyond the value of `Reduced Hessian dimension`, the reduced Hessian cannot be stored and the solver switches to `QPSolver CG`. The Cholesky solver is reactivated if the number of superbasics stabilizes at a value less than `Reduced Hessian dimension`.

`QPSolver QN` solves the QP using a quasi-Newton method similar to that of MINOS. In this case, R is the factor of a quasi-Newton approximate Hessian.

`QPSolver CG` uses an active-set method similar to `QPSolver QN`, but uses the conjugate-gradient method to solve all systems involving the reduced Hessian.

- The Cholesky QP solver is the most robust, but may require a significant amount of computation if there are many superbasics (degrees of freedom).
- The quasi-Newton QP solver does not require computation of the exact R at the start of phase 2. It may be appropriate when the number of superbasics is large but relatively few iterations are needed to reach a solution (e.g., if `sqOpt` is called with a Warm or Hot start).
- The conjugate-gradient QP solver is appropriate for problems with many degrees of freedom (say, more than 1000 superbasics).

`Reduced Hessian dimension` i Default = $\min\{1000, \texttt{ncolH} + 1\}$
same as `Hessian dimension`

This specifies that an $r \times r$ triangular matrix R is to be available for use by the Cholesky QP solver (to define the reduced Hessian according to $R^TR = Z^THZ$).

`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 [2]). 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 Section 5.8.

`Superbasics limit`	i	Default = `ncolH` + 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.

`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.

`System Information`	`No`	Default
`System Information`	`Yes`	

This option allows the knowledgeable user to print some additional information on the progress of the major and minor iterations.

`Timing level`	i	Default = 3

$i = 0$ suppresses output of cpu times. (Intended for installations with dysfunctional timing routines.)

`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.

5. 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 Section 5.8.

5.1. 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 $A_{\tt pp}$ or $I_{\tt pp}$, 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 superbasic variable chosen to become nonbasic.

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

`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 $\mathtt{L} + \mathtt{U}$ 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.

5.2. Basis factorization statistics

If `Print level` ≥ 10, the following items are output to the PRINT file whenever the basis B or the rectangular matrix $B_S = (\,B \;\; S\,)^T$ is factorized.

Note that B_S may be factorized at just some of the iterations. It is immediately followed by a factorization of B itself.

Gaussian elimination is used to compute a sparse LU factorization of B or B_S, where PLP^T and PUQ are lower and upper triangular matrices for some permutation matrices P and Q. Stability is ensured as described under `LU factor tolerance` in Section 4.6.

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`	The number of updates reached the `Factorization Frequency`.
`2`	The nonzeros in the updated factors have increased significantly.
`7`	Not enough storage to update factors.
`10`	Row residuals too large (see the description of `Check Frequency`).
`11`	Ill-conditioning has caused inconsistent results.

`Itn`	The current minor iteration number.
`Nonlin`	The number of nonlinear variables in the current basis B.
`Linear`	The number of linear variables in B.
`Slacks`	The number of slack variables in B.
`B BR BS` or `BT factorize`	The type of LU factorization.

`B`	Periodic factorization of the basis B.
`BR`	More careful rank-revealing factorization of B using threshold rook pivoting. This occurs mainly at the start, if the first basis factors seem singular or ill-conditioned. Followed by a normal `B` factorize.
`BS`	B_S is factorized to choose a well-conditioned B from the current $(\,B \;\; S\,)$. Followed by a normal `B` factorize.
`BT`	Same as `BS` except the current B is tried first and accepted if it appears to be not much more ill-conditioned than after the previous `BS` factorize.

`m`	The number of rows in B or B_S.
`n`	The number of columns in B or B_S. Preceded by "`=`" or "`>`" respectively.

`Elems`	The number of nonzero elements in B or B_S.
`Amax`	The largest nonzero in B or B_S.
`Density`	The percentage nonzero density of B or B_S.
`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 `n` 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.
`Cmpressns`	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.
`Incres`	The percentage increase in the number of nonzeros in L and U relative to the number of nonzeros in B or B_S.
`Utri`	is the number of triangular rows of B or B_S at the top of U.
`lenU`	The number of nonzeros in U.
`Ltol`	The maximum subdiagonal element allowed in L. This is the specified `LU factor tolerance` or a smaller value that is currently being used for greater stability.
`Umax`	The maximum nonzero element in U.
`Ugrwth`	The ratio `Umax/Amax`, which ideally should not be substantially larger than 10.0 or 100.0. If it is orders of magnitude larger, it may be advisable to reduce the `LU factor tolerance` to 5.0, 4.0, 3.0 or 2.0, say (but bigger than 1.0). As long as `Lmax` is not large (say 10.0 or less), max{`Amax`, `Umax`} / `DUmin` gives an estimate of the condition number of B. If this is extremely large, the basis is nearly singular. Slacks are used to replace suspect columns of B and the modified basis is refactored.
`Ltri`	is the number of triangular columns of B or B_S at the left of L.
`dense1`	is the number of columns remaining when the density of the basis matrix being factorized reached 0.3.
`Lmax`	The actual maximum subdiagonal element in L (bounded by `Ltol`).
`Akmax`	The largest nonzero generated at any stage of the LU factorization. (Values much larger than `Amax` indicate instability.)
`growth`	The ratio `Akmax/Amax`. Values much larger than 100 (say) indicate instability.
`bump`	is the size of the "bump" or block to be factorized nontrivially after the triangular rows and columns of B or B_S have been removed.
`dense2`	is the number of columns remaining when the density of the basis matrix being factorized reached 0.6. (The Markowitz pivot strategy searches fewer columns at that stage.)
`DUmax`	The largest diagonal of PUQ.

`DUmin`	The smallest diagonal of PUQ.
`condU`	The ratio `DUmax/DUmin`, which estimates the condition number of U (and of B if `Ltol` is less than 100, say).

5.3. Crash statistics

When `Print Level` $\geq$ 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 Section 4.6). 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.

5.4. EXIT conditions

When `sqOpt` or one of its auxiliary routines terminates, a message is printed that summarizes what happened during the run. The general form of the output message is:

```
SOLVER EXIT  e -- exit condition
SOLVER INFO  i -- informational message
```

where e is an integer that labels the particular *exit condition*, and i is one of several alternative *informational messages* that elaborate on the exit condition. For example, solver `sqOpt` may print the message:

```
SQOPT EXIT 20 -- the problem appears to be unbounded
SQOPT INFO 21 -- unbounded objective
```

Note that in this example, the exit condition gives a broad definition of what happened, while the informational message is more specific about the cause of the termination.

The integer i associated with the informational message is the output value of the argument `INFO`. Note that the integer e associated with the exit condition may always be recovered from `INFO` by stripping off the least significant decimal digit.

Possible exit conditions for `sqOpt` are:

0	Finished successfully
10	The problem appears to be infeasible
20	The problem appears to be unbounded
30	Resource limit error
40	Terminated after numerical difficulties
50	Error in the user-supplied functions
60	Undefined user-supplied functions
70	User requested termination
80	Insufficient storage allocated
90	Input arguments out of range
100	Finished successfully (associated with `sqOpt` auxiliary routines)
110	Errors while processing MPS data
130	Errors while reading the OPTIONS file
140	System error

The exit conditions 0–20 arise when a solution exists (though it may not be optimal). A BASIS file may be saved, and the solution will be output to the PRINT or SOLUTION files if requested.

If exit conditions 80–100 occur during the *first* basis factorization, the primal and dual variables `x` and `pi` will have their original input values. BASIS files will be saved if requested, but certain values in the printed solution will not be meaningful.

5.5. Description of the EXIT messages

Next we describe each message and suggest possible courses of action.

```
EXIT --  0  Finished successfully
INFO --  1  optimality conditions satisfied
INFO --  2  feasible point found
INFO --  4  weak QP minimizer
```

These messages usually indicate a successful run. If requested, the BASIS files will have been saved, and the solution will be printed and/or saved on the SOLUTION file.

For `INFO 1` 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.

For `INFO 4` 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).

One caution about "`Optimality conditions satisfied`". Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. Some information concerning the run can be obtained from the short summary given at the end of the print and summary files. Here is an example from the problem `sqmain2` in the SQOPT distribution:

```
SQOPT  EXIT   0 -- finished successfully
SQOPT  INFO   1 -- optimality conditions satisfied

Problem name                 sqProb 1
No. of iterations                  11   Objective value     -2.0436650381E+06
No. of Hessian products            15   Objective row        0.0000000000E+00
                                        Quadratic objective -2.0436650381E+06
No. of superbasics                  1   No. of basic nonlinears           2
No. of degenerate steps            0  Percentage                      0.00
Max x       (scaled)         3 2.2E-01  Max pi      (scaled)         6 3.1E+07
Max x                        3 6.2E+02  Max pi                       7 9.6E+03
Max Prim inf(scaled)         0 0.0E+00  Max Dual inf(scaled)         5 1.0E-08
Max Primal infeas            0 0.0E+00  Max Dual infeas              9 3.1E-12
```

`Max Primal infeas` refers to the largest bound infeasibility and which variable is involved. If it is too large, consider restarting with a smaller `Feasibility tolerance` (say 10 times smaller) and perhaps `Scale option 0`.

Similarly, `Max Dual infeas` indicates which variable is most likely to be at a non-optimal value. Broadly speaking, if

$$\texttt{Max Dual infeas/Max pi} = 10^{-d},$$

then the objective function would probably change in the dth significant digit if optimization could be continued. If d seems too large, consider restarting with a smaller `Optimality tolerance`.

```
EXIT -- 10  The problem appears to be infeasible
INFO -- 11  infeasible linear constraints
INFO -- 12  infeasible linear equalities
INFO -- 14  infeasibilities minimized
```

This exit occurs if `sqOpt` is unable to find a point that satisfies the constraints.

The output messages are based on a relatively reliable indicator of infeasibility. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. 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.

For the cases `INFO 11` and `12`, the sum of infeasibilities will usually not have been *minimized* when `sqOpt` recognizes that the constraints are infeasible and exits. There may exist other points that have a significantly lower sum of infeasibilities.

If the problem is infeasible and the option `Elastic mode` is nonzero, then `sqOpt` will optimize the original QP objective and the sum of the infeasibilities—suitably weighted using the `Elastic weight` parameter. In elastic mode, some of the bounds on the rows and columns are "elastic"—i.e., they are allowed to violate their specified bounds. 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. If the problem has no feasible solution, `sqOpt` will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, `sqOpt` would locally minimize the constraint violations subject to the nonelastic constraints and bounds.)

```
EXIT -- 20  The problem appears to be unbounded
INFO -- 21  unbounded objective
```

Unboundedness is detected by the simplex method when a nonbasic variable can 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.

```
EXIT -- 30  Resource limit error
INFO -- 31  iteration limit reached
INFO -- 33  the superbasics limit is too small
```

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.

If the superbasics limit is too small, then 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 `Superbasics limit` 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 `Reducd 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.

```
EXIT -- 40  Terminated after numerical difficulties
INFO -- 42  singular basis
INFO -- 43  cannot satisfy the general constraints
INFO -- 44  ill-conditioned null-space basis
```

Termination because of a singular basis 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 U were respectively zero or smaller than a certain tolerance.) The associated variables are replaced by slacks and the modified basis is refactorized, but singularity persists. This must mean that the problem is badly scaled, or the `LU factor tolerance` is too much larger than 1.0.

If the general constraints cannot be satisfied, 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 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. If there are some linear constraints and variables, try `Scale option 1` if scaling has not yet been used.

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 Section 5.2, and set the `LU factor tolerance` to 2.0 (or possibly even smaller, but not less than 1.0).

```
EXIT -- 50  Error in the user-supplied functions
INFO -- 54  the QP Hessian is indefinite
```

An indefinite matrix was detected during the computation of the reduced Hessian factor R such that $R^TR = Z^THZ$. This may be caused by the matrix H being indefinite, i.e., there may exist a vector y such that $y^THy < 0$. In this case, the QP problem is not convex and cannot be solved using this version of `sqOpt`. You should check that `qpHx` is coded correctly and that all relevant components of `Hx` are assigned their correct values.

If `qphx` is coded correctly with H symmetric positive semidefinite, then the problem may be ill-conditioning of the reduced Hessian caused by ill-conditioning in either H or Z. Ill-conditioning in Z may be alleviated by reducing the values of the optional parameters `LU factor tolerance` and `LU update tolerance` (see Page 35).

```
EXIT -- 80  Insufficient storage allocated
INFO -- 81  work arrays must have at least 500 elements
INFO -- 82  not enough character storage
INFO -- 83  not enough integer storage
INFO -- 84  not enough real storage
```

SNOPT cannot start to solve a problem unless the char, int and real work arrays are at least 500 elements.

If the main character, integer or real storage arrays `cw(*)`, `iw(*)` and `rw(*)` are not large enough for the current problem, the routine declaring `cw(*)`, `iw` and `rw` should be recompiled with a larger dimensions for those arrays. The new values should also be assigned to `lencw`, `leniw` and `lenrw`. An estimate of the additional storage required is given in messages preceding the exit.

If `rw(*)` is not large enough, be sure that the `Hessian dimension` is not unreasonably large.

```
EXIT -- 90  Input arguments out of range
INFO -- 91  invalid input argument
INFO -- 92  basis file dimensions do not match this problem
```

This exit occurs if some data associated with the problem is out of range. If `INFO = 91`, at least one input argument of `sqOpt` is invalid. The PRINT and SUMMARY files provide more detail about which argument(s) must be modified.

If `INFO = 92` an OLD BASIS file could not be loaded properly. (In this situation, New BASIS files cannot be saved, and there is no solution to print.) On the first line of the OLD BASIS file, the dimensions labeled `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 another problem.

The basis file state vector will not match the current problem if, 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 line, or some of the `2`'s in the map did not have a corresponding "$j \quad x_j$" entry following the map.

```
EXIT -- 140  System error
INFO -- 141  wrong number of basic variables
INFO -- 142  error in basis package
```

These messages arise if 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.

An inconsistency in the number of basic variables should never happen. It may indicate that the wrong `sqOpt` source files have been compiled, or arguments of incorrect type have been used in the calling sequence.

Check that all integer variables and arrays are declared `integer` in your calling program, and that all "real" variables and arrays are declared consistently. (They should be `double precision` on most machines.)

If there is an 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 values of the arrays `indA`, `Acol` and `locA`.

5.6. 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 $l \le Ax \le u$. The ith constraint is therefore of the form

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

Internally, the constraints take the form $Ax - s = 0$, where s is the set of slack variables (which happen to satisfy the bounds $l \le s \le u$). 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^Tx.

`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 $\alpha \le x_j \le \beta$. 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 $q(x) + c^Tx$. (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 Section 5. 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.

5.7. 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.)

5.8. 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` line 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 example `sqmain` in the SQOPT distribution. The problem is Example 1.1 (Page 5) with $n = 30$ and $x_0 = (\frac{1}{2}, \frac{1}{2}, \cdots, \frac{1}{2})^T$. The number of general constraints is $m = 30$. The output was generated with the option (`Summary frequency = 1`).

```
==============================
S Q O P T  7.2-1    (Jul 2005)
==============================

Begin sqmain (Example program for sqopt)

Nonlinear constraints       0     Linear constraints      30
Nonlinear variables        30     Linear variables         0
Jacobian  variables         0     Objective variables     30
Total constraints          30     Total variables         30

   Itn       dj     Step   nInf  sInf,Objective
     0                        1  1.40000000E+01
     1  1.0E+00  1.0E+00      1  1.30000000E+01
     2  1.0E+00  1.0E+00      1  1.20000000E+01
     3  1.0E+00  1.0E+00      1  1.10000000E+01
     4  1.0E+00  1.0E+00      1  1.00000000E+01
     5  1.0E+00  1.0E+00      1  9.00000000E+00
     6  1.0E+00  1.0E+00      1  8.00000000E+00
     7  1.0E+00  1.0E+00      1  7.00000000E+00
     8  1.0E+00  1.0E+00      1  6.00000000E+00
     9  1.0E+00  1.0E+00      1  5.00000000E+00

   Itn       dj     Step   nInf  sInf,Objective
    10  1.0E+00  1.0E+00      1  4.00000000E+00
    11  1.0E+00  1.0E+00      1  3.00000000E+00
    12  1.0E+00  1.0E+00      1  2.00000000E+00
    13  1.0E+00  1.0E+00      1  1.00000000E+00

This is problem  sqmain.   ncolH =  30

Itn     14: Feasible linear constraints

   Itn       dj     Step   nInf  sInf,Objective  Norm rg     nS
    14                           3.75000000E+00
    15  1.0E+00  9.8E-11         3.75000000E+00
    16 -1.9E+01  9.8E-10         3.75000000E+00
    17 -2.9E+01  1.0E+00         3.26666667E+00  2.9E-08      1
    18           1.0E+00         3.26666667E+00               1

SQOPT  EXIT   0 -- finished successfully
SQOPT  INFO   1 -- optimality conditions satisfied

Problem name                 sqProb
No. of iterations                  18   Objective value      3.2666666667E+00
No. of Hessian products            11   Objective row        0.0000000000E+00
                                        Quadratic objective -4.8333333333E-01
No. of superbasics                  1   No. of basic nonlinears            29
No. of degenerate steps             1   Percentage                       5.56
Max x       (scaled)         30 3.3E-02   Max pi      (scaled)       30 4.7E-01
Max x                        30 3.3E-02   Max pi                     30 4.7E-01
Max Prim inf(scaled)          0 0.0E+00   Max Dual inf(scaled)        0 0.0E+00
Max Primal infeas             0 0.0E+00   Max Dual infeas             0 0.0E+00

Finished         sqmain.
```

6. 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
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 BASIS files are saved in that order every kth iteration, 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:

```
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 economic-growth model MANNE. (The problem has 10 nonlinear constraints, 10 linear constraints, and 30 variables.) Selected column numbers are included to define significant data fields.

6.1. 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` $= m$, the number of rows in the constraint matrix, `N` $= n$, the number of columns in the constraint matrix, and `SB` = the number of superbasic variables. Any undefined names will be printed with a blank entry.

3. A set of $(n+m-1)/80+1$ 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 $j = 1 : n+m$ 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 nonbasic, it may be *fixed* (lower bound = upper bound), or *free* (infinite bounds), or it may be strictly between its bounds. In such cases, `hs`$(j) = 0$. (Free variables will almost always be basic.)

4. A set of lines of the form

$$j \qquad\qquad x_j$$

written with `format(i8, 1p, e24.14)` 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 $(j - n)$th slack, if $j > n$. 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. The list also includes nonbasic variables that lie strictly between their bounds.

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 labeled `M` and `N` on the second line must agree with m and n for the problem that has just been defined. The value labeled `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 its current value (even if it is not equal to one of its bounds).

6.2. 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.

```
sqProb 2  ITN      32    Optimal Soln  NINF      0      OBJ   9.130712687760E-01
OBJ=         RHS=         RNG=         BND=         M=     31 N=     30 SB=    8
0333333330333333333333333333331111111311121212121212121212121330
      49   -1.42537551933494E-02
      57   -3.01054650047995E-02
      55   -2.61425375519345E-02
      47   -1.02908277404919E-02
      53   -2.21796100990641E-02
      45   -6.32790028763443E-03
      51   -1.82166826462067E-02
      43   -2.36497283477702E-03
       0
```

Figure 1: Format of NEW and OLD BASIS files for example `sqmain2` in the SQOPT distribution

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, or zero for free variables.

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.

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.

6.3. 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. `BS` causes *Name* to become basic.
3. `SB` causes *Name* to become superbasic at the specified *Value*.
4. `LL` or `UL` cause *Name* to be nonbasic at the specified *Value*.
5. 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*.)
6. 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.
7. (*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.)
8. (*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          sqProb 2  PUNCH/INSERT
 XU x       2  r       1   5.11608E-19
 XU x       3  r       2   5.11608E-19
 XU x       4  r       3   5.11608E-19
 XU x       5  r       4   5.11608E-19
 XU x       6  r       5   5.11608E-19
 XU x       7  r       6   5.11608E-19
 XU x       8  r       7   5.11608E-19
 XU x       9  r       8   5.11608E-19
 XU x      11  r      10  -3.31931E-20
 XU x      12  r      11  -3.31931E-20
 XU x      13  r      12  -3.31931E-20
 XL x      14  r      13   2.36497E-03
 XU x      15  r      14   2.36497E-03
 XL x      16  r      15   8.69287E-03
 XU x      17  r      16   8.69287E-03
 XL x      18  r      17   1.89837E-02
 XU x      19  r      18   1.89837E-02
 XL x      20  r      19   3.32375E-02
 XU x      21  r      20   3.32375E-02
 XL x      22  r      21   5.14541E-02
 XU x      23  r      22   5.14541E-02
 XL x      24  r      23   7.36337E-02
 XU x      25  r      24   7.36337E-02
 XL x      26  r      25   9.97763E-02
 XU x      27  r      26   9.97763E-02
 XL x      28  r      27   1.29882E-01
 XU x      29  r      28   1.29882E-01
 XL x      30  r      31   1.63950E-01
 SB r      13             -2.36497E-03
 SB r      15             -6.32790E-03
 SB r      17             -1.02908E-02
 SB r      19             -1.42538E-02
 SB r      21             -1.82167E-02
 SB r      23             -2.21796E-02
 SB r      25             -2.61425E-02
 SB r      27             -3.01055E-02
ENDATA
```

Figure 2: Format of PUNCH/INSERT files

```
NAME          sqProb 2        DUMP/LOAD
 LL x       1                0.00000E+00
 BS x       2                5.11608E-19
 BS x       3                5.11608E-19
 BS x       4                5.11608E-19
 BS x       5                5.11608E-19
 BS x       6                5.11608E-19
 BS x       7                5.11608E-19
 BS x       8                5.11608E-19
 BS x       9                5.11608E-19
 LL x      10                0.00000E+00
 BS x      11               -3.31931E-20
 BS x      12               -3.31931E-20
 BS x      13               -3.31931E-20
 BS x      14                2.36497E-03
 BS x      15                2.36497E-03
 .. .       .                .
 BS x      29                1.29882E-01
 BS x      30                1.63950E-01
 UL r       1                0.00000E+00
 UL r       2                0.00000E+00
 UL r       3                0.00000E+00
 UL r       4                0.00000E+00
 UL r       5                0.00000E+00
 UL r       6                0.00000E+00
 UL r       7                0.00000E+00
 UL r       8                0.00000E+00
 BS r       9                9.00555E-19
 UL r      10                0.00000E+00
 UL r      11                0.00000E+00
 UL r      12                0.00000E+00
 SB r      13               -2.36497E-03
 UL r      14                0.00000E+00
 SB r      15               -6.32790E-03
 .. .       .                .
 BS r      30               -1.63950E-01
 LL r      31                1.00000E+00
ENDATA
```

Figure 3: Format of DUMP/LOAD files

6.4. Restarting modified problems

Sections 6.1–6.3 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.

References

[1] S. I. Feldman, D. M. Gay, M. W. Maimone, and N. L. Schryer, *A Fortran-to-C converter*, Computing Science Technical Report 149, AT&T Bell Laboratories, Murray Hill, NJ, 1990.

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

[3] Philip E. Gill and Walter Murray, *Numerically stable methods for quadratic programming*, Math. Program., 14 (1978), pp. 349–372.

[4] Philip E. Gill, Walter Murray, and Michael 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.

[5] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright, *Maintaining LU factors of a general sparse matrix*, Linear Algebra Appl., 88/89 (1987), pp. 239–270.

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

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

[8] 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. Report MS 96-010, Department of Mathematics and Statistics, University of Edinburgh, 1996.