USER'S GUIDE FOR TOMLAB /SNOPT[1]

Kenneth Holmström[2], Anders O. Göran[3] and Marcus M. Edvall[4]

February 13, 2008

*More information available at the TOMLAB home page: http://tomopt.com. E-mail: **tomlab@tomopt.com**.

†Professor in Optimization, Mälardalen University, Department of Mathematics and Physics, P.O. Box 883, SE-721 23 Västerås, Sweden, kenneth.holmstrom@mdh.se.

‡Tomlab Optimization AB, Västerås Technology Park, Trefasgatan 4, SE-721 30 Västerås, Sweden, anders@tomopt.com.

§Tomlab Optimization Inc., 855 Beech St #121, San Diego, CA, USA, medvall@tomopt.com.

Contents

1 Introduction

1.1 Overview

Welcome to the TOMLAB /SNOPT User's Guide. TOMLAB /SNOPT includes a set of solvers and MATLAB embedded interfaces. The solver package includes binaries for the following solvers:

MINOS - For large-scale sparse general nonlinear programming problems.

LP-MINOS - For large-scale sparse linear programming problems.

QP-MINOS - For large-scale sparse quadratic programming problems.

LPOPT - For dense linear programming problems.

QPOPT - For dense convex quadratic programming problems.

SNOPT - For large-scale, sparse, linear and nonlinear programming.

SQOPT - For sparse linear and quadratic programming.

Please visit http://tomopt.com/tomlab/products/sol/ for more information.

The interface between TOMLAB /SNOPT, Matlab and TOMLAB consists of two layers. The first layer gives direct access from Matlab to SNOPT, via calling a Matlab function that calls a pre-compiled MEX file (DLL under Windows, shared library in UNIX) that defines and solves the problem in SNOPT. The second layer is a Matlab function that takes the input in the TOMLAB format, and calls the first layer function. On return the function creates the output in the TOMLAB format.

1.2 Contents of this Manual

- Section 2 gives the basic information needed to run the Matlab interface.
- Section 3 provides all the solver references for MINOS, LP-MINOS, QP-MINOS, LPOPT, QPOPT, SNOPT, and SQOPT.
- Section 4 discusses the use of TOMLAB /SNOPT in more detail.
- Section 6 contains details about MINOS.
- Section 8 provides detailed information about the SNOPT solver in the TOMLAB /SNOPT package for the advanced user.

1.3 Prerequisites

In this manual we assume that the user is familiar with SNOPT, the various SOL Reference Manuals, TOMLAB and the Matlab language.

2 Using the Matlab Interface

The main routines in the two-layer design of the interface are shown in Table 1. Page and section references are given to detailed descriptions on how to use the routines.

Table 1: The interface routines.

Function	Description	Section	Page
minos	The layer one Matlab interface routine, calls the MEX-file interface *minos.dll*	3.1.1	9
minosTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *minos.m*.	3.1.2	15
minosLPTL	The layer two TOMLAB interface routine that calls *minosLPTL.m*. Converts the input *Prob* format before calling *minos.m* and converts back to the output *Result* structure. This option only handles linear programming problems	3.2.1	22
minosQPTL	The layer two TOMLAB interface routine that calls *minosQPTL.m*. Converts the input *Prob* format before calling *minos.m* and converts back to the output *Result* structure. This option only handles quadratic programming problems	3.3.1	28
lpopt	The layer one Matlab interface routine, calls the MEX-file interface *lpopt.dll*	3.4.1	35
lpoptTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *lpopt.m*.	3.4.2	38
qpopt	The layer one Matlab interface routine, calls the MEX-file interface *qpopt.dll*	3.5.1	42
qpoptTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *qpopt.m*.	3.5.2	46
snopt	The layer one Matlab interface routine, calls the MEX-file interface *snopt.dll*	3.6.1	51
snoptTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *snopt.m*.	3.6.2	57
sqopt	The layer one Matlab interface routine, calls the MEX-file interface *sqopt.dll*	3.7.1	66
sqoptTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *sqopt.m*.	3.7.2	71

The SOL control parameters are possible to set from Matlab.

They can be set as inputs to the interface routine *minos* for example and the others. The user sets fields in a structure called *Prob.SOL.optPar*, where the subfield names follow the SOL standard for setting solver options. The following example shows how to set the maximum number of iterations.

```
Prob.SOL.optPar(30)   = 500;    % Setting maximum number of iterations
```

3 TOMLAB /SNOPT Solver Reference

The SNOPT solvers are a set of Fortran solvers that were developed by the Stanford Systems Optimization Laboratory (SOL). Table 2 lists the solvers included in TOMLAB /SNOPT. The solvers are called using a set of MEX-file interfaces developed as part of TOMLAB. All functionality of the SOL solvers are available and changeable in the TOMLAB framework in Matlab.

Detailed descriptions of the TOMLAB /SNOPT solvers are given in the following sections. Also see the M-file help for each solver.

The solvers reference guides for the TOMLAB /SNOPT solvers are available for download from the TOMLAB home page http://tomopt.com. There is also detailed instruction for using the solvers in Section 4. Extensive TOMLAB m-file help is also available, for example *help snoptTL* in Matlab will display the features of the SNOPT solver using the TOMLAB format.

TOMLAB /SNOPT solves **nonlinear optimization** problems (**con**) defined as

$$\begin{aligned} \min_x \quad & f(x) \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \\ c_L & \leq & c(x) & \leq & c_U \end{array} \end{aligned} \tag{1}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{m_1}$ and $c_L, c(x), c_U \in \mathbb{R}^{m_2}$.

quadratic programming (**qp**) problems defined as

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2} x^T F x + c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{2}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

linear programming (**lp**) problems defined as

$$\begin{aligned} \min_x \quad & f(x) = c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{3}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

Table 2: The SOL optimization solvers in TOMLAB /SNOPT.

Function	Description	Reference	Page
MINOS 5.5	Sparse linear and nonlinear programming with linear and nonlinear constraints.	[25]	
LP-MINOS	A special version of the *MINOS 5.5* MEX-file interface for sparse linear programming.	[25]	
QP-MINOS	A special version of the *MINOS 5.5* MEX-file interface for sparse quadratic programming.	[25]	
LPOPT 1.0-10	Dense linear programming.	[15]	
QPOPT 1.0-10	Non-convex quadratic programming with dense constraint matrix and sparse or dense quadratic matrix.	[15]	
SNOPT 6.1-1	Large, sparse linear and nonlinear programming with linear and nonlinear constraints.	[18, 16]	
SQOPT 6.1-1	Sparse convex quadratic programming.	[17]	

3.1 MINOS

3.1.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.1.2 for information on how to use MINOS with TOMLAB.

Purpose
minos solves nonlinear optimization problems defined as

$$\begin{aligned} \min_x \quad & f(x) \\ s/t \quad & b_L \leq \begin{matrix} x \\ Ax \\ c(x) \end{matrix} \leq b_U \end{aligned}, \tag{4}$$

where $x \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{n+m_1+m_2}$ and $c(x) \in \mathbb{R}^{m_2}$.

or quadratic optimization problems defined as

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2} x^T F x + c^T x \\ s/t \quad & b_L \leq \begin{matrix} x \\ g(x) \\ Ax \end{matrix} \leq b_U \end{aligned}, \tag{5}$$

where $c, x \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

The full input matrix A has three parts A = [d/dx g(x); A; c'];

Calling Syntax

The file 'funfdf.m' must be defined and contain: function [mode, f, g] = funfdf(x, Prob, mode, nstate) to compute the objective function f and the gradient g at the point x.

The file 'funcdc.m' must be defined and contain: function [mode ,c ,dcS] = funcdc(x, Prob, mode, nstate) to compute the nonlinear constraint value c and the constraint Jacobian dcS for the nonlinear constraints at the point x.

NOTE: The matrix dcS MUST be a SPARSE MATLAB matrix. Do dcS = sparse(dcS); after dcS has been computed.

[hs, xs, pi, rc, Inform, nS, nInf, sInf, Obj, iwCount, gObj, fCon, gCon] = minos(H, A, bl, bu, nnCon, nnObj, nnJac, Prob, iObj, optPar, Warm, hs, xs, pi, nS, SpecsFile, PrintFile, SummFile, PriLev, ObjAdd, moremem, ProbName);

Description of Inputs

The following fields are used:

H Matrix n x n in a quadratic programming (QP) problem. DENSE or SPARSE. Leave empty if LP, or NLP problem.

A Constraint matrix, m x n SPARSE (nonlinear, linear and objective) m > 0 always!!! Define dummy constraint for unconstrained problems.

bl Lower bounds on (x,g(x),Ax,c').

bu Upper bounds on (x,g(x),Ax,c').

NOTE! The bl and bu values for the last nonlinear constraint c must have reverse signs and be put in each other places: If $c_L <= c(x) <= c_U$, then $bl = -c_U$ and $bu = -c_L$. This is because the bounds acts as the constraints on the slack variables for the nonlinear constraints.

nnCon Number of nonlinear constraints.

nnObj Number of nonlinear objective variables.

nnJac Number of nonlinear Jacobian variables.

Prob Must be a structure. No check is made in the MEX interface. If TOMLAB calls minos, then Prob is the standard TOMLAB problem structure, otherwise the user should set:

Prob.P = ProblemNumber, where ProblemNumber is some integer.

The following fields are used:, continued

If the problem is a LP or QP problem (H defined), the user does not have to specify anything else in the structure.

For a general nonlinear objective or nonlinear constraints names of two user written routines must be given:

funfdf, actual name stored in Prob.FUNCS.fg, with syntax [mode, f, g] = funfdf(x, Prob, mode, nstate).

funcdc, actual name stored in Prob.FUNCS.cdc, with syntax [mode, c, dcS] = funcdc(x, Prob, mode, nstate).

MINOS is calling the TOMLAB routines nlp_fg.m and nlp_cdcS.m in the callback, and they call funfdf and funcdc, respectively.

If these fields in Prob are empty (Prob.FUNCS.fg, Prob.FUNCS.cdc), the TOMLAB callback routines calls the usual function routines. Then the Prob struct should be normally defined, and the fields Prob.FUNCS.f, Prob.FUNCS.g, Prob.FUNCS.c, Prob.FUNCS.dc be set in the normal way (e.g. by the routine mFiles.m, or one of the Assign-routines like conAssign.m).

If the mode parameter is 0, funfdf should return f, otherwise both f and the gradient vector g. If the mode parameter is 0, funcdc should return c, otherwise both c and dcS. Note that each row in dcS corresponds to a constraint, and that dcS must be a SPARSE matrix.

The user could also write his own versions of the routines nlp_fg.m and nlp_cdcS.m and put them before in the path.

iObj Says which row of A is a free row containing a linear objective vector c. If there is no such vector, iObj = 0. Otherwise, this row must come after any nonlinear rows, so that nnCon <= iObj <= m.

optPar Vector with optimization parameters overriding defaults and the optionally specified SPECS file. If using only default options, set optPar as an empty matrix.

Warm Flag, if true: warm start. Default cold start (if empty). If 'Warm Start' xs, nS and hs must be supplied with correct values.

hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu) 2=superbasic (between bounds), 3=basic (between bounds).

The following fields are used:, continued

xs Initial vector, optionally including m slacks at the end. If warm start, full xs must be supplied.

pi Lagrangian multipliers for the nnCon nonlinear constraints. If empty, set as 0.

nS # of superbasics. Only used if calling again with a Warm Start.

SpecsFile Name of the SPECS input parameter file, see TOMLAB Guide.

PrintFile Name of the Print file. Name includes the path, maximal number of characters = 500.

SummFile Name of the Summary file. Name includes the path, maximal number of characters = 500.

PriLev Printing level in the minos m-file and minos MEX-interface.

= 0 Silent
= 1 Summary information
= 2 More detailed information

ObjAdd Constant added to the objective for printing purposes, typically 0.

morememem Add extra memory for the sparse LU, might speed up the optimization. 1E6 is 10MB of memory. If empty, set as 0.

ProbName Name of the problem. ¡=100 characters are used in the MEX interface. In the MINOS solver the first 8 characters are used in the printed solution and in some routines that output BASIS files. Blank is OK.

Description of Outputs

The following fields are used:

hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu) 2=superbasic (between bounds), 3=basic (between bounds).
Basic and superbasic variables may be outside their bounds by as much as the `Feasibility tolerance`. Note that if scaling is specified, the `Feasibility tolerance` applies to the variables of the *scaled* problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled. Check the "Primal infeasibility" printed after the `EXIT` message.

The following fields are used:, continued

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 `xn(`j`)` lies strictly between its bounds.

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

xs Solution vector (n+m by 1) with n decision variable values together with the m slack variables.

pi Lagrangian multipliers (dual solution vector) (m x 1 vector)

rc Vector of reduced costs, $g - (\ A \ \ I\ )^T\pi$, where g is the gradient of the objective function if `xn` is feasible, or the gradient of the Phase-1 objective otherwise. If `ninf` $= 0$, the last m entries are $-\pi$. Reduced costs vector is of n+m length.

Inform Result of MINOS run.

0 Optimal solution found.
1 The problem is infeasible.
2 The problem is unbounded (or badly scaled).
3 Too many iterations.
4 Apparent stall. The solution has not changed for a large number of iterations (e.g. 1000).
5 The `Superbasics limit` is too small.
6 User requested termination (by returning bad value).
7 Gradient seems to be giving incorrect derivatives.
8 Jacobian seems to be giving incorrect derivatives.
9 The current point cannot be improved.
10 Numerical error in trying to satisfy the linear constraints (or the linearized nonlinear constraints). The basis is very ill-conditioned.
11 Cannot find a superbasic to replace a basic variable.
12 Basis factorization requested twice in a row. Should probably be treated as `inform` $= 9$.
13 Near-optimal solution found. Should probably be treated as `inform` $= 9$.

20 Not enough storage for the basis factorization.
21 Error in basis package.
22 The basis is singular after several attempts to factorize it (and add slacks where necessary).
30 An OLD BASIS file had dimensions that did not match the current problem.
32 System error. Wrong number of basic variables.
40 Fatal errors in the MPS file.
41 Not enough storage to read the MPS file.
42 Not enough storage to solve the problem.

The following fields are used:, continued

nS	# of superbasics.
nInf	Number of infeasibilities.
sInf	Sum of infeasibilities.
Obj	Objective function value at optimum.
iwCount	Number of iterations (major and minor), function and constraint calls.
gObj	Gradient of the nonlinear objective.
fCon	Nonlinear constraint vector.
gCon	Gradient vector (non-zeros) of the nonlinear constraint vector.

3.1.2 Using TOMLAB

Purpose
minosTL solves nonlinear optimization problems defined as

$$\begin{array}{lrcccl} \min\limits_{x} & f(x) & & & & \\ & & & & & \\ & x_L & \leq & x & \leq & x_U, \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{6}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{m_1}$ and $c_L, c(x), c_U \in \mathbb{R}^{m_2}$.

Calling Syntax
Using the driver routine *tomRun*:

```
Prob = ◇Assign( ... );
Result = tomRun('minos', Prob, 1);
```

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
c_L, c_U	Bounds on nonlinear constraints.
A	Linear constraint matrix.
QP.c	Linear coefficients in objective function.
PriLevOpt	Print level.
WarmStart	If true, use warm start, otherwise cold start.
SOL.xs	Solution and slacks from previous run.
SOL.hs	State for solution and slacks from previous run.
SOL.nS	Number of superbasics from previous run.
SOL.moremem	Add more memory if MINOS stops with not enough storage message. 1E6 is 10MB of memory. Default n+m1+m2 (number of variables + linear + nonlinear constraints).

Prob, The following fields are used:, continued

SOL.SpecsFile	Name of user defined SPECS file, read BEFORE optPar() is used.
SOL.PrintFile	Name of SOL Print file. Amount and type of printing determined by SPECS parameters or optPar parameters.
SOL.SummFile	Name of SOL Summary File.
SOL.optPar	Elements > -999 takes precedence over corresponding TOMLAB params. See Table 34.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Gradient of the function.
c_k	Nonlinear constraint residuals.
cJac	Nonlinear constraint gradients.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
cState	State of nonlinear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from minos.m (similar to TOMLAB).
Inform	Result of MINOS run.

0 Optimal solution found.
1 The problem is infeasible.
2 The problem is unbounded (or badly scaled).
3 Too many iterations.
4 Apparent stall. The solution has not changed for a large number of iterations (e.g. 1000).
5 The `Superbasics limit` is too small.
6 User requested termination (by returning bad value).

Result, The following fields are used:, continued

7 Gradient seems to be giving incorrect derivatives.
8 Jacobian seems to be giving incorrect derivatives.
9 The current point cannot be improved.
10 Numerical error in trying to satisfy the linear constraints (or the linearized nonlinear constraints). The basis is very ill-conditioned.
11 Cannot find a superbasic to replace a basic variable.
12 Basis factorization requested twice in a row. Should probably be treated as `inform` = 9.
13 Near-optimal solution found. Should probably be treated as `inform` = 9.

20 Not enough storage for the basis factorization.
21 Error in basis package.
22 The basis is singular after several attempts to factorize it (and add slacks where necessary).
30 An OLD BASIS file had dimensions that did not match the current problem.
32 System error. Wrong number of basic variables.
40 Fatal errors in the MPS file.
41 Not enough storage to read the MPS file.
42 Not enough storage to solve the problem.

rc Vector of reduced costs, $g - (\ A \ \ I\ )^T\pi$, where g is the gradient of the objective function if `xn` is feasible, or the gradient of the Phase-1 objective otherwise. If `ninf` = 0, the last m entries are $-\pi$. Reduced costs vector is of n+m length.

Iter Number of iterations.
FuncEv Number of function evaluations.
GradEv Number of gradient evaluations.
ConstrEv Number of constraint evaluations.

QP.B Basis vector in TOMLAB QP standard.

MinorIter Number of minor iterations.

Solver Name of the solver (minos).
SolverAlgorithm Description of the solver.

SOL.xs Solution and slack variables.

SOL.hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu) 2=superbasic (between bounds), 3=basic (between bounds).

Result, The following fields are used:, continued

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

Very occasionally some nonbasic variables may be outside their bounds by as much as the `Feasibility tolerance`, and there may be some nonbasics for which `xn(`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).

SOL.nS # of superbasics.

SOL.nInf # of infeasibilities.

SOL.sInf Sum of infeasibilities.

3.1.3 optPar

Description

Use missing value (-999 or less), when no change of parameter setting is wanted. The default value will then be used by MINOS, unless the value is altered in the SPECS file.
Definition: nnL = max(nnObj,nnJac))

Description of Inputs

Table 7: The following fields are used:

#	SPECS keyword text	Lower	Default	Upper	Comment
Printing					
1.	PRINT LEVEL	0	0	11111	JFLXB: Jac, fCon, lambda, x, B=LU stats
Frequencies I					
5.	PRINT FREQUENCY	0	100		
6.	SUMMARY FREQUENCY	0	100		
7.	SOLUTION YES/NO	0	1	1	1 = YES; 0 = NO
8.	SUPPRESS PARAMETERS	0	0	1	1 = True
Convergence Tolerances					
9.	ROW TOLERANCE	> 0	1E-6		
10.	OPTIMALITY TOLERANCE	> 0	$max(1E-6, (10eps_R)^{0.5}) = 1.73E\text{-}6$		
11.	FEASIBILITY TOLERANCE	> 0	1E-6		
Derivative checking					
13.	VERIFY LEVEL	-1	-1	3	-1,0,1,2,3
14.	START OBJECTIVE CHECK AT COL	0	1	nnObj	
15.	STOP OBJECTIVE CHECK AT COL	0	nnObj	nnObj	
16.	START CONSTRAINT CHECK AT COL	0	1	nnJac	
17.	STOP CONSTRAINT CHECK AT COL	0	nnJac	nnJac	
Scaling					
18.	SCALE OPTION See Section 6.2 for more information.	0	1 or 2	2	2 if LP,1 if NLP
19.	SCALE TOLERANCE	> 0	0.9	< 1	
20.	SCALE PRINT	0	0	1	1 = True
21.	CRASH TOLERANCE	0	0.1	< 1	
Other Tolerances					
22.	LINESEARCH TOLERANCE	> 0	0.1	< 1	
LU I					

Table 7: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
23.	LU FACTORIZATION TOLERANCE	1	100 or 5		100 if LP
24.	LU UPDATE TOLERANCE	1	10 or 5		10 if LP
25.	LU SWAP TOLERANCE	> 0	1.22E-4		$eps^{1/4}$
26.	LU SINGULARITY TOLERANCE	> 0	3.25E-11		$eps^{0.67}$
LP or LC subproblems					
27.	PIVOT TOLERANCE	> 0	3.25E-11		$eps^{0.67}$
28.	CRASH OPTION	0	3	3	0,1,2,3
29.	WEIGHT ON LINEAR OBJECTIVE	0.0	0.0		during Phase 1
30.	ITERATIONS LIMIT	0	3(m+m3) + 10nnL		
	m3=1 if length(Prob.QP.c) > 0, otherwise m3=0.				
	TOMLAB default: max(10000,3(m+m3) + 10nnL).				
31.	PARTIAL PRICE	1	10 or 1		10 for LP
SLC method					
32.	MAXIMIZE	0	0	1	1=maximize
33.	LAGRANGIAN	0	1	1	1=YES, 0=NO
34.	PENALTY PARAMETER	0.0	1.0		
35.	MAJOR ITERATIONS LIMIT	> 0	50		
36.	MINOR ITERATIONS LIMIT	> 0	40		
37.	MAJOR DAMPING PARAMETER	> 0	2.0		
38.	MINOR DAMPING PARAMETER	> 0	2.0		
39.	DERIVATIVE LEVEL	0	3	3	0,1,2,3
	Is always set by minosTL dependent on Prob.ConsDiff, Prob.NumDiff.				
40.	RADIUS OF CONVERGENCE	0.0	0.01		
41.	FUNCTION PRECISION	> 0	3.0E-13		$eps^{0.8} = eps_R$
42.	DIFFERENCE INTERVAL	> 0	5.48E-7		$eps^{0.4}$
43.	CENTRAL DIFFERENCE INTERVAL	> 0	6.69E-5		$eps^{0.8/3}$
44.	COMPLETION	0	1 LC, 0 NC	1	0=PARTIAL 1=FULL
45.	UNBOUNDED STEP SIZE	> 0	1E10		
46.	UNBOUNDED OBJECTIVE	> 0	1E20		
Hessian approximation					
47.	HESSIAN DIMENSION	1	50	1+nnL	
48.	SUPERBASICS LIMIT	1	50	1+nnL	
	TOMLAB default (to avoid termination with Superbasics Limit too small):				
	If n <= 5000: $max(50, n + 1)$				
	If n > 5000: $max(500, n + 200 - size(A, 1) - length(c_L))$				
	Avoid setting REDUCED HESSIAN (number of columns in reduced Hessian).				
	It will then be set to the same value as the SUPERBASICS LIMIT by MINOS.				
Frequencies II					

Table 7: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
51.	CHECK FREQUENCY	> 0	60		
52.	EXPAND FREQUENCY	> 0	10000		
53.	FACTORIZATION FREQUENCY	> 0	50		
LU II					
63.	LU PARTIAL PIVOTING	0	0	3	0=partial
	or LU COMPLETE PIVOTING				1=complete
	or LU ROOK PIVOTING				2=rook
Additional parameters					
67.	AIJ TOLERANCE	0	1E-10		
	Elements $\|a(i,j)\| < AIJ$ TOLERANCE are set as 0				
70.	SUBSPACE	> 0	0.5	1	Subspace tolerance
	Convergence tolerance in current subspace before consider moving off another constraint.				

3.2 LP-MINOS

3.2.1 Using TOMLAB

Purpose
minosLPTL solves linear optimization problems defined as

$$\begin{array}{lll} \min\limits_{x} & f(x) = c^T x & \\ & & \\ s/t & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{array} \tag{7}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

Calling Syntax
Using the driver routine *tomRun*:

```
Prob = lpAssign( ... );
Result = tomRun('lp-minos', Prob ... );
```

or

```
Prob = ProbCheck( ... );
Result = minosLPTL(Prob);
```

Call Prob = lpAssign(...) or Prob = ProbDef to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
A	Linear constraint matrix.
QP.c	Linear coefficients in objective function.
PriLevOpt	Print level.
WarmStart	If true, use warm start, otherwise cold start.
SOL.xs	Solution and slacks from previous run.
SOL.hs	State for solution and slacks from previous run.
SOL.nS	Number of superbasics from previous run (always 0 for LP).

Prob, The following fields are used:, continued

SOL.moremem	Add more memory if MINOS stops with not enough storage message. 1E6 is 10MB of memory. Default n+m (number of variables + linear constraints).
SOL.SpecsFile	Name of user defined SPECS file, read BEFORE optPar() is used.
SOL.PrintFile	Name of SOL Print file. Amount and type of printing determined by SPECS parameters or optPar parameters.
SOL.SummFile	Name of SOL Summary File.
SOL.optPar	Elements > -999 takes precedence over corresponding TOMLAB params. See Table 34.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Gradient c (linear objective).
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from minos.m (similar to TOMLAB).
Inform	Result of MINOS run.

0 Optimal solution found.
1 The problem is infeasible.
2 The problem is unbounded (or badly scaled).
3 Too many iterations.
4 Apparent stall. The solution has not changed for a large number of iterations (e.g. 1000).
5 The `Superbasics limit` is too small.
6 User requested termination (by returning bad value).
7 Gradient seems to be giving incorrect derivatives.
8 Jacobian seems to be giving incorrect derivatives.
9 The current point cannot be improved.

Result, The following fields are used:, continued

10 Numerical error in trying to satisfy the linear constraints (or the linearized nonlinear constraints). The basis is very ill-conditioned.
11 Cannot find a superbasic to replace a basic variable.
12 Basis factorization requested twice in a row. Should probably be treated as `inform` = 9.
13 Near-optimal solution found. Should probably be treated as `inform` = 9.

20 Not enough storage for the basis factorization.
21 Error in basis package.
22 The basis is singular after several attempts to factorize it (and add slacks where necessary).
30 An OLD BASIS file had dimensions that did not match the current problem.
32 System error. Wrong number of basic variables.
40 Fatal errors in the MPS file.
41 Not enough storage to read the MPS file.
42 Not enough storage to solve the problem.

rc Vector of reduced costs, $g - (\ A\ \ I\ )^T\pi$, where g is the gradient of the objective function if `xn` is feasible, or the gradient of the Phase-1 objective otherwise. If `ninf` = 0, the last m entries are $-\pi$. Reduced costs vector is of n+m length.

Iter Number of iterations.
FuncEv Number of function evaluations.
ConstrEv Number of constraint evaluations.

QP.B Basis vector in TOMLAB QP standard.

MinorIter Number of minor iterations.

Solver Name of the solver (minos).
SolverAlgorithm Description of the solver.

SOL.xs Solution and slack variables.

SOL.hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu) 2=superbasic (between bounds), 3=basic (between bounds).
Basic and superbasic variables may be outside their bounds by as much as the `Feasibility tolerance`. Note that if scaling is specified, the `Feasibility tolerance` applies to the variables of the *scaled* problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled. Check the "Primal infeasibility" printed after the `EXIT` message.

Result, The following fields are used:, continued

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 `xn(`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).

SOL.nS # of superbasics.

SOL.nInf # of infeasibilities.

SOL.sInf Sum of infeasibilities.

3.2.2 optPar

Description

Use missing value (-999 or less), when no change of parameter setting is wanted. The default value will then be used by MINOS, unless the value is altered in the SPECS file.

Description of Inputs

Table 10: The following fields are used:

#	SPECS keyword text	Lower	Default	Upper	Comment
Printing					
1.	PRINT LEVEL	0	0	1	0=brief 1=LU stats
Frequencies I					
5.	PRINT FREQUENCY	0	100		
6.	SUMMARY FREQUENCY	0	100		
7.	SOLUTION YES/NO	0	1	1	1 = YES; 0 = NO
8.	SUPPRESS PARAMETERS	0	0	1	1 = True
Convergence Tolerances					
10.	OPTIMALITY TOLERANCE	> 0	1E-6		
11.	FEASIBILITY TOLERANCE	> 0	1E-6		
Scaling					
18.	SCALE OPTION See Section 6.2 for more information.	0	2	2	0,1,2
19.	SCALE TOLERANCE	> 0	0.9	< 1	
20.	SCALE PRINT	0	0	1	1 = True
21.	CRASH TOLERANCE	0	0.1	< 1	
LU I					
23.	LU FACTORIZATION TOLERANCE	1	100		
24.	LU UPDATE TOLERANCE	1	10		
25.	LU SWAP TOLERANCE	> 0	1.22e-4		$eps^{1/4}$
26.	LU SINGULARITY TOLERANCE	> 0	3.25E-11		$eps^{0.67}$
LP parameters					
27.	PIVOT TOLERANCE	> 0	3.25E-11		$eps^{0.67}$
28.	CRASH OPTION	0	3	3	0,1,2,3
29.	WEIGHT ON LINEAR OBJECTIVE	0.0	0.0		during Phase 1
30.	ITERATIONS LIMIT	0	3m		
31.	PARTIAL PRICE	1	10		

Table 10: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
Solve with tight or loose tols (applies to LP but probably not useful)					
44.	COMPLETION	0	1	1	0=PARTIAL 1=FULL
Frequencies II					
51.	CHECK FREQUENCY	> 0	60		
52.	EXPAND FREQUENCY	> 0	10000		
53.	FACTORIZATION FREQUENCY	> 0	100		
LU II					
63.	LU PARTIAL PIVOTING or LU COMPLETE PIVOTING or LU ROOK PIVOTING	0	0	3	0=partial 1=complete 2=rook
Additional parameters					
67.	AIJ TOLERANCE Elements $\|a(i,j)\| < AIJ$ TOLERANCE are set as 0	0	1E-10		
70.	SUBSPACE Convergence tolerance in current subspace before consider moving off another constraint.	> 0	0.5	1	Subspace tolerance

3.3 QP-MINOS

3.3.1 Using TOMLAB

Purpose
minosQPTL solves quadratic optimization problems defined as

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2} x^T F x + c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{8}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

Calling Syntax
Using the driver routine *tomRun*:

```
Prob = qpAssign( ... );
Result = tomRun('qp-minos', Prob ... );
```

or

```
Prob = ProbCheck( ... );
Result = minosQPTL(Prob);
```

Call Prob = qpAssign(...) or Prob=ProbDef to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
A	Linear constraint matrix.
QP.c	Linear coefficients in objective function.
QP.F	Quadratic matrix of size nnObj x nnObj. nnObj < n is OK.
PriLevOpt	Print level.
WarmStart	If true, use warm start, otherwise cold start.
SOL.xs	Solution and slacks from previous run.

Prob, The following fields are used:, continued

SOL.hs	State for solution and slacks from previous run.
SOL.nS	Number of superbasics from previous run.
SOL.moremem	Add more memory if MINOS stops with not enough storage message. 1E6 is 10MB of memory. Default n+m (number of variables + linear constraints).
SOL.SpecsFile	Name of user defined SPECS file, read BEFORE optPar() is used.
SOL.PrintFile	Name of SOL Print file. Amount and type of printing determined by SPECS parameters or optPar parameters.
SOL.SummFile	Name of SOL Summary File.
SOL.optPar	Elements > -999 takes precedence over corresponding TOMLAB params. See Table 34.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Gradient c (linear objective).
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from minos.m (similar to TOMLAB).
Inform	Result of MINOS run.

0 Optimal solution found.
1 The problem is infeasible.
2 The problem is unbounded (or badly scaled).
3 Too many iterations.
4 Apparent stall. The solution has not changed for a large number of iterations (e.g. 1000).
5 The `Superbasics limit` is too small.

Result, The following fields are used:, continued

6 User requested termination (by returning bad value).
7 Gradient seems to be giving incorrect derivatives.
8 Jacobian seems to be giving incorrect derivatives.
9 The current point cannot be improved.
10 Numerical error in trying to satisfy the linear constraints (or the linearized nonlinear constraints). The basis is very ill-conditioned.
11 Cannot find a superbasic to replace a basic variable.
12 Basis factorization requested twice in a row. Should probably be treated as `inform` = 9.
13 Near-optimal solution found. Should probably be treated as `inform` = 9.

20 Not enough storage for the basis factorization.
21 Error in basis package.
22 The basis is singular after several attempts to factorize it (and add slacks where necessary).
30 An OLD BASIS file had dimensions that did not match the current problem.
32 System error. Wrong number of basic variables.
40 Fatal errors in the MPS file.
41 Not enough storage to read the MPS file.
42 Not enough storage to solve the problem.

rc Vector of reduced costs, $g - (\ A \ \ I\ )^T\pi$, where g is the gradient of the objective function if `xn` is feasible, or the gradient of the Phase-1 objective otherwise. If `ninf` = 0, the last m entries are $-\pi$. Reduced costs vector is of n+m length.

Iter Number of iterations.
FuncEv Number of function evaluations. Set to Iter.
GradEv Number of gradient evaluations. Set to Iter.
ConstrEv Number of constraint evaluations. Set to 0.

QP.B Basis vector in TOMLAB QP standard.

MinorIter Number of minor iterations. NOT SET.

Solver Name of the solver (minos).
SolverAlgorithm Description of the solver.

SOL.xs Solution and slack variables.

SOL.hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu) 2=superbasic (between bounds), 3=basic (between bounds).

Result, The following fields are used:, continued

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

Very occasionally some nonbasic variables may be outside their bounds by as much as the `Feasibility tolerance`, and there may be some nonbasics for which `xn(`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).

SOL.nS # of superbasics.

SOL.nInf # of infeasibilities.

SOL.sInf Sum of infeasibilities.

3.3.2 optPar

Description

Use missing value (-999 or less), when no change of parameter setting is wanted. The default value will then be used by MINOS, unless the value is altered in the SPECS file.

Description of Inputs

Table 13: The following fields are used:

#	SPECS keyword text	Lower	Default	Upper	Comment
Printing					
1.	PRINT LEVEL	0	0	1	0=brief 1=LU stats
Frequencies I					
5.	PRINT FREQUENCY	0	100		
6.	SUMMARY FREQUENCY	0	1		
7.	SOLUTION YES/NO	0	1	1	1 = YES; 0 = NO
8.	SUPPRESS PARAMETERS	0	0	1	1 = True
Convergence Tolerances					
10.	OPTIMALITY TOLERANCE	> 0	$max(1E-6, (10eps_R)^{0.5})$ = 1.73E-6		
11.	FEASIBILITY TOLERANCE	> 0	1E-6		
Derivative checking					
13.	VERIFY LEVEL	-1	-1	3	-1,0,1,2,3
14.	START OBJECTIVE CHECK AT COL	0	1	nnObj	
15.	STOP OBJECTIVE CHECK AT COL	0	nnObj	nnObj	
Scaling					
18.	SCALE OPTION See Section 6.2 for more information.	0	1	2	
19.	SCALE TOLERANCE	> 0	0.9	< 1	
20.	SCALE PRINT	0	0	1	1 = True
21.	CRASH TOLERANCE	0	0.1	< 1	
Other Tolerances					
22.	LINESEARCH TOLERANCE	> 0	0.1	< 1	
LU I					
23.	LU FACTOR TOLERANCE	1	5.0		
24.	LU UPDATE TOLERANCE	1	5.0		
25.	LU SWAP TOLERANCE	> 0	1.22E-4		$eps^{1/4}$
26.	LU SINGULARITY TOLERANCE	> 0	3.25E-11		$eps^{0.67}$

Table 13: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
LP parameters					
27.	PIVOT TOLERANCE	> 0	3.25E-11		$eps^{0.67}$
28.	CRASH OPTION	0	3	3	0,1,2,3
29.	WEIGHT ON LINEAR OBJECTIVE	0.0	0.0		during Phase 1
30.	ITERATIONS LIMIT	0	3(m+m3) + 10nnL		
	m3=1 if length(Prob.QP.c) > 0, otherwise m3=0.				
	TOMLAB default: max(10000,3(m+m3) + 10nnL).				
31.	PARTIAL PRICE	1	1		
32.	MAXIMIZE	0	0	1	1=maximize
Reduced-gradient method					
39.	DERIVATIVE LEVEL	0	3	3	0,1,2,3
	Is always set by minosqpTL to 3.				
41.	FUNCTION PRECISION	> 0	3.0E-13		$eps^{0.8} = eps_R$
42.	DIFFERENCE INTERVAL	> 0	5.48E-8		$eps^{0.4}$
43.	CENTRAL DIFFERENCE INTERVAL	> 0	6.69E-5		$eps^{0.8/3}$
44.	COMPLETION	0	1 LC, 0 NC	1	0=PARTIAL 1=FULL
45.	UNBOUNDED STEP SIZE	> 0	1E10		
46.	UNBOUNDED OBJECTIVE	> 0	1E20		
Hessian approximation					
47.	HESSIAN DIMENSION	1	50	1+nnL	
48.	SUPERBASICS LIMIT	1	50	1+nnL	
	TOMLAB default (to avoid termination with Superbasics Limit too small):				
	If n <= 5000: $max(50, n + 1)$				
	If n > 5000: $max(500, n + 200 - size(A, 1) - length(c_L))$				
	Avoid setting REDUCED HESSIAN (number of columns in reduced Hessian).				
	It will then be set to the same value as the SUPERBASICS LIMIT by MINOS.				
Frequencies II					
51.	CHECK FREQUENCY	> 0	60		
52.	EXPAND FREQUENCY	> 0	10000		
53.	FACTORIZATION FREQUENCY	> 0	50		
LU II					
63.	LU PARTIAL PIVOTING	0	0	3	0=partial
	or LU COMPLETE PIVOTING				1=complete
	or LU ROOK PIVOTING				2=rook
Additional parameters					
67.	AIJ TOLERANCE	0	1E-10		
	Elements $\|a(i, j)\| < AIJ$ TOLERANCE are set as 0				

Table 13: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
70.	SUBSPACE Convergence tolerance in current subspace before consider moving off another constraint.	> 0	0.5	1	Subspace tolerance

3.4 LPOPT

3.4.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.4.2 for information on how to use LPOPT with TOMLAB.

Purpose
lpopt solves dense linear optimization problems defined as

$$\begin{aligned} \min_x \quad & f(x) = c^T x \\ s/t \quad & \begin{matrix} & & x & & \\ b_L & \leq & Ax & \leq & b_U \end{matrix} \end{aligned} \tag{9}$$

where $c, x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{n+m_1}$.

If isempty(c) (or all(c==0), then a feasible point problem is solved (FP). Otherwise a standard linear programming problem is solved (LP)

Calling Syntax

[Inform, Iter, iState, Ax, cLamda, Obj, x] = lpopt(A, bl, bu, c, Warm, x, iState, SpecsFile, PrintFile, SummFile, PriLev, optPar);

Description of Inputs

The following fields are used:

A	Constraint matrix, m x n (DENSE).
bl	Lower bounds on (x,Ax), m+n x 1 vector (DENSE).
bu	Upper bounds on (x,Ax), m+n x 1 vector (DENSE).
c	Linear objective function cost coeffs, n x 1 (DENSE). If length(c) < n, setting c(1:n)=0;
Warm	If Warm > 0, then warm start, otherwise cold Start. Default 0. If warm start, then x and iState must be set properly. Normally the values from last call to lpopt are used.
x	Initial estimate of solution vector x. (DENSE) If length(x) ¡ n, the rest of the elements in x are set to 0.
iState	Working set (if Warm start) (n+m) x 1 (DENSE) If length(iState) ¡ n+m, setting iState(1:n+m)=0;

The following fields are used:, continued

iState(i)=0:	Corresponding constraint not in the initial QP working set.
iState(i)=1:	Inequality constraint at its lower bound in QP working set.
iState(i)=2:	Inequality constraint at its upper bound in QP working set.
iState(i)=3:	Equality constraint in the initial QP working set,bl(i)==bu(i).

SpecsFile Name of the OPTIONS File, see TOMLAB Guide.

PrintFile Name of the Print file. Name includes the path, maximal number of characters = 500.

SummFile Name of the Summary file. Name includes the path, maximal number of characters = 500.

PriLev Print level in the lpopt MEX-interface.
= 0 Silent.
= 1 Summary information.
= 2 More detailed information.
if isempty(PriLev), set as 0.

optPar Vector with optimization parameters overriding defaults and the optionally specified SPECS file. If length(optPar) ¡ 62, lpopt sets the rest of the values to missing value (-999).

Description of Outputs

The following fields are used:

Inform Result of LPOPT run.
0 = Optimal solution with unique minimizer found.
1 = A dead point was reached.
2 = The solution appears to be unbounded (or badly scaled).
3 = The constraints could not be satisfied. The problem has no feasible solution.
4 = Too many iterations, in either phase.
5 = The Maximum degrees of freedom is too small. The reduced Hessian must expand if further progress is too be made.
6 = An input parameter was invalid.
7 = The problem type was not recognized.
Other = UNKNOWN LPOPT Inform value.

Iter Number of iterations.

iState Status of working set, se input description of iState.

The following fields are used:, continued

Ax	A*x.
cLamda	Lagrangian multipliers (dual solution vector) (m x 1 vector).
Obj	Objective function value at optimum.
x	Solution vector with decision variable values (n x 1 vector).

3.4.2 Using TOMLAB

Purpose
lpoptTL solves dense linear optimization problems defined as

$$\begin{array}{lll} \min\limits_{x} & f(x) = c^T x & \\ s/t & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{array} \tag{10}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

Calling Syntax
Using the driver routine *tomRun*:

```
Prob = lpAssign( ... );
Result = tomRun('lpopt', Prob ... );
```

or

```
Prob = ProbCheck( ... );
Result = lpoptTL(Prob);
```

Call Prob = lpAssign(...) or Prob=ProbDef to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
A	Linear constraint matrix.
QP.c	Linear coefficients in objective function.
PriLevOpt	Print level.
WarmStart	If true, use warm start, otherwise cold start.
SOL.xs	Solution and slacks from previous run.
SOL.iState	Working set (if Warm start) (n+m) x 1 (DENSE) If length(iState) ¡ n+m, setting iState(1:n+m)=0;
iState(i)=0:	Corresponding constraint not in the initial QP working set.
iState(i)=1:	Inequality constraint at its lower bound in QP working set.
iState(i)=2:	Inequality constraint at its upper bound in QP working set.

Prob, The following fields are used:, continued

iState(i)=3:	Equality constraint in the initial QP working set, bl(i)==bu(i).
SOL.SpecsFile	Name of user defined SPECS file, read BEFORE optPar() is used.
SOL.PrintFile	Name of SOL Print file. Amount and type of printing determined by SPECS parameters or optPar parameters.
SOL.SummFile	Name of SOL Summary File.
SOL.optPar	Elements > -999 takes precedence over corresponding TOMLAB params. See Table 34.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Gradient c (linear objective).
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from lpopt.m (similar to TOMLAB).
Inform	LPOPT information parameter. 0 = Optimal solution with unique minimizer found. 1 = A dead point was reached. 2 = The solution appears to be unbounded (or badly scaled). 3 = The constraints could not be satisfied. The problem has no feasible solution. 4 = Too many iterations, in either phase. 5 = The Maximum degrees of freedom is too small. The reduced Hessian must expand if further progress is too be made. 6 = An input parameter was invalid. 7 = The problem type was not recognized. Other = UNKNOWN LPOPT Inform value.
rc	Reduced costs. If ninf=0, last m == -v_k.

Result, The following fields are used:, continued

Iter	Number of iterations.
FuncEv	Number of function evaluations.
ConstrEv	Number of constraint evaluations.
QP.B	Basis vector in TOMLAB QP standard.
MinorIter	Number of minor iterations. Not Set.
Solver	Name of the solver (lpopt).
SolverAlgorithm	Description of the solver.
SOL.xs	Solution and slack variables.
SOL.iState	State for variables and constraints in iState.

3.4.3 optPar

Description
Use missing value (-999 or less), when no change of parameter setting is wanted. No call is then done to internal routines lpprmi, lpprmr and lpprm. The default value will then be used by LPOPT, unless the value is altered in the SPECS file.

Description of Inputs

Table 18: The following fields are used:

#	SPECS keyword text	Lower	Default	Upper	Comment
Printing					
1.	PRINT LEVEL	0	10		0,1,5,10,20,30
Convergence Tolerances					
	In LPOPT/QPOPT: macheps = 2^{-53}; eps in Matlab is = 2^{-52};				
10.	OPTIMALITY TOLERANCE	> 0	1.05E-8		sqrt(macheps)
11.	FEASIBILITY TOLERANCE	> 0	1.05E-8		sqrt(macheps)
Other Tolerances					
21.	CRASH TOLERANCE	> 0	0.01	< 1	
27.	RANK TOLERANCE	> 0	1.11E-14		100*macheps
30.	ITERATION LIMIT	>0	max(2000,5(n+m))		
33.	MIN SUM YES (or NO)	0	0	1	1=min infeas.
	IF 1 (MIN SUM YES), minimize the infeasibilities before return.				
36.	FEASIBILITY PHASE ITERATIONS	>0	max(2000,5(n+m))		
45.	INFINITE STEP SIZE	>0	1E20		
Frequencies					
51.	CHECK FREQUENCY	> 0	50		
52.	EXPAND FREQUENCY	> 0	5		

3.5 QPOPT

3.5.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.5.2 for information on how to use QPOPT with TOMLAB.

Purpose
qpopt solves dense quadratic optimization problems defined as

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}x^T F x + c^T x + d^T x \\ s/t \quad & \begin{matrix} & x & \\ b_L & \leq & Ax & \leq & b_U \end{matrix} , \end{aligned} \tag{11}$$

where $c, x \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{n+m_1}$.

If isempty(H) and isempty(c), then a feasible point problem is solved (FP).
If isempty(H), then a linear programming problem is solved (LP).
If isempty(c), then a quadratic programming problem is solved (QP1).
Otherwise a standard quadratic programming problem is solved (QP2).

Calling Syntax

[Inform, Iter, iState, Ax, cLamda, Obj, x] = qpopt (H, A, bl, bu, c, Warm, x, iState, SpecsFile, PrintFile, SummFile, PriLev, optPar);

Description of Inputs

The following fields are used:

H	Matrix H in quadratic part of objective function (SPARSE or DENSE).
A	Constraint matrix, m x n. (DENSE).
bl	Lower bounds on (x, Ax), m+n x 1 vector (DENSE).
bu	Upper bounds on (x, Ax), m+n x 1 vector (DENSE).
c	Linear objective function cost coeffs, n x 1. (DENSE) If length(c) < n, setting c(1:n)=0;.
Warm	If Warm > 0, then warm start, otherwise cold Start. Default 0. If warm start, then x and iState must be set properly. Normally the values from last call to qpopt are used.
x	Initial estimate of solution vector x. (DENSE) If length(x) ¡ n, the rest of the elements in x are set to 0.

The following fields are used:, continued

iState Working set (if Warm start) (n+m) x 1 (DENSE) If length(iState) ¡ n+m, setting iState(1:n+m)=0;

iState(i)=0: Corresponding constraint not in the initial QP working set.
iState(i)=1: Inequality constraint at its lower bound in QP working set.
iState(i)=2: Inequality constraint at its upper bound in QP working set.
iState(i)=3: Equality constraint in the initial QP working set, bl(i)==bu(i).

SpecsFile Name of the OPTIONS File, see TOMLAB Guide.

PrintFile Name of the Print file. Name includes the path, maximal number of characters = 500.

SummFile Name of the Summary file. Name includes the path, maximal number of characters = 500.

PriLev Print level in the qpopt MEX-interface.
= 0 Silent.
= 1 Summary information.
= 2 More detailed information.
if isempty(PriLev), set as 0.

optPar Vector with optimization parameters overriding defaults and the optionally specified SPECS file. If length(optPar) ¡ 62, qpopt sets the rest of the values to missing value (-999).

Description of Outputs

The following fields are used:

Inform Result of QPOPT run. 0 = Optimal solution found.

0: `x` is a unique local minimizer. This means that `x` is *feasible* (it satisfies the constraints to the accuracy requested by the `Feasibility tolerance`), the reduced gradient is negligible, the Lagrange multipliers are optimal, and the reduced Hessian is positive definite. If H is positive definite or positive semidefinite, `x` is a *global minimizer*. (All other feasible points give a higher objective value.) Otherwise, the solution is a *local minimizer*, which may or may not be global. (All other points in the immediate neighborhood give a higher objective.)

The following fields are used:, continued

1: A dead-point was reached. This might occur if H is not sufficiently positive definite. If H is positive semidefinite, the solution is a *weak minimizer*. (The objective value is a global optimum, but there may be infinitely many neighboring points with the same objective value.) If H is indefinite, a feasible direction of decrease may or may not exist (so the point may not be a local or weak minimizer).

At a dead-point, the necessary conditions for optimality are satisfied (`x` is feasible, the reduced gradient is negligible, the Lagrange multipliers are optimal, and the reduced Hessian is positive semidefinite.) However, the reduced Hessian is nearly singular, and/or there are some very small multipliers. If H is indefinite, `x` is not necessarily a local solution of the problem. Verification of optimality requires further information, and is in general an NP-hard problem [37].

2: The solution appears to be unbounded. The objective is not bounded below in the feasible region, if the elements of `x` are allowed to be arbitrarily large. This occurs if a step larger than `Infinite Step` would have to be taken in order to continue the algorithm, or the next step would result in a component of `x` having magnitude larger than `Infinite Bound`. It should not occur if H is sufficiently positive definite.

3: The constraints could not be satisfied. The problem has no feasible solution.

4: One of the iteration limits was reached before normal termination occurred. See `Feasibility Phase Iterations` and `Optimality Phase Iterations`.

5: The `Maximum degrees of freedom` is too small. The reduced Hessian must expand if further progress is to be made.

6: An input parameter was invalid.

7: The `Problem type` was not recognized.

Iter Number of iterations.

iState Status of working set, se input description of iState.

Ax A*x.

cLamda Lagrangian multipliers (dual solution vector) (m x 1 vector).

Obj Objective function value at optimum.

x Solution vector with decision variable values (n x 1 vector).

The following fields are used:, continued

3.5.2 Using TOMLAB

Purpose
qpoptTL solves quadratic optimization problems defined as

$$\begin{array}{lll} \min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x & \\ s/t & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} & \end{array} \tag{12}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

Calling Syntax
Using the driver routine *tomRun*:

```
Prob = qpAssign( ... );
Result = tomRun('qpopt', Prob ... );
```

or

```
Prob = ProbCheck( ... );
Result = qpoptTL(Prob);
```

Call Prob = qpAssign(...) or Prob=ProbDef to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
A	Linear constraint matrix.
QP.c	Linear coefficients in objective function.
QP.F	Quadratic matrix of size nnObj x nnObj. nnObj < n is OK.
PriLevOpt	Print level.
WarmStart	If true, use warm start, otherwise cold start.
SOL.xs	Solution and slacks from previous run.
SOL.iState	Working set (if Warm start) (n+m) x 1 (DENSE) If length(iState) ¡ n+m, setting iState(1:n+m)=0;

Prob, The following fields are used:, continued

iState(i)=0:	Corresponding constraint not in the initial QP working set.
iState(i)=1:	Inequality constraint at its lower bound in QP working set.
iState(i)=2:	Inequality constraint at its upper bound in QP working set.
iState(i)=3:	Equality constraint in the initial QP working set, bl(i)==bu(i).

SOL.cLamda	Lagrangian multipliers from previous run (Warm start).
SOL.SpecsFile	Name of user defined SPECS file, read BEFORE optPar() is used.
SOL.PrintFile	Name of SOL Print file. Amount and type of printing determined by SPECS parameters or optPar parameters.
SOL.SummFile	Name of SOL Summary File.
SOL.optPar	Elements > -999 takes precedence over corresponding TOMLAB params. See Table 34.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Exact gradient computed at optimum.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from qpopt.m (similar to TOMLAB).
Inform	Result of QPOPT run. 0 = Optimal solution found.

Result, The following fields are used:, continued

0: `x` is a unique local minimizer. This means that `x` is *feasible* (it satisfies the constraints to the accuracy requested by the `Feasibility tolerance`), the reduced gradient is negligible, the Lagrange multipliers are optimal, and the reduced Hessian is positive definite. If H is positive definite or positive semidefinite, `x` is a *global minimizer*. (All other feasible points give a higher objective value.) Otherwise, the solution is a *local minimizer*, which may or may not be global. (All other points in the immediate neighborhood give a higher objective.)
1: A dead-point was reached. This might occur if H is not sufficiently positive definite. If H is positive semidefinite, the solution is a *weak minimizer*. (The objective value is a global optimum, but there may be infinitely many neighboring points with the same objective value.) If H is indefinite, a feasible direction of decrease may or may not exist (so the point may not be a local or weak minimizer).

At a dead-point, the necessary conditions for optimality are satisfied (`x` is feasible, the reduced gradient is negligible, the Lagrange multipliers are optimal, and the reduced Hessian is positive semidefinite.) However, the reduced Hessian is nearly singular, and/or there are some very small multipliers. If H is indefinite, `x` is not necessarily a local solution of the problem. Verification of optimality requires further information, and is in general an NP-hard problem [37].

2: The solution appears to be unbounded. The objective is not bounded below in the feasible region, if the elements of `x` are allowed to be arbitrarily large. This occurs if a step larger than `Infinite Step` would have to be taken in order to continue the algorithm, or the next step would result in a component of `x` having magnitude larger than `Infinite Bound`. It should not occur if H is sufficiently positive definite.

3: The constraints could not be satisfied. The problem has no feasible solution.

4: One of the iteration limits was reached before normal termination occurred. See `Feasibility Phase Iterations` and `Optimality Phase Iterations`.

5: The `Maximum degrees of freedom` is too small. The reduced Hessian must expand if further progress is to be made.

6: An input parameter was invalid.

7: The `Problem type` was not recognized.

rc Reduced costs. If ninf=0, last m == -v_k.

Iter Number of iterations.

Result, The following fields are used:, continued

FuncEv	Number of function evaluations. Set to Iter.
GradEv	Number of gradient evaluations. Set to Iter.
ConstrEv	Number of constraint evaluations. Set to 0.
QP.B	Basis vector in TOMLAB QP standard.
MinorIter	Number of minor iterations. Not Set.
Solver	Name of the solver (QPOPT).
SolverAlgorithm	Description of the solver.
SOL.xs	Solution and slack variables.
SOL.cLamda	Lagrangian multipliers.
SOL.iState	State for variables and constraints in iState.

3.5.3 optPar

Description

Use missing value (-999 or less), when no change of parameter setting is wanted. The default value will then be used by QPOPT, unless the value is altered in the SPECS file.

Description of Inputs

Table 23: The following fields are used:

#	SPECS keyword text	Lower	Default	Upper	Comment
Printing					
1.	PRINT LEVEL	0	10		0,1,5,10,20,30
Convergence Tolerances					
	In LPOPT/QPOPT: macheps = 2^{-53}; eps in Matlab is = 2^{-52};				
10.	OPTIMALITY TOLERANCE	> 0	1.05E-8		sqrt(macheps)
11.	FEASIBILITY TOLERANCE	> 0	1.05E-8		sqrt(macheps)
Other Tolerances					
21.	CRASH TOLERANCE	0	0.01	< 1	
27.	RANK TOLERANCE	> 0	1.1E-14		100*macheps
30.	ITERATION LIMIT	>0	max(2000,5(n+m))		
33.	MIN SUM YES (or NO)	0	0	1	1=min infeas.
	IF 1 (MIN SUM YES), minimize the infeasibilities before return.				
36.	FEASIBILITY PHASE ITERATIONS	>0	max(2000,5(n+m))		
45.	INFINITE STEP SIZE	>0	1E20		
47.	HESSIAN ROWS	0	n	n	0 if FP or LP
	Implicitly given by the dimensions of H in the call from Matlab				
48.	MAX DEGREES OF FREEDOM	0	n	n	
	ONLY USED IF HESSIAN ROWS == N				
Frequencies					
51.	CHECK FREQUENCY	> 0	50		
52.	EXPAND FREQUENCY	> 0	5		

3.6 SNOPT

3.6.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.6.2 for information on how to use SNOPT with TOMLAB.

Purpose
snopt solves sparse nonlinear optimization problems defined as

$$\begin{aligned} \min_x \quad & f(x) \\ s/t \quad & b_L \leq \begin{matrix} x \\ Ax \\ c(x) \end{matrix} \leq b_U \end{aligned}, \tag{13}$$

where $x, \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{n+m_1+m_2}$ and $c(x) \in \mathbb{R}^{m_2}$.

Calling Syntax

The file 'funfdf.m' must be defined and contain: function [mode, f, g] = funfdf(x, Prob, mode, nstate) to compute the objective function f and the gradient g at the point x.

The file 'funcdc.m' must be defined and contain: function [mode ,c ,dcS] = funcdc(x, Prob, mode, nstate) to compute the nonlinear constraint value c and the constraint Jacobian dcS for the nonlinear constraints at the point x.

Note that Matlab has dynamic sparse matrix handling and Fortran has static handling. The returned vector of constraint gradient values must always match the pattern of nonzeros as defined in the call to snopt. One approach for a general solution to this is given in the Tomlab callback routine nlp_cdcS.m, which calls the user defined 'funcdc' function.

The fields Prob.P and Prob.ConsPattern must be set, see below.

[hs, xs, pi, rc, Inform, nS, nInf, sInf, Obj, iwCount, gObj, fCon, gCon] = snopt(A, bl, bu, nnCon, nnObj, nnJac, Prob, iObj, optPar, Warm, hs, xs, pi, nS, SpecsFile, PrintFile, SummFile, PriLev, ObjAdd, moremem, ProbName);

Description of Inputs

The following fields are used:

A	Constraint matrix, m x n SPARSE (A consists of nonlinear part, linear part and one row for the linear objective). m > 0 always.

The following fields are used:, continued

bl Lower bounds on (x,g(x),Ax,c').

bu Upper bounds on (x,g(x),Ax,c').

nnCon Number of nonlinear constraints.

nnObj Number of nonlinear objective variables.

nnJac Number of nonlinear Jacobian variables.

Prob Must be a structure. No check is made in the MEX interface!

If TOMLAB calls snopt, then Prob is the standard TOMLAB problem structure, otherwise the user should set:

Prob.P = ProblemNumber, where ProblemNumber is some integer.

Prob.ConsPattern = []; or as the nonzero pattern for the constraint Jacobian as Prob.ConsPattern = ConsPattern;

ConsPattern is a nnCon x n zero-one sparse or dense matrix, where 0 values indicate zeros in the constraint Jacobian and ones indicate values that might be non-zero.

If the problem is a LP or QP problem (H defined), then the user does not have to specify anything more in the structure.

For a general nonlinear objective, or nonlinear constraints names of two user written routines must be given:

funfdf, actual name stored in Prob.FUNCS.fg, with syntax [mode, f, g] = funfdf(x, Prob, mode, nstate)

funcdc, actual name stored in Prob.FUNCS.cdc, with syntax [mode, c, dcS] = funcdc(x, Prob, mode, nstate)

SNOPT is calling the TOMLAB routines nlp_fg.m and nlp_cdcS.m in the callback, and they call funfdf and funcdc, respectively.

If these fields in Prob are empty (Prob.FUNCS.fg, Prob.FUNCS.cdc), the TOMLAB callback routines calls the usual function routines. Then the Prob struct should be normally defined, and the fields Prob.FUNCS.f, Prob.FUNCS.g, Prob.FUNCS.c, Prob.FUNCS.dc be set in the normal way (e.g. by the routine mFiles.m).

The following fields are used:, continued

If the mode parameter is 0, funfdf should return f, otherwise both f and the gradient vector g. If the mode parameter is 0, funcdc should return c, otherwise both c and dcS. Note that each row in dcS corresponds to a constraint, and that dcS must be a SPARSE matrix.

The user could also write his own versions of the routines nlp_fg.m and nlp_cdcS.m and put them before in the path.

iObj Says which row of A is a free row containing a linear objective vector c. If there is no such vector, iObj = 0. Otherwise, this row must come after any nonlinear rows, so that nnCon <= iObj <= m.

optPar Vector with optimization parameters overriding defaults and the optionally specified SPECS file. If using only default options, set optPar as an empty matrix.

Warm Flag, if true: warm start. Default cold start (if empty). If 'Warm Start' xS, nS and hs must be supplied with correct values.

hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu) 2=superbasic (between bounds), 3=basic (between bounds).

xs Initial vector, optionally including m slacks at the end. If warm start, full xs must be supplied.

pi Lagrangian multipliers for the nnCon nonlinear constraints. If empty, set as 0.

nS # of superbasics. Only used if calling again with a Warm Start.

SpecsFile Name of the SPECS input parameter file, see SNOPT User's Guide.

PrintFile Name of the Print file. Name includes the path, maximal number of characters = 500.

SummFile Name of the Summary file. Name includes the path, maximal number of characters = 500.

PriLev Printing level in the snopt m-file and snopt MEX-interface.

= 0 Silent
= 1 Summary information
= 2 More detailed information

ObjAdd Constant added to the objective for printing purposes, typically 0.

The following fields are used:, continued

mormem Add extra memory for the sparse LU, might speed up the optimization. 1E6 is 10MB of memory. If empty, set as 0.

ProbName Name of the problem. ¡=100 characters are used in the MEX interface. In the SNOPT solver the first 8 characters are used in the printed solution and in some routines that output BASIS files. Blank is OK.

Description of Outputs

The following fields are used:

hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu), 2=superbasic (between bounds), 3=basic (between bounds).

Basic and superbasic variables may be outside their bounds by as much as the `Minor feasibility tolerance`. Note that if scaling is specified, the feasibility tolerance applies to the variables of the *scaled* problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled.

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

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

xs Solution vector (n+m by 1) with n decision variable values together with the m slack variables.

pi The vector of dual variables π (a set of Lagrange multipliers for the general constraints). (m x 1 vector).

rc Vector of reduced costs, $g-(\ A\ \ -I\ )^T\pi$, where g is the gradient of the objective if `xs` is feasible (or the gradient of the Phase-1 objective otherwise). The last m entries are π. The vector is n+m. If nInf=0, last m == pi.

Inform Result of SNOPT run.

Finished successfully
1 optimality conditions satisfied
2 feasible point found

The following fields are used:, continued

3 requested accuracy could not be achieved

The problem appears to be infeasible
11 infeasible linear constraints
12 infeasible linear equalities
13 nonlinear infeasibilities minimized
14 infeasibilities minimized

The problem appears to be unbounded
21 unbounded objective
22 constraint violation limit reached

Resource limit error
31 iteration limit reached
32 major iteration limit reached
33 the superbasics limit is too small

Terminated after numerical difficulties
41 current point cannot be improved
42 singular basis
43 cannot satisfy the general constraints
44 ill-conditioned null-space basis

Error in the user-supplied functions
51 incorrect objective derivatives
52 incorrect constraint derivatives

Undefined user-supplied functions
61 undefined function at the first feasible point
62 undefined function at the initial point
63 unable to proceed into undefined region

User requested termination
72 terminated during constraint evaluation
73 terminated during objective evaluation
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

The following fields are used:, continued

92 basis file dimensions do not match this problem

System error
141 wrong number of basic variables
142 error in basis package

nS The final number of superbasic variables.

nInf Gives the number and the sum (next parameter) of the infeasibilities of constraints that lie outside their bounds by more than the `Feasibility tolerance`.

If the *linear* constraints are infeasible, `xs` minimizes the sum of the infeasibilities of the linear constraints subject to the upper and lower bounds being satisfied. In this case `nInf` gives the number of components of $A_L x$ lying outside their upper or lower bounds. The nonlinear constraints are not evaluated.

Otherwise, `xs` minimizes the sum of the infeasibilities of the *nonlinear* constraints subject to the linear constraints and upper and lower bounds being satisfied. In this case `nInf` gives the number of components of $f(x)$ lying outside their upper or lower bounds.

sInf Sum of infeasibilities. See *nInf* above.

Obj Objective function value at optimum.

iwCount Number of iterations minor (iwCount(1)) and major (iwCount(2)), function (iwCount(3:6)) and constraint (iwCount(7:10)) calls.

gObj Gradient of the nonlinear objective.

fCon Nonlinear constraint vector.

gCon Gradient vector (non-zeros) of the nonlinear constraint vector.

3.6.2 Using TOMLAB

Purpose
snoptTL solves nonlinear optimization problems defined as

$$\begin{array}{lrcccl} \min\limits_{x} & f(x) & & & & \\ & & & & & \\ & x_L & \leq & x & \leq & x_U, \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{14}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{m_1}$ and $c_L, c(x), c_U \in \mathbb{R}^{m_2}$.

Calling Syntax
Using the driver routine *tomRun*:

```
Prob = ⋄Assign( ... );
Result = tomRun('snopt', Prob ... );
```

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
c_L, c_U	Bounds on nonlinear constraints.
A	Linear constraint matrix.
QP.c	Linear coefficients in objective function.
PriLevOpt	Print level.
WarmStart	If true, use warm start, otherwise cold start.
SOL.xs	Solution and slacks from previous run.
SOL.hs	State for solution and slacks from previous run.
SOL.nS	Number of superbasics from previous run.
SOL.hElastic	Defines which variables are elastic in elastic mode. hElastic(j): 0 = variable j is non-elastic and cannot be infeasible. 1 = variable j can violate its lower bound.

Prob, The following fields are used:, continued

	2 = variable j can violate its upper bound. 3 = variable j can violate either its lower or upper bound.
SOL.moremem	Add more memory if SNOPT stops with not enough storage message. 1E6 is 10MB of memory. Default 0.
SOL.SpecsFile	Name of user defined SPECS file, read BEFORE optPar() is used.
SOL.PrintFile	Name of SOL Print file. Amount and type of printing determined by SPECS parameters or optPar parameters.
SOL.SummFile	Name of SOL Summary File.
SOL.optPar	Elements > -999 takes precedence over corresponding TOMLAB params. See Table 34.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Gradient of the function.
c_k	Nonlinear constraint residuals.
cJac	Nonlinear constraint gradients.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
cState	State of nonlinear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from snoptTL.m (similar to TOMLAB).
Inform	Result of SNOPT run.
	Finished successfully 1 optimality conditions satisfied 2 feasible point found 3 requested accuracy could not be achieved

Result, The following fields are used:, continued

The problem appears to be infeasible
11 infeasible linear constraints
12 infeasible linear equalities
13 nonlinear infeasibilities minimized
14 infeasibilities minimized

The problem appears to be unbounded
21 unbounded objective
22 constraint violation limit reached

Resource limit error
31 iteration limit reached
32 major iteration limit reached
33 the superbasics limit is too small

Terminated after numerical difficulties
41 current point cannot be improved
42 singular basis
43 cannot satisfy the general constraints
44 ill-conditioned null-space basis

Error in the user-supplied functions
51 incorrect objective derivatives
52 incorrect constraint derivatives

Undefined user-supplied functions
61 undefined function at the first feasible point
62 undefined function at the initial point
63 unable to proceed into undefined region

User requested termination
72 terminated during constraint evaluation
73 terminated during objective evaluation
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

Result, The following fields are used:, continued

	System error 141 wrong number of basic variables 142 error in basis package
rc	Vector of reduced costs, $g-(\ A\ \ -I\ )^T\pi$, where g is the gradient of the objective if `xs` is feasible (or the gradient of the Phase-1 objective otherwise). The last m entries are π. The vector is n+m. If nInf=0, last m == pi.
Iter	Number of iterations.
FuncEv	Number of function evaluations.
GradEv	Number of gradient evaluations.
ConstrEv	Number of constraint evaluations.
QP.B	Basis vector in TOMLAB QP standard.
MinorIter	Number of minor iterations.
Solver	Name of the solver (snopt).
SolverAlgorithm	Description of the solver.
SOL.xs	Solution vector (n+m by 1) with n decision variable values together with the m slack variables.
SOL.hs	Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu), 2=superbasic (between bounds), 3=basic (between bounds). Basic and superbasic variables may be outside their bounds by as much as the `Minor feasibility tolerance`. Note that if scaling is specified, the feasibility tolerance applies to the variables of the *scaled* problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled. Very occasionally some nonbasic variables may be outside their bounds by as much as the `Minor feasibility tolerance`, and there may be some nonbasics for which `xs`(j) lies strictly between its bounds. If `nInf` > 0, some basic and superbasic variables may be outside their bounds by an arbitrary amount (bounded by `sInf` if scaling was not used).
SOL.nS	The final number of superbasic variables.

Result, The following fields are used:, continued

SOL.nInf Gives the number and the sum (next parameter) of the infeasibilities of constraints that lie outside their bounds by more than the `Feasibility tolerance`.

If the *linear* constraints are infeasible, `xs` minimizes the sum of the infeasibilities of the linear constraints subject to the upper and lower bounds being satisfied. In this case `nInf` gives the number of components of $A_L x$ lying outside their upper or lower bounds. The nonlinear constraints are not evaluated.

Otherwise, `xs` minimizes the sum of the infeasibilities of the *nonlinear* constraints subject to the linear constraints and upper and lower bounds being satisfied. In this case `nInf` gives the number of components of $f(x)$ lying outside their upper or lower bounds.

SOL.sInf Sum of infeasibilities. See *nInf* above.

3.6.3 optPar

Description

Use missing value (-999 or less), when no change of parameter setting is wanted. The default value will then be used by SNOPT, if not the value is altered in the SPECS file (input SpecsFile).
Definition: nnL = max(nnObj,nnJac)) - Used in #38 and #47.

Description of Inputs

Table 28: The following fields are used:

#	SPECS keyword text	Lower	Default	Upper	Comment
The SQP method I - Printing					
1.	MAJOR PRINT LEVEL	0	1	11111	
QP subproblems I - Printing					
2.	MINOR PRINT LEVEL	0	1	10	0, 1 or 10
Frequencies I					
5.	PRINT FREQUENCY	0	100		
6.	SUMMARY FREQUENCY	0	100		
7.	SOLUTION YES/NO	0	1	1	1 = YES; 0 = NO
8.	SUPPRESS OPTIONS LISTING Also called SUPPRESS PARAMETERS	0	0	1	1 = True
The SQP Method II - Convergence Tolerances					
9.	MAJOR FEASIBILITY TOLERANCE	> 0	1E-6		
Nonlinear constraints I					
10.	MAJOR OPTIMALITY TOLERANCE eps_R == optPar(41), Default relative function precision	> 0	$max(2E-6, (10eps_R)^{0.5}) = 1.73$E-6 eps_R gives $(10 * eps_R)^0.5 = 1.73E-6$.		
QP subproblems II - Convergence Tolerances					
11.	MINOR FEASIBILITY TOLERANCE Feasibility tolerance on linear constraints .	> 0	1E-6		
12.	MINOR OPTIMALITY TOLERANCE	> 0	1E-6		
Derivative checking					
13.	VERIFY LEVEL	-1	-1	3	-1,0,1,2,3
14.	START OBJECTIVE CHECK AT COL	0	1	nnObj	
15.	STOP OBJECTIVE CHECK AT COL	0	nnObj	nnObj	
16.	START CONSTRAINT CHECK AT COL	0	1	nnJac	
17.	STOP CONSTRAINT CHECK AT COL	0	nnJac	nnJac	
QP subproblems III					
18.	SCALE OPTION	0	0 or 2	2	2 if LP,0 if NLP

Table 28: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
19.	SCALE TOLERANCE	> 0	0.9	< 1	
20.	SCALE PRINT	0	0	1	1 = True
21.	CRASH TOLERANCE	0	0.1	< 1	
The SQP Method III					
22.	LINESEARCH TOLERANCE	> 0	0.9	< 1	
LU I					
23.	LU FACTORIZATION TOLERANCE	1	100/3.99		100 if LP
24.	LU UPDATE TOLERANCE	1	10/3.99		10 if LP
25.	LU SWAP TOLERANCE	> 0	1.22E-4		$eps^(1/4)$
26.	LU SINGULARITY TOLERANCE	> 0	3.25E-11		$eps^{0.67}$
QP subproblems IV					
27.	PIVOT TOLERANCE	> 0	3.25E-11		$eps^(0.67)$
28.	CRASH OPTION	0	3	3	0,1,2,3
29.	ELASTIC WEIGHT	0	10000.0		
30.	ITERATIONS LIMIT	0	10000		or 20m, if more
	Maximal sum of minor iterations				
31.	PARTIAL PRICE	0	10 or 1		10 for LP
The SQP Method IV					
32.	MAXIMIZE	0	0	1	1=maximize
33.	FEASIBLE POINT	0	0	1	1=feasible pnt
Nonlinear constraints I					
34.	VIOLATION LIMIT	> 0	$1e6$		
The SQP Method V					
35.	MAJOR ITERATIONS LIMIT	> 0	$max(1000, 3 * max(n, m))$		
	Maximal number of major iterations				
36.	MINOR ITERATIONS LIMIT	> 0	500		
	Maximal number of minor iterations, i.e. in the solution of QP or simplex				
37.	MAJOR STEP LIMIT	> 0	2		
Hessian Approximation I					
38.	HESSIAN FREQUENCY	> 0	99999999		
The SQP Method VI					
39.	DERIVATIVE LEVEL	0	3	3	0,1,2,3
40.	DERIVATIVE LINESEARCH	0	1	1	0=NONDERIVATIVE
	0 is quadratic - gives quadratic, without gradient values				
	1 is cubic - gives cubic, always using gradient values				

Table 28: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
	Default: 0 if numerical derivatives, otherwise 1				
41.	FUNCTION PRECISION	> 0	3.0E-13		$eps^{0.8} = eps_R$
42.	DIFFERENCE INTERVAL	> 0	5.48E-7		$eps^{0.4}$
43.	CENTRAL DIFFERENCE INTERVAL	> 0	6.70E-5		$eps^{0.8/3}$
44.	PROXIMAL POINT METHOD	1	1	2	1,2
	Minimize the 1-norm (or 2-norm) of $\|\|(x - x_0)\|\|$ to find an initial point				
	that is feasible subject to simple bounds and linear constraints.				
45.	UNBOUNDED STEP SIZE	> 0	1E20		
46.	UNBOUNDED OBJECTIVE	> 0	1E15		
	Hessian Approximation II				
47.	HESSIAN FULL MEMORY	0	1	1	=1 if nnL <= 75
	or HESSIAN LIMITED MEMORY				=0 if nnL > 75
	The SQP Method VII				
48.	SUPERBASICS LIMIT	> 0	max(500,n+1)		
	TOMLAB extension (to avoid termination with Superbasics Limit too small):				
	Set $= n + 1$ if $n - size(A, 1) - length(c_L) > 450$ and $n <= 5000$				
	If $n > 5000$: $max(500, n - size(A, 1) - length(c_L))$				
	Avoid setting REDUCED HESSIAN (number of columns in reduced Hessian).				
	It will then be set to the same value as the SUPERBASICS LIMIT by SNOPT.				
	Hessian Approximation III				
49.	HESSIAN UPDATES	> 0	20		
	Maximum number of QN (Quasi-Newton) updates.				
	If HESSIAN FULL MEMORY, default is 99999999, otherwise 20.				
50.	HESSIAN FLUSH	> 0	99999999		
	Frequencies II				
51.	CHECK FREQUENCY	> 0	60		
52.	EXPAND FREQUENCY	> 0	10000		
53.	FACTORIZATION FREQUENCY	> 0	50		
	LU II				
63.	LU PARTIAL PIVOTING	0	0	3	0=partial
	or LU COMPLETE PIVOTING				1=complete
	or LU ROOK PIVOTING				2=rook
	or LU DIAGONAL PIVOTING				3=diagonal
	The SQP Method VIII				
64.	PENALTY PARAMETER	$>= 0$	0.0		
	Initial penalty parameter.				

Table 28: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
QP subproblems V					
65.	NEW SUPERBASICS	> 0	99		
	Also MINOR SUPERBASICS. Maximal number of new superbasics per major iteration.				
66.	QPSOLVER CHOLESKY or QPSOLVER CG or QPSOLVER QN	0	0	2	0=Cholesky 1=CG 2=Quasi-Newton CG
Conjugate-Gradient QP solver					
67.	CG TOLERANCE	> 0	$1e-2$		
68.	CG ITERATIONS	> 0	100		Max number of CG iters
69.	QPSOLVER CHOLESKY	0	0	1	QN preconditioned CG
	also called QG PRECONDITIONING. Default 1 if QPSOLVER QN.				
70.	SUBSPACE Quasi-Newton QP rg tolerance.	0	0.1	1	Subspace tolerance
The SQP Method IX					
71.	HESSIAN DIMENSION	> 0	$min(2000, nnL+1)$		=1 if LP problem (n upper limit)
	also called REDUCED HESSIAN. Number of columns in Reduced Hessian.				

3.7 SQOPT

3.7.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.7.2 for information on how to use SQOPT with TOMLAB.

Purpose

sqopt solves dense quadratic optimization problems defined as

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}x^T F x + c^T x + d^T x \\ s/t \quad & \begin{matrix} b_L & \leq & \begin{matrix} x \\ Ax \end{matrix} & \leq & b_U \end{matrix} \end{aligned} \qquad (15)$$

where $c, x \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{n+m_1}$.

Calling Syntax

The full input matrix A has two parts A = [A; d']; The position of the row d' is iObj. iObj=0 means no linear part in A.

NOTE: There are two ways to give the linear objective: either explicit as vector c or as part of the sparse matrix A, as d (or both ways).

[xs, hs, pi, rc, Inform, nS, nInf, sInf, Obj, iwCount] = sqopt(A, bl, bu, H, c, hElast, iObj, optPar, Warm, hs, xs, nS, SpecsFile, PrintFile, SummFile, ObjAdd, moremem, ProbName, Prob);

Description of Inputs

The following fields are used:

A Constraint matrix, m x n (SPARSE).

bl Lower bounds on (x,Ax,d').

bu Upper bounds on (x,Ax,d').

H Quadratic matrix, n x n, SPARSE or DENSE, empty if LP problem. If H is a string, H should be the name of a function routine, e.g if H = 'HxComp' then the function routine:

function Hx = HxComp(x, nState, Prob)

should compute H * x. The user must define this routine. nState == 1 if calling for the first time, otherwise 0. Third argument, the Prob structure, should only be used if calling SQOPT with the additional input parameter Prob, see below.

The following fields are used:, continued

Tomlab implements this callback to the predefined Matlab function HxFunc.m, using the call if Prob.SOL.callback == 1.

c Linear objective.

hElast Defines which bounds are elastic in elastic mode. hElast(j):

0 = variable j cannot be infeasible.
1 = variable j can violate its lower bound.
2 = variable j can violate its upper bound.
3 = variable j can violate either its lower or upper bound.

iObj Says which row of A is a free row containing a linear objective vector d. If there is no such vector, iObj = 0.

optPar Vector with optimization parameters overriding defaults and the optionally specified SPECS file. Set empty if only using default parameters.

Warm Flag, if true: warm start. Default cold start (if empty). If 'Warm Start' xS, nS and hs must be supplied with correct values.

hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu), 2=superbasic (between bounds), 3=basic (between bounds).

xs Initial x vector (nx1), optionally including m slacks at the end. If Warm start, full n+m vector xs must be supplied.

nS # of superbasics. Used if a Warm Start, otherwise set to 0.

SpecsFile Name of the SPECS input parameter file.

PrintFile Name of the Print file. Name includes the path, maximal number of characters = 500.

SummFile Name of the Summary file. Name includes the path, maximal number of characters = 500.

ObjAdd Constant added to the objective for printing purposes, typically 0.

moremem Add extra memory for the sparse LU, might speed up the optimization. 1E6 is 10MB of memory. If empty, set to 0.

The following fields are used:, continued

ProbName Name of the problem. ¡=100 characters are used in the MEX interface. In the SQOPT solver the first 8 characters are used in the printed solution and in some routines that output BASIS files. Blank is OK.

Prob Sending the Prob structure is optional, only of use if sending H as a function string, see input H.

Description of Outputs

The following fields are used:

xs Solution vector (n+m by 1) with n decision variable values together with the m slack variables.

hs Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu), 2=superbasic (between bounds), 3=basic (between bounds).

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

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

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

pi Lagrangian multipliers (dual solution vector) (m x 1 vector).

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

Inform Result of SQOPT run.

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

The following fields are used:, continued

10 the problem appears to be infeasible
11 infeasible linear constraints
12 infeasible linear equalities
14 infeasibilities minimized

20 the problem appears to be unbounded
21 unbounded objective

30 resource limit error
31 iteration limit reached
33 the superbasics limit is too small

40 terminated after numerical difficulties
42 singular basis
43 cannot satisfy the general constraints
44 ill-conditioned null-space basis

50 error in the user-supplied functions
53 the QP Hessian is indefinite

70 user requested termination
73 terminated during QP objective evaluation
74 terminated from monitor routine

80 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

90 input arguments out of range
91 invalid input argument
92 basis file dimensions do not match this problem
93 the QP Hessian is indefinite

140 system error
141 wrong number of basic variables
142 error in basis package

nS # of superbasics.

nInf Number of infeasibilities.

sInf Sum of infeasibilities.

The following fields are used:, continued

Obj Objective function value at optimum.

iwCount Number of QP iterations in iwCount(1), number of Hx products.

3.7.2 Using TOMLAB

Purpose
sqoptTL solves nonlinear optimization problems defined as

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}x^T F x + c^T x \\ s/t \quad & \begin{array}{rcccl} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{16}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

Calling Syntax
Using the driver routine *tomRun*:

```
Prob = ⋄Assign( ... );
Result = tomRun('sqopt', Prob ... );
```

or

```
Prob = ProbCheck( ... );
Result = sqoptTL(Prob);
```

Call Prob = ⋄Assign(...) or Prob=ProbDef; to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
A	Linear constraint matrix.
QP.c	Linear coefficients in objective function.
QP.F	Quadratic matrix of size nnObj x nnObj. nnObj < n is OK.
PriLevOpt	Print level.
WarmStart	If true, use warm start, otherwise cold start.
SOL.callback	If 1, use a callback to Matlab to compute QP.F * x for different x. Faster when F is very large and almost dense, avoiding copying of F from Matlab to MEX.
SOL.xs	Solution and slacks from previous run.

Prob, The following fields are used:, continued

SOL.hs	State for solution and slacks from previous run.
SOL.nS	Number of superbasics from previous run.
SOL.hElastic	Defines which variables are elastic in elastic mode. hElastic(j): 0 = variable j is non-elastic and cannot be infeasible. 1 = variable j can violate its lower bound. 2 = variable j can violate its upper bound. 3 = variable j can violate either its lower or upper bound.
SOL.moremem	Add more memory if SQOPT stops with not enough storage message. 1E6 is 10MB of memory. Default 0.
SOL.SpecsFile	Name of user defined SPECS file, read BEFORE optPar() is used.
SOL.PrintFile	Name of SOL Print file. Amount and type of printing determined by SPECS parameters or optPar parameters.
SOL.SummFile	Name of SOL Summary File.
SOL.optPar	Elements > -999 takes precedence over corresponding TOMLAB params. See Table 34.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Gradient of the function.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from sqopt.m (similar to TOMLAB).
Inform	Result of SQOPT run.

Result, The following fields are used:, continued

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

10 the problem appears to be infeasible
11 infeasible linear constraints
12 infeasible linear equalities
14 infeasibilities minimized

20 the problem appears to be unbounded
21 unbounded objective

30 resource limit error
31 iteration limit reached
33 the superbasics limit is too small

40 terminated after numerical difficulties
42 singular basis
43 cannot satisfy the general constraints
44 ill-conditioned null-space basis

50 error in the user-supplied functions
53 the QP Hessian is indefinite

70 user requested termination
73 terminated during QP objective evaluation
74 terminated from monitor routine

80 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

90 input arguments out of range
91 invalid input argument
92 basis file dimensions do not match this problem
93 the QP Hessian is indefinite

140 system error
141 wrong number of basic variables
142 error in basis package

Result, The following fields are used:, continued

rc	A vector of reduced costs, $g - (\ A \quad -I\ )^T\pi$, where g is the gradient of the objective if `xs` is feasible (or the gradient of the Phase-1 objective otherwise). The last m entries are π.
Iter	Number of iterations.
FuncEv	Number of function evaluations. Set to Iter.
GradEv	Number of gradient evaluations. Set to Iter.
ConstrEv	Number of constraint evaluations. Set to 0.
QP.B	Basis vector in TOMLAB QP standard.
Solver	Name of the solver (sqopt).
SolverAlgorithm	Description of the solver.
SOL.hs	Basis status of variables + constraints (n+m x 1 vector). State of variables: 0=nonbasic (on bl), 1=nonbasic (on bu), 2=superbasic (between bounds), 3=basic (between bounds). Basic and superbasic variables may be outside their bounds by as much as the value of the optional parameter `Feasibility tolerance`. Note that if scaling is specified, the `Feasibility tolerance` applies to the variables of the *scaled* problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled. Check the "Primal infeasibility" printed after the EXIT message. Very occasionally some nonbasic variables may be outside their bounds by as much as the `Feasibility tolerance`, and there may be some nonbasics for which `xs`(j) lies strictly between its bounds. If `nInf` > 0, some basic and superbasic variables may be outside their bounds by an arbitrary amount (bounded by `sInf` if scaling was not used).
SOL.hs	State for variables and slacks in xs.
SOL.nS	# of superbasics.
SOL.nInf	# of infeasibilities.
SOL.sInf	Sum of infeasibilities.

3.7.3 optPar

Description

Use missing value (-999 or less), when no change of parameter setting is wanted. The default value will then be used by SQOPT, unless the value is altered in the SPECS file (input SpecsFile).
See TOMLAB User's Guide for the SPECS keywords and description.

Description of Inputs

Table 33: The following fields are used:

#	SPECS keyword text	Lower	Default	Upper	Comment
LP/QP Parameters I - Printing					
2.	PRINT LEVEL	0	0	10	0, 1 or 10
Frequencies I					
5.	PRINT FREQUENCY	0	100		
6.	SUMMARY FREQUENCY	0	100		
7.	SOLUTION YES/NO	0	1	1	1 = YES; 0 = NO
8.	SUPPRESS PARAMETERS	0	0	1	1 = True
LP/QP Parameters II - Convergence Tolerances					
11.	FEASIBILITY TOLERANCE	> 0	1E-6		
12.	OPTIMALITY TOLERANCE	> 0	1E-6		
Scaling					
18.	SCALE OPTION	0	2	2	
19.	SCALE TOLERANCE	> 0	0.9	< 1	
20.	SCALE PRINT	0	0	1	1 = True
21.	CRASH TOLERANCE	0	0.1	< 1	
LU I					
23.	LU FACTOR TOLERANCE	1	100		
24.	LU UPDATE TOLERANCE	1	10		
25.	LU SWAP TOLERANCE	> 0	1.22E-4		$eps^{1/4}$
26.	LU SINGULARITY TOLERANCE	> 0	3.25E-11		$eps^{0.67}$
LP/QP Parameters III					
27.	PIVOT TOLERANCE	> 0	3.25E-11		$eps^{0.67}$
28.	CRASH OPTION	0	0	3	0,1,2,3
29.	ELASTIC WEIGHT	0	1		
30.	ITERATIONS LIMIT	0	10000		
31.	PARTIAL PRICE	1	10		
32.	MAXIMIZE	0	0	1	1=maximize

Table 33: The following fields are used:, continued

#	SPECS keyword text	Lower	Default	Upper	Comment
QP Objective					
45.	UNBOUNDED STEP SIZE	> 0	1E20		
48.	SUPERBASICS LIMIT	> 0	min(500,1+nnObj)		
LP/QP Parameters IV					
49.	ELASTIC MODE	0	1		0,1,2
50.	ELASTIC OBJECTIVE	0	2		0,1,2
Frequencies II					
51.	CHECK FREQUENCY	> 0	60		
52.	EXPAND FREQUENCY	> 0	10000		
53.	FACTORIZATION FREQUENCY	> 0	50		
LU II					
63.	LU COMPLETE PIVOTING or LU PARTIAL PIVOTING	0	0	1	1=complete, 0=partial

4 Using the SNOPT Solvers in TOMLAB

This section discusses the use of the TOMLAB solvers from Stanford Systems Optimization Laboratory (SOL). In order to use these solvers efficiently, it is recommended to read the corresponding solver details as well. It is important to do help on the m-files corresponding to each solver as well as the TOMLAB interface routine. The names for *MINOS* solver are *minos.m* and *minosTL.m*, and similar for other solvers.

To learn all the different parameter settings for a solver it is useful to run the GUI, where all parameters are selectable, and all default values are shown. Furthermore there are short help available for the different solver parameters in the drag menus. Even if the user is not using the GUI to solve the particular user problem, it might be useful to run the test problems defined in TOMLAB to learn how to use the SOL solvers in the most efficient way.

4.1 Setting Solver Parameters

To handle the use of the SOL solvers, a special field in the *Prob* structure, *Prob.SOL*, is used to send information to the SOL solvers. It is also used to store the information needed for warm starts of the SOL solvers.

The vector *Prob.SOL.optPar* of length 71 holds most of the different parameters that control the performance of the SOL solvers. All parameters have default values. If calling the SOL solver directly, not using TOMLAB, the user should set the values wanted in the *optPar* vector. The rest should have the value -999, which gives the default value used by the solver. For information on other parameters that could effect the solution process see the Table 34. The TOMLAB interface routine always has the name of the routine, with the additional two letters *TL*, e.g. for *MINOS* the TOMLAB interface routine is *minosTL*.

Other important fields to set when using the SOL solvers are the print and summary files that the solvers create. These files are very important to look through if any problems are occurring, to find what the causes might be, and if any warnings or errors are reported from the SOL solvers. To create a print and summary file, one example when running *MINOS* is the following statements

```
Prob.SOL.optPar(1) = 111111;     % Maximal print level
Prob.SOL.PrintFile = 'minos.pri' % Print file called minos.pri
Prob.SOL.SummFile  = 'minos.sum' % Summary file called minos.sum
Prob.NumDiff       = 6;          % Tell TOMLAB that minos is estimating
Prob.ConsDiff      = 6;          % all derivatives (gradient and Jacobian)
```

If *MINOS* is told that no derivatives are given, then *MINOS* will try to estimate them, and then TOMLAB must not do the same, i.e. *Prob.NumDiff* and *Prob.ConsDiff* must be set to six (internal solver estimation of derivatives). If *MINOS* is told that all derivatives are given, then TOMLAB might estimate them for *MINOS* using any of the five methods possible, or by using automatic differentiation.

4.2 Derivatives for the Solvers

The TOMLAB solvers from Stanford Systems Optimization Laboratory (SOL), have some useful special features, which influence the way that input is prepared to the solvers.

When defining the gradient vector and the constraint Jacobian matrix it is often the case that they are only partially known. The SOL solvers give the possibility to mark these elements. They will then be estimated by finite differences.

In TOMLAB the gradient and the constraint Jacobian matrix are defined in two separate routines. If any element is unknown, it is just marked with the standard Matlab element *NaN*. The TOMLAB SOL interface routines will estimate the *NaN* elements if *Prob.CheckNaN* is set to 1.

If any gradient or constraint Jacobian element is infinite, in Matlab set as *Inf* or *-Inf*, this element is converted to a big number, 10^{20}, in the TOMLAB SOL interface.

The following applies to the sparse nonlinear programming solvers *MINOS* and *SNOPT*. When the constraint Jacobian matrix is sparse, then only the nonzero elements should be given. The sparse pattern is given as a sparse matrix *Prob.ConsPattern*. In this matrix nonzero elements are marking nonzero elements in the constraint Jacobian. This pattern is static, i.e. given once before the call to the SOL solver. One problem is that a sparse matrix in Matlab is dynamic, i.e. only the nonzero elements of the matrix are stored. As soon as an element becomes zero, the vector of nonzeros are decreased one element. A gradient element that is normally nonzero might become zero during the optimization. Therefore care must be taken by the interface to return the correct values, because the SOL solvers assume the possible non-zero elements of the constraint Jacobian to be returned in the correct order.

The TOMLAB interface assumes the following conventions for the constraint Jacobian matrix:

- If the user returns a sparse matrix, and the number of nonzeros are equal to the number of nonzeros in *Prob.ConsPattern*, no checks are done.
- If the user returns a sparse matrix, and the number of nonzeros are not equal to the number of nonzeros in *Prob.ConsPattern*, the interface is matching all elements in the sparse array to find which nonzeros they represent, and returns the correct vector of static nonzeros.
- If the user returns a sparse matrix, and has given no pattern of nonzeros in *Prob.ConsPattern*, i.e. it is an empty array, then the solver and the interface assumes a full, dense matrix and the interface makes a full matrix before returning the elements, column by column, to the solver.
- If the user returns a dense matrix, the interface just feeds all elements, column by column, back to the solver.
- If too few elements are returned, the solver will estimate the rest using finite differences.

When using the dense SOL nonlinear programming solvers, the constraint Jacobian matrix is always assumed to be dense. The interface will convert any sparse matrix returned by the user to a dense, full matrix, and return the elements, column by column, to the solver.

If no derivatives are available, it might be better to use the *Nonderivative linesearch* in *SNOPT*. It is based on safeguarded quadratic interpolation. The default is to use a safeguarded cubic interpolation if derivatives are available. To select *Nonderivative linesearch* set the following parameter:

```
Prob.SOL.optPar(40) = 0; % Use Nonderivative instead of Derivative Linesearch
```

4.3 Solver Output to Files

The SOL solvers print different amount of information on ASCII files, one *Print File* with more information, and one *Summary File* with less information. *SNOPT* is using *snoptpri.txt* and *snoptsum.txt* as default names. *MINOS* is using *minospri.txt* and *minossum.txt* as default names. The following example shows how to set new names, other than the default, for these files.

```
Prob.SOL.PrintFile = 'snoptp.out';   % New name for Print File
Prob.SOL.SummFile  = 'snopts.out';   % New name for Summary File
```

The *SQOPT* solver by default also defines the two log files as *sqoptpri.txt* and *sqoptsum.txt*. If the print level is 0 no output will occur, unless some errors are encountered. It is possible to make *SQOPT* totally silent and avoid any opening of files by the following statements.

```
Prob.SOL.PrintFile = '';
Prob.SOL.SummFile  = '';
Prob.SOL.optPar(2) = 0;
Prob.SOL.optPar(3) = 0;
```

The amount of printing is determined by a print level code, which is different for the solvers. See the help and manual for each solver. Some solvers also have two print levels, one major print level and one minor print level. This applies for *SNOPT*. There are also different other parameters that influence how much output is written on the files. The following example show how to get maximum output for *SNOPT* on files with user defined names.

```
Prob.SOL.PrintFile = 'sn.p'; % New name for Print File
Prob.SOL.SummFile  = 'sn.s'; % New name for Summary File
Prob.SOL.optPar(1) = 111111; % Major print level, combination of six 0/1
Prob.SOL.optPar(2) = 10;     % Minor print level, 0, 1 or 10. 10 is maximum
Prob.SOL.optPar(5) = 1;      % Print Frequency
Prob.SOL.optPar(6) = 1;      % Summary Frequency
Prob.SOL.optPar(7) = 1;      % Solution yes. 0 = Solution not printed
Prob.SOL.optPar(8) = 1;      % Full options listing, not default
```

The other SOL solvers, *QPOPT* and *LPOPT*, only define the *Print File* and *Summary File* if the user has defined names for them. See the help for each solver on how to set the correct print level and other parameters.

4.4 Warm Starts for the Solvers

In TOMLAB warm starts for the SOL solvers are automatically handled. The only thing needed is to call the routine *WarmDefSOL* after having solved the problem the first time, as the following principal example shows doing repeated calls to *SNOPT*.

```
...                             % Define first problem
Result = tomRun('snopt',Prob); % Solve problem at t=1
... for t=2:N
   ...                              % Changes at time t in Prob structure
   Prob      = WarmDefSOL('snopt', Prob, Result(t-1));
   Result(t) = tomRun('snopt',Prob);   % Call tomRun to solve again
   ...                              % Postprocessing
end
```

The *WarmDefSOL* routine are setting the warm start flag *Prob.WarmStart*, as true.

```
Prob.WarmStart = 1;
```

It is also moving subfields on the *Result.SOL* structure into *Prob.SOL* for the next run. For *SNOPT*, *SQOPT* and *MINOS* the following commands are needed.

```
Prob.SOL.xs = Result.SOL.xs
Prob.SOL.hs = Result.SOL.hs
Prob.SOL.nS = Result.SOL.nS
```

For *QPOPT* and the other SOL solvers the commands are

```
Prob.SOL.xs      = Result.SOL.xs
Prob.SOL.iState  = Result.SOL.iState
Prob.SOL.cLamda  = Result.SOL.cLamda
Prob.SOL.H       = Result.SOL.H
```

The fields *SOL.cLamda* and *SOL.H* are not used for *QPOPT*, and *LPOPT*.

Note that for all solvers the new initial value are taken from the field *Prob.SOL.xs* and any user change in the standard initial value *Prob.x_0* is neglected.

4.5 Memory for the Solvers

Some users have encountered problems where SOL solvers report insufficient memory on machines where memory should not be an issue. The solvers estimate their internal memory requirements at startup. This estimate is not always large enough so the user might have to specify additional memory. This can be accomplished by

```
Prob.SOL.moremem = 1000; % or even larger if necessary
```

4.6 Parameters in Prob.optParam

Table 34: Information stored in the structure *Prob.optParam*. Default values in parenthesis.

Field	Description
PreSolve	Flag if presolve analysis is to be applied on linear constraints (0).
DiffInt	Difference interval in derivative estimates (used for *Prob.NumDiff/ConsDiff* -1 or 1).
CentralDiff	Central difference interval in derivative estimates (used for *Prob.NumDiff/ConsDiff* -2 or 2).
splineSmooth	Smoothness parameter sent to the SPLINE Toolbox routine *csaps.m* when computing numerical approximations of the derivatives (0.4) (used for *Prob.NumDiff/ConsDiff* -3 or 3).
VARsplineTol	Tolerance parameter sent to the SPLINE Toolbox routine *spaps.m* when computing numerical approximations of the derivatives (10^{-3}) (used for *Prob.NumDiff/ConsDiff* -4 or 4).

Table 34: Information stored in the structure *Prob.optParam*, continued.

Field	**Description**
CHECK	If true, no more check is done on the structure. Set to true (=1) after first call to solver (which then calls *optParamSet*).

5 QPOPT details

5.1 Introduction

TOMLAB /QPOPT (hereafter referred to as QPOPT) is based on an inertia-controlling method that maintains a Cholesky factorization of the reduced Hessian (see below). The method follows Gill and Murray [13] and is described in [32]. Here we briefly summarize the main features of the method.

5.1.1 Overview

QPOPT's method has a *feasibility phase* (finding a feasible point by minimizing the sum of infeasibilities) and an *optimality phase* (minimizing the quadratic objective function within the feasible region). The computations in both phases are performed by the same subroutines, but with different objective functions. The feasibility phase does *not* perform the standard simplex method; i.e., it does not necessarily find a vertex (with n constraints active), except in the `LP` case if $m_L \leq n$. Once an iterate is feasible, all subsequent iterates remain feasible. Once a vertex is reached, all subsequent iterates are at a vertex.

QPOPT is designed to be efficient when applied to a *sequence* of related problems—for example, within a sequential quadratic programming method for nonlinearly constrained optimization (e.g., the NPOPT package [33]). In particular, the user may specify an initial working set (the indices of the constraints believed to be satisfied exactly at the solution); see the discussion of `Warm Start`.

In general, an iterative process is required to solve a quadratic program. Each new iterate $\bar{x}$ is defined by

$$\bar{x} = x + \alpha p, \tag{17}$$

where the *step length* α is a non-negative scalar, and p is called the *search direction*. (For simplicity, we shall consider a typical iteration and avoid reference to the iteration index.)

5.1.2 The working set

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

The definition of the search direction ensures that constraints in the working set remain *unaltered* for any value of the step length. Thus,

$$Wp = 0. \tag{18}$$

In order to compute p, a TQ *factorization* of W is used:

$$WQ = \begin{pmatrix} 0 & T \end{pmatrix}, \tag{19}$$

where T is a nonsingular $m_w \times m_w$ upper-triangular matrix, and Q is an $n \times n$ nonsingular matrix constructed from a product of orthogonal transformations(see [27]). If the columns of Q are partitioned so that

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

where Y is $n \times m_w$ and Z is $n \times n_Z$ (where $n_Z = n - m_w$), then the columns of Z form a basis for the null space of W. Let n_R be an integer such that $0 \leq n_R \leq n_Z$, and let Z_R denote a matrix whose n_R columns are a subset

of the columns of Z. (The integer n_R is the quantity "`Zr`" in the printed output from `qpopt`). In many cases, Z_R will include *all* the columns of Z. The direction p will satisfy (18) if

$$p = Z_R p_R, \tag{20}$$

where p_R is any n_R-vector.

5.1.3 The reduced Hessian

Let g_Q and H_Q denote the *transformed gradient* and *transformed Hessian*:

$$g_Q = Q^T g(x) \qquad \text{and} \qquad H_Q = Q^T H Q.$$

The first n_R elements of the vector g_Q will be denoted by g_R, and the first n_R rows and columns of the matrix H_Q will be denoted by H_R. The quantities g_R and H_R are known as the *reduced gradient* and *reduced Hessian* of $q(x)$, respectively. Roughly speaking, g_R and H_R describe the first and second derivatives of an *unconstrained* problem for the calculation of p_R.

At each iteration, a triangular factorization of H_R is available. If H_R is positive definite, $H_R = R^T R$, where R is the upper-triangular Cholesky factor of H_R. If H_R is not positive definite, $H_R = R^T D R$, where $D = \text{diag}(1, 1, \ldots, 1, \omega)$, with $\omega \leq 0$.

In QPOPT, the computation is arranged so that the reduced-gradient vector is a multiple of e_R, a vector of all zeros except in the last (n_Rth) position. This allows p_R in (20) to be computed from a single back-substitution,

$$R p_R = \gamma e_R, \tag{21}$$

where γ is a scalar whose definition depends on whether the reduced Hessian is positive definite at x. In the positive-definite case, $x + p$ is the minimizer of the objective function subject to the working-set constraints being treated as equalities. If H_R is not positive definite, p_R satisfies

$$p_R^T H_R p_R < 0 \text{and} g_R^T p_R \leq 0,$$

allowing the objective function to be reduced by any step of the form $x + \alpha p$, $\alpha > 0$.

5.1.4 Optimality conditions

If the reduced gradient is zero, x is a constrained stationary point in the subspace defined by Z. During the feasibility phase, the reduced gradient will usually be zero only at a vertex (although it may be zero elsewhere in the presence of constraint dependencies). During the optimality phase, a zero reduced gradient implies that x minimizes the quadratic objective when the constraints in the working set are treated as equalities. At a constrained stationary point, Lagrange multipliers λ are defined from the equations

$$W^T \lambda = g(x). \tag{22}$$

A Lagrange multiplier λ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\lambda_j \leq \sigma$ when the associated constraint is at its *upper bound*, or if $\lambda_j \geq -\sigma$ when the associated constraint is at its *lower bound*, where σ depends on the `Optimality tolerance`. If a multiplier is non-optimal, the objective function (either the true objective or the sum of infeasibilities) can be reduced by deleting the corresponding constraint from the working set (with index `Jdel`; see Section 5.6).

If optimal multipliers occur during the feasibility phase but the sum of infeasibilities is not zero, there is no feasible point. The user can request QPOPT to continue until the sum of infeasibilities is minimized (see the discussion of `Min sum`). At such a point, the Lagrange multiplier λ_j corresponding to an inequality constraint in the working set will be such that $-(1+\sigma) \leq \lambda_j \leq \sigma$ when the associated constraint is at its *upper bound*, and $-\sigma \leq \lambda_j \leq 1+\sigma$ when the associated constraint is at its *lower bound*. Lagrange multipliers for equality constraints will satisfy $|\lambda_j| \leq 1+\sigma$.

If the reduced gradient is not zero, Lagrange multipliers need not be computed and the search direction p is given by $Z_R p_R$ (see (21)). The step length is chosen to maintain feasibility with respect to the satisfied constraints. If H_R is positive definite and $x+p$ is feasible, α is defined to be one. In this case, the reduced gradient at $\bar{x}$ will be zero, and Lagrange multipliers are computed. Otherwise, α is set to α_M, the step to the "nearest" constraint (with index `Jadd`; see Section 5.6). This constraint is added to the working set at the next iteration.

If the reduced Hessian H_R is not positive definite and α_M does not exist (i.e., no positive step α_M reaches the boundary of a constraint not in the working set), then QPOPT terminates at x and declares the problem to be unbounded.

5.2 Further Details of the Method

The following sections are not essential knowledge for normal users. They give background on the active-set strategy and the anti-cycling procedure.

5.2.1 Treatment of simple upper and lower bounds

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

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

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

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

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

5.2.2 The initial working set

For a cold start, the initial working set includes equality constraints and others that are close to being satisfied at the starting point. ("Close" is defined under `Crash tolerance`.) For a warm start, the initial working is specified by the user (and possibly revised to improve the condition of W).

At the start of the optimality phase, QPOPT must ensure that the initial reduced Hessian H_R is positive-definite. It does so by including a suitably large number of constraints (real or artificial) in the initial working set. (When W contains n constraints, H_R has no rows and columns. Such a matrix is positive definite by definition.)

Let H_Z denote the first n_Z rows and columns of $H_Q = Q^T H Q$ at the beginning of the optimality phase. A partial Cholesky factorization with interchanges is used to find an upper-triangular matrix R that is the factor of the largest positive-definite leading submatrix of H_Z. The use of interchanges tends to maximize the dimension of R. (The condition of R may be controlled by setting the `Rank Tolerance`.) Let Z_R denote the columns of Z corresponding to R, and let Z be partitioned as $Z = \begin{pmatrix} Z_R & Z_A \end{pmatrix}$. A working set for which Z_R defines the null space can be obtained by including *the rows of* Z_A^T as "artificial constraints" (with bounds equal to the current value of $Z_A^T x$). Minimization of the objective function then proceeds within the subspace defined by Z_R, as described in Section 5.1.

The artificially augmented working set is given by

$$\bar{W} = \begin{pmatrix} Z_A^T \\ W \end{pmatrix},$$

so that p will satisfy $Wp = 0$ *and* $Z_A^T p = 0$. By definition of the TQ factors of W, we have

$$\bar{W}Q = \begin{pmatrix} Z_A^T \\ W \end{pmatrix} Q = \begin{pmatrix} Z_A^T \\ W \end{pmatrix} \begin{pmatrix} Z_R & Z_A & Y \end{pmatrix} = \begin{pmatrix} 0 & \bar{T} \end{pmatrix},$$

where

$$\bar{T} = \begin{pmatrix} I & 0 \\ 0 & T \end{pmatrix}.$$

Hence the TQ factors of $\bar{W}$ are available trivially.

The matrix Z_A is not kept fixed, since its role is purely to define an appropriate null space; the TQ factorization can therefore be updated in the normal fashion as the iterations proceed. No work is required to "delete" the artificial constraints associated with Z_A when $Z_R^T g = 0$, since this simply involves repartitioning Q. The "artificial" multiplier vector associated with the rows of Z_A^T is equal to $Z_A^T g$, and the multipliers corresponding to the rows of the "true" working set are the multipliers that would be obtained if the artificial constraints were not present. If an artificial constraint is "deleted" from the working set, an `A` appears alongside the entry in the `Jdel` column of the printed output (see Section 5.6). The multiplier may have either sign.

The number of columns in Z_A and Z_R, the Euclidean norm of $Z_R^T g$, and the condition estimator of R appear in the printed output as `Art`, `Zr`, `Norm gZ` and `Cond Rz` (see Section 5.6).

Under some circumstances, a different type of artificial constraint is used when solving a linear program. Although the algorithm of `qpopt` does not usually perform simplex steps (in the traditional sense), there is one exception: a linear program with fewer general constraints than variables (i.e., $m_L \leq n$). (Use of the simplex method in this situation leads to savings in storage.) At the starting point, the "natural" working set (the set of constraints exactly or nearly satisfied at the starting point) is augmented with a suitable number of "temporary" bounds, each of which has the effect of temporarily fixing a variable at its current value. In subsequent iterations, a temporary bound is treated similarly to normal constraints until it is deleted from the working set, in which case it is never added again. If a temporary bound is "deleted" from the working set, an `F` (for "Fixed") appears alongside the entry in the `Jdel` column of the printed output (see Section 5.6). Again, the multiplier may have either sign.

5.2.3 The anti-cycling procedure

The EXPAND procedure [30] is used to reduce the possibility of cycling at a point where the active constraints are nearly linearly dependent. The main feature of EXPAND is that the feasibility tolerance is increased slightly at the start of every iteration. This allows a positive step to be taken every iteration, perhaps at the expense of violating the constraints slightly.

Suppose that the `Feasibility tolerance` is δ. Over a period of K iterations (where K is defined by the `Expand frequency`), the feasibility tolerance actually used by QPOPT—the *working* feasibility tolerance—increases from 0.5δ to δ (in steps of $0.5\delta/K$).

At certain stages the following "resetting procedure" is used to remove constraint infeasibilities. First, all variables whose upper or lower bounds are in the working set are moved exactly onto their bounds. A count is kept of the number of nontrivial adjustments made. If the count is positive, iterative refinement is used to give variables that satisfy the working set to (essentially) machine precision. Finally, the working feasibility tolerance is reinitialized to 0.5δ.

If a problem requires more than K iterations, the resetting procedure is invoked and a new cycle of iterations is started with K incremented by 10. (The decision to resume the feasibility phase or optimality phase is based on comparing any constraint infeasibilities with δ.)

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

The EXPAND procedure not only allows a positive step to be taken at every iteration, but also provides a potential *choice* of constraints to be added to the working set. Let α_M denote the maximum step at which $x + \alpha_M p$ does not violate any constraint by more than its feasibility tolerance. All constraints at distance α ($\alpha \leq \alpha_M$) along p from the current point are then viewed as acceptable candidates for inclusion in the working set. The constraint whose normal makes the largest angle with the search direction is added to the working set. This strategy helps keep the working-set matrix W well-conditioned.

5.3 The Options File

Observe that options are normally set in *Prob.SOL.optPar*.

Several choices in QPOPT's algorithm logic may be defined by various *optional parameters* (more briefly known as *options* or *parameters*).

In order to reduce the number of subroutine parameters for `qpopt`, the options have *default values* that are appropriate for most problems. Options need be specified only if their values should be different from the default.

5.3.1 Format of option strings

Each optional parameter is defined by an *option string* of up to 72 characters, containing one or more *items* separated by spaces or equal signs (`=`). Alphabetic characters may be in upper or lower case. An example option string is `Print level = 5`. In general, an option string contains the following items:

1. A *keyword* such as `Print`.
2. A *phrase* such as `level` that qualifies the keyword. (Typically 0, 1 or 2 words.)
3. A *number* that specifies either an `integer` or a `real` value (only for some options). Such numbers may be up to 16 contiguous characters in Fortran 77's `F`, `E` or `D` formats, terminated by a space.

Blank strings and comments may be used to improve readability. A *comment* begins with an asterisk (`*`) and all subsequent characters are ignored. Synonyms are recognized for some of the keywords, and abbreviations may be used if there is no ambiguity.

The following are examples of valid option strings for QPOPT:

```
NOLIST
COLD START
Warm start
Problem type  = LP
Problem type  = Quadratic Program        * Same as QP or QP2
Problem Type    QP4
Min sum         Yes
Feasibility Phase iteration limit   100
Feasibility tolerance            1.0e-8  * for IEEE double precision
Crash tolerance                   0.002
Defaults
* This string will be ignored.             So will a blank line.
```

5.4 Description of the optional parameters

Permissible options are defined below in alphabetical order. For each option, we give the keyword, any essential qualifiers, the default value, and the definition. The minimum abbreviation of each keyword and qualifier is underlined. If no characters of a qualifier are underlined, the qualifier may be omitted. The letters i and r denote `integer` and `real` values required for certain options. The letter a denotes a character string value. The number **u** represents unit roundoff for floating-point arithmetic (typically about 10^{-16}).

`Check frequency` i Default = 50

Every ith

iteration, a numerical test is made to see if the current solution x satisfies the constraints in the working set. If the largest residual of the constraints in the working set is judged to be too large, the working-set matrix is refactorized and the variables are recomputed to satisfy the constraints more accurately.

`Cold start` Default = `Coldstart`

`Warm start`

This option specifies how the initial working set is chosen. With a cold start, QPOPT chooses the initial working set based on the values of the variables and constraints at the initial point. Broadly speaking, the first working set will include all equality constraints and also any bounds or inequality constraints that are "nearly" satisfied (to within the `Crash tolerance`).

With a warm start, the user must provide a valid definition of every element of the array `istate`. The specification of `istate` will be overridden if necessary, so that a poor choice of the working set will not cause a fatal error. A warm start will be advantageous if a good estimate of the initial working set is available—for example, when `qpopt` is called repeatedly to solve related problems.

`Crash tolerance` r Default = 0.01

This value is used for cold starts when QPOPT selects an initial working set. Bounds and inequality constraints are selected if they are satisfied to within r. More precisely, a constraint of the form $a_j^T x \geq l$ will be included in the initial working set if $|a_j^T x - l| \leq r(1 + |l|)$. If $r < 0$ or $r > 1$, the default value is used.

`Defaults`

This is a special option to reset all options to their default values.

`Expand frequency` i Default = 5

This defines the initial value of an integer K that is used in an anti-cycling procedure designed to guarantee progress even on highly degenerate problems. See Section 5.2.3.

If $i \geq 9999999$, no anti-cycling procedure is invoked.

`Feasibility tolerance` r Default = $\sqrt{\mathbf{u}}$

This defines the maximum acceptable *absolute* violation in each constraint at a "feasible" point. For example, if the variables and the coefficients in the general constraints are of order unity, and the latter are correct to about 6 decimal digits, it would be appropriate to specify r as 10^{-6}. If $r < \mathbf{u}$, the default value is used.

Before optimizing the objective function, QPOPT must find a feasible point for the constraints. If the sum of infeasibilities cannot be reduced to zero and `Min sum = Yes` is requested, QPOPT will find the minimum value of the sum. Let `sinf` be the corresponding sum of infeasibilities. If `sinf` is quite small, it may be appropriate to raise r by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

`Feasibility Phase Iteration Limit` i_1 Default $= \max(50, 5(n + m_L))$

`Optimality Phase Iteration Limit` i_2 Default $= \max(50, 5(n + m_L))$

The scalars i_1 and i_2 specify the maximum number of iterations allowed in the feasibility and optimality phases. `Optimality Phase iteration limit` is equivalent to `Iteration limit`. Setting $i_1 = 0$ and `PrintLevel` > 0 means that the workspace needed will be computed and printed, but no iterations will be performed.

`Hessian rows` i Default $= 0$ or n

This specifies m, the number of rows in the Hessian matrix H or its trapezoidal factor G (as used by the default subroutine `qpHess`).

For problem type `FP` or `LP`, the default value is $m = 0$.

For problems `QP1` or `QP2`, the first m rows and columns of H are obtained from `H`, and the remainder are assumed to be zero. For problems `QP3` or `QP4`, the factor G is assumed to have m rows and n columns. They are obtained from the associated rows of `H`.

If a nonstandard subroutine `qpHess` is provided, it may access the problem type and m via the lines

```
integer            lqptyp, mHess
common    /sol1qp/ lqptyp, mHess
```

For example, `Problem type FP`, `LP` or `QP4` sets `lqptyp` $= 1$, 2 or 6 respectively, and `Hessian rows 20` sets `mHess` $=$ 20. `Infinite Bound size` r Default $= 10^{20}$

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

`Infinite Step size` r Default $= \max(\texttt{bigbnd}, 10^{20})$

If $r > 0$, r specifies the magnitude of the change in variables that will be considered a step to an unbounded solution. (Note that an unbounded solution can occur only when the Hessian is not positive definite.) If the change in x during an iteration would exceed the value of `Infinite Step`, the objective function is considered to be unbounded below in the feasible region. If $r \leq 0$, the default value is used.

`Iteration limit` i Default $= \max(50, 5(n + m_L))$

`Iters`

`Itns`

This is equivalent to `Optimality Phase iteration limit`. See `Feasibility Phase`.

List

If Nolist was previously specified, List restores output to the Print file whenever an optional parameter is reset.

Maximum degrees of freedom i Default = n

This places a limit on the storage allocated for the triangular factor R of the reduced Hessian H_R. Ideally, i should be set slightly larger than the value of n_R expected at the solution. (See Sections 5.1.2 and 5.1.3.) It need not be larger than $m_N + 1$, where m_N is the number of variables that appear nonlinearly in the quadratic objective function. For many problems it can be much smaller than m_N.

For quadratic problems, a minimizer may lie on any number of constraints, so that n_R may vary between 1 and n. The default value of i is therefore the number of variables n. If Hessian rows m is specified, the default value of i is the same number, m.

Min sum a Default = No

This option comes into effect if the constraints cannot be satisfied. If Min sum = No, QPOPT terminates as soon as it is evident that no feasible point exists. The final point will generally not be the point at which the sum of infeasibilities is minimized. If Min sum = Yes, QPOPT will continue until either the sum of infeasibilities is minimized or the iteration limit is reached, whichever occurs first.

Nolist

This suppresses output to the Print file whenever an optional parameter is reset.

Optimality tolerance r Default = $\sqrt{\mathbf{u}}$

This affects the tolerance used to determine if the Lagrange multipliers associated with the bounds and general constraints have the right "sign" for the solution to be judged optimal. Increasing r tends to cause earlier termination. For example, if $r = 1.0e-4$, the final objective value will probably agree with the true optimum to about 4 digits.

Print level i Default = 10

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

i	
0	No output.
1	The final solution only, sent to the Print file.
5	One line of output for each iteration (no printout of the final solution).
≥ 10	The final solution and one line of output for each iteration (Print file only).
≥ 20	At each iteration, the Lagrange multipliers, the variables x, the constraint values Ax and the constraint status (Print file only).

≥ 30 At each iteration, the diagonal elements of the upper-triangular matrix T associated with the TQ factorization (19) of the working set, and the diagonal elements of the upper-triangular matrix R (Print file only).

`Problem type` a Default = `QP2`

This option specifies the type of objective function to be minimized during the optimality phase. The following are the six values of a and the dimensions of the arrays that must be specified to define the objective function:

`FP`	`H` and `cvec` not accessed;
`LP`	`H` not accessed, `cvec(n)` required;
`QP1`	`H(ldH,*)` symmetric, `cvec` not referenced;
`QP2`	`H(ldH,*)` symmetric, `cvec(n)`;
`QP3`	`H(ldH,*)` upper-trapezoidal, `cvec` not referenced;
`QP4`	`H(ldH,*)` upper-trapezoidal, `cvec(n)`;

`Linear program` is equivalent to `LP`. `Quadratic program` and `QP` are equivalent to the default option `QP2`. For the QP options, the default subroutine `qpHess` requires array `H(ldH,*)` as shown. If a non-standard `qpHess` is provided, `H(*,*)` may be used in any convenient way.

`Rank tolerance` r Default = $100\mathbf{u}$

This parameter enables the user to control the condition number of the triangular factor R (see Section 5.1). If ρ_i denotes the function $\rho_i = \max\{|R_{11}|, |R_{22}|, \ldots, |R_{ii}|\}$, the dimension of R is defined to be smallest index i such that $|R_{i+1,i+1}| \leq \sqrt{r}|\rho_{i+1}|$. If $r \leq 0$, the default value is used.

`Summary file` i Default = 6

This specifies the unit number for the Summary file (see Section 5.6).

If $i > 0$ and `PrintLevel` > 0, a brief log in 80-column format is output to unit i. On many systems, the default value refers to the screen. `Summary file = 0` suppresses output, *including error messages*.

`Warm start`

See `Cold start`.

5.5 Optional parameter checklist and default values

For easy reference, the following list shows all valid options and their default values. The quantity **u** represents floating-point precision ($\approx 1.1 \times 10^{-16}$ in IEEE double-precision arithmetic).

`Check frequency`	`50`	*	
`Cold start`		*	
`Crash tolerance`	`.01`	*	
`Expand frequency`	`5`	*	
`Feasibility tolerance`	`1.1e-8`	*	$\sqrt{\mathbf{u}}$
`Feasibility Phase iteration limit`	`50`	*	or $5(n + m_L)$
`Optimality Phase iteration limit`	`50`	*	or $5(n + m_L)$
`Hessian rows`	`n`	*	
`Infinite bound size`	`1.0e+20`	*	Plus infinity
`Infinite step size`	`1.0e+20`	*	
`Iteration limit`	`50`	*	or $5(n + m_L)$
`List`		*	
`Maximum degrees of freedom`	`n`	*	
`Min sum`	`No`	*	
`Optimality tolerance`	`1.1e-8`	*	$\sqrt{\mathbf{u}}$
`Print file`	`9`	*	
`Print level`	`10`	*	
`Problem type`	`QP`	*	or `QP2`
`Rank tolerance`	`1.1e-14`	*	$100\mathbf{u}$
`Summary file`	`6`	*	

Other options may be set as follows:

```
Defaults
Nolist
Warm start
```

5.6 The Summary File

The Summary file records an iteration log and error messages. The file name is set in *Prob.SOL.SummFile*.

5.6.1 Constraint numbering and status

For items `Jdel` and `Jadd` in the iteration log, indices 1 through `n` refer to the bounds on the variables, and indices `n` + 1 through `n` + `nclin` refer to the general constraints.

When the status of a constraint changes, the index of the constraint is printed, along with the designation `L` (lower bound), `U` (upper bound), `E` (equality), `F` (temporarily fixed variable) or `A` (artificial constraint).

5.6.2 The iteration log

The following items are printed *after* each iteration.

`Itn` is the iteration count (including those from the feasibility phase).

`Jdel` is the index of the constraint deleted from the working set. If `Jdel` is zero, no constraint was deleted.

`Jadd` is the index of the constraint added to the working set. If `Jadd` is zero, no constraint was added.

`Step` is the step taken along the computed search direction. If a constraint is added during the current iteration (i.e., `Jadd` is positive), `Step` will be the step to the nearest constraint. During the optimality phase, the step can be greater than one only if the reduced Hessian is not positive definite.

`Ninf` is the number of violated constraints (infeasibilities). This number will be zero during the optimality phase.

`Sinf/Objective` is the value of the current objective function. If `x` is not feasible, `Sinf` gives a weighted sum of the magnitudes of constraint violations. If `x` is feasible, `Objective` is the value of the objective function. The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true objective at the first feasible point.

During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists. Note that the *sum* of the infeasibilities may increase or decrease during this part of the feasibility phase. However, once optimal phase-one multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities must either remain constant or be reduced until the minimum sum of infeasibilities is found.

In the optimality phase, the value of the objective is non-increasing.

`Norm gZ` is $\|Z_R^T g\|$, the Euclidean norm of the reduced gradient with respect to Z_R. During the optimality phase, this norm will be approximately zero after a unit step.

`Zr` is the number of columns of Z_R (see Section 5.1). `Zr` is the dimension of the subspace in which the objective is currently being minimized. The value of `Zr` is the number of variables minus the number of constraints in the working set.

`Art` is the number of artificial constraints in the working set, i.e., the number of columns of Z_A (see Section 5.2). At the start of the optimality phase, `Art` provides an estimate of the number of nonpositive eigenvalues in the reduced Hessian.

5.6.3 Summary file from the example problem

Following is a Summary file example.

```
                   QPOPT  ---  Version 1.0-10     Sep  1995
                   ========================================

 Itn Jdel  Jadd     Step Ninf  Sinf/Objective Norm gZ   Zr  Art
   0    0     0  0.0E+00    0  0.00000000E+00 0.0E+00    0    6
 Itn     0 -- Feasible point found.
   0    0     0  0.0E+00    0  1.51638000E+03 9.8E+01    1    5
   1    0     8U 2.8E-01    0  1.72380000E+02 0.0E+00    0    5
   2    1L   10L 3.1E-03    0  1.68083225E+02 0.0E+00    0    5
   3    5A   11L 1.2E-02    0  1.57176475E+02 0.0E+00    0    4


 Itn Jdel  Jadd     Step Ninf  Sinf/Objective Norm gZ   Zr  Art
   4    4A   12L 3.2E-02    0  1.38528925E+02 0.0E+00    0    3
   5    3A   13L 6.9E-02    0  1.11295925E+02 0.0E+00    0    2
   6    2A   14L 1.3E-01    0  7.41228000E+01 0.0E+00    0    1
   7    1A    1U 8.4E-01    0 -5.85162625E+01 0.0E+00    0    0
   8   13L    0  1.0E+00    0 -8.72144740E+01 1.3E-15    1    0


 Itn Jdel  Jadd     Step Ninf  Sinf/Objective Norm gZ   Zr  Art
   9    1U    6U 2.5E+00    0 -3.12744888E+02 1.4E+02    1    0
  10    0     1L 1.4E-01    0 -5.62265012E+02 0.0E+00    0    0
  11   14L    7U 1.3E-01    0 -6.21487825E+02 0.0E+00    0    0

 Exit from QP problem after   11 iterations.  Inform =  0


                   QPOPT  ---  Version 1.0-10     Sep  1995
                   ========================================

 Itn Jdel  Jadd     Step Ninf  Sinf/Objective Norm gZ   Zr  Art
   0    0     0  0.0E+00    3  2.35500000E+01 1.7E+00    0    3
   1    2U   10L 4.0E+00    2  1.96000000E+01 1.4E+00    0    3
   2    4U   12L 7.8E+00    1  1.17500000E+01 1.0E+00    0    3
   3    6U   14L 1.2E+01    0  0.00000000E+00 0.0E+00    0    3
 Itn     3 -- Feasible point found.
   3    0     0  0.0E+00    0  8.66526437E+02 1.5E+02    1    2


 Itn Jdel  Jadd     Step Ninf  Sinf/Objective Norm gZ   Zr  Art
   4    0     9L 1.0E-01    0  4.98244375E+01 0.0E+00    0    2
   5    2A   11L 4.5E-01    0 -5.62265013E+02 0.0E+00    0    1
   6    1A    6U 5.7E-13    0 -5.62265013E+02 0.0E+00    0    0
   7   14L    7U 1.3E-01    0 -6.21487825E+02 0.0E+00    0    0

 Exit from QP problem after    7 iterations.  Inform =  0
```

5.7 The Print File

The Print file records specified options, error messages, a detailed iteration log, and the final solution. The print file is specified in *Prob.SOL.PrintFile*.

5.7.1 Constraint numbering and status

Items `Jdel` and `Jadd` in the iteration log are the same as in the Summary file. Please see Section 5.6.1.

5.7.2 The iteration log

When `PrintLevel` ≥ 5, a line of output is produced at every iteration. The quantities printed are those in effect *on completion* of the iteration. Several items are the same as in the Summary file. Please see Section 5.6.2.

`Itn` Same as Summary file.

`Jdel` Same as Summary file.

`Jadd` Same as Summary file.

`Step` Same as Summary file.

`Ninf` Same as Summary file.

`Sinf/Objective` Same as Summary file.

`Bnd` is the number of simple bound constraints in the current working set.

`Lin` is the number of general linear constraints in the current working set.

`Art` Same as Summary file.

`Zr` Same as Summary file. `Zr` = `n` − (`Bnd` + `Lin` + `Art`).

The number of columns of Z (see Section 5.1) can be calculated as `Nz` = `n`−(`Bnd`+`Lin`) = `Zr`+`Art`. If `Nz` is zero, x lies at a vertex of the feasible region.

`Norm gZ` Same as Summary file.

`NOpt` is the number of nonoptimal Lagrange multipliers at the current point. `NOpt` is not printed if the current `x` is infeasible or no multipliers have been calculated. At a minimizer, `NOpt` will be zero.

`Min LM` is the value of the Lagrange multiplier associated with the deleted constraint. If the `Min LM` is negative, a lower bound constraint has been deleted, if `Min LM` is positive, an upper bound constraint has been deleted. If no multipliers are calculated during a given iteration, `Min LM` will be zero.

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

`Cond Rz` is a lower bound on the condition number of the triangular factor R (the Cholesky factor of the current reduced Hessian H_R, whose dimension is `Zr`). If the problem type is `LP`, `Cond Rz` is not printed.

`Rzz` is the last diagonal element ω of the matrix D associated with the $R^T DR$ factorization of the reduced Hessian H_R (see Section 5.1). `Rzz` is only printed if H_R is not positive definite (in which case $\omega \neq 1$). If the printed value of `Rzz` is small in absolute value, then H_R is approximately singular. A negative value of `Rzz` implies that the objective function has negative curvature on the current working set.

5.7.3 Printing the solution

When `PrintLevel` $= 1$ or `PrintLevel` ≥ 10, the final output from `qpopt` includes a listing of the status of every variable and constraint. Numerical values that are zero are printed as "`.`". In the "Variables" section, the following output is given for each variable x_j ($j = 1$ to `n`).

`Variable` gives j, the number of the variable.

`State` gives the state of the variable. The possible states are as follows, where δ is the `Feasibility tolerance`.

- `FR` The variable lies between its upper and lower bound.
- `EQ` The variable is a fixed variable, with x_j equal to its upper and lower bound.
- `LL` The variable is active at its lower bound (to within δ).
- `UL` The variable is active at its upper bound (to within δ).
- `TF` The variable is temporarily fixed at its current value.
- `--` The lower bound is violated by more than δ.
- `++` The upper bound is violated by more than δ.

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

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

`Value` is the final value of the variable x_j.

`Lower bound` is the lower bound specified for x_j. "`None`" indicates that `bl`$(j) \leq -$`bigbnd`.

`Upper bound` is the upper bound specified for x_j. "`None`" indicates that `bu`$(j) \geq$ `bigbnd`.

`Lagr multiplier` is the Lagrange multiplier for the associated bound. This will be zero if `State` is `FR`. If `x` is optimal, the multiplier should be non-negative if `State` is `LL`, and non-positive if `State` is `UL`.

`Slack` is the difference between the variable "`Value`" and the nearer of its (finite) bounds `bl`(j) and `bu`(j). A blank entry indicates that the associated variable is not bounded (i.e., `bl`$(j) \leq -$`bigbnd` and `bu`$(j) \geq$ `bigbnd`).

In the "Constraints" section, similar output is given for each constraint $a_i^T x$, $i = 1$ to `nclin`. The word "variable" must be replaced by "constraint", and x_j should be changed to $a_i^T x$, and (j) should be changed to $(\texttt{nclin} + i)$. "Movement off a constraint" means allowing the entry in the `slack` column to become positive.

5.7.4 Interpreting the printout

The input data for `qpopt` should always be checked (even if it terminates with `inform` = 0!). Two common sources of error are uninitialized variables and incorrectly dimensioned array arguments. The user should check that all components of `A`, `bl`, `bu` and `x` are defined on entry to `qpopt`, and that `qpHess` computes all relevant components of `Hx`.

In the following, we list the different ways in which `qpopt` terminates abnormally and discuss what further action may be necessary.

`Underflow` A single underflow will always occur if machine constants are computed automatically (as in the distributed version of QPOPT). Other floating-point underflows may occur occasionally, but can usually be ignored.

`Overflow` If the printed output before the overflow error contains a warning about serious ill-conditioning in the working set when adding the jth constraint, it may be possible to avoid the difficulty by increasing the `Feasibility tolerance`. If the message recurs, the offending linearly dependent constraint (with index "j") must be removed from the problem. If a warning message did not precede the fatal overflow, contact the authors.

`inform` = 3 The problem appears to have no feasible point. Check that there are no conflicting constraints, such as $x_1 \geq 1$, $x_2 \geq 2$ and $x_1 + x_2 = 0$. If the data for the constraints are accurate to the absolute precision σ, make sure that the `Feasibility tolerance` is *greater* than σ. For example, if all elements of `A` are of order unity and are accurate to only three decimal places, the `Feasibility tolerance` should be at least 10^{-3}.

`inform` = 4 One of the iteration limits may be too small. (See `Feasibility Phase` and `Optimality Phase`.) Increase the appropriate limit and rerun `qpopt`.

`inform` = 5 The `Maximum Degrees of Freedom` is too small. Rerun `qpopt` with a larger value (possibly using the warm start facility to specify the initial working set).

`inform` = 6 An input parameter is invalid. The printed output will indicate which parameter(s) must be redefined. Rerun with corrected values.

`inform` = 7 The specified problem type was not `FP`, `LP`, `QP1`, `QP2`, `QP3`, or `QP4`. Rerun `qpopt` with `Problem type` set to one of these values.

6 MINOS details

6.1 Introduction

TOMLAB /MINOS (hereafter referred to as MINOS) is a linear and nonlinear programming system, designed to solve large-scale constrained optimization problems of the following form:

$$\underset{x,\,y}{\text{minimize}} \qquad F(x) + c^T x + d^T y \tag{23}$$

$$\text{subject to} \quad b_1 \;\leq\; f(x) + A_1 y \;\leq\; b_2, \tag{24}$$

$$b_3 \;\leq\; A_2 x + A_3 y \;\leq\; b_4, \tag{25}$$

$$l \;\leq\; (x,\; y) \;\leq\; u, \tag{26}$$

where the vectors b_i, c, d, l, u and the matrices A_i are constant, $F(x)$ is a nonlinear function of some of the variables, and $f(x)$ is a vector of nonlinear functions. The nonlinearities (if present) may be of a general nature but must be smooth and preferably "almost linear", in the sense that they should not change radically with small changes in the variables. We make the following definitions:

x	the nonlinear variables
y	the linear variables
$(x,\; y)$	the vector $\binom{x}{y}$
(1.1)	the objective function
(1.2)	the nonlinear constraints
(1.3)	the linear constraints
(1.4)	the bounds on the variables
m	the total number of general constraints in (2) and (3)
n	the total number of variables in x and y
m_1	the number of nonlinear constraints (the dimension of $f(x)$)
n_1	the number of nonlinear variables (the dimension of x)
n_1'	the number of nonlinear objective variables (in $F(x)$)
n_1''	the number of nonlinear Jacobian variables (in $f(x)$)

A large-scale problem is one in which m and n are several hundred or several thousand. MINOS takes advantage of the fact that the constraint matrices A_i and the partial derivatives $\partial f_i(x)/\partial x_j$ are typically sparse (contain many zeros).

The dimensions n_1' and n_1'' allow for the fact that $F(x)$ and $f(x)$ may involve different sets of nonlinear variables "x". The two sets of variables *always overlap*, in the sense that the shorter "x" is always the same as the beginning of the other. If x is the same in both cases, we have $n_1 = n_1' = n_1''$. Otherwise, we define the number of nonlinear variables to be $n_1 = \max(n_1', n_1'')$.

In the following sections we introduce more terminology and give an overview of the MINOS optimization algorithms and the main system features.

6.1.1 Linear Programming

When $F(x)$ and $f(x)$ are absent, the problem becomes a *linear program*. Since there is no need to distinguish between linear and nonlinear variables, we use x rather than y. We also convert all general constraints into

equalities with the aid of slack variables s, so that the only inequalities are simple bounds on the variables. Thus, we write linear programs in the form

$$\underset{x,\ s}{\text{minimize}} \quad c^T x \text{subject to} Ax + s = b, \quad l \leq (x,\ s) \leq u. \tag{27}$$

When the constraints are linear, the bounds on the slacks are defined so that $b = 0$. When there are nonlinear constraints, some elements of b are nonzero.

In the mathematical programming world, x and s are sometimes called *structural* variables and *logical* variables. Their upper and lower bounds are fundamental to problem formulations and solution algorithms. Some of the components of l may be $-\infty$ and those of u may be $+\infty$. If $l_j = u_j$, a variable is said to be *fixed*, and if its bounds are $-\infty$ and $+\infty$, the variable is called *free*.

Within MINOS, a point (x, s) is said to be *feasible* if the following are true:

- The constraints $Ax + s = b$ are satisfied to within machine precision $\approx 10^{-15}$.
- The bounds $l \leq (x,\ s) \leq u$ are satisfied to within a *feasibility tolerance* $\delta_{\text{fea}} \approx 10^{-6}$.
- The nonlinear constraints (24) are satisfied to within a *row tolerance* $\delta_{\text{row}} \approx 10^{-6}$.

Tolerances such as δ_{fea} and δ_{row} may be specified by setting `Feasibility tolerance` and `Row tolerance`.

MINOS solves linear programs using a reliable implementation of the *primal simplex method* [8], in which the constraints $Ax + s = b$ are partitioned into the form

$$Bx_B + Nx_N = b, \tag{28}$$

where the *basis matrix* B is a square and nonsingular submatrix of $(\ A\ \ I\ )$. The elements of x_B and x_N are called the basic and nonbasic variables respectively. Together, they are a permutation of the vector $(x,\ s)$. Certain *dual variables* π and *reduced costs* d_N are defined by the equations

$$B^T\pi = c_B, \qquad d_N = c_N - N^T\pi, \tag{29}$$

where $(c_B,\ c_N)$ is a permutation of the objective vector $(c,\ 0)$.

At a feasible point, nonbasic variables are typically equal to one of their bounds, and basic variables are somewhere between their bounds. To reach an optimal solution, the simplex method performs a sequence of *iterations* of the following general nature. With guidance from d_N, a nonbasic variable is chosen to move from its current value, and the basic variables are adjusted to satisfy the constraints in (27). Usually one of the basic variables reaches a bound. The basis partition is then redefined with a column of B being replaced by a column of N. When no such interchange can be found to reduce the value of c^Tx, the current solution is optimal.

The simplex method

For convenience, let x denote the variables $(x,\ s)$. The main steps in a simplex iteration are as follows:

Compute dual variables: Solve $B^T\pi = c_B$.

Price: Compute some or all of the reduced costs $d_N = c_N - N^T\pi$ to determine if a favorable nonbasic column a_q exists.

Compute search direction: Solve $Bp_B = \pm a_q$ to determine the basic components of a search direction p along which the objective is improved. (The nonbasic elements of p are $p_N = 0$, except for ± 1 for the element corresponding to a_q.)

Find maximum steplength: Find the largest steplength $\alpha_{\max}$ such that $x + \alpha_{\max} p$ continues to satisfy the bounds on the variables. The steplength may be determined by the new nonbasic variable reaching its opposite bound, but normally some basic variable will reach a bound first.

Update: Take the step $\alpha_{\max}$. If this was determined by a basic variable, interchange the corresponding column of B with column a_q from N.

When a starting basis is chosen and the basic variables x_B are first computed, if any components of x_B lie significantly outside their bounds, we say that the current point is *infeasible*. In this case, the simplex method uses a "Phase 1" procedure to reduce the sum of infeasibilities. This is similar to the subsequent "Phase 2" procedure just described.

The feasibility tolerance δ_{fea} is used to determine which Phase is in effect. A similar *optimality tolerance* δ_{opt} is used during pricing to judge whether any reduced costs are significantly large. (This tolerance is scaled by $\|\pi\|$, a measure of the size of the current π.)

If the solution procedures are interrupted, some of the nonbasic variables may lie strictly *between* their bounds: $l_j < x_j < u_j$. In addition, at a "feasible" or "optimal" solution, some of the basic variables may lie slightly outside their bounds: $l_j - \delta_{\text{fea}} \leq x_j \leq u_j + \delta_{\text{fea}}$. In rare cases, even a few nonbasic variables might lie outside their bounds by as much as δ_{fea}.

MINOS maintains a sparse LU factorization of the basis matrix B, using a Markowitz ordering scheme and Bartels-Golub updates, as implemented in the Fortran package LUSOL [29]; see [6, 5, 39, 40]. The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.

6.1.2 Problems with a Nonlinear Objective

When nonlinearities are confined to the term $F(x)$ in the objective function, the problem is a linearly constrained nonlinear program. MINOS solves such problems using a *reduced-gradient* method [42] combined with a *quasi-Newton* method [9, 21] that generally leads to superlinear convergence. The implementation follows that described in Murtagh and Saunders [22].

As a slight generalization of (28), the constraints $Ax + s = b$ are partitioned into the form

$$Bx_B + Sx_S + Nx_N = b, \tag{30}$$

where x_S is a set of *superbasic variables*. As before, the nonbasic variables are normally equal to one of their bounds, while the basic *and* superbasic variables lie somewhere between their bounds (to within δ_{fea}). Let the number of superbasic variables be n_S, the number of columns in S. At a solution, n_S will be no more than n_1, the number of nonlinear variables, and it is often much smaller than this. In many real-life cases we have found that n_S remains reasonably small, say 200 or less, regardless of the size of the problem. This is one reason why MINOS has proved to be a practical tool.

In the reduced-gradient method, x_S is regarded as a set of "independent variables" that are allowed to move in any desirable direction to reduce the objective function (or the sum of infeasibilities). The basic variables are then adjusted in order to continue satisfying the linear constraints. If it appears that no improvement can be made with the current definition of B, S and N, one of the nonbasic variables is selected to be added to S, and the process is repeated with an increased value of n_S. At all stages, if a basic or superbasic variable encounters one of its bounds, that variable is made nonbasic and the value of n_S is reduced by one.

For linear programs, we may interpret the simplex method as being the same as the reduced-gradient method, with the number of superbasic variables oscillating between 0 and 1. (In general, a step of the simplex method *or* the reduced-gradient method is called a *minor iteration*.)

A certain matrix Z is needed for descriptive purposes. It takes the form

$$Z = \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}, \tag{31}$$

though it is never computed explicitly. Given LU factors of the basis matrix B, it is possible to compute products of the form Zq and Z^Tg by solving linear equations involving B or B^T. This in turn allows optimization to be performed on the superbasic variables, while the basic variables are adjusted to satisfy the general linear constraints. (In the description below, the reduced-gradient vector satisfies $d_S = Z^Tg$, and the search direction satisfies $p = Zp_S$.)

An important part of MINOS is the quasi-Newton method used to optimize the superbasic variables. This can achieve superlinear convergence during any sequence of iterations for which the B, S, N partition remains constant. It requires a dense upper-triangular matrix R of dimension n_S, which is updated in various ways to approximate the *reduced Hessian*:

$$R^TR \approx Z^THZ, \tag{32}$$

where H is the *Hessian* of the objective function, i.e. the matrix of second derivatives of $F(x)$. As for unconstrained optimization, the storage required for R is sometimes a limiting factor.

The reduced-gradient method

Let g be the gradient of the nonlinear objective (23). The main steps in a reduced-gradient iteration are as follows:

Compute dual variables and reduced gradient: Solve $B^T\pi = g_B$ and compute the reduced-gradient vector $d_S = g_S - S^T\pi$.

Price: If $\|d_S\|$ is sufficiently small, compute some or all of the reduced costs $d_N = g_N - N^T\pi$ to determine if a favorable nonbasic column a_q exists. If so, move that column from N into S, expanding R accordingly.

Compute search direction: Solve $R^TRp_S = -d_S$ and $Bp_B = -Sp_S$ to determine the superbasic and basic components of a search direction p along which the objective is improved. (The nonbasic elements of p are $p_N = 0$.)

Find maximum steplength: Find the largest steplength $\alpha_{\max}$ such that $x + \alpha_{\max}p$ continues to satisfy the bounds on the variables.

Perform linesearch: Find an approximate solution to the one-dimensional problem

$$\underset{\alpha}{\text{minimize}} \quad F(x + \alpha p) \text{subject to} 0 \leq \alpha \leq \alpha_{\max}.$$

Update (quasi-Newton): Take the step α. Apply a quasi-Newton update to R to account for this step.

Update (basis change): If a superbasic variable reached a bound, move it from S into N. If a basic variable reached a bound, find a suitable superbasic variable to move from S into B, and move the basic variable into N. Update R if necessary.

At an optimum, the reduced gradient d_S should be zero. MINOS terminates when $\|d_S\| \leq \delta_{\text{opt}}\|\pi\|$ and the reduced costs (component of d_N) are all sufficiently positive or negative, as judged by the same quantity $\delta_{\text{opt}}\|\pi\|$.

In the linesearch, $F(x + \alpha p)$ really means the objective function (23) evaluated at the point $(x,\ y,\ s) + \alpha p$. This steplength procedure is another important part of MINOS. Two different procedures are used, depending on whether or not all gradients are known analytically; see [26, 35]. The number of nonlinear function evaluations required may be influenced by setting the `Linesearch tolerance` in the SPECS file.

Normally, the objective function $F(x)$ will never be evaluated at a point x unless that point satisfies the linear constraints and the bounds on the variables. An exception is during a finite-difference check on the calculation of gradient elements. This check is performed at the *starting point* x_0, which takes default values or may be specified. MINOS ensures that the bounds $l \leq x_0 \leq u$ are satisfied, but in general the starting point will not satisfy the general linear constraints. If $F(x_0)$ is undefined, the gradient check should be suppressed (`Verify level -1`), or the starting point should be redefined.

6.1.3 Problems with Nonlinear Constraints

If any of the constraints are nonlinear, MINOS employs a *projected Lagrangian* algorithm, based on a method due to Robinson [41]; see Murtagh and Saunders [24]. This involves a sequence of *major iterations*, each of which requires the solution of a *linearly constrained subproblem*. Each subproblem contains linearized versions of the nonlinear constraints, as well as the original linear constraints and bounds.

At the start of the k-th major iteration, let $(x_k,\ y_k)$ be an estimate of the variables, and let λ_k be an estimate of the Lagrange multipliers (dual variables) associated with the nonlinear constraints. The constraints are linearized by changing $f(x)$ in Equation (2) to its linear approximation:

$$\bar{f}(x, x_k) = f(x_k) + J(x_k)(x - x_k),$$

or more briefly $\bar{f} = f_k + J_k(x - x_k)$, where $J(x_k)$ is the *Jacobian matrix* evaluated at x_k. (The i-th row of the Jacobian is the gradient vector for the i-th nonlinear constraint function.) The subproblem to be solved during the k-th major iteration is then

$$\underset{x,\,y}{\text{minimize}} \qquad F(x) + c^T x + d^T y - \lambda_k^T f_d + \tfrac{1}{2}\rho_k \|f_d\|^2 \tag{33}$$

$$\text{subject to} \quad b_1 \leq \bar{f} + A_1 y \leq b_2, \tag{34}$$

$$b_3 \leq A_2 x + A_3 y \leq b_4, \tag{35}$$

$$l \leq (x,\ y) \leq u, \tag{36}$$

where $f_d = f - \bar{f}$ is the difference between $f(x)$ and its linearization. The objective function (33) is called an *augmented Lagrangian*. The scalar ρ_k is a *penalty parameter*, and the term involving ρ_k is a modified *quadratic penalty function*. MINOS uses the reduced-gradient method to solve each subproblem. As before, slack variables are introduced and the vectors b_i are incorporated into the bounds on the slacks. The linearized constraints take the form

$$\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} J_k x_k - f_k \\ 0 \end{pmatrix}.$$

We refer to this system as $Ax + s = b$ as in the linear case. The Jacobian J_k is treated as a sparse matrix, the same as the matrices A_1, A_2 and A_3. The quantities J_k, b, λ_k and ρ_k change each major iteration.

The projected Lagrangian method

For convenience, suppose that all variables and constraints are nonlinear. The main steps in a major iteration are as follows:

Solve subproblem: Find an approximate solution $(\bar{x}, \bar{\lambda})$ to the kth subproblem (33)–(36).

Compute search direction: Adjust the elements of $\bar{\lambda}$ if necessary (if they have the wrong sign). Define a search direction $(\Delta x,\ \Delta\lambda) = (\bar{x} - x_k,\ \bar{\lambda} - \lambda_k)$.

Find steplength: Choose a steplength σ such that some merit function $M(x, \lambda)$ has a suitable value at the point $(x_k + \sigma\Delta x,\ \lambda_k + \sigma\Delta\lambda)$.

Update: Take the step σ to obtain $(x_{k+1},\ \lambda_{k+1})$. In some cases, adjust ρ_k.

For the first major iteration, the nonlinear constraints are ignored and minor iterations are performed until the original linear constraints are satisfied.

The initial Lagrange multiplier estimate is typically $\lambda_k = 0$ (though it can be provided by the user). If a subproblem terminates early, some elements of the new estimate $\bar{\lambda}$ may be changed to zero.

The penalty parameter initially takes a certain default value $\rho_k = 100.0/m_1$, where m_1 is the number of nonlinear constraints. (A value r times as big is obtained by specifying `Penalty parameter` r.) For later major iterations, ρ_k is reduced in stages when it appears that the sequence $\{x_k, \lambda_k\}$ is converging. In many cases it is safe to specify `Penalty parameter 0.0` at the beginning, particularly if a problem is only mildly nonlinear. This may improve the overall efficiency.

In the output from MINOS, the term `Feasible subproblem` indicates that the *linearized constraints* (34)–(36) have been satisfied. In general, the nonlinear constraints are satisfied only in the limit, so that *feasibility* and *optimality* occur at essentially the same time. The nonlinear constraint violation is printed every major iteration. Even if it is zero early on (say at the initial point), it may increase and perhaps fluctuate before tending to zero. On "well behaved" problems, the constraint violation will decrease quadratically (i.e., very quickly) during the final few major iterations.

For certain rare classes of problem it is safe to request the values $\lambda_k = 0$ and $\rho_k = 0$ for all subproblems by specifying `Lagrangian = No` (in which case the nonlinear constraint functions are evaluated only once per major iteration). However for general problems, convergence is much more likely with the default setting, `Lagrangian = Yes`.

The merit function

Unfortunately, it is not known how to define a merit function $M(x, \lambda)$ that can be *reduced* at every major iteration. As a result, there is no guarantee that the projected Lagrangian method described above will converge from an arbitrary starting point. This has been the principal theoretical gap in MINOS, finally resolved by the PhD research of Michael Friedlander [12]. The main features needed to stabilize MINOS are:

- To relax the linearized constraints via an ℓ_1 penalty function.
- To repeat a major iteration with increased ρ_k (and more relaxed linearized constraints) if the nonlinear constraint violation would increase too much.

In practice, the method of MINOS 5.51 often does converge and a good *rate* of convergence is often achieved in the final major iterations, particularly if the constraint functions are "nearly linear". As a precaution, MINOS

prevents radical changes from one major iteration to the next. Where possible, the steplength is chosen to be $\sigma = 1$, so that each new estimate of the solution is $(x_{k+1},\ \lambda_{k+1}) = (\bar{x}, \bar{\lambda})$, the solution of the subproblem. If this point is "too different", a shorter steplength $\sigma < 1$ is chosen.

If the major iterations for a particular model do not appear to be converging, some of the following actions may help:

1. Specify initial activity levels for the nonlinear variables as carefully as possible.
2. Include sensible upper and lower bounds on all variables.
3. Specify a `Major damping parameter` that is lower than the default value. This tends to make σ smaller.
4. Specify a `Penalty parameter` that is higher than the default value. This tends to prevent excessive departures from the constraint linearization.

6.1.4 Problem Formulation

In general, it is worthwhile expending considerable prior analysis to make the constraints completely linear if at all possible. Sometimes a simple transformation will suffice. For example, a pipeline optimization problem has pressure drop constraints of the form

$$\frac{K_1}{d_1^{4.814}} + \frac{K_2}{d_2^{4.814}} + \cdots \leq P_T^2 - P_0^2,$$

where d_i are the design variables (pipe diameters) and the other terms are constant. These constraints are highly nonlinear, but by redefining the decision variables to be $x_i = 1/d_i^{4.814}$ we can make the constraints linear. Even if the objective function becomes more nonlinear by such a transformation (and this usually happens), the advantages of having linear constraints greatly outweigh this.

Similarly, it is usually best not to move nonlinearities from the objective function into the constraints. For example, we should *not* replace "minimize $F(x)$" by

$$\text{minimize} \quad z \quad \text{subject to} \quad F(x) - z = 0.$$

Scaling is a very important matter during problem formulation. A general rule is to scale both the data and the variables to be as close to 1.0 as possible. In general we suggest the range 1.0 to 10.0. When conflicts arise, one should sacrifice the objective function in favor of the constraints. Real-world problems tend to have a natural scaling within each constraint, as long as the variables are expressed in consistent physical units. Hence it is often sufficient to apply a scale factor to each row. MINOS has options to scale the rows and columns of the constraint matrix automatically. By default, only the linear rows and columns are scaled, and the procedure is reliable. If you request that the nonlinear constraints and variables be scaled, bear in mind that the scale factors are determined by the initial Jacobian $J(x_0)$, which may differ considerably from $J(x)$ at a solution.

Finally, *upper and lower bounds* on the variables (and on the constraints) are extremely useful for confining the region over which optimization has to be performed. If sensible values are known, they should always be used. They are also important for avoiding singularities in the nonlinear functions. Note that bounds may be violated slightly by as much as the feasibility tolerance δ_{fea}. Hence, if $\sqrt{x_2}$ or $\log x_2$ appear (for example) and if $\delta_{\text{fea}} = 10^{-6}$, the lower bound on x_2 would normally have to be at least 10^{-5}. If it is *known* that x_2 will be at least 0.5 (say) at a solution, then its lower bound should be 0.5.

For a detailed discussion of many aspects of numerical optimization, see Gill, Murray and Wright [35]; in particular, see Chapter 8 for much invaluable advice on problem formulation and assessment of results.

6.1.5 Restrictions

MINOS is designed to find solutions that are *locally optimal*. The nonlinear functions in a problem must be *smooth* (i.e., their first derivatives must exist), especially near the desired solution. The functions need not be separable.

A certain "feasible" region is defined by the general constraints and the bounds on the variables. If the objective is convex within this region and if the feasible region itself is convex, any optimal solution obtained will be a *global optimum*. Otherwise there may be several local optima, and some of these may not be global. In such cases the chances of finding a global optimum are usually increased by choosing a starting point that is "sufficiently close", but there is no general procedure for determining what "close" means, or for verifying that a given local optimum is indeed global.

Integer restrictions cannot be imposed directly. If a variable x_j is required to be 0 or 1, a common ploy is to include a quadratic term $x_j(1 - x_j)$ in the objective function. MINOS will indeed terminate with $x_j = 0$ or 1, but inevitably the final solution will just be a local optimum. (Note that the quadratic is negative definite. MINOS will find a global minimum for quadratic functions that are positive definite or positive semidefinite, assuming the constraints are linear.)

6.2 Solver Options

The following sections describes some of the solver options depending on problem type.

6.2.1 Options for Linear Programming

The following options apply specifically to linear programs.

`Crash option`	i	Default = 3
`Crash tolerance`	t	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 $(A \;\; I)$. 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.

$i = 0$ 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 A.

2 Crash is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the first major iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the second major iteration and Crash is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).

3 Crash is called up to three times (if there are nonlinear constraints). The first two calls treat *linear equalities* and *linear inequalities* separately. As before, the last call treats nonlinear rows at the start of the second major iteration.

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` allows 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 $|a_{ij}| \leq a_{\max} \times t$. (To be meaningful, t should be in the range $0 \leq t < 1$.)

When $t > 0.0$, the bases obtained by Crash may not be strictly triangular, but are likely to be nonsingular and almost triangular. The intention is to choose a 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 are 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` t for some value of $t > 1/4$.

6.2.2 Options for All Problems

The following options have the same purpose for all problems, whether they linear or nonlinear.

`Check frequency` k Default = 60

Every k-th minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax + s = b$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax - s$ 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.

`Cycle limit`	l	Default = 1
`Cycle print`	p	Default = 1
`Cycle tolerance`	t	Default = 0.0
`Phantom columns`	c	Default = 0
`Phantom elements`	e	Default = 0

`Debug level` l Default = 0

This causes various amounts of information to be output to the Print file. Most debug levels are not helpful to normal users, but they are listed here for completeness.

- $l = 0$ No debug output.
- $l = 2$ (or more) Output from `m5setx` showing the maximum residual after a row check.
- $l = 40$ Output from `lu8rpc` (which updates the LU factors of the basis matrix), showing the position of the last nonzero in the transformed incoming column.
- $l = 50$ Output from `lu1mar` (which computes the LU factors each refactorization), showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.
- $l = 100$ Output from `m2bfac` and `m5log` listing the basic and superbasic variables and their values at every iteration.

`Expand frequency` k Default = 10000

This option is part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems. See [31].

"Cycling" can occur only if zero steplengths are allowed. Here, 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 k iterations, the tolerance actually used by MINOS increases from 0.5δ to δ (in steps of $0.5\delta/k$).

Every k iterations, or when feasibility and optimality are first detected, a resetting procedure eliminates any infeasible nonbasic variables. Some additional iterations may be needed to restore feasibility and optimality. Increasing k reduces that likelihood, but it gives less freedom to choose large pivot elements during basis changes. (See `Pivot tolerance`.)

`Factorization frequency` k Default = 100 (LP) or 50 (NLP)

With linear programs, most iterations cause a basis change, in which one column of the basis matrix B is replaced by another. The LU factors of B must be updated accordingly. At most k updates are performed before the current B is factorized directly.

Each update tends to add nonzeros to the LU factors. Since the updating method is stable, k mainly affects the efficiency of minor iterations, rather than stability.

High values of k (such as 100 or 200) may be more efficient on "dense" problems, when the factors of B tend to have two or three times as many nonzeros as B itself. Lower values of k may be more efficient on problems that are very sparse.

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

This sets the feasibility tolerance $\delta_{\text{fea}} = t$ (see §6.2.1). A variable or constraint is considered *feasible* if it does not lie outside its bounds by more than δ_{fea}.

MINOS first attempts to satisfy the linear constraints and bounds. If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible. Let *sinf* be the corresponding sum of infeasibilities. 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. If *sinf* is not small, there may be other points that have a significantly smaller sum of infeasibilities. MINOS does not attempt to find a solution that minimizes the sum.

For `Scale option` 1 or 2, feasibility is defined in terms of the *scaled* problem (since it is then more likely to be meaningful). The final unscaled solution can therefore be infeasible by an unpredictable amount, depending on the size of the scale factors. Precautions are taken so that in a "feasible solution" the original variables will never be infeasible by more than 0.1. Values that large are very unlikely.

`Iterations limit` k Default = $3m$

MINOS stops after k iterations even if the simplex method has not yet reached a solution. If $k = 0$, no iterations are performed, but the starting point is tested for both feasibility and optimality.

`LU factor tolerance` t_1 Default = 100.0 (LP) or 5.0 (NLP)
`LU update tolerance` t_2 Default = 10.0 (LP) or 5.0 (NLP)

These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy $t_1, t_2 \geq 1.0$. The matrix L is a product of matrices of the form

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

where the multipliers μ satisfy $|\mu| \leq t_i$. Values near 1.0 favor stability, while larger values favor sparsity. The default values usually strike a good compromise. For large and relatively dense problems, $t_1 = 10.0$ 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 t_1 and/or t_2 in order to

achieve stability. For example, if the columns of A include a submatrix of the form

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

one should set both t_1 and t_2 to values in the range $1.0 \leq t_i < 4.0$.

`LU density tolerance`	t_3	Default = 0.5
`LU singularity tolerance`	t_4	Default = $\epsilon^{0.67} \approx 3.25e-11$

The density tolerance t_3 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 t_3, the Markowitz strategy for choosing pivots is altered to reduce the time spent searching for each remaining pivot. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.

The singularity tolerance t_4 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \leq t_4$ or $|U_{jj}| < t_4 \max_i |U_{ij}|$, the j-th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)

In some cases, the Jacobian matrix may converge to values that make the basis exactly singular. (For example, a whole row of the Jacobian could be zero at an optimal solution.) Before exact singularity occurs, the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting t_4 = `1.0e-5`, say, may help cause a judicious change of basis.

`Maximize`	
`Minimize`	Default

This specifies the required direction of optimization.

`Multiple price`	k	Default = 1

It is not normal to set $k > 1$ for linear programs, as it causes MINOS to use the reduced-gradient method rather than the simplex method. The number of iterations, and the total work, are likely to increase.

The reduced-gradient iterations do *not* correspond to the very efficient multiple pricing "minor iterations" carried out by certain commercial linear programming systems. Such systems require storage for k dense vectors of dimension m, so that k is usually limited to 5 or 6. In MINOS, the total storage requirements increase only slightly with k. (The `Superbasics limit` must be at least k.)

`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 gradient of the objective

function corresponding to the j-th variable, a_j is the associated column of the constraint matrix (or Jacobian), and π is the set of dual variables.

By construction, the reduced gradients for basic variables are always zero. Optimality is declared if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy $d_j/\|\pi\| \geq -t$ or $d_j/\|\pi\| \leq t$ respectively, and if $|d_j|/\|\pi\| \leq t$ for superbasic variables.

In those tests, $\|\pi\|$ is a measure of the size of the dual variables. It is included to make the tests independent of a large scale factor on the objective function. The quantity actually used is defined by $\sigma = \sum_{i=1}^{m} |\pi_i|$, $\|\pi\| = \max\{\sigma/\sqrt{m}, 1.0\}$, so that only scale factors larger than 1.0 are allowed for. If the objective is scaled down to be very *small*, the optimality test reduces to comparing d_j against t.

`Partial price` p Default = 10 (LP) or 1 (NLP)

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 basic or superbasic).

When $p = 1$, all columns of the constraint matrix (A I) are searched. Otherwise, A and I are partitioned to give p roughly equal segments A_j, I_j ($j = 1$ to p). 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}. (Subscripts are modulo p.)

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. (Several may be selected if multiple pricing has been specified.) 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` t Default = $\epsilon^{2/3} \approx 10^{-11}$

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 t. For nonlinear problems, elements smaller than $t\|p\|$ are ignored.

It is common for two or more variables to reach a bound at essentially the same time. In such cases, the `Feasibility tolerance` provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small Feasibility tolerances should therefore not be specified.

To a lesser extent, the `Expand frequency` also provides some freedom to maximize the pivot element. Excessively *large* Expand frequencies should therefore not be specified.

`Scale option`	l	Default = 2 (LP) or 1 (NLP)
`Scale`	`Yes`	
`Scale`	`No`	
`Scale linear variables`		Same as `Scale option 1`
`Scale nonlinear variables`		Same as `Scale option 2`
`Scale all variables`		Same as `Scale option 2`
`Scale tolerance`	t	Default = 0.9
`Scale, Print`		
`Scale, Print, Tolerance`	t	

Three scale options are available as follows:

- $l = 0$ No scaling. If storage is at a premium, this option saves $m + n$ words of workspace.
- $l = 1$ Linear 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, 1982). This sometimes improves the performance of the solution procedures.
- $l = 2$ All 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 $(A \;\; I)$ that are fixed or have positive lower bounds or negative upper bounds.

`Scale Yes` sets the default scaling. (*Caution*: If all variables are nonlinear, `Scale Yes` unexpectedly does *nothing*, because there are no linear variables to scale.) `Scale No` suppresses scaling (equivalent to `Scale option 0`).

If nonlinear constraints are present, `Scale option 1` or `0` should generally be tried at first. `Scale option 2` gives scales that depend on the initial Jacobian, and should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

`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 $j > n$.

All forms except `Scale option` may specify a tolerance t, where $0 < t < 1$ (for example: `Scale, Print, Tolerance = 0.99`). This 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 t times its previous value, another scaling pass is performed to adjust the row and column scales. Raising t from 0.9 to 0.99 (say) usually increases the number of scaling passes through A. At most 10 passes are made.

If a `Scale option` has not already been specified, `Scale, Print` or `Scale tolerance` both set the default scaling.

`Weight on linear objective`	w	Default = 0.0

This keyword invokes the so-called *composite objective* technique, if the first solution obtained is infeasible, and if the objective function contains linear terms. While trying to reduce the sum of infeasibilities, the method also

attempts to optimize the linear objective. At each infeasible iteration, the objective function is defined to be

$$\underset{x}{\text{minimize}} \quad \sigma w(c^T x) + (\text{sum of infeasibilities}),$$

where $\sigma = 1$ for minimization, $\sigma = -1$ for maximization, and c is the linear objective. If an "optimal" solution is reached while still infeasible, w is reduced by a factor of 10. This helps to allow for the possibility that the initial w is too large. It also provides dynamic allowance for the fact that the sum of infeasibilities is tending towards zero.

The effect of w is disabled after 5 such reductions, or if a feasible solution is obtained.

The `Weight` option is intended mainly for linear programs. It is unlikely to be helpful on nonlinear problems.

6.2.3 Options for Nonlinear Objectives

The following options apply to nonlinear programs whose constraints are linear.

`Crash option`	l	Default = 3
`Crash tolerance`	t	Default = 0.1

These options are the same as for linear programs.

6.2.4 Options for All Nonlinear problems

`Expand frequency`	k	Default = 10000

This option is used the same as for linear programs, but takes effect only when there is just one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region. Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.) Increasing k 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`).

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

When the constraints are linear, a *feasible solution* is one in which all variables, including slacks, satisfy their upper and lower bounds to within the absolute tolerance t. (Since slacks are included, this means that the general linear constraints are also satisfied to within t.)

When nonlinear constraints are present, a *feasible subproblem* is one in which the linear constraints and bounds, as well as the current linearization of the nonlinear constraints, are satisfied to within the tolerance t.

MINOS first attempts to satisfy the linear constraints and bounds. If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible.

Normally, the nonlinear functions $F(x)$ and $f(x)$ are evaluated only at points x that satisfy the linear constraints and bounds. If the functions are undefined in certain regions, every attempt should be made to eliminate these regions from the problem. For example, for a function $F(x) = \sqrt{x_1} + \log x_2$, it would be essential to place lower bounds on both variables. If `Feasibility tolerance` $= 10^{-6}$, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might

be appropriate. (The log singularity is more serious; in general, keep variables as far away from singularities as possible.)

An exception is during optional gradient checking (see `Verify`), which occurs before any optimization takes place. The points at which the functions are evaluated satisfy the bounds but not necessarily the general constraints. If this causes difficulty, gradient checking should be suppressed by setting `Verify level -1`.

If a subproblem is infeasible, the bounds on the linearized constraints are relaxed in several stages until the subproblem appears feasible. (The true bounds are restored for the next subproblem.) This approach sometimes allows the optimization to proceed successfully. In general, infeasible subproblems are a symptom of difficulty and it may be necessary to increase the `Penalty parameter` or alter the starting point.

Note: Feasibility with respect to the nonlinear constraints is measured against the `Row tolerance`, not the `Feasibility tolerance`.

`Hessian dimension` r Default = 50

This specifies that an $r \times r$ triangular matrix R is to be available for use by the quasi-Newton algorithm (to approximate the reduced Hessian matrix according to $Z^THZ \approx R^TR$). Suppose there are s superbasic variables at a particular iteration. *Whenever possible, r should be greater than s.*

If $r \geq s$, the first s columns of R are used to approximate the reduced Hessian in the normal manner. If there are no further changes to the set of superbasic variables, the rate of convergence is usually superlinear. If $r < s$, a matrix of the form

$$R = \begin{pmatrix} R_r & 0 \\ & D \end{pmatrix}$$

is used to approximate the reduced Hessian, where R_r is an $r \times r$ upper triangular matrix and D is a *diagonal* matrix of order $s - r$. The rate of convergence is no longer superlinear (and may be arbitrarily slow).

The storage required is of order $\frac{1}{2}r^2$, which is substantial if r is as large as 1000 (say). In general, r should be a slight over-estimate of the final number of superbasic variables, whenever storage permits. It need not be larger than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems it can be much smaller than n_1.

`Iterations limit` k Default = $3m + 10n_1$

If the constraints are linear, this is the maximum number of iterations allowed for the simplex method or the reduced-gradient method. Otherwise, it is the maximum number of *minor* iterations, summed over all major iterations.

If $k = 0$, no minor iterations are performed, but the starting point is tested for both feasibility and optimality.

`Linesearch tolerance` t Default = 0.1

For nonlinear problems, this controls the accuracy with which a steplength α is located during one-dimensional searches of the form

$$\underset{\alpha}{\text{minimize}} \quad F(x + \alpha p) \text{subject to} 0 < \alpha \leq \beta.$$

A linesearch occurs on most minor iterations for which x is feasible. (If the constraints are nonlinear, the function being minimized is the augmented Lagrangian in equation (5).)

The value of t must satisfy $0.0 \leq t < 1.0$. The default value $t = 0.1$ requests a moderately accurate search, and should be satisfactory in most cases. If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate; try $t = 0.01$ or $t = 0.001$. The number of iterations should decrease, and this will reduce total run time if there are many linear or nonlinear constraints. If the nonlinear functions are *expensive* to evaluate, a less accurate search may be appropriate; try $t = 0.5$ or perhaps $t = 0.9$. (The number of iterations will probably increase, but the total number of function evaluations may decrease enough to compensate.)

`LU singularity tolerance`	t_3	Default $= \epsilon^{0.67} \approx 3.25e-11$
`LU swap tolerance`	t_4	Default $= \epsilon^{1/4} \approx 10^{-4}$

The singularity tolerance t_3 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \leq t_3$ or $|U_{jj}| < t_3 \max_i |U_{ij}|$, the j-th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)

In some cases, the Jacobian matrix may converge to values that make the basis exactly singular. (For example, a whole row of the Jacobian could be zero at an optimal solution.) Before exact singularity occurs, the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting $t_3 =$ `1.0e-5`, say, may help cause a judicious change of basis.

The `LU swap tolerance` is somewhat similar but can take effect more easily. It is again used only after a basis factorization, and normally just at the start of a major iteration. If a diagonal of U seems to be rather small (as measured by t_4) relative to the biggest diagonal of U, a basis change is made in which the basic variable associated with the small diagonal of U is swapped with a carefully chosen superbasic variable (if there are any). The number of superbasic variables stays the same. A message is printed to advise that a swap has occurred.

In practice this tends to help problems whose basis is becoming ill-conditioned. If the number of swaps becomes excessive, set `LU swap tolerance 1.0e-6`, say, or perhaps smaller.

`Minor damping parameter`	d	Default $= 2.0$

This parameter limits the change in x during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

A linesearch of the form $\text{minimize}_\alpha\, F(x + \alpha p)$ is performed over the range $0 < \alpha \leq \beta$, where β is the step to the nearest upper or lower bound on x. Normally, the first steplength tried is $\alpha_1 = \min(1, \beta)$, but in some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow.

The parameter d is therefore used to define a limit $\bar{\beta} = d(1 + \|x\|)/\|p\|$, and the first evaluation of $F(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \bar{\beta}, \beta)$.

Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The `Minor damping parameter` provides an additional safeguard. The default value $d = 2.0$ should not affect progress on well behaved problems, but setting $d = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A "good" starting point may be required. An important application is to the class of nonlinear least-squares problems.

In cases where several local optima exist, specifying a small value for d may help locate an optimum near the starting point.

`Multiple price`	k	Default = 1

"Pricing" refers to a scan of the current nonbasic variables to determine if any should be changed from their current value (by allowing them to become superbasic or basic).

If multiple pricing is in effect, the k best nonbasic variables are selected for admission to the superbasic set. ("Best" means the variables with largest reduced gradients of appropriate sign.) If partial pricing is also in effect, the k best variables are selected from the current partition of A and I.

On large nonlinear problems it may help to set $k > 1$ if there are not many superbasic variables at the starting point but many at the optimal solution.

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

`Partial price`	p	Default = 10 (LP) or 1 (NLP)

This parameter may be useful for large problems that have significantly more variables than constraints. Larger values reduce the work required for each "pricing" operation (when a nonbasic variable is selected to become basic or superbasic).

`Scale option`	l	Default = 2 (LP) or 1 (NLP)
`Scale`	`Yes`	
`Scale`	`No`	
`Scale linear variables`		Same as `Scale option 1`
`Scale nonlinear variables`		Same as `Scale option 2`
`Scale all variables`		Same as `Scale option 2`
`Scale tolerance`	t	Default = 0.9
`Scale, Print`		
`Scale, Print, Tolerance`	t	

Three scale options are available as follows:

- $l = 0$ No scaling. If storage is at a premium, this option saves $m + n$ words of workspace.
- $l = 1$ If some of the variables are linear, the constraints and linear variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0.
- $l = 2$ 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 $(A \;\; I)$ that are fixed or have positive lower bounds or negative upper bounds.

`Scale option 1` is the default for nonlinear problems. (Only linear variables are scaled.)

`Scale Yes` sets the default. (*Caution*: If all variables are nonlinear, `Scale Yes` unexpectedly does *nothing*, because there are no linear variables to scale.) `Scale No` suppresses scaling (equivalent to `Scale option 0`).

The `Scale tolerance` and `Scale, Print` options are the same as for linear programs.

`Subspace tolerance` t Default = 0.5

This controls the extent to which optimization is confined to the current set of basic and superbasic variables (Phase 4 iterations), before one or more nonbasic variables are added to the superbasic set (Phase 3). The value specified must satisfy $0 < t \leq 1$.

When a nonbasic variable x_j is made superbasic, the norm of the reduced-gradient vector (for all superbasics) is recorded. Let this be $\|Z^T g_0\|$. (In fact, the norm is $|d_j|$, the size of the reduced gradient for the new superbasic variable x_j.)

Subsequent Phase 4 iterations continue at least until the norm of the reduced-gradient vector satisfies $\|Z^T g\| \leq t \times \|Z^T g_0\|$. ($\|Z^T g\|$ is the size of the largest reduced-gradient among the superbasic variables.)

A smaller value of t is likely to increase the total number of iterations, but may reduce the number of basis changes. A larger value such as $t = 0.9$ may sometimes lead to improved overall efficiency, if the number of superbasic variables is substantially larger at the optimal solution than at the starting point.

Other convergence tests on the change in the function being minimized and the change in the variables may prolong Phase 4 iterations. This helps to make the overall performance insensitive to larger values of t.

`Superbasics limit` s Default = 50

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

For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables". Normally, s need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems, s may be considerably smaller than n_1. This saves storage if n_1 is very large.

If `Hessian dimension` r is specified, the default value of s is the same number (and conversely). This is a safeguard to ensure superlinear convergence wherever possible. Otherwise, the default for both r and s is 50.

`Unbounded objective value` r_1 Default = `1.0e+20`
`Unbounded step size` r_2 Default = `1.0e+10`

These parameters are intended to detect unboundedness in nonlinear problems. During a linesearch of the form $\min_\alpha F(x + \alpha p)$, if $|F|$ exceeds r_1 or if α exceeds r_2, iterations are terminated with the exit message `Problem is unbounded (or badly scaled)`.

If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$), before the test against r_1 can be made.

Unboundedness in x is best avoided by placing finite upper and lower bounds on the variables. See also the `Minor damping parameter`.

`Verify level`	l	Default = 0
`Verify objective gradients`		Same as `Verify level 1`
`Verify constraint gradients`		Same as `Verify level 2`
`Verify`		Same as `Verify level 3`
`Verify gradients`		Same as `Verify level 3`
`Verify`	`Yes`	Same as `Verify level 3`
`Verify`	`No`	Same as `Verify level 0`

These options refer to a check on the gradients computed by your nonlinear function routines `funobj` and `funcon` at the starting point (the initial value of the nonlinear variables `x(*)`). Values output in the gradient array `g(*)` are compared with estimates obtained by finite differences.

$l = 0$	Only a "cheap" test is performed, requiring three evaluations of the nonlinear objective (if any) and two evaluations of the nonlinear constraints.
$l = 1$	A more reliable check is made on each component of the objective gradient.
$l = 2$	A check is made on each column of the Jacobian matrix associated with the nonlinear constraints.
$l = 3$	A detailed check is made on both the objective and the Jacobian.
$l = -1$	No checking is performed. This may be necessary if the functions are undefined at the starting point.

`Verify level 3` is recommended for a new function routine, particularly if the "cheap" test indicates error. Missing gradients are not checked (so there is no overhead). If there are many nonlinear variables, the `Start` and `Stop` keywords may be used to limit the check to a subset.

As noted, gradient verification occurs at the starting point, before a problem is scaled, and before the first basis is factorized. The bounds on x will be satisfied, but the general linear constraints may not. If the nonlinear objective or constraint functions are undefined, you could initially specify

```
Objective = NONE
Nonlinear objective variables   0
Major iterations                1
Verify level                   -1
```

to obtain a point that satisfies the linear constraints, and then restart with the correct linear and nonlinear objective, along with

```
Verify level                    3
```

6.2.5 Options for Nonlinear Constraints

The following options apply to problems with nonlinear constraints.

`Completion`	`Partial`	Default
`Completion`	`Full`	

When there are nonlinear constraints, this determines whether subproblems should be solved to moderate accuracy

(partial completion), or to full accuracy (full completion). MINOS implements the option by using two sets of convergence tolerances for the subproblems.

Use of partial completion may reduce the work during early major iterations, unless the `Minor iterations` limit is active. The optimal set of basic and superbasic variables will probably be determined for any given subproblem, but the reduced gradient may be larger than it would have been with full completion.

An automatic switch to full completion occurs when it appears that the sequence of major iterations is converging. The switch is made when the nonlinear constraint error is reduced below 100*(`Row tolerance`), the relative change in λ_k is 0.1 or less, and the previous subproblem was solved to optimality.

Full completion tends to give better Lagrange-multiplier estimates. It may lead to fewer major iterations, but may result in more minor iterations.

`Crash option`	l	Default = 3
`Crash tolerance`	t	Default = 0.1

Let A refer to the linearized constraint matrix.

- $l = 0$ The initial basis contains only slack variables: $B = I$.
- $l = 1$ A is evaluated at the starting point. Crash is called once, looking for a triangular basis in all rows and columns of A.
- $l = 2$ A is evaluated only after the linear constraints are satisfied. Crash is called twice. The first call looks for a triangular basis in linear rows. The first major iteration proceeds with simplex-type iterations until the linear constraints are satisfied. A is then evaluated for the second major iteration and Crash is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).
- $l = 3$ Crash is called three times, treating *linear equalities* and *linear inequalities* separately, with simplex-type iterations in between. As before, the last call treats nonlinear rows at the start of the second major iteration.

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

A "feasible subproblem" is one in which the linear constraints and bounds, as well as the current linearization of the nonlinear constraints, are satisfied to within t.

Note that feasibility with respect to the nonlinear constraints is determined by the `Row tolerance` (not the `Feasibility tolerance`).

MINOS first attempts to satisfy the linear constraints and bounds. If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible.

If `Scale option` = 1 or 2, feasibility is defined in terms of the scaled problem (since it is then more likely to be meaningful).

Normally, the nonlinear functions $F(x)$ and $f(x)$ are evaluated only at points x that satisfy the linear constraints and bounds. If the functions are undefined in certain regions, every attempt should be made to eliminate these regions from the problem. For example, for a function $F(x) = \sqrt{x_1} + \log x_2$, it would be essential to place lower bounds on both variables. If `Feasibility tolerance` = 10^{-6}, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might

be appropriate. (The log singularity is more serious; in general, keep variables as far away from singularities as possible.)

An exception is during optional gradient checking (see `Verify`), which occurs before any optimization takes place. The points at which the functions are evaluated satisfy the bounds but not necessarily the general constraints. If this causes difficulty, gradient checking should be suppressed by setting `Verify level -1`.

If a subproblem is infeasible, the bounds on the linearized constraints are relaxed in several stages until the subproblem appears feasible. (The true bounds are restored for the next subproblem.) This approach sometimes allows the optimization to proceed successfully. In general, infeasible subproblems are a symptom of difficulty and it may be necessary to increase the `Penalty parameter` or alter the starting point.

`Lagrangian`	`Yes`	Default
`Lagrangian`	`No`	

This determines the form of the objective function used for the linearized subproblems. The default value `Yes` is highly recommended. The `Penalty parameter` value is then also relevant.

If `No` is specified, the nonlinear constraint functions are evaluated only twice per major iteration. Hence this option may be useful if the nonlinear constraints are very expensive to evaluate. However, in general there is a great risk that convergence may not be achieved.

`Major damping parameter`	d	Default = 2.0

This parameter may assist convergence on problems that have highly nonlinear constraints. It is intended to prevent large relative changes between subproblem solutions (x_k, λ_k) and (x_{k+1}, λ_{k+1}). For example, the default value 2.0 prevents the relative change in either x_k or λ_k from exceeding 200 per cent. It will not be active on well behaved problems. (If all components of x_k or λ_k are small, the norms of those vectors will not be allowed to increase beyond about 2.0.)

The parameter is used to interpolate between the solutions at the beginning and end of each major iteration. Thus, x_{k+1} and λ_{k+1} are changed to

$$x_k + \sigma(x_{k+1} - x_k) \text{and} \lambda_k + \sigma(\lambda_{k+1} - \lambda_k)$$

for some steplength $\sigma < 1$. In the case of nonlinear equations (where the number of constraints is the same as the number of variables) this gives a *damped Newton method*.

This is a very crude control. If the sequence of major iterations does not appear to be converging, one should first re-run the problem with a higher `Penalty parameter` (say 2, 4 or 10). (Skip this re-run in the case of a square system of nonlinear equations: there are no degrees of freedom and the `Penalty parameter` value has essentially no effect.)

If the subproblem solutions continue to change violently, try reducing d to 0.2 or 0.1 (say).

`Major iterations`	k	Default = 50

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the nonlinear constraints, since in some cases the sequence of major iterations may not converge.

The progress of the major iterations can be best monitored using `Print level 0` (the default).

`Minor iterations` k Default = 40

This is the maximum number of minor iterations allowed during a major iteration, *after* the linearized constraints for that subproblem have been satisfied. (An arbitrary number of minor iterations may be needed to find a feasible point for each subproblem.) The `Iterations limit` provides an independent limit on the total minor iterations (across all subproblems).

A moderate value (e.g., $30 \leq k \leq 200$) prevents excessive effort being expended on early major iterations, but allows later subproblems to be solved to completion.

The first major iteration is special: it terminates as soon as the linear constraints and bounds are satisfied (if possible), ignoring the nonlinear constraints.

In general it is unsafe to specify a value as small as $k = 1$ or 2. (Even when an optimal solution has been reached, a few minor iterations may be needed for the corresponding subproblem to be recognized as optimal.)

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

`Penalty parameter` r Default = 1.0

This specifies that the initial value of ρ_k in the augmented Lagrangian (5) should be r times a certain default value $100/m_1$, where m_1 is the number of nonlinear constraints. It is used when `Lagrangian = Yes` (the default setting).

For early runs on a problem with unknown characteristics, the default value should be acceptable. If the problem is highly nonlinear and the major iterations do not converge, a larger value such as 2 or 5 may help. In general, a positive r may be necessary to ensure convergence, *even for convex programs.*

On the other hand, if r is too large, the rate of convergence may be unnecessarily slow. If the functions are not highly nonlinear or a good starting point is known, it is often safe to specify `Penalty parameter 0.0`.

If several related problems are to be solved, the following strategy for setting the `Penalty parameter` may be useful:

1. Initially, use a moderate value for r (such as the default) and a reasonably low `Iterations` and/or `Major iterations` limit.
2. If successive major iterations appear to be terminate with radically different solutions, try increasing the penalty parameter. (See also the `Major damping parameter`.)
3. If there appears to be little progress between major iterations, it may help to reduce the penalty parameter.

`Radius of convergence` r Default = 0.01

This determines when the penalty parameter ρ_k is reduced (if initialized to a positive value). Both the nonlinear constraint violation (see *rowerr* below) and the relative change in consecutive Lagrange multiplier estimates must be less than r at the start of a major iteration before ρ_k is reduced or set to zero.

A few major iterations later, full completion is requested if not already set, and the remaining sequence of major iterations should converge quadratically to an optimum.

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

This specifies how accurately the nonlinear constraints should be satisfied at a solution. The default value is usually small enough, since model data is often specified to about that accuracy.

Let *viol* be the maximum violation of the nonlinear constraints (2), and let $rowerr = viol/(1 + xnorm)$, where $xnorm$ is a measure of the size of the current solution (x, y). The solution is regarded as acceptably feasible if $rowerr \leq t$.

If the problem functions involve data that is known to be of low accuracy, a larger `Row tolerance` may be appropriate. On the other hand, nonlinear constraints are often satisfied with rapidly increasing accuracy during the last few major iterations. It is common for the final solution to satisfy $rowerr = O(\epsilon)$.

`Scale option`	l	Default = 2 (LP) or 1 (NLP)
`Scale`	`Yes`	
`Scale`	`No`	
`Scale linear variables`		Same as `Scale option 1`
`Scale nonlinear variables`		Same as `Scale option 2`
`Scale all variables`		Same as `Scale option 2`
`Scale tolerance`	r	Default = 0.9
`Scale, Print`		
`Scale, Print, Tolerance`	r	

Three scale options are available as follows:

- $l = 0$ No scaling. If storage is at a premium, this option saves $m + n$ words of workspace.
- $l = 1$ Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0.
- $l = 2$ All 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 $(A \;\; I)$ that are fixed or have positive lower bounds or negative upper bounds.

`Scale option 1` is the default for nonlinear problems. (Only linear variables are scaled.)

`Scale Yes` sets the default scaling. *Caution*: If all variables are nonlinear, `Scale Yes` unexpectedly does nothing, because there are no linear variables to scale.) `Scale No` suppresses scaling (equivalent to `Scale option 0`).

With nonlinear constraints, `Scale option 1` or `0` should generally be tried first. `Scale option 2` gives scales that depend on the initial Jacobian, and should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

`Verify level`	l	Default = 0
`Verify objective gradients`		Same as `Verify level 1`
`Verify constraint gradients`		Same as `Verify level 2`
`Verify`		Same as `Verify level 3`
`Verify gradients`		Same as `Verify level 3`
`Verify`	`Yes`	Same as `Verify level 3`
`Verify`	`No`	Same as `Verify level 0`

This option refers to a finite-difference check on the gradients (first derivatives) of each nonlinear function. It occurs before a problem is scaled, and before the first basis is factorized. (Hence, the variables may not yet satisfy the general linear constraints.)

$l = 0$	Only a "cheap" test is performed, requiring three evaluations of the nonlinear objective (if any) and two evaluations of the nonlinear constraints.
$l = 1$	A more reliable check is made on each component of the objective gradient.
$l = 2$	A check is made on each column of the Jacobian matrix associated with the nonlinear constraints.
$l = 3$	A detailed check is made on both the objective and the Jacobian.
$l = -1$	No checking is performed. This may be necessary if the functions are undefined at the starting point.

6.2.6 Options for Input and Output

The following options specify various files to be used, and the amount of printed output required.

`Print level`	l	Default = 0
`Print frequency`	k	Default = 100

The PRINT file provides more complete information than the SUMMARY file. It includes a listing of the main options, statistics about the problem, scaling information, the iteration log, the exit condition, and (optionally) the final solution. It also includes error messages.

The files are specified in *Prob.SOL.PrintFile* and *Prob.SOL.SummFile*.

For problems with linear constraints, `Print level 0` gives most of the normal output. `Print level 1` produces statistics for the basis matrix B and its LU factors each time the basis is factorized. `Print frequency` k produces one line of the iteration log every k minor iterations. `Print frequency 1` causes every minor iteration to be logged. `Print frequency 0` is shorthand for $k = 99999$.

For problems with nonlinear constraints, `Print level 0` produces just one line of output per major iteration. This provides a short summary of the progress of the optimization. The `Print frequency` is ignored. If `Print level` > 0, certain quantities are printed at the start of each major iteration, and minor iterations are logged according to the `Print frequency`.

In the latter case, the value of l is best thought of as a binary number of the form

```
Print level   JFLXB
```

where each letter stands for a digit that is either 0 or 1. The quantities referred to are:

- `B` Basis statistics, as mentioned above.
- `X` x_k, the nonlinear variables involved in the objective function or the constraints.
- `L` λ_k, the Lagrange-multiplier estimates for the nonlinear constraints. (Suppressed if `Lagrangian = No`, since then $\lambda_k = 0$.)
- `F` $f(x_k)$, the values of the nonlinear constraint functions.
- `J` $J(x_k)$, the Jacobian matrix.

To obtain output of any item, set the corresponding digit to 1, otherwise to 0. For example, `Print level 10` sets `X` = 1 and the other digits equal to zero. The nonlinear variables will be printed each major iteration.

If `J` = 1, the Jacobian is output column-wise. Column j is preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if `J` = 1, there is no reason to specify `X` = 1 unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
   3  1.250000D+01 BS      1  1.00000D+00      4  2.00000D+00
```

which means that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.

`Solution`	`Yes`	Default
`Solution`	`No`	

The `Yes` and `No` options control whether the final solution is output to the PRINT file. Numerical values are printed in `f16.5` format where possible.

The special values 0, 1 and -1 are printed as `.`, `1.0` and `-1.0`. Bounds outside the range $(-10^{20}, 10^{20})$ appear as the word `None`.

`Summary file`	f	Default = 6 (typically)
`Summary level`	l	Default = 0
`Summary frequency`	k	Default = 100

The SUMMARY file provides a brief form of the iteration log and the exit condition. It also includes error messages. In an interactive environment, the output normally appears at the screen and allows a run to be monitored.

For problems with linear constraints, `Summary level 0` produces brief output. `Summary level 1` gives a few additional messages. `Summary frequency` k produces one line of the iteration log every k minor iterations. `Summary frequency 1` causes every minor iteration to be logged. `Summary frequency 0` is shorthand for $k = 99999$.

For problems with nonlinear constraints, `Summary level 0` produces one line of output per major iteration. This provides a short summary of the progress of the optimization. The `Summary frequency` is ignored. If `Summary level` > 0, certain quantities are printed at the start of each major iteration, and minor iterations are logged according to the `Summary frequency`.

```
Major minor   total ninf step     objective     Feasible Optimal  nsb   ncon     LU penalty BSwap
    1     1T      1    0 0.0E+00  0.00000000E+00 0.0E+00 1.2E+01    8      4     31 1.0E-01     0
    2    13      14    0 1.0E+00  2.67011596E+00 4.4E-06 2.8E-03    7     23     56 1.0E-01     8
Completion Full    now requested
    3     3      17    0 1.0E+00  2.67009870E+00 3.1E-08 1.4E-06    7     29     41 1.0E-01     0
    4     0      17    0 1.0E+00  2.67009863E+00 5.6E-17 1.4E-06    7     30     41 1.0E-02     0
```

Figure 1: The Major Iteration log

6.3 File Output

6.3.1 The PRINT file

The following information is output to the PRINT file during the solution process. The longest line of output is 124 characters.

- A listing of the SPECS file, if any.
- The selected options.
- An estimate of the storage needed and the amount available.
- Some statistics about the problem data.
- The storage available for the LU factors of the basis matrix.
- A log from the scaling procedure, if `Scaleoption` > 0.
- Notes about the initial basis obtained from CRASH or a BASIS file.
- The major iteration log.
- The minor iteration log.
- Basis factorization statistics.
- The EXIT condition and some statistics about the solution obtained.
- The printed solution, if requested.

The last five items are described in the following sections.

6.3.2 The major iteration log

Problems with nonlinear constraints require several *major iterations* to reach a solution, each involving the solution of an *LC subproblem* (a linearly constrained subproblem that generates search directions for x and λ). If `Printlevel` $= 0$, one line of information is output to the PRINT file each major iteration. An example log is shown in Figure 1.

Label	*Description*
`Major`	The current major iteration number.
`minor`	is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, `Mnr` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution.
`total`	The total number of minor iterations.

`ninf` The number of infeasibilities in the LC subproblem. Normally 0, because the bounds on the linearized constraints are relaxed in several stages until the constraints are "feasible".

`step` The step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. On reasonably well-behaved problems, `step` = 1.0 as the solution is approached, meaning the new estimate of (x, λ) is the solution of the LC subproblem.

`objective` The value of true objective function.

`Feasible` The value of `rowerr`, the maximum component of the scaled nonlinear constraint residual. The solution is regarded as acceptably feasible if `Feasbl` is less than the `Row tolerance`.

`Optimal` The value of `maxgap`, the maximum complementarity gap. It is an estimate of the degree of nonoptimality of the reduced costs. Both `Feasible` and `Optimal` are small in the neighborhood of a solution.

`nsb` The current number of superbasic variables.

`ncon` The number of times subroutine `funcon` has been called to evaluate the nonlinear constraint functions. The Jacobian has been evaluated or approximated essentially the same number of times. (Function evaluations needed to estimate the Jacobian by finite differences are not included.)

`LU` The number of nonzeros in the sparse LU factors of the basis matrix on completion of the LC subproblem. (The factors are computed at the start of each major iteration, and updated during minor iterations whenever a basis change occurs.)

As the solution is approached and the minor iterations decrease towards zero, `LU` reflects the number of nonzeros in the LU factors at the start of the LC subproblem.

`penalty` The penalty parameter ρ_k used in the modified augmented Lagrangian that defines the objective function for the LC subproblem.

`BSwap` The number of columns of the basis matrix B that were swapped with columns of S to improve the condition of B. The swaps are determined by an LU factorization of the rectangular matrix $B_S = (B \;\; S)^T$ with stability being favored more than sparsity.

```
   Itn ph pp     rg    +sbs  -sbs   -bs  step    pivot   ninf  sinf,objective     L     U ncp  nobj  ncon  nsb Hmod cond(H) conv
     1  1  1 -1.0E+00     2     2     1 3.0E+01  1.0E+00     1  1.35000000E+02     0    19   0
     2  1  1 -1.0E+00    27    27   102 7.0E+01  1.0E+00     1  1.05000000E+02     0    19   0
     3  1  1 -1.0E+00     3     3    27 3.0E+01 -1.0E+00     1  3.50000000E+01     1    19   0
     4  1  1 -1.0E+00    28    28    26 4.9E-11  1.0E+00     1  5.00000000E+00     1    20   0
     5  1  1 -1.0E+00    47    47     2 4.9E-11  1.0E+00     1  5.00000000E+00     1    20   0
     6  1  1  1.0E+00    27    27   101 5.0E+00 -1.0E+00     1  5.00000000E+00     2    20   0

 Itn      6 -- feasible solution.  Objective =  -1.818044887E+02

     7  3  1 -1.7E+01    87     0     0 1.0E+00  0.0E+00     0 -2.77020571E+02     4    21   0     6     0    1  1 0 1.0E+00 FFTT
     8  3  1 -1.7E+01    72     0     0 1.9E-01  0.0E+00     0 -3.05336895E+02     4    21   0     8     0    2  1 0 5.5E+00 FFTT
     9  3  1 -2.3E+01    41     0     0 1.0E+00  0.0E+00     0 -4.43743832E+02     4    21   0     9     0    3  1 0 6.5E+00 FFFF
    10  4  1  6.6E-01     0     0     0 6.0E+00  0.0E+00     0 -5.64075338E+02     4    21   0    11     0    3  1 0 3.5E+00 FFTT
...

   Itn ph pp     rg    +sbs  -sbs   -bs  step    pivot   ninf  sinf,objective     L     U ncp  nobj  ncon  nsb Hmod cond(H) conv
   161  4  1  8.8E-03     0    73    71 4.2E+00  1.0E+00     0 -1.73532497E+03     4    20   0   340     0   17  1 1 9.6E+00 TTTF
   162  3  1 -3.5E-02     6     0     0 1.5E+00  0.0E+00     0 -1.73533264E+03     4    20   0   342     0   18  1 0 1.3E+02 TTFF
   163  4  1  2.9E-02     0     0     0 4.5E+00  0.0E+00     0 -1.73533617E+03     4    20   0   344     0   18  1 0 2.0E+01 TTFF
   164  4  1  2.1E-02     0     0     0 2.3E+01  0.0E+00     0 -1.73538331E+03     4    20   0   347     0   18  1 0 9.8E+00 TTFF
   165  4  1  3.0E-02     0     0     0 5.0E+00  0.0E+00     0 -1.73552261E+03     4    20   0   349     0   18  1 0 2.1E+01 TTFF
   166  4  1  1.2E-02     0     0     0 1.0E+00  0.0E+00     0 -1.73556089E+03     4    20   0   350     0   18  1 0 2.2E+01 TTTF
 tolrg  reduced to  1.162E-03      lvltol = 1
   167  4  1  2.3E-03     0     0     0 1.0E+00  0.0E+00     0 -1.73556922E+03     4    20   0   351     0   18  1 0 2.2E+01 TTFF
   168  4  1  1.2E-03     0     0     0 7.9E-01  0.0E+00     0 -1.73556953E+03     4    20   0   353     0   18  1 0 2.1E+01 TTFF
   169  4  1  1.0E-04     0     0     0 1.0E+00  0.0E+00     0 -1.73556958E+03     4    20   0   354     0   18  1 0 2.0E+01 TTTT
 tolrg  reduced to  1.013E-05      lvltol = 1
   170  4  1  2.9E-05     0     0     0 1.1E+00  0.0E+00     0 -1.73556958E+03     4    20   0   356     0   18  1 0 1.7E+01 TTFF
   171  4  1  1.0E-05     0     0     0 1.0E+00  0.0E+00     0 -1.73556958E+03     4    20   0   357     0   18  1 0 1.7E+01 TTFF
   172  4  1  1.5E-06     0     0     0 1.2E+00  0.0E+00     0 -1.73556958E+03     4    20   0   359     0   18  1 0 1.7E+01 TTTF
 tolrg  reduced to  1.000E-06      lvltol = 2
   173  4  1  2.4E-07     0     0     0 1.0E+00  0.0E+00     0 -1.73556958E+03     4    20   0   360     0   18  1 0 1.7E+01 TTTF

 Biggest dj =  3.583E-03 (variable     25)   norm rg =  2.402E-07   norm pi =  1.000E+00
```

Figure 2: The Minor Iteration log

6.3.3 The minor iteration log

If `Printlevel` ≥ 1, one line of information is output to the PRINT file every kth minor iteration, where k is the specified `Print frequency` (default $k =$ `100`). A heading is printed periodically. Problem `t5weapon` gives the log shown in Figure 2.

Label — *Description*

`Itn` The current minor iteration number.

`ph` The current phase of the solution procedure:

1. Phase 1 simplex method, trying to satisfy the linear constraints. The current solution is an infeasible vertex.
2. Phase 2 simplex method, solving a linear program.
3. Reduced-gradient method. A nonbasic variable has just become superbasic.
4. Reduced-gradient method, optimizing the current set of superbasic variables.

`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.

A PRICE operation is defined to be the process by which a nonbasic variable is selected to become a new superbasic. The selected variable is denoted by `jq`. Variable `jq` often becomes basic immediately.

Otherwise it remains superbasic, unless it reaches its opposite bound and becomes nonbasic again. If `Partial price` is in effect, variable `jq` is selected from $A_{\mathtt{pp}}$ or $I_{\mathtt{pp}}$, the `pp`th segments of the constraint matrix (A I).

`rg` In Phase 1, 2 or 3, this is d_j, the reduced cost (reduced gradient) of the variable `jq` selected by PRICE at the start of the present iteration. Algebraically, $d_j = g_j - \pi^T a_j$ for $j =$ `jq`, where g_j is the gradient of the current objective function, π is the vector of dual variables for the problem (or LC subproblem), and a_j is the jth column of the current (A I).

In Phase 4, `rg` is the largest reduced gradient among the superbasic variables.

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

`-sbs` The variable chosen to leave the set of superbasics. It has become basic if the entry under `-bs` is nonzero; otherwise it has 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 $x + \alpha p$.

`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` (because 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`.

`ninf` The number of infeasibilities *before* the present iteration. This number decreases monotonically.

`sinf,objective` If `ninf` > 0, this is `sinf`, the sum of infeasibilities before the present iteration. It usually decreases at each nonzero `step`, but if `ninf` decreases by 2 or more, `sinf` may occasionally increase.

Otherwise it is the value of the current objective function *after* the present iteration. For linear programs, it means the true linear objective function. For problems with linear constraints, it means the sum of the linear objective and the value returned by subroutine `funobj`. For problems with nonlinear constraints, it is the quantity just described if `Lagrangian = No`; otherwise it is the value of the augmented Lagrangian for the current major iterations (which tends to the true objective as convergence is approached).

`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 workspace available to MINOS should be increased by a significant amount. As a suggestion, the work array `z(*)` should be extended by $2(\mathtt{L} + \mathtt{U})$ elements.

The following items are printed if the problem is nonlinear or if the superbasic set is non-empty (i.e., if the current solution is not a vertex).

Label	*Description*
`nobj`	The number of times subroutine `funobj` has been called.
`ncon`	The number of times subroutine `funcon` has been called.
`nsb`	The current number of superbasic variables.
`Hmod`	An indication of the type of modifications made to the triangular matrix R that is used to approximate the reduced Hessian matrix. Two integers i_1 and i_2 are shown. They will remain zero for linear problems. If $i_1 = 1$, a BFGS quasi-Newton update has been made to R, to account for a move within the current subspace. (This will not occur if the solution is infeasible.) If $i_2 = 1$, R has been modified to account for a change in basis. This will sometimes occur even if the solution is infeasible (if a feasible point was obtained at some earlier stage). Both updates are implemented by triangularizing the matrix $R + vw^T$ for some vectors v and w. If an update fails for numerical reasons, i_1 or i_2 will be set to 2, and the resulting R will be nearly singular. (However, this is highly unlikely.)
`cond(H)`	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—a lower bound on the condition number of the matrix $R^T R$ that approximates the reduced Hessian. `cond(H)` gives a rough indication of whether or not the optimization procedure is having difficulty. The reduced-gradient algorithm will make slow progress if `cond(H)` becomes as large as 10^8, and will probably fail to find a better solution if `cond(H)` reaches 10^{12} or more. To guard against high values of `cond(H)`, attention should be given to the scaling of the variables and the constraints. In some cases it may be necessary to add upper or lower bounds to certain variables to keep them a reasonable distance from singularities in the nonlinear functions or their derivatives.
`conv`	A set of four logical variables C_1, C_2, C_3, C_4 that are used to determine when to discontinue optimization in the current subspace (Phase 4) and consider releasing a nonbasic variable from its bound (the PRICE operation of Phase 3). Let `rg` be the norm of the reduced gradient, as described above. The meaning of the variables C_j is as follows: C_1 is `true` if the change in x was sufficiently small; C_2 is `true` if the change in the objective was sufficiently small; C_3 is `true` if `rg` is smaller than some loose tolerance `TOLRG`; C_4 is `true` if `rg` is smaller than some tighter tolerance. The test used is of the form *if* (C_1 *and* C_2 *and* C_3) *or* C_4 *then go to Phase 3.* At present, `tolrg` $= t|$`dj`$|$, where t is the `Subspace tolerance` (nominally 0.5) and `dj` is the reduced-gradient norm at the most recent Phase 3 iteration. The "tighter tolerance" is the maximum of 0.1 `tolrg` and $10^{-7}\|\pi\|$. Only the tolerance t can be altered at run-time.

6.3.4 Crash statistics

The following items are output to the PRINT file when no warm start is used. They refer to the number of columns that the CRASH procedure selects during several passes through A while 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, including those whose bounds are rather far apart.
`Preferred`	is the number of "preferred" columns in the basis (i.e., $\texttt{hs}(j) = 3$ for some $j \leq n$). It will be a subset of the columns for which $\texttt{hs}(j) = 3$ was specified.
`Unit`	is the number of unit columns in the basis.
`Double`	is the number of columns in the basis containing 2 nonzeros.
`Triangle`	is the number of triangular columns in the basis with 3 or more nonzeros.
`Pad`	is the number of slacks used to pad the basis (to make it a nonsingular triangle).

6.3.5 Basis factorization statistics

If `Printlevel` ≥ 1, the following items are output to the PRINT file whenever the basis B or the rectangular matrix $B_S = (\,B \quad S\,)^T$ is factorized. Note that B_S may be factorized at the start of just some of the major 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.

Label	*Description*
`Factorize`	The number of factorizations since the start of the run.
`Demand`	A code giving the reason for the present factorization.
`Itn`	The current 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.
`m`	The number of rows in the matrix being factorized (B or B_S).
`n`	The number of columns in the matrix being factorized. Preceded by "=" if the matrix is B; by ">" if it is B_S.
`Elems`	The number of nonzero elements in B or B_S.
`Amax`	The largest nonzero in B or B_S.
`Density`	The density of the matrix (percentage of nonzeros).

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

`lenL` The number of nonzeros in the factor L.

`L+U` The number of nonzeros in both L and U.

`Cmprssns` 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 MINOS 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` The size of the "backward triangle" in B or B_S. These top rows of U come directly from the matrix.

`lenU` The number of nonzeros in the factor U.

`Ltol` The maximum allowed size of nonzeros in L. Usually equal to the `LU factor tolerance`.

`Umax` The maximum nonzero in U.

`Ugrwth` The ratio Umax / Amax.

`Ltri` The size of the "forward triangle" in B or B_S. These initial columns of L come directly from the matrix.

`dense1` is the number of columns remaining when the density of the basis matrix being factorized reached 0.3.

`Lmax` The maximum nonzero in L (no larger than `Ltol`).

`Akmax` The maximum nonzero arising during the factorization. (Printed only if Threshold Complete Pivoting is in effect.)

`Agrwth` The ratio Akmax / Amax. (Printed only if Threshold Complete Pivoting is in effect.)

`bump` The number of columns of B or B_S excluding `Utri` and `Ltri`.

`dense2` The number of columns remaining when the density of the basis matrix being factorized reached 0.6.

`DUmax` The largest diagonal of U (really PUQ).

`DUmin` The smallest diagonal of U.

`condU` The ratio `DUmax/DUmin`. As long as `Ltol` is not large (say 10.0 or less), `condU` is an estimate of the condition number of B. If this number is extremely large, the basis is nearly singular and some numerical difficulties might occur. (However, an effort is made to avoid near-singularity by using slacks to replace columns of B that would have made `Umin` extremely small. Messages are issued to this effect, and the modified basis is refactored.)

6.3.6 EXIT conditions

When the solution procedure terminates, an `EXIT --` message is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

The number associated with each EXIT is the output value of the integer variable `inform`.

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

```
-2  EXIT -- input error.   MINOS encountered end-of-file or an
    endrun card before finding a SPECS file.
```

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

```
-1  ENDRUN
```

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

The following messages arise when a solution exists (though it may not be optimal).

```
 0  EXIT -- optimal solution found
```

This is the message we all hope to see! It is certainly preferable to every other message, and we naturally want to believe what it says, because this is surely one situation where *the computer knows best*. There may be cause for celebration if the objective function has reached an astonishing new high (or low).

In all cases, a distinct level of caution is in order, even if it can wait until next morning. For example, if the objective value *is* much better than expected, we may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder. It is good practice in the function subroutines to print any data that is input during the first entry.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are *minimizing* the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the *same* objective value, particularly if the objective is largely linear.) Conversely, if we are *maximizing* a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Always specify a good starting point if possible, especially for nonlinear variables, and include reasonable upper and lower bounds on the variables to confine the solution to the specific region of interest. We expect modelers to *know something about their problem*, and to make use of that knowledge as well as they can.

One other caution about "`Optimal solution`"s. Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. `Max Primal infeas` refers to the largest bound infeasibility and

which variable (or slack) 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

$$\texttt{MaxDualinfeas/Normofpi} = 10^{-d},$$

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 smaller `Optimality tolerance`s.

Finally, `Nonlinear constraint violn` shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller `Row tolerance`.

```
1  EXIT -- the problem is infeasible
```

When the constraints are linear, this message can probably be trusted. 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.

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each linearly constrained (LC) subproblem, MINOS is prepared to relax the bounds on the slacks associated with nonlinear rows.

If an LC subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), MINOS enters so-called "nonlinear elastic" mode. The subproblem includes 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 nonlinear rows "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 original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, MINOS will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, MINOS would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though MINOS locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

```
2  EXIT -- the problem is unbounded  (or badly scaled)

   EXIT -- violation limit exceeded -- the problem may be unbounded
```

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

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

For nonlinear problems, MINOS monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the `Unbounded` parameters, the problem

is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the `Violation limit`.

`3 EXIT -- major iteration limit exceeded`

`EXIT -- minor iteration limit exceeded`

`EXIT -- too many iterations`

Either the `Iterations limit` or the `Major 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 *WarmDefSOL* at the end of the run.

`4 EXIT -- requested accuracy could not be achieved`

A feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but MINOS is within 10^{-2} of satisfying the `Major optimality tolerance`. Check that the `Major optimality tolerance` is not too small.

`5 EXIT -- the superbasics limit is too small: nnn`

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

In general, raise the `Superbasics limit` s by a reasonable amount, bearing in mind the storage needed for the reduced Hessian (about $\frac{1}{2}s^2$ double words).

`6 EXIT -- constraint and objective values could not be calculated`

This exit occurs if a value `mode` ≤ -1 is set during some call to `funobj` or `funcon`. MINOS assumes that you want the problem to be abandoned forthwith.

In some environments, this exit means that your subroutines were not successfully linked to MINOS. If the default versions of `funobj` and `funcon` are ever called, they issue a warning message and then set `mode` to terminate the run.

`7 EXIT -- subroutine funobj seems to be giving incorrect gradients`

A check has been made on some individual elements of the objective gradient array at the first point that satisfies the linear constraints. At least one component `gObj(`j`)` is being set to a value that disagrees markedly with a forward-difference estimate of $\partial f/\partial x_j$. (The relative difference between the computed and estimated values is 1.0 or more.) This exit is a safeguard, since MINOS will usually fail to make progress when the computed gradients are seriously inaccurate. In the process it may expend considerable effort before terminating with EXIT 9 below.

Check the function and gradient computation *very carefully* in `funobj`. A simple omission (such as forgetting to divide `fObj` by 2) could explain everything. If `fObj` or `gObj(`j`)` is very large, then give serious thought to scaling the function or the nonlinear variables.

If you feel *certain* that the computed `gObj(`j`)` is correct (and that the forward-difference estimate is therefore wrong), you can specify `Verify level 0` to prevent individual elements from being checked. However, the optimization procedure may have difficulty.

8 EXIT -- subroutine funcon seems to be giving incorrect gradients

This is analogous to the preceding exit. At least one of the computed Jacobian elements is significantly different from an estimate obtained by forward-differencing the constraint vector $F(x)$. Follow the advice given above, trying to ensure that the arrays `fCon` and `gCon` are being set correctly in `funcon`.

9 EXIT -- the current point cannot be improved upon

Several circumstances could lead to this exit.

1. Subroutines `funobj` or `funcon` could be returning accurate function values but inaccurate gradients (or vice versa). This is the most likely cause. Study the comments given for EXIT 7 and 8, and do your best to ensure that the coding is correct.

2. The function and gradient values could be consistent, but their precision could be too low. For example, accidental use of a `real` data type when `double precision` was intended would lead to a relative function precision of about 10^{-6} instead of something like 10^{-15}. The default `Optimality tolerance` of 10^{-6} would need to be raised to about 10^{-3} for optimality to be declared (at a rather suboptimal point). Of course, it is better to revise the function coding to obtain as much precision as economically possible.

3. If function values are obtained from an expensive iterative process, they may be accurate to rather few significant figures, and gradients will probably not be available. One should specify

`Function precision`	t
`Major optimality tolerance`	$\sqrt{t}$

   but even then, if t is as large as 10^{-5} or 10^{-6} (only 5 or 6 significant figures), the same exit condition may occur. At present the only remedy is to increase the accuracy of the function calculation.

10 EXIT -- cannot satisfy the general constraints

An LU factorization of the basis has just been obtained and used to recompute the basic variables x_B, given the present values of the superbasic and nonbasic variables. A 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. Try `Scale option 1` if scaling has not yet been used and there are some linear constraints and variables.

For certain highly structured basis matrices (notably those with band structure), a systematic growth may occur in the factor U. Consult the description of `Umax`, `Umin` and `Growth` in §6.3.5, and set the `LU factor tolerance` to 2.0 (or possibly even smaller, but not less than 1.0).

12 EXIT -- terminated from subroutine s1User

The user has set the value `iAbort` = 1 in subroutine `s1User`. MINOS assumes that you want the problem to be abandoned forthwith.

> If the following exits 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.

`20 EXIT -- not enough integer/real storage for the basis factors`

The main integer or real storage array `iw(*)` or `rw(*)` is apparently not large enough for this problem. The routine declaring `iw` and `rw` should be recompiled with a larger dimensions for those arrays. The new values should also be assigned to `leniw` and `lenrw`.

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

`21 EXIT -- error in basis package`

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

`22 EXIT -- singular basis after nnn factorization attempts`

This exit is highly unlikely to occur. The first factorization attempt will have found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix 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 following messages arise, either an OLD BASIS file could not be loaded properly, or some fatal system error has occurred. New BASIS files cannot be saved, and there is no solution to print. The problem is abandoned.

`30 EXIT -- the basis file dimensions do not match this problem`

On the first line of the OLD BASIS file, the dimensions 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.

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

`31 EXIT -- the basis file state vector does not match this problem`

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

`32 EXIT -- system error. Wrong no. of basic variables: nnn`

This exit should never happen. It may indicate that the wrong MINOS source files have been compiled, or incorrect parameters have been used in the call to subroutine `minoss`.

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

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

`42 EXIT -- not enough 8-character storage to start solving the problem`

The main character storage array `cw(*)` is not large enough.

`43 EXIT -- not enough integer storage to start solving the problem`

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

`44 EXIT -- not enough real storage to start solving the problem`

The main storage array `rw(*)` is not large enough to provide workspace for the optimization procedure. Be sure that the `Superbasics limit` is not unreasonably large. Otherwise, see the advice for EXIT 20.

6.3.7 Solution output

At the end of a run, the final solution is 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 certain commercial systems, though there is no industry standard.

An example of the printed solution is given in §6.3. In general, numerical values are output with format `f16.5`. The maximum record length is 111 characters, including the first (carriage-control) character.

To reduce clutter, a dot "`.`" is printed for any numerical value that is exactly zero. The values ± 1 are also printed specially as `1.0` and `-1.0`. Infinite bounds ($\pm 10^{20}$ or larger) are printed as `None`.

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

The ROWS section

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

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

and the value of $a^T x$ is called the *row activity*. Internally, the linear constraints take the form $Ax - s = 0$, where the slack variables s should satisfy the bounds $l \leq s \leq u$. For the ith "row", it is the slack variable s_i that is directly available, and it is sometimes convenient to refer to its state. Slacks may be basic or nonbasic (but not superbasic).

Nonlinear constraints $\alpha \leq F_i(x) + a^T x \leq \beta$ are treated similarly, except that the row activity and degree of infeasibility are computed directly from $F_i(x) + a^T x$ rather than from s_i.

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 limits are the same ($\alpha = \beta$).

`BS` The constraint is not binding. s_i is basic.

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 from its current value, there would be no change in the objective function. The values of the basic and superbasic variables *might* change, giving a genuine alternative solution. The values of the dual variables *might* also change.

`D` *Degenerate.* The slack is basic, but it is equal to (or very close to) one of its bounds.

`I` *Infeasible.* The slack is basic and is currently violating one of its bounds by more than the `Feasibility tolerance`.

`N` *Not precisely optimal.* The slack is nonbasic. Its reduced gradient is larger than the `Optimality tolerance` .

Note: If `Scaleoption` $>$ `0`, the tests for assigning `A`, `D`, `I`, `N` are made on the scaled problem, since the keys are then more likely to be meaningful.

`Activity` The row value a^Tx (or $F_i(x) + a^Tx$ for nonlinear rows).

`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_j, $j = 1 : n$. We assume that a typical variable has bounds $\alpha \leq x_j \leq \beta$.

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 at some value strictly between its bounds: $\alpha < x_j < \beta$.

`BS` x_j is basic. Usually $\alpha < x_j < \beta$.

`SBS` x_j is superbasic. Usually $\alpha < x_j < \beta$.

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

`A` *Alternative optimum possible.* The variable is nonbasic, but its reduced gradient is essentially zero. This means that if x_j were allowed to start moving from its current value, there would be no change in the objective function. The values of the basic and superbasic variables *might* change, giving a genuine alternative solution. The values of the dual variables *might* also change.

`D` *Degenerate.* x_j is basic, but it is equal to (or very close to) one of its bounds.

`I` *Infeasible.* x_j is basic and is currently violating one of its bounds by more than the `Feasibility tolerance`.

`N` *Not precisely optimal.* x_j is nonbasic. Its reduced gradient is larger than the `Optimality tolerance` .

Note: If `Scaleoption` $>$ `0`, the tests for assigning `A`, `D`, `I`, `N` are made on the scaled problem, since the keys are then more likely to be meaningful.

`Activity` The value of the variable x_j.

`Obj Gradient` g_j, the jth component of the gradient of the (linear or nonlinear) objective function. (If any x_j is infeasible, g_j is the gradient of the sum of infeasibilities.)

`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 $m + j$.

6.3.8 The SUMMARY file

If `Summaryfile` > 0, the following information is output to the SUMMARY file. (It is a brief form of the PRINT file.) All output lines are less than 72 characters.

- The `Begin` line from the SPECS file, if any.
- The basis file loaded, if any.
- A brief Major iteration log.
- A brief Minor iteration log.
- The EXIT condition and a summary of the final solution.

The following SUMMARY file is from example problem `t6wood` using `Print level 0` and `Major damping parameter 0.5`.

```
     ==============================
     M I N O S  5.51     (Nov 2002)
     ==============================

Begin t6wood   (WOPLANT test problem; optimal obj = -15.55716)

Name   WOPLANT
===>  Note:  row  OBJ       selected as linear part of objective.
Rows          9
Columns      12
Elements     73

Scale option  2,    Partial price   1

Itn      0 -- linear constraints satisfied.

This is problem t6wood.  Derivative level =  3

funcon  sets      36   out of      50   constraint gradients.

Major minor  step     objective  Feasible Optimal  nsb   ncon penalty BSwap
    1     0T 0.0E+00  0.00000E+00 5.9E-01 1.1E+01    0      4 1.0E+00     0
    2    22  5.0E-01 -1.56839E+01 2.7E-01 1.6E+01    3     47 1.0E+00     0
    3    10  6.0E-01 -1.51527E+01 1.5E-01 9.9E+00    2     68 1.0E+00     2
    4    21  5.7E-01 -1.53638E+01 6.4E-02 3.6E+00    3    113 1.0E+00     1
    5    15  1.0E+00 -1.55604E+01 2.7E-02 1.4E-01    3    144 1.0E+00     0
    6     5  1.0E+00 -1.55531E+01 6.4E-03 2.2E-01    3    154 1.0E+00     0
    7     4  1.0E+00 -1.55569E+01 3.1E-04 7.0E-04    3    160 1.0E-01     0
    8     2  1.0E+00 -1.55572E+01 1.6E-08 1.1E-04    3    163 1.0E-02     0
    9     1  1.0E+00 -1.55572E+01 5.1E-14 2.2E-06    3    165 1.0E-03     0

EXIT -- optimal solution found

Problem name                WOPLANT
No. of iterations                80   Objective value     -1.5557160112E+01
No. of major iterations           9   Linear objective    -1.5557160112E+01
Penalty parameter            0.000100   Nonlinear objective  0.0000000000E+00
No. of calls to funobj            0   No. of calls to funcon            165
No. of superbasics                3   No. of basic nonlinears             6
No. of degenerate steps           0   Percentage                       0.00
Norm of x   (scaled)          9.8E-01   Norm of pi  (scaled)          1.8E+02
Norm of x                     3.2E+01   Norm of pi                    1.6E+01
Max Prim inf(scaled)        0 0.0E+00   Max Dual inf(scaled)        1 2.2E-06
Max Primal infeas           0 0.0E+00   Max Dual infeas             1 5.8E-08
Nonlinear constraint violn    5.1E-14



Solution printed on file   9

funcon called with nstate =   2

Time for MPS input                          0.00 seconds
Time for solving problem                    0.04 seconds
Time for solution output                    0.00 seconds
Time for constraint functions               0.00 seconds
Time for objective function                 0.00 seconds
Endrun
```

7 SQOPT details

7.1 Introduction

TOMVIEW /SQOPT (hereafter referred to as SQOPT) is a program for solving the large-scale *linear or quadratic programming problem*, which is assumed to be stated in following form:

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

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

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

The possible forms for the function $q(x)$ are summarized in Table 35. The most general form for $q(x)$ is

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

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

Table 35: Choices for the objective function $q(x)$.

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

If $H = 0$, then $q(x) = f + c^T x$ and the problem is known as a *linear program* (LP).

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.

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

7.2 Brief description of the method

Here we briefly describe some features of the algorithm used in SQOPT and introduce some terminology used in the description of the subroutine and its arguments. For further details, see §7.6.

7.2.1 Formulation of the problem

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

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

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

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

Phase 1 involves solving a linear program of the form

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

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

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

To make the relaxed problem meaningful, SQOPT minimizes q while (in some sense) finding the "smallest" violation

of the elastic variables. In the situation where all the variables are elastic, the relaxed problem has the form

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

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

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

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

7.2.2 The main iteration

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

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

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

Associated with each of the m equality constraints $Ax - s = 0$ is a *dual variable* π_i. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j (also known as a *reduced cost*). The reduced gradients for the variables x are the quantities $g - A^T\pi$, where g is the gradient of the QP objective; and the reduced gradients for the slacks s are the dual variables π. The QP subproblem is optimal if $d_j \ge 0$ 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 §7.3.2).

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

7.3 The SPECS file

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

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

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

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

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

It may be useful to include a comment on the first (`Begin`) line of the file. This line is echoed to the SUMMARY file.

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.

7.3.1 SPECS File Checklist and Defaults

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

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

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

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

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

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

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

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

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

7.3.2 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.

`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 $r > 1/4$.

`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 minimizes 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 `Elasticmode` = 2.
2	Include only the elastic variables in the composite objective. The elastics are weighted by $\gamma = 1$. This choice minimizes the violations of the elastic variable at the expense of possibly increasing the true objective. This option can be used to find a point that minimizes the sum of the violations of a subset of constraints determined by the parameter `helast`.

`Elastic weight` r Default = 1.0

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

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

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

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

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

`Expand frequency` i Default = 10000

This option is part of the EXPAND anti-cycling procedure [3] designed to make progress even on highly degenerate problems.

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

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

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

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

- With linear programs, the basis factors are usually updated every iteration. The default k is reasonable for typical problems. Higher values up to $k =$ `100` (say) may be more efficient on problems that are extremely sparse and well scaled.
- When the objective function is quadratic, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the `Check frequency`) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of k updates is reached.

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

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

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

`Infinite Bound Size` r Default = `1.0E` + `20`

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

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

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.

`Log Frequency` k Default = 1

see `Print Frequency`

`LU factor tolerance` r_1 Default = 100.0

`LU update tolerance` r_2 Default = 10.0

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

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

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

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

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

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

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

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

`LU density tolerance` r_1 Default = 0.6

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

The density tolerance r_1 is used during LU factorization of the basis matrix. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds r_1, the Markowitz strategy for choosing pivots is altered to reduce the time spent searching for each remaining pivot. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.

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

`Minimize` Default

`Maximize`

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

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

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

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

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

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

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

- The quantity $\|\pi\|$ actually used is defined by

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

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

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

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

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

- When $i = 1$, all columns of the constraint matrix ($A \quad -I$) are searched.

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

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

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

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

`Print frequency` k Default = 1

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

`Print level` k Default = 1

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

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

`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 [11]). 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`

`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 §7.5.1.

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

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

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

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

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

`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.

7.4 File 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 §7.5.1.

7.4.1 The iteration Log

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

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

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

Note that dj is the norm of the reduced-gradient vector at the start of the iteration, just after the PRICE operation.

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

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

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

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

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

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

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

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

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

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

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

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

Note: If Elasticmode $= 2$, 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.

7.4.2 Basis Factorization Statistics

When `PrintLevel` ≥ 20 and `Printfile` > 0, the following lines of intermediate printout (< 120 characters) are produced on the unit number specified by `Print file` whenever the matrix B or $B_S = (\,B \;\; S\,)^T$ is factorized. Gaussian elimination is used to compute an LU factorization of B or B_S, where PLP^T is a lower triangular matrix and PUQ is an upper triangular matrix for some permutation matrices P and Q. This factorization is stabilized in the manner described under `LU factor tolerance` in §7.3.2.

Label *Description*

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

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

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

`Iteration` The current iteration number.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`Bmax` The maximum nonzero element in B.

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

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

`Umax` is not printed if B_S is factorized.

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

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

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

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

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

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

7.4.3 Crash statistics

When `PrintLevel` ≥ 20 and `Printfile` > 0, the following CRASH statistics (< 120 characters) are produced on the unit number specified by `Print file` whenever `Start` = 'Cold' (see §7.3.2). 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.

7.4.4 EXIT conditions

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

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

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

`-2. EXIT -- input error.`

Otherwise, the SPECS file may be empty, or cards containing the keywords `Skip` or `Endrun` may imply that all problems should be ignored (see §7.3).

`-1. ENDRUN`

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

The following messages arise when optimization terminates gracefully.

`0. EXIT -- optimal solution found`

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

`1. EXIT -- the problem is infeasible`

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

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

`2. EXIT -- the problem is unbounded (or badly scaled)`

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

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

`3. EXIT -- iteration limit exceeded`

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

`4. EXIT -- QP Hessian appears to be indefinite`

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

`5. EXIT -- the superbasics limit is too small: nnn`

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

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

`6. EXIT -- weak solution found`

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

This exit will occur when (i) the problem is feasible, (ii) the reduced gradient is negligible, (iii) the Lagrange multipliers are optimal, and (iv) the reduced Hessian is singular or there are some very small multipliers. This exit cannot occur if H is positive definite (i.e., $q(x)$ is strictly convex).

`10. EXIT -- cannot satisfy the general constraints`

An LU factorization of the basis has just been obtained and used to recompute the basic variables x_B, given

the present values of the superbasic and nonbasic variables. A single step of "iterative refinement" has also been applied to increase the accuracy of x_B. However, a row check has revealed that the resulting solution does not satisfy the current constraints $Ax - s = 0$ sufficiently well.

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

For certain highly structured basis matrices (notably those with band structure), a systematic growth may occur in the factor U. Consult the description of `Umax`, `Umin` and `Growth` in §7.4.2, and set the `LU factor tolerance` to 2.0 (or possibly even smaller, but not less than 1.0).

> If the following exits occur during the *first* basis factorization, the basic variables x_B will have certain default values that may not be particularly meaningful, and the dual vector π will be zero.

`20. EXIT -- not enough integer/real storage for the basis factors`

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

`21. EXIT -- error in basis package`

A preceding message will describe the error in more detail. One such message says that the current basis has more than one element in row i and column j.

`22. EXIT -- singular basis after nnn factorization attempts`

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

`32. EXIT -- system error. Wrong no. of basic variables: nnn`

This exit should never happen. If it does, something is seriously awry in the SQOPT source code.

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

`42. EXIT -- not enough 8-character storage to start solving the problem`

`43. EXIT -- not enough integer storage to start solving the problem`

`44. EXIT -- not enough real storage to start solving the problem`

7.5 Solution Output

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

The ROWS section

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

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

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

Label — *Description*

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

`Row` The name of the ith row.

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

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

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

- `A` *Alternative optimum possible.* The slack is nonbasic, but its reduced gradient is essentially zero. This means that if the slack were allowed to start moving away from its bound, there would be no change in the value of the objective function. The values of the basic and superbasic variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case, the values of dual variables *might* also change.
- `D` *Degenerate.* The slack is basic or superbasic, but it is equal to (or very close to) one of its bounds.
- `I` *Infeasible.* The slack is basic or superbasic and it is currently violating one of its bounds by more than the `Feasibility tolerance`.
- `N` *Not precisely optimal.* The slack is nonbasic or superbasic. If the `Optimality tolerance` were tightened by a factor of 10 (e.g., if it were reduced from 10^{-5} to 10^{-6}), the solution would not be declared optimal because the reduced gradient for the slack would not be considered negligible. (If a loose tolerance has been used, or if the run was terminated before optimality, this key might be helpful in deciding whether or not to restart the run.)

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

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

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

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

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

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

`I` The constraint number, i.

The COLUMNS section

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

`Label` *Description*

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

`Column` The name of x_j.

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

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

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

`Activity` The value of the variable x_j.

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

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

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

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

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

An example of the printed solution is given in §7.4. 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.

7.5.1 The SUMMARY file

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

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

The following information is included:

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

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

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

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

Begin sqmain (Example program for sqopt)


Scale option  2,    Partial price   1


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

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

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


This is problem  sqmain.   ncolH =   5

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

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

Solution printed on file   9
```

7.6 Algorithmic Details

SQOPT is based on an inertia-controlling method that maintains a Cholesky factorization of the reduced Hessian (see below). The method follows Gill and Murray [14] and is described in [19]. Here we briefly summarize the main features of the method. Where possible, explicit reference is made to items listed in the printed output, and to the names of the relevant optional parameters.

7.6.1 Overview

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

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

$$\begin{pmatrix} \bar{x} \\ \bar{s} \end{pmatrix} = \begin{pmatrix} x \\ s \end{pmatrix} + \alpha p, \tag{37}$$

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

7.6.2 Definition of the working set

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

The search direction is defined so that constraints in the working set remain *unaltered* for any value of the step length. It follows that p must satisfy the identity $Wp = 0$. This characterization allows p to be computed using any $n \times n_Z$ full-rank matrix Z that spans the null space of W. (Thus, $n_Z = n - m_w$ and $WZ = 0$.) The null-space matrix Z is defined from a sparse LU factorization of part of W; see (38)–(39) below). The direction p will satisfy $Wp = 0$ if $p = Zp_Z$ for any n_Z-vector p_Z.

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

The working-set matrix W can be represented in terms of a certain column partition of the matrix $\begin{pmatrix} A & -I \end{pmatrix}$. As in §7.2 we partition the constraints $Ax - s = 0$ so that

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

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

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

If P is a permutation such that $\begin{pmatrix} A & -I \end{pmatrix} P = \begin{pmatrix} B & S & N \end{pmatrix}$, then the working-set matrix W satisfies

$$WP = \begin{pmatrix} B & S & N \\ 0 & 0 & I_N \end{pmatrix}, \tag{38}$$

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

The null-space matrix Z is defined from a sparse LU factorization of part of W. In particular, Z is maintained in "reduced-gradient" form, using the package LUSOL [2] to maintain sparse LU factors of the basis matrix B that alters as the working set W changes. Given the permutation P, the null-space basis is given by

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}. \tag{39}$$

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

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

$$g_Z = Z^Tg \qquad \text{and} \qquad H_Z = Z^THZ, \tag{40}$$

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

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

If the QP contains linear variables, H is positive semi-definite and R may be singular with at least one zero diagonal. However, an inertia-controlling active-set strategy is used to ensure that only the last diagonal of R can be zero. (See [19] for discussion of a similar strategy for indefinite quadratic programming.)

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

7.6.3 The main iteration

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

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

The special form (38) of the working set allows the multiplier vector λ, the solution of $W^T\lambda = g$, to be written in terms of the vector

$$d = \begin{pmatrix} g \\ 0 \end{pmatrix} - \begin{pmatrix} A & -I \end{pmatrix}^T \pi = \begin{pmatrix} g - A^T\pi \\ \pi \end{pmatrix}, \tag{41}$$

where π satisfies the equations $B^T\pi = g_B$, and g_B denotes the basic components of g. The components of π are the Lagrange multipliers λ_j associated with the equality constraints $Ax - s = 0$. The vector d_N of nonbasic components of d consists of the Lagrange multipliers λ_j associated with the upper and lower bound constraints in the working set. The vector d_S of superbasic components of d is the reduced gradient g_Z (40). The vector d_B of basic components of d is zero, by construction. (The Euclidean norm of d_S, and the final values of d_S, g and π are the quantities `norm rg`, `Reduced Gradnt`, `Obj Gradient` and `Dual Activity` in the PRINT file output.)

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

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

$$R^T R p_Z = -g_Z, \tag{42}$$

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

If H_Z is singular, then R must also be singular, and an inertia-controlling strategy is used to ensure that only the last diagonal element of R is zero. (See [19] for discussion of a similar strategy for indefinite quadratic programming.) In this case, p_Z satisfies

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

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

7.6.4 Miscellaneous

If the basis matrix is not chosen carefully, the condition of the null-space matrix Z (39) could be arbitrarily high. To guard against this, SQOPT implements a "basis repair" feature in the following way. LUSOL is used to compute the rectangular factorization

$$\begin{pmatrix} B & S \end{pmatrix}^T = LU, \tag{43}$$

returning just the permutation P that makes PLP^T unit lower triangular. The pivot tolerance is set to require $|PLP^T|_{ij} \leq 2$, and the permutation is used to define P in (38). It can be shown that $\|Z\|$ is likely to be little more than 1. Hence, Z should be well-conditioned *regardless of the condition of* W.

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

The EXPAND procedure (see Gill *et al.* [3]) is used to reduce the possibility of cycling at a point where the active constraints are nearly linearly dependent. Although there is no absolute guarantee that cycling will not occur, the probability of cycling is extremely small (see Hall and McKinnon [20]). The main feature of expand is that the feasibility tolerance is increased at every iteration, perhaps at the expense of violating the bounds on (x, s) by a simple amount.

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

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

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

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

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

8 SNOPT details

8.1 Introduction

TOMLAB /SNOPT (hereafter referred to as SNOPT) is a general-purpose system for constrained optimization. It minimizes a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. It is suitable for large-scale linear and quadratic programming and for linearly constrained optimization, as well as for general nonlinear programs of the form

SparseNP	$\underset{x}{\text{minimize}}$	$f_0(x)$
	subject to	$l \leq \begin{pmatrix} x \\ f(x) \\ A_L x \end{pmatrix} \leq u,$

where l and u are constant lower and upper bounds, $f_0(x)$ is a smooth scalar objective function, A_L is a sparse matrix, and $f(x)$ is a vector of smooth nonlinear constraint functions $\{f_i(x)\}$. An optional parameter `maximize` may specify that $f_0(x)$ should be maximized instead of minimized.

Ideally, the first derivatives (gradients) of $f_0(x)$ and $f_i(x)$ should be known and coded by the user.

Note that upper and lower bounds are specified for all variables and constraints. This form allows full generality in specifying various types of constraint. Special values are used to indicate absent bounds ($l_j = -\infty$ or $u_j = +\infty$ for appropriate j). Free variables and free constraints ("free rows") are ones that have both bounds infinite. Fixed variables and equality constraints have $l_j = u_j$.

8.1.1 Problem types

If $f_0(x)$ is linear and $f(x)$ is absent, SparseNP is a *linear program* (LP) and SNOPT applies the primal simplex method [8]. Sparse basis factors are maintained by LUSOL [2] as in MINOS [4].

If only the objective is nonlinear, the problem is *linearly constrained* (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). For both cases, SNOPT applies a sparse sequential quadratic programming (SQP) method [34], using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for steplength control is an augmented Lagrangian, as in the dense SQP solver NPSOL [28, 36].

In general, SNOPT requires less matrix computation than NPSOL and fewer evaluations of the functions than the nonlinear algorithms in MINOS [23, 1].

It is suitable for nonlinear problems with thousands of constraints and variables, and is efficient if many constraints and bounds are active at a solution. (Thus, ideally there should not be thousands of degrees of freedom.)

8.2 Description of the SQP method

Here we summarize the main features of the SQP algorithm used in SNOPT and introduce some terminology used in the description of the subroutine and its arguments. The SQP algorithm is fully described by Gill, Murray and Saunders [34].

8.2.1 Constraints and slack variables

The upper and lower bounds on the m components of $f(x)$ and $A_L x$ are said to define the *general constraints* of the problem. SNOPT 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 linear constraint $5 \leq 2x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \leq s_1 \leq +\infty$. SparseNP can therefore be written in the equivalent form

$$\begin{aligned} &\underset{x,s}{\text{minimize}} \quad f_0(x) \\ &\text{subject to } \begin{pmatrix} f(x) \\ A_L x \end{pmatrix} - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u. \end{aligned}$$

The general constraints become the equalities $f(x) - s_N = 0$ and $A_L x - s_L = 0$, where s_L and s_N are known as the *linear* and *nonlinear* slacks.

8.2.2 Major iterations

The basic structure of the SQP algorithm involves *major* and *minor* iterations. The major iterations generate a sequence of iterates $\{x_k\}$ that satisfy the linear constraints and converge to a point that satisfies the first-order conditions for optimality. At each iterate a QP subproblem is used to generate a search direction towards the next iterate x_{k+1}. The constraints of the subproblem are formed from the linear constraints $A_L x - s_L = 0$ and the nonlinear constraint linearization

$$f(x_k) + f'(x_k)(x - x_k) - s_N = 0,$$

where $f'(x_k)$ denotes the *Jacobian matrix*, whose elements are the first derivatives of $f(x)$ evaluated at x_k. The QP constraints therefore comprise the m linear constraints

$$\begin{aligned} f'(x_k)x - s_N &= -f(x_k) + f'(x_k)x_k, \\ A_L x - s_L &= 0, \end{aligned}$$

where x and s are bounded above and below by u and l as before. If the $m \times n$ matrix A and m-vector b are defined as

$$A = \begin{pmatrix} f'(x_k) \\ A_L \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -f(x_k) + f'(x_k)x_k \\ 0 \end{pmatrix},$$

then the QP subproblem can be written as

$$\underset{x,s}{\text{minimize}} \;\; q(x) \quad \text{subject to} \;\; Ax - s = b, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u, \tag{44}$$

where $q(x)$ is a quadratic approximation to a modified Lagrangian function [34].

8.2.3 Minor iterations

Solving the QP subproblem is itself an iterative procedure. The iterations of the QP solver are the *minor* iterations of the SQP method. At each minor iteration, the constraints $Ax - s = b$ are (conceptually) partitioned into the form

$$Bx_B + Sx_S + Nx_N = b,$$

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

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

Associated with each of the m equality constraints $Ax - s = b$ are the *dual variables* π. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j. The reduced gradients for the variables x are the quantities $g - A^T\pi$, where g is the gradient of the QP objective, and the reduced gradients for the slacks are the dual variables π. The QP subproblem is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for other variables, including superbasics. In practice, an *approximate* QP solution $(\widehat{x}_k, \hat{s}_k, \widehat{\pi}_k)$ is found by relaxing these conditions.

8.2.4 The merit function

After a QP subproblem has been solved, new estimates of the SparseNP solution are computed using a line search on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f_0(x) - \pi^T\big(f(x) - s_N\big) + \tfrac{1}{2}\big(f(x) - s_N\big)^T D\big(f(x) - s_N\big), \tag{45}$$

where D is a diagonal matrix of penalty parameters ($D_{ii} \geq 0$). If (x_k, s_k, π_k) denotes the current solution estimate and $(\widehat{x}_k, \hat{s}_k, \widehat{\pi}_k)$ denotes the QP solution, the line search determines a step α_k ($0 < \alpha_k \leq 1$) such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \widehat{x}_k - x_k \\ \hat{s}_k - s_k \\ \widehat{\pi}_k - \pi_k \end{pmatrix} \tag{46}$$

gives a *sufficient decrease* in the merit function (45). When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ [36].

As in NPSOL, s_N is adjusted to minimize the merit function as a function of s prior to the solution of the QP subproblem. For more details, see [28, 10].

8.2.5 Treatment of constraint infeasibilities

SNOPT makes explicit allowance for infeasible constraints. First, infeasible *linear* constraints are detected by solving the linear program

$$\text{FLP} \qquad \begin{array}{l} \underset{x,v,w}{\text{minimize}} \quad e^T(v+w) \\ \text{subject to} \quad l \le \begin{pmatrix} x \\ A_L x - v + w \end{pmatrix} \le u, \quad v \ge 0, \quad w \ge 0, \end{array}$$

where e is a vector of ones, and the nonlinear constraint bounds are temporarily excluded from l and u. This is equivalent to minimizing the sum of the general linear constraint violations subject to the bounds on x. (The sum is the ℓ_1-norm of the linear constraint violations. In the linear programming literature, the approach is called *elastic programming*.)

The linear constraints are infeasible if the optimal solution of FLP has $v \neq 0$ or $w \neq 0$. SNOPT then terminates without computing the nonlinear functions.

Otherwise, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) SNOPT proceeds to solve SparseNP as given, using search directions obtained from the sequence of QP subproblems (44).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the nonlinear constraints become large), SNOPT enters "elastic" mode and thereafter solves the problem

$$\text{NP}(\gamma) \qquad \begin{array}{l} \underset{x,v,w}{\text{minimize}} \quad f_0(x) + \gamma e^T(v+w) \\ \text{subject to} \quad l \le \begin{pmatrix} x \\ f(x) - v + w \\ A_L x \end{pmatrix} \le u, \quad v \ge 0, \quad w \ge 0, \end{array}$$

where γ is a nonnegative parameter (the *elastic weight*), and $f_0(x) + \gamma e^T(v+w)$ is called a *composite objective* (the ℓ_1 penalty function for the nonlinear constraints).

The value of γ may increase automatically by multiples of 10 if the optimal v and w continue to be nonzero. If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar ℓ_1 formulation of SparseNP is fundamental to the Sℓ_1QP algorithm of Fletcher [38]. See also Conn [7].

The initial value of γ is controlled by the optional parameters `Elastic mode` and `Elastic weight` (p. 180).

8.3 Optional parameters

The performance of each SNOPT interface is controlled by a number of parameters or "options". Each option has a default value that should be appropriate for most problems. The options are normally set in *optPar* or *Prob.SOL.optPar* before calling the solver. 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*.

8.3.1 The SPECS file

The specs file contains a list of option definitions, using data in the following general form:

```
Begin options
   Iterations limit             500
   Minor feasibility tolerance  1.0e-7
   Solution                     Yes
End options
```

We call such data a SPECS file because it specifies various options. The file starts with the keyword `Begin` and ends with `End`. Each line specifies a single option in free format, 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.

It may be useful to include a comment on the first (`Begin`) line of the file. This line is echoed to the SUMMARY file.

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.

8.3.2 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
  Major print level              1       * one-line major iteration log
  Minor print level              1       * one-line minor iteration log
  Print   file                   9       *
  Summary file                   6       * typically the screen
  Print   frequency            100       * minor iterations log on PRINT file
  Summary frequency            100       * minor 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
  Major feasibility tolerance    1.0e-6  * target nonlinear constraint violation
  Major optimality  tolerance    1.0e-6  * target complementarity gap
  Minor feasibility tolerance    1.0e-6  * for satisfying the QP bounds

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

* Scaling
  Scale option                   1       * linear constraints and variables
  Scale tolerance                0.9     *
* Scale Print                             * default: scales are not printed

* Other Tolerances
  Crash tolerance                0.1     *
  Linesearch tolerance           0.9     * smaller for more accurate search
  Pivot tolerance                3.7e-11 * eps^(2/3)

* QP subproblems
  QPSolver                     Cholesky  * default
  Crash option                   3       * first basis is essentially triangular
  Elastic mode                   No      * until it seems necessary
  Elastic weight                 1.0e+4  * used only during elastic mode
  Iterations limit               10000   * or 20m if that is more
  Partial price                  1       * 10 for large LPs

* SQP method
  Major iterations limit         1000    * or m if that is more
```

```
  Minor iterations limit          500    *
  Major step limit                2.0    *
  Superbasics limit              n + 1
  Hessian dimension               750    * or Superbasics limit if that is less
  Derivative level               3       *
  Derivative linesearch                  *
* Nonderivative linesearch               *
  Function precision             3.0e-13 * ε^0.8 (almost full accuracy)
  Difference interval            5.5e-7  * (Function precision)^1/2
  Central difference interval    6.7e-5  * (Function precision)^1/3
  New superbasics limit          99      * controls early termination of QPs
  Objective row                  ObjRow  * row number of objective in F(x)
  Penalty parameter               0.0    * initial penalty parameter
  Proximal point method             1    * satisfies linear constraints near x_0
  Violation limit                10.0    * unscaled constraint violation limit
  Unbounded step size            1.0e+18 *
  Unbounded objective            1.0e+15 *

* Hessian approximation
  Hessian                    full memory * default if n ≤ 75
  Hessian                 limited memory * default if n > 75
  Hessian frequency              999999  * for full Hessian (never reset)
  Hessian updates                20      * for limited memory Hessian
  Hessian flush                  999999  * no flushing

* Frequencies
  Check frequency                60      * test row residuals ||Ax - s||
  Expand frequency               10000   * for anti-cycling procedure
  Factorization frequency        50      * 100 for LPs
  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       3.25e-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

* 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
```

8.3.3 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. In the description of the options we use the notation of the problem format SparseNP to refer to the objective and constraint functions.

`Central difference interval` r Default $= \epsilon^{1/3} \approx$ `6.0e-6`

When `Derivative level` < 3), the central-difference interval r is used near an optimal solution to obtain more accurate (but more expensive) estimates of gradients. Twice as many function evaluations are required compared to forward differencing. The interval used for the jth variable is $h_j = r(1+|x_j|)$. The resulting derivative estimates should be accurate to $O(r^2)$, unless the functions are badly scaled.

`Check frequency` i Default $= 60$

Every ith minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax - s = b$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax + s$ 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 $= 3$

`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 twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and CRASH is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).
3	CRASH is called up to three times (if there are nonlinear constraints). The first two calls treat *linear equalities* and *linear inequalities* separately. As before, the last call treats nonlinear rows before the first major iteration.

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 are 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 $r > 1/4$.

`Derivative level` i Default = 3

The keyword `Derivative level` specifies which nonlinear function gradients are known analytically and will be supplied to SNOPT. This is normally automatically set by TOMLAB.

i	*Meaning*
3	All objective and constraint gradients are known.
2	All constraint gradients are known, but some or all components of the objective gradient are unknown.
1	The objective gradient is known, but some or all of the constraint gradients are unknown.
0	Some components of the objective gradient are unknown and some of the constraint gradients are unknown.

The value $i =$ `3` should be used whenever possible. It is the most reliable and will usually be the most efficient.

If $i =$ `0` or `2`, SNOPT will *estimate* the missing components of the objective gradient, using finite differences. However, it could increase the total run-time substantially, and there is less assurance that an acceptable solution will be located. If the nonlinear variables are not well scaled, it may be necessary to specify a nonstandard `Difference interval` (see below).

If $i =$ `0` or `1`, SNOPT will estimate missing elements of the Jacobian. For each column of the Jacobian, one call is needed to estimate all missing elements in that column, if any. If `Jacobian = sparse` and the sparsity pattern of the Jacobian happens to be

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

where `*` indicates known gradients and `?` indicates unknown elements, SNOPT will use one call to estimate the missing element in column 2, and another call to estimate both missing elements in column 3. No calls are needed for columns 1 and 4.

At times, central differences are used rather than forward differences.

`Derivative linesearch` Default

`Nonderivative linesearch`

At each major iteration a line search is used to improve the merit function. A `Derivative linesearch` uses safeguarded cubic interpolation and requires both function and gradient values to compute estimates of the step α_k. If some analytic derivatives are not provided, or a `Nonderivative linesearch` is specified, SNOPT employs a line search based upon safeguarded quadratic interpolation, which does not require gradient evaluations.

A nonderivative line search can be slightly less robust on difficult problems, and it is recommended that the default be used if the functions and derivatives can be computed at approximately the same cost. If the gradients are very expensive relative to the functions, a nonderivative line search may give a significant decrease in computation time.

Comment: Derivative linesearch is only default if analytic derivatives are provided.

`Difference interval` h_1 Default $= \epsilon^{1/2} \approx$ `1.5e-8`

This alters the interval h_1 that is used to estimate gradients by forward differences in the following circumstances:

- In the initial ("cheap") phase of verifying the problem derivatives.
- For verifying the problem derivatives.
- For estimating missing derivatives.

In all cases, a derivative with respect to x_j is estimated by perturbing that component of x to the value $x_j + h_1(1 + |x_j|)$, and then evaluating $f_0(x)$ or $f(x)$ at the perturbed point. The resulting gradient estimates should be accurate to $O(h_1)$ unless the functions are badly scaled. Judicious alteration of h_1 may sometimes lead to greater accuracy.

`Elastic mode` `No` Default

`Elastic mode` `Yes`

Normally SNOPT initiates elastic mode only when it seems necessary. Option `Yes` causes elastic mode to be entered from the beginning.

`Elastic weight` ω Default $= 10^4$

This keyword determines the initial weight γ associated with problem NP(γ) on p. 174.

At major iteration k, if elastic mode has not yet started, a scale factor $\sigma_k = 1 + \|g(x_k)\|_\infty$ is defined from the current objective gradient. Elastic mode is then started if the QP subproblem is infeasible, or the QP dual variables are larger in magnitude than $\sigma_k\omega$. The QP is re-solved in elastic mode with $\gamma = \sigma_k\omega$.

Thereafter, major iterations continue in elastic mode until they converge to a point that is optimal for problem NP(γ). If the point is feasible for SparseNP ($v = w = 0$), it is declared locally optimal. Otherwise, γ is increased by a factor of 10 and major iterations continue. If γ has already reached a maximum allowable value, SparseNP is declared locally infeasible.

`Expand frequency` i Default = 10000

This option is part of the EXPAND anti-cycling procedure [3] designed to make progress even on highly degenerate problems.

For linear models, 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 `Minor feasibility tolerance` is δ. Over a period of i iterations, the tolerance actually used by SNOPT increases from 0.5δ to δ (in steps of $0.5\delta/i$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

Increasing i helps reduce the number of slightly infeasible nonbasic 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 = 50

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 nonlinear, 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`

see `Minor feasibility tolerance`

`Feasible point`

see `Minimize`

`Function precision` ϵ_R Default = $\epsilon^{0.8} \approx$ `3.7e-11`

The *relative function precision* ϵ_R is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ is computed as `1000.56789` for some relevant x and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_R would be `1.0e-6`.

(Ideally the functions $f(x)$ or $F_i(x)$ should have magnitude of order 1. If all functions are substantially *less* than 1 in magnitude, ϵ_R should be the *absolute* precision. For example, if $f(x) =$ `1.23456789e-4` at some point and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_R would be `1.0e-10`.)

- The default value of ϵ_R is appropriate for simple analytic functions.

- In some cases the function values will be the result of extensive computation, possibly involving an iterative procedure that can provide rather few digits of precision at reasonable cost. Specifying an appropriate `Function precision` may lead to savings, by allowing the line search procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

`Hessian dimension` r Default = `Superbasics limit` or 750

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 $R^T R = Z^T H Z$).

`Hessian full memory` Default = `Full` if $n_1 \leq 75$

`Hessian limited memory`

These options select the method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

If `Hessian full memory` is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of nonlinear variables n_1 is not too large (say, less than 75). In this case, the storage requirement is fixed and one can expect 1-step Q-superlinear convergence to the solution.

`Hessian limited memory` should be used on problems where n_1 is very large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation H_r a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after H_r has been reset to their diagonal.)

`Hessian frequency` i Default = 999999

If `Hessian Full` is selected and i BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)

`Hessian Full memory` and `Hessian frequency = 20` have a similar effect to `Hessian Limited memory` and `Hessian updates = 20` (except that the latter retains the current diagonal during resets).

`Hessian updates` i Default = 20

If `Hessian Limited memory` is selected and i BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.

Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the more vectors stored, the greater the cost of each QP iteration. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., $i = 5$).

`Iterations limit` k Default = $\max\{10000, 20m\}$

This is the maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations.

`Infinite Bound size` r Default = `1.0e+20`

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

`Linesearch tolerance` t Default = 0.9

This controls the accuracy with which a steplength will be located along the direction of search each iteration. At the start of each line search a target directional derivative for the merit function is identified. This parameter determines the accuracy to which this target value is approximated.

- t must be a real value in the range $0.0 \leq t \leq 1.0$.
- The default value $t = 0.9$ requests just moderate accuracy in the line search.
- If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate; try $t = 0.1$, 0.01 or 0.001. The number of major iterations might decrease.
- If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. *If all gradients are known*, try $t = 0.99$. (The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.)
- If not all gradients are known, a moderately accurate search remains appropriate. Each search will require only 1–5 function values (typically), but many function calls will then be needed to estimate missing gradients for the next iteration.

`Log frequency` k Default = 100

see `Print frequency`

`LU factor tolerance` r_1 Default = 100.0 (LP) or 3.99 (NLP)

`LU update tolerance` r_2 Default = 10.0 (LP) or 3.99 (NLP)

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, $r_1 = 5.0$ (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

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

  both r_1 and r_2 should be in the range $1.0 \leq r_i < 4.0$.

`LU Partial Pivoting` Default

`LU Rook Pivoting`

`LU Complete Pivoting`

The LU factorization implements a Markowitz-type search for a pivot that locally minimizes the fill-in subject to a threshold pivoting stability criterion. The default option is to use threshold 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 = 0.5

`LU singularity tolerance` r_2 Default = $\epsilon^{0.67} \approx$ `3.25e-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 $|U_{jj}| \leq r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the jth column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)

In some cases, the Jacobian matrix may converge to values that make the basis exactly singular. (For example, a whole row of the Jacobian could be zero at an optimal solution.) Before exact singularity occurs, the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting a larger tolerance $r_2 =$ `1.0e-5`, say, may help cause a judicious change of basis.

`Major feasibility tolerance` ϵ_r Default = `1.0e-6`

This specifies how accurately the nonlinear constraints should be satisfied. The default value of `1.0e-6` is appropriate when the linear and nonlinear constraints contain data to about that accuracy.

Let `rowerr` be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$\texttt{rowerr} = \max_i \ \texttt{viol}_i/\|x\| \ \leq \ \epsilon_r, \tag{47}$$

where $\texttt{viol}_i$ is the violation of the ith nonlinear constraint ($i = 1 : \texttt{nnCon}$).

In the major iteration log, `rowerr` appears as the quantity labeled "`Feasibl`". If some of the problem functions are known to be of low accuracy, a larger `Major feasibility tolerance` may be appropriate.

`Major optimality tolerance` ϵ_d Default = `1.0e-6`

This specifies the final accuracy of the dual variables. On successful termination, SNOPT will have computed a solution (x, s, π) such that

$$\texttt{maxComp} = \max_j \ \texttt{Comp}_j/\|\pi\| \ \leq \ \epsilon_d, \tag{48}$$

where $\texttt{Comp}_j$ is an estimate of the complementarity slackness for variable j ($j = 1 : n + m$). The values $\texttt{Comp}_j$ are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$ (where g_j is the jth component of the objective gradient, a_j is the associated column of the constraint matrix $(A \quad -I)$, and π is the set of QP dual variables):

$$\texttt{Comp}_j = \begin{cases} d_j \min\{x_j - l_j, 1\} & \text{if } d_j \geq 0; \\ -d_j \min\{u_j - x_j, 1\} & \text{if } d_j < 0. \end{cases}$$

In the major iteration log, `maxComp` appears as the quantity labeled "`Optimal`".

`Major iterations limit` k Default = $\max\{1000, m\}$

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints.

`Major print level` p Default = 00001

This controls the amount of output to the PRINT and SUMMARY files each major iteration. `Major print level 1` gives normal output for linear and nonlinear problems, and `Major print level 11` gives addition details of the Jacobian factorization that commences each major iteration.

In general, the value being specified may be thought of as a binary number of the form

`Major print level JFDXbs`

where each letter stands for a digit that is either `0` or `1` as follows:

- `s` a single line that gives a summary of each major iteration. (This entry in `JFDXbs` is not strictly binary since the summary line is printed whenever $\texttt{JFDXbs} \geq 1$).
- `b` BASIS statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if $\texttt{JFDXbs} \geq 10$).
- `X` x_k, the nonlinear variables involved in the objective function or the constraints.
- `D` π_k, the dual variables for the nonlinear constraints.

`F` $F(x_k)$, the values of the nonlinear constraint functions.

`J` $J(x_k)$, the Jacobian matrix.

To obtain output of any items `JFDXbs`, set the corresponding digit to `1`, otherwise to `0`.

If `J=1`, the Jacobian matrix will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if `J=1`, there is no reason to specify `X=1` unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3  1.250000D+01 BS      1  1.00000E+00      4  2.00000E+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.

`Major print level 0` suppresses most output, except for error messages.

`Major step limit` r Default = 2.0

This parameter limits the change in x during a line search. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

1. A line search determines a step α over the range $0 < \alpha \leq \beta$, where β is 1 if there are nonlinear constraints, or the step to the nearest upper or lower bound on x if all the constraints are linear. Normally, the first steplength tried is $\alpha_1 = \min(1, \beta)$.
2. In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The parameter r is therefore used to define a limit $\bar{\beta} = r(1 + \|x\|)/\|p\|$ (where p is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \bar{\beta}, \beta)$.
3. Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The `Major step limit` provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A "good" starting point may be required. An important application is to the class of nonlinear least-squares problems.
4. In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

`Minimize` Default

`Maximize`

`Feasible point`

The keywords `Minimize` and `Maximize` specify the required direction of optimization. It applies to both linear and nonlinear terms in the objective.

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

`Minor iterations limit` k Default = 500

If the number of minor iterations for the optimality phase of the QP subproblem exceeds k, then all nonbasic QP variables that have not yet moved are frozen at their current values and the reduced QP is solved to optimality.

Note that more than k minor iterations may be necessary to solve the reduced QP to optimality. These extra iterations are necessary to ensure that the terminated point gives a suitable direction for the line search.

In the major iteration log, a `t` at the end of a line indicates that the corresponding QP was artificially terminated using the limit k.

Note that `Iterations limit` defines an independent *absolute* limit on the *total* number of minor iterations (summed over all QP subproblems).

`Minor feasibility tolerance` t Default = `1.0e-6`

SNOPT tries to ensure that all variables eventually satisfy their upper and lower bounds to within the tolerance t. This includes slack variables. Hence, general linear constraints should also be satisfied to within t.

Feasibility with respect to nonlinear constraints is judged by the `Major feasibility tolerance` (not by t).

- If the bounds and linear constraints cannot be satisfied to within t, the problem is declared *infeasible*. Let `sInf` be the corresponding sum of infeasibilities. 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.
- Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem.

  For example, if $f(x) = \sqrt{x_1} + \log x_2$, it is essential to place lower bounds on both variables. If $t =$ `1.0e-6`, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious. In general, keep x as far away from singularities as possible.)
- If `Scale option` ≥ 1, feasibility is defined in terms of the *scaled* problem (since it is then more likely to be meaningful).
- In reality, SNOPT uses t as a feasibility tolerance for satisfying the bounds on x and s in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible. SNOPT is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). See the `Elastic` options.

`Minor print level` k Default = 1

This controls the amount of output to the PRINT and SUMMARY files during solution of the QP subproblems. The value of k has the following effect:

0	No minor iteration output except error messages.
≥ 1	A single line of output each minor iteration (controlled by `Print frequency` and `Summary frequency`).

≥ 10 Basis factorization statistics generated during the periodic refactorization of the basis (see `Factorization frequency`). Statistics for the *first factorization* each major iteration are controlled by the `Major print level`.

`New superbasics limit` i Default = 99

This option causes early termination of the QP subproblems if the number of free variables has increased significantly since the first feasible point. If the number of new superbasics is greater than i the nonbasic variables that have not yet moved are frozen and the resulting smaller QP is solved to optimality.

In the major iteration log, a "`T`" at the end of a line indicates that the QP was terminated early in this way.

`Partial price` i Default = 10 (LP) or 1 (NLP)

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

During solution of QP subproblems, 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.
- 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 `Minor 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.

`Proximal point method` i Default = 1

$i = 1$ or 2 specifies minimization of $\|x - x_0\|_1$ or $\frac{1}{2}\|x - x_0\|_2^2$ when the starting point x_0 is changed to satisfy the linear constraints (where x_0 refers to nonlinear variables).

`QPSolver Cholesky` Default

`QPSolver CG`

`QPSolver QN`

Specifies the algorithm used to solve the QP subproblem. `QPSolver Cholesky` uses the active-set QP method of SQOPT, which holds the full Cholesky factor R of the reduced Hessian $Z^T H Z$. As the QP iterations proceed, the dimension of R changes as the number of superbasic variables changes. If it is necessary that the number of superbasic variables increases beyond the value of `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 `Hessian dimension`.

`QPSolver QN` solves the QP subproblem using a quasi-Newton method similar to 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 the number of superbasics is large.
- The quasi-Newton QP solver does not require the computation of the R at the start of each QP and may be appropriate when the number of superbasics is large, but each QP subproblem requires relatively few minor iterations.
- The conjugate-gradient QP solver is appropriate for problems with large numbers of degrees of freedom.

`Reduced Hessian dimension` i Default = $\min\{750, n_1 + 1\}$

see `Hessian dimension`

`Scale option` i Default = 2 (LP) or 1 (NLP)

`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 (and Jacobian) never have very large elements (say, larger than 1000).

1 Linear 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 [11]). This will sometimes improve the performance of the solution procedures.

2 All constraints and variables are scaled by the iterative procedure. Also, an additional scaling is performed that takes into account columns of $\begin{pmatrix} A & -I \end{pmatrix}$ that are fixed or have positive lower bounds or negative upper bounds.

If nonlinear constraints are present, the scales depend on the Jacobian at the first point that satisfies the linear constraints. `Scale option 2` should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

`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`

`Start Objective Check at Column`	k	Default = 1
`Start Constraint Check at Column`	k	Default = 1
`Stop Objective Check at Column`	l	Default = n_1'
`Stop Constraint Check at Column`	l	Default = n_1''

If `Verify level` > 0, they may be used to abbreviate the verification of individual derivative elements. For example:

- If the first 100 objective gradients appeared to be correct in an earlier run, and if you have just found a bug in that ought to fix up the 101-th component, then you might as well specify `Start Objective Check at Column 101`. Similarly for columns of the Jacobian.
- If the first 100 variables occur nonlinearly in the constraints, and the remaining variables are nonlinear only in the objective, then one must set the first 100 components of `g(*)` to zero, but these hardly need to be verified. The above option would again be appropriate.

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

If $f > 0$ and `Minor print level` > 0, a line of the QP iteration log will be output to file f every kth minor iteration.

`Superbasics limit`	i	Default = $n_1 + 1$

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

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

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

Normally, i need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems, i may be considerably smaller than n_1. This will save storage if n_1 is very large.

`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 objective value`	$f_{\max}$	Default = `1.0e+15`
`Unbounded step size`	$\alpha_{\max}$	Default = `1.0e+18`

These parameters are intended to detect unboundedness in nonlinear problems. (They may not achieve that purpose!) During a line search, f_0 is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. if $|f_0|$ exceeds $f_{\max}$ or α exceeds $\alpha_{\max}$, iterations are terminated with the exit message `Problem is unbounded (or badly scaled)`.

If singularities are present, unboundedness in $f_0(x)$ may be manifested by a floating-point overflow (during the evaluation of $f_0(x + \alpha p)$), before the test against $f_{\max}$ can be made.

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

`Verify level`	l	Default = 0

This option refers to finite-difference checks on the derivatives computed by the user-provided routines. Derivatives are checked at the first point that satisfies all bounds and linear constraints.

l	*Meaning*
0	Only a "cheap" test will be performed.
1	Individual gradients will be checked (with a more reliable test). A key of the form "`OK`" or "`Bad?`" indicates whether or not each component appears to be correct.
2	Individual columns of the problem Jacobian will be checked.
3	Options 2 and 1 will both occur (in that order).
−1	Derivative checking is disabled.

`Verify level 3` should be specified whenever a new function routine is being developed. The `Start` and `Stop` keywords may be used to limit the number of nonlinear variables checked. Missing derivatives are not checked, so they result in no overhead.

`Violation limit` τ Default = 10

This keyword defines an absolute limit on the magnitude of the maximum constraint violation after the line search. On completion of the line search, the new iterate x_{k+1} satisfies the condition

$$v_i(x_{k+1}) \leq \tau \max\{1, v_i(x_0)\}, \tag{49}$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the ith nonlinear constraint violation $v_i(x) = \max(0, l_i - f_i(x), f_i(x) - u_i)$.

The effect of this violation limit is to restrict the iterates to lie in an *expanded* feasible region whose size depends on the magnitude of τ. This makes it possible to keep the iterates within a region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then τ may be any large positive value.

8.4 File Output

The files can be directed with the `Print file` and `Summary file` options (or suppressed).

8.4.1 The PRINT file

If `Print file` is set (not done through *optPar*, see the help for calling the solver), the following information is output to the PRINT file during the solution process. All printed lines are less than 131 characters.

- A listing of the SPECS file, if any.
- A listing of the options that were or could have been set in the SPECS file.
- An estimate of the working storage needed and the amount available.
- Some statistics about the problem being solved.
- The storage available for the LU factors of the basis matrix.
- A summary of the scaling procedure, if `Scale option` > 0.
- Notes about the initial basis resulting from a CRASH procedure or a BASIS file.
- The major iteration log.
- The minor iteration log.
- Basis factorization statistics.
- The EXIT condition and some statistics about the solution obtained.
- The printed solution, if requested.

The last five items are described in the following sections.

8.4.2 The major iteration log

If `Major print level` > 0, one line of information is output to the PRINT file every kth minor iteration, where k is the specified `Print frequency` (default $k =$ `1`).

Label	*Description*
`Itns`	The cumulative number of minor iterations.
`Major`	The current major iteration number.
`Minors`	is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, `Minors` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see §8.2).
`Step`	The step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.
`nCon`	The number of times user subroutines have been called to evaluate the nonlinear problem functions. Evaluations needed for the estimation of the derivatives by finite differences are not included. `nCon` is printed as a guide to the amount of work required for the line search.
`Feasible`	is the value of `rowerr`, the maximum component of the scaled nonlinear constraint residual (47). The solution is regarded as acceptably feasible if `Feasible` is less than the `Major feasibility tolerance`. In this case, the entry is contained in parenthesis.

If the constraints are linear, all iterates are feasible and this entry is not printed.

`Optimal` is the value of `maxgap`, the maximum complementarity gap (48). It is an estimate of the degree of nonoptimality of the reduced costs. Both `Feasbl` and `Optimal` are small in the neighborhood of a solution.

`MeritFunction` is the value of the augmented Lagrangian merit function (see (45)). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see §8.2). As the solution is approached, `Merit` will converge to the value of the objective at the solution.

In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight.

If the constraints are linear, this item is labeled `Objective`, the value of the objective function. It will decrease monotonically to its optimal value.

`L+U` The number of nonzeros representing the basis factors L and U on completion of the QP subproblem.

If nonlinear constraints are present, the basis factorization $B = LU$ is computed at the start of the first minor iteration. At this stage, `LU` = `lenL` + `lenU`, where `lenL`, the number of subdiagonal elements in the columns of a lower triangular matrix and `lenU` is the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix.

As columns of B are replaced during the minor iterations, `LU` may fluctuate up or down but in general will tend to increase. As the solution is approached and the minor iterations decrease towards zero, `LU` will reflect the number of nonzeros in the LU factors at the start of the QP subproblem.

If the constraints are linear, refactorization is subject only to the `Factorize frequency`, and `LU` will tend to increase between factorizations.

`BSwap` The number of columns of the basis matrix B that were swapped with columns of S to improve the condition of B. The swaps are determined by an LU factorization of the rectangular matrix $B_S = (\,B \;\; S\,)^T$ with stability being favored more than sparsity.

`nS` The current number of superbasic variables.

`CondHz` An estimate of the condition number of R^TR, an estimate of Z^THZ, the reduced Hessian of the Lagrangian. It is the square of the ratio of the largest and smallest diagonals of the upper triangular matrix R (which is a lower bound on the condition number of R^TR). `Cond Hz` gives a rough indication of whether or not the optimization procedure is having difficulty. If ϵ is the relative precision of the machine being used, the SQP algorithm will make slow progress if `Cond Hz` becomes as large as $\epsilon^{-1/2} \approx 10^8$, and will probably fail to find a better solution if `Cond Hz` reaches $\epsilon^{-3/4} \approx 10^{12}$.

To guard against high values of `Cond Hz`, attention should be given to the scaling of the variables and the constraints. In some cases it may be necessary to add upper or lower bounds to certain variables to keep them a reasonable distance from singularities in the nonlinear functions or their derivatives.

`Penalty` is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if there are no nonlinear constraints.

The summary line may include additional code characters that indicate what happened during the course of the major iteration.

Code	*Meaning*

`c` Central differences have been used to compute the unknown components of the objective and constraint gradients. A switch to central differences is made if either the line search gives a small step, or x is close to being optimal. In some cases, it may be necessary to re-solve the QP subproblem with the central-difference gradient and Jacobian.

`d` During the line search it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of `Violation limit`.

`l` The norm-wise change in the variables was limited by the value of the `Major step limit`. If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of `Major step limit`.

`i` If SNOPT is not in elastic mode, an "`i`" signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem NP(γ).

If SNOPT is already in elastic mode, an "`i`" indicates that the minimizer of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

`M` An extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.

`m` This is the same as "`M`" except that it was also necessary to modify the update to include an augmented Lagrangian term.

`n` No positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.

`R` The approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the `Hessian frequency` and `Hessian updates` keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

`r` The approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

`s` A self-scaled BFGS update was performed. This update is always used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.

`t` The minor iterations were terminated because of the `Minor iterations limit`.

`T` The minor iterations were terminated because of the `New superbasics limit`.

`u` The QP subproblem was unbounded.

`w` A weak solution of the QP subproblem was found.

`z` The `Superbasics limit` was reached.

8.4.3 The minor iteration log

If `Minor print level` > 0, one line of information is output to the PRINT file every kth minor iteration, where k is the specified `Minor 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 a nonbasic variable is selected to become superbasic (in addition to those already in the superbasic set). The selected variable is denoted by `jq`. Variable `jq` often becomes basic immediately. Otherwise it remains superbasic, unless it reaches its opposite bound and returns to the nonbasic set.

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

Label *Description*

`Itn` The current iteration number.

`RedCost,QPmult` This is 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 for the QP subproblem, and a_j is the jth column of $(A \quad -I)$.

Note that `dj` is the 1-norm of the reduced-gradient vector at the start of the iteration, just after the PRICE operation.

`LPstep,QPstep` 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 (`+SBS` > 0), `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.

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

`SumInf` If `nInf` > 0, this is `sInf`, the sum of infeasibilities after the present iteration. It usually decreases at each nonzero `Step`, but if `nInf` decreases by 2 or more, `SumInf` may occasionally increase.

In elastic mode, the heading is changed to `Composite Obj`, and the value printed decreases monotonically.

`rgNorm` The norm of the reduced-gradient vector at the start of the iteration. (It is the 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.

(The heading is not printed if the problem is linear.)

`LPobjective,QPobjective,Elastic QPobj` The QP objective function after the present iteration. In elastic mode, the heading is changed to `Elastic QPobj`. In either case, the value printed decreases monotonically.

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

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

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

`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+U` The number of nonzeros representing the basis factors L and U. Immediately after a basis factorization $B = LU$, this is `lenL+lenU`, the number of subdiagonal elements in the columns of a lower triangular matrix and the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix.

Further nonzeros are added to `L` when various columns of B are later replaced. As columns of B are replaced, the matrix U is maintained explicitly (in sparse form). The value of `L` will steadily increase, whereas the value of `U` may fluctuate up or down. Thus, in general, the value of `L+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 SNOPT should be increased by a significant amount. As a suggestion, the work arrays `iw(*)` and `rw(*)` should be extended by `L + U` elements.

`nS` The current number of superbasic variables. (The heading is not printed if the problem is linear.)

`cond Hz` See the major iteration log. (The heading is not printed if the problem is linear.)

8.4.4 Basis factorization statistics

If `Major 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 before solution of the next QP subproblem.

Note that B_S may be factorized at the start of just some of the major 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 §8.3.3.

If `Minor print level` ≥ 10, the same items are printed during the QP solution whenever the current B is factorized.

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 SNOPT 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).

8.4.5 Crash statistics

If `Major print level` ≥ 10, the following items are output to the PRINT file. They refer to the number of columns that the CRASH procedure selects during several passes through A while 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, including those whose bounds are rather far apart.

`Preferred` is the number of "preferred" columns in the basis (i.e., `hs`$(j) = 3$ for some $j \leq n$). It will be a subset of the columns for which `hs`$(j) = 3$ was specified.

`Unit` is the number of unit columns in the basis.

`Double` is the number of columns in the basis containing 2 nonzeros.

`Triangle` is the number of triangular columns in the basis with 3 or more nonzeros.

`Pad` is the number of slacks used to pad the basis (to make it a nonsingular triangle).

8.4.6 EXIT conditions

When any solver or auxiliary routine in the SNOPT package 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, the solver may print the message:

```
SNOPTA EXIT 20 -- the problem appears to be unbounded
SNOPTA 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 number i associated with the informational message is the output value of the argument `inform`. Note that the number e associated with the exit condition may always be recovered from `inform` by stripping off the least significant decimal digit.

The list of possible exit conditions 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 SNOPT auxiliary routines)

110 Errors while processing MPS data

120 Errors while estimating Jacobian structure

130 Errors while reading `OPTIONS` file

140 System error

> The exit conditions 0–20 arise when a solution exists (though it may not be optimal).

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

```
EXIT --  0  Finished successfully
INFO --  1  optimality conditions satisfied
INFO --  2  feasible point found    (from option Feasible point only)
INFO --  3  requested accuracy could not be achieved
```

This message should be the cause of guarded optimism! It is certainly preferable to every other message, and we naturally want to believe what it says.

In all cases, a distinct level of caution is in order. For example, if the objective value is much better than expected, we may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder. It is good practice in the function subroutines to print any data that is input during the first entry.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are *minimizing* the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the *same* objective value, particularly if the objective is largely linear.) Conversely, if we are *maximizing* a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Our advice is always to specify a starting point that is as good an estimate as possible, and to include reasonable upper and lower bounds on all variables, in order to confine the solution to the specific region of interest. We expect modelers to *know something about their problem*, and to make use of that knowledge as they themselves know best.

One other 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.

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

 Problem name                 Toy1
 No. of iterations                7   Objective value     -1.0000000008E+00
 No. of major iterations          7   Linear objective     0.0000000000E+00
 Penalty parameter          3.253E-02 Nonlinear objective -1.0000000008E+00
 No. of calls to funobj           9   No. of calls to funcon             9
 No. of degenerate steps          0   Percentage                      0.00
 Max x                      2 1.0E+00 Max pi                    1 1.2E-01
 Max Primal infeas          0 0.0E+00 Max Dual infeas           2 8.0E-10
 Nonlinear constraint violn   6.4E-09
```

`Max Primal infeas` refers to the largest bound infeasibility and which variable is involved. If it is too large, consider restarting with a smaller `Minor 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

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

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 `Major optimality tolerance`.

Finally, `Nonlinear constraint violn` shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller `Major feasibility tolerance`.

If the requested accuracy could not be achieved, a feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but SNOPT is within 10^{-2} of satisfying the `Major optimality tolerance`. Check that the `Major optimality tolerance` is not too small.

`EXIT -- 10  The problem appears to be infeasible`

`INFO -- 11  infeasible linear constraints`

`INFO -- 12  infeasible linear equalities`

`INFO -- 13  nonlinear infeasibilities minimized`

`INFO -- 14  infeasibilities minimized`

This exit occurs if SNOPT unable to find a point that satisfies the constraints. When the constraints are linear, these message can probably be trusted. 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 `Minor feasibility tolerance` are ignored, but at least one component of x or s violates a bound by more than the tolerance.

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each QP subproblem, SNOPT is prepared to relax the bounds on the slacks associated with nonlinear rows.

If a QP subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), SNOPT enters so-called "nonlinear elastic" mode. The subproblem includes 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 nonlinear rows "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 original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, SNOPT will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, SNOPT would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though SNOPT locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

`EXIT -- 20  The problem appears to be unbounded`

`INFO -- 21  unbounded objective`

`INFO -- 22  constraint violation limit reached`

For linear problems, 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.

For nonlinear problems, SNOPT monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the `Unbounded` parameters—see §8.3.3), the problem is terminated and declared unbounded. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second informational message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the `Violation limit`.

`EXIT -- 30  Resource limit error`

`INFO -- 31  iteration limit reached`

`INFO -- 32  major iteration limit reached`

`INFO -- 33  the superbasics limit is too small`

Either the `Iterations limit` or the `Major 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 *WarmDefSOL* 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).

`EXIT -- 40  Terminated after numerical difficulties`

`INFO -- 41  current point cannot be improved`

`INFO -- 42  singular basis`

`INFO -- 43  cannot satisfy the general constraints`

`INFO -- 44  ill-conditioned null-space basis`

Several circumstances may lead to SNOPT not being able to improve on a non-optimal point.

1. Subroutines could be returning accurate function values but inaccurate gradients (or vice versa). This is the most likely cause. Study the comments given for `INFO 51` and `52`, and do your best to ensure that the coding is correct.

2. The function and gradient values could be consistent, but their precision could be too low. For example, accidental use of a `real` data type when `double precision` was intended would lead to a relative function precision of about 10^{-6} instead of something like 10^{-15}. The default `Major optimality tolerance` of 10^{-6} would need to be raised to about 10^{-3} for optimality to be declared (at a rather suboptimal point). Of course, it is better to revise the function coding to obtain as much precision as economically possible.

3. If function values are obtained from an expensive iterative process, they may be accurate to rather few

significant figures, and gradients will probably not be available. One should specify

`Function precision`	t
`Major optimality tolerance`	$\sqrt{t}$

but even then, if t is as large as 10^{-5} or 10^{-6} (only 5 or 6 significant figures), the same exit condition may occur. At present the only remedy is to increase the accuracy of the function calculation.

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 §8.4.4, 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 -- 51  incorrect objective derivatives`

`INFO -- 52  incorrect constraint derivatives`

This exit implies that there may be errors in the subroutines that define the problem objective and constraints. If the objective derivatives appear to incorrect, a check has been made on some individual elements of the objective gradient array at the first point that satisfies the linear constraints. At least one component (`G(`k`)` or `gObj(`j`)`) is being set to a value that disagrees markedly with its associated forward-difference estimate $\partial f_0/\partial x_j$. (The relative difference between the computed and estimated values is 1.0 or more.) This exit is a safeguard, since SNOPT will usually fail to make progress when the computed gradients are seriously inaccurate. In the process it may expend considerable effort before terminating with `INFO` 41 above.

Check the function and gradient computation *very carefully*. A simple omission (such as forgetting to divide f_0 by 2) could explain everything. If f_0 or a component $\partial f_0/\partial x_j$ is very large, then give serious thought to scaling the function or the nonlinear variables.

If you feel *certain* that the computed `gObj(`j`)` is correct (and that the forward-difference estimate is therefore wrong), you can specify `Verify level 0` to prevent individual elements from being checked. However, the optimization procedure may have difficulty.

If some constraint derivatives appear to be incorrect, then at least one of the computed constraint derivatives is significantly different from an estimate obtained by forward-differencing the vector $F(x)$. Follow the advice given above for the objective function, trying to ensure that the arrays `F` and `G` are being set correctly.

```
EXIT -- 60  Undefined user-supplied functions
INFO -- 61  undefined function at the first feasible point
INFO -- 62  undefined function at the initial point
INFO -- 63  unable to proceed into undefined region
```

```
EXIT -- 70  User requested termination
INFO -- 71  terminated during function evaluation
INFO -- 72  terminated during constraint evaluation
INFO -- 73  terminated during objective evaluation
INFO -- 74  terminated from monitor routine
```

These exits occur when `Status` < -1 is set during some call to the user-defined routines. SNOPT assumes that you want the problem to be abandoned forthwith.

> If the following exits occur during the *first* basis factorization, the primal and dual variables `x` and `pi` will have their original input values.

```
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
```

```
EXIT -- 140  System error
INFO -- 141  wrong number of basic variables
```

8.4.7 Solution output

At the end of a run, the final solution is 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 certain commercial systems, though there is no industry standard.

An example of the printed solution is given in §8.4. In general, numerical values are output with format `f16.5`. The maximum record length is 111 characters, including the first (carriage-control) character.

To reduce clutter, a dot "`.`" is printed for any numerical value that is exactly zero. The values ± 1 are also printed specially as `1.0` and `-1.0`. Infinite bounds ($\pm 10^{20}$ or larger) are printed as `None`.

Note: If two problems are the same except that one minimizes an objective $f_0(x)$ and the other maximizes $-f_0(x)$, their solutions will be the same but the signs of the dual variables π_i and the reduced gradients d_j will be reversed.

The ROWS section

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

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

and the value of $a^T x$ is called the *row activity*. Internally, the linear constraints take the form $Ax - s = 0$, where the slack variables s should satisfy the bounds $l \leq s \leq u$. For the ith "row", it is the slack variable s_i that is directly available, and it is sometimes convenient to refer to its state. Slacks may be basic or nonbasic (but not superbasic).

Nonlinear constraints $\alpha \leq f_i(x) + a^T x \leq \beta$ are treated similarly, except that the row activity and degree of infeasibility are computed directly from $f_i(x) + a^T x$ rather than s_i.

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 limits are the same ($\alpha = \beta$).
- `BS` The constraint is not binding. s_i is basic.

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 from its current value, there would be no change in the objective function. The values of the basic and superbasic variables *might* change, giving a genuine alternative solution. The values of the dual variables *might* also change.
- `D` *Degenerate.* The slack is basic, but it is equal to (or very close to) one of its bounds.

`I` *Infeasible.* The slack is basic and is currently violating one of its bounds by more than the `Feasibility tolerance`.

`N` *Not precisely optimal.* The slack is nonbasic. Its reduced gradient is larger than the `Major optimality tolerance` .

Note: If `Scale option > 0`, the tests for assigning `A`, `D`, `I`, `N` are made on the scaled problem because the keys are then more likely to be meaningful.

`Activity` The row value a^Tx (or $f_i(x) + a^Tx$ for nonlinear rows).

`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_j, $j = 1 : n$. We assume that a typical variable has bounds $\alpha \leq x_j \leq \beta$.

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 at some value strictly between its bounds: $\alpha < x_j < \beta$.

`BS` x_j is basic. Usually $\alpha < x_j < \beta$.

`SBS` x_j is superbasic. Usually $\alpha < x_j < \beta$.

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

`A` *Alternative optimum possible.* The variable is nonbasic, but its reduced gradient is essentially zero. This means that if x_j were allowed to start moving from its current value, there would be no change in the objective function. The values of the basic and superbasic variables *might* change, giving a genuine alternative solution. The values of the dual variables *might* also change.

`D` *Degenerate.* x_j is basic, but it is equal to (or very close to) one of its bounds.

`I` *Infeasible.* x_j is basic and is currently violating one of its bounds by more than the `Feasibility tolerance`.

`N` *Not precisely optimal.* x_j is nonbasic. Its reduced gradient is larger than the `Major optimality tolerance` .

Note: If `Scale option > 0`, the tests for assigning `A`, `D`, `I`, `N` are made on the scaled problem because the keys are then more likely to be meaningful.

`Activity` The value of the variable x_j.

`Obj Gradient` g_j, the jth component of the gradient of the (linear or nonlinear) objective function. (If any x_j is infeasible, g_j is the gradient of the sum of infeasibilities.)

`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 $m + j$.

8.4.8 The SUMMARY file

If `Summary file` > 0, the following information is output to the SUMMARY file. (It is a brief form of the PRINT file.) All output lines are less than 72 characters.

- The `Begin` line from the SPECS file, if any.
- The basis file loaded, if any.
- A brief Major iteration log.
- A brief Minor iteration log.
- The EXIT condition and a summary of the final solution.

The following SUMMARY file is from the example, using `Major print level 1` and `Minor print level 0`.

```
==============================
S N O P T  7.1-1(2) (Jun 2004)
==============================

SNSPEC EXIT 100 -- finished successfully
SNSPEC INFO 101 -- OPTIONS file read



Nonlinear constraints       2     Linear constraints       0
Nonlinear variables         2     Linear variables         0
Jacobian  variables         2     Objective variables      2
Total constraints           2     Total variables          2




This is problem  Toy1
The user has defined       6   out of       6   first  derivatives


Major Minors     Step   nCon Feasible  Optimal  MeritFunction    nS Penalty
    0      2              1  4.1E-01  5.0E-01  1.0000000E+00     2           r
    1      2  1.0E+00     2 (0.0E+00) 3.1E-01 -4.1421356E-01     1         n rl
    2      1  1.0E+00     3  1.4E+00  1.5E-01 -9.3802987E-01     1         s
    3      1  1.0E+00     4  1.8E-01  3.3E-02 -9.6547072E-01     1 2.8E-03
    4      1  1.0E+00     5  3.4E-02  8.9E-03 -9.9129962E-01       2.8E-03
    5      0  1.0E+00     6  4.2E-02  4.8E-03 -1.0000531E+00       2.8E-03
    6      0  1.0E+00     7  1.9E-04  2.0E-05 -9.9999997E-01       3.3E-02
    7      0  1.0E+00     8 (3.7E-09)(4.0E-10)-1.0000000E+00       3.3E-02

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

Problem name                 Toy1
No. of iterations                7   Objective value     -1.0000000008E+00
No. of major iterations          7   Linear objective     0.0000000000E+00
Penalty parameter          3.253E-02   Nonlinear objective -1.0000000008E+00
No. of calls to funobj           9   No. of calls to funcon              9
No. of degenerate steps          0   Percentage                       0.00
Max x                      2 1.0E+00   Max pi                     1 1.2E-01
Max Primal infeas          0 0.0E+00   Max Dual infeas            2 8.0E-10
Nonlinear constraint violn    6.4E-09



Solution printed on file  15

Finished problem Toy1

Time for MPS input                          0.00 seconds
Time for solving problem                    0.00 seconds
Time for solution output                    0.00 seconds
Time for constraint functions               0.00 seconds
Time for objective function                 0.00 seconds

snOptA finished.
INFO  = 1
nInf  = 0
sInf  =  0.
Obj   = -1.
```

References

[1] A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. *Math. Program.*, 16:84–117, 1982.

[2] Maintaining LU factors of a general sparse matrix. *Linear Algebra and its Applications*, 88, 1987.

[3] A practical anti-cycling procedure for linearly constrained optimization. *Math. Program.*, 45:437–474, 1989.

[4] MINOS 5.4 User's Guide. *Journal of Global Optimization*, 1995.

[5] R. H. Bartels. A stabilization of the simplex method. *Numerische Mathematik*, 16:414–434, 1971.

[6] R. H. Bartels and G. H. Golub. The simplex method of linear programming using the *lu* decomposition. *Communications of the ACM*, 12:266–268, 1969.

[7] A. R. Conn. Constrained optimization using a nondifferentiable penalty function. *SIAM J. Numer. Anal.*, 10:760–779, 1973.

[8] G. B. Dantzig. Linear programming and extensions. 1963.

[9] W. C. Davidon. Variable metric methods for minimization. *A.E.C. Research and Development Report ANL-599*, 1959.

[10] S. K. Eldersveld. Large-scale sequential quadratic programming algorithms. 1991.

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

[12] M. P. Friedlander. *A Globally Convergent Linearly Constrained Lagrangian Method for Nonlinear Optimization.* 2002.

[13] P. E. Gill and W. Murray. Numerically stable methods for quadratic programming. Mathematical Programming, 14:349–372, 1978.

[14] P. E. Gill and W. Murray. *Numerically stable methods for quadratic programming. Math. Program.*, 14:349–372, 1978.

[15] Philip E. Gill, Walter Murray, and Michael A. Saunders. User's guide for QPOPT 1.0: A Fortran package for Quadratic programming. Technical Report SOL 95-4, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305-4022, 1995.

[16] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for Large-Scale constrained programming. Technical Report SOL 97-3, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305-4022, 1997.

[17] Philip E. Gill, Walter Murray, and Michael A. Saunders. User's guide for SQOPT 5.3: A Fortran package for Large-Scale linear and quadratic programming. Technical Report Draft October 1997, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305-4022, 1997.

[18] Philip E. Gill, Walter Murray, and Michael A. Saunders. User's guide for SNOPT 5.3: A Fortran package for Large-Scale nonlinear programming. Technical Report SOL 98-1, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305-4022, 1998.

[19] Hans-Martin Gutmann. *Inertia-controlling methods for general quadratic programming. SIAM Rev.*, 33:1–36, 1991.

[20] 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*, 19:201–227, 1996.

[21] Jr. J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization*. 1983.

[22] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.

[23] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Math. Program.*, 14:41–72, 1978.

[24] B. A. Murtagh and M. A. Saunders. A projected lagrangian algorithm and its implementation for sparse nonlinear constraints. *Mathematical Programming Study*, 16:84–117, 1982.

[25] Bruce A. Murtagh and Michael A. Saunders. MINOS 5.5 USER'S GUIDE. Technical Report SOL 83-20R, Revised July 1998, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305-4022, 1998.

[26] M. A. Saunders P. E. Gill, W. Murray and M. H. Wright. Two step-length algorithms for numerical optimization. 1979.

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

[28] M. A. Saunders P. E. Gill, W. Murray and M. H. Wright. User's guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming. 1986.

[29] M. A. Saunders P. E. Gill, W. Murray and M. H. Wright. Maintaining *lu* factors of a general sparse matrix. *Linear Algebra and its Applications*, 88, 1987.

[30] M. A. Saunders P. E. Gill, W. Murray and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. Mathematical Programming, 45:437–474, 1989.

[31] M. A. Saunders P. E. Gill, W. Murray and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45:437–474, 1989.

[32] M. A. Saunders P. E. Gill, W. Murray and M. H. Wright. Inertia-controlling methods for general quadratic programming. SIAM Review, 33:1–36, 1991.

[33] W. Murray P. E. Gill and M. A. Saunders. User's guide for npopt: a fortran package for nonlinear programming.

[34] W. Murray P. E. Gill and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM J. Optim.*, 12:979–1006, 2002.

[35] W. Murray P. E. Gill and M. H. Wright. *Practical Optimization*. 1981.

[36] ed. P. M. Pardalos. Some theoretical properties of an augmented Lagrangian merit function. *Advances in Optimization and Parallel Computing, North Holland*, pages 101–128, 1992.

[37] P. M. Pardalos and G. Schnitger. Checking local optimality in constrained quadratic programming is NP-hard. Operations Research Letters, 7:33–35, 1988.

[38] R. H. Byrd R. Fletcher, P. T. Boggs and R. B. Schnabel. An ℓ_1 penalty method for nonlinear constraints. *Numerical Optimization 1984, eds., Philadelphia*, pages 26–40, 1985.

[39] J. K. Reid. Fortran subroutines for handling sparse linear programming bases. *Report R8269*, 1976.

[40] J. K. Reid. A sparsity-exploiting variant of the bartels-golub decomposition for linear programming bases. *Mathematical Programming*, 24:55–69, 1982.

[41] S. M. Robinson. A quadratically convergent algorithm for general nonlinear programming problems. *Mathematical Programming*, 3:145–156, 1972.

[42] P. Wolfe. The reduced-gradient method. 1962.