SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4023

MINOS 5.5 USER'S GUIDE

by

Bruce A. Murtagh* and Michael A. Saunders

TECHNICAL REPORT SOL 83-20R

December 1983
Revised Jan 1987, Mar 1993, Feb 1995, Jul 1998

*Graduate School of Management, Macquarie University, Sydney, NSW 2109, Australia.

Research and reproduction of this report were partially supported by National Science Foundation Grants DCR-8413211, ECS-8312142 and DDM-9204208; US Department of Energy Contract DE-AA03-76SF000326 PA# DE-AS03-76ER72018 and Grant DE-FG03-92ER25117; Office of Naval Research Grants N00014-85-K-0343 and N00014-90-J-1242; and US Army Research Office Contract DAAG29-84-K-0156.

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the above sponsors.

PREFACE

Since the middle of 1980, approximately 150 academic and research institutions around the world have installed MINOS/AUGMENTED, the predecessor of the present system. About 30 further installations exist in private industry. With enquiries continuing to arrive almost daily, the need for a combined linear and nonlinear programming system is apparent in both environments. To date, many users have been able to develop substantial nonlinear models and have come to be fairly confident that the *Optimal Solution* message actually means what it says. Certainly, other less joyful exit messages will often have greeted eager eyes. These serve to emphasize that model building remains an *art*, and that nonlinear programs can be *arbitrarily difficult to solve*. Nevertheless, the success rate has been high, and the positive response from users with diverse applications has inspired us to pursue further development.

MINOS 5.0 is the result of prolonged refinements to the same basic algorithms that were in MINOS/AUGMENTED:

- the simplex method (Dantzig, 1951, 1963),
- a quasi-Newton method (very many authors from Davidon, 1959, onward),
- the reduced-gradient method (Wolfe, 1962), and
- a projected Lagrangian method (Robinson, 1972; Rosen and Kreuser, 1972).

From numerous potential options, it has been possible to develop these particular algorithms into a relatively harmonious whole. The resulting system permits the solution of both small and large problems in the four main areas of *smooth* optimization:

- linear programming,
- unconstrained optimization,
- linearly constrained optimization, and
- nonlinearly constrained optimization.

In rare cases, the quasi-Newton method may require excessive storage. We have chosen not to provide a nonlinear conjugate-gradient method, or a truncated linear conjugate-gradient method, for this situation. Instead, we retain the quasi-Newton method throughout, restricting it to certain subspaces where necessary. (The strategy for altering the subspaces remains experimental.)

We regret that other obvious algorithms (such as integer programming, piece-wise smooth optimization, the dual simplex method) are still not available. Nor are ranging procedures or parametric algorithms. Sensitivity analysis is still confined to the usual interpretation of Lagrange multipliers.

As before, MINOS 5.0 is a stand-alone system that is intended for use alongside commercial mathematical programming systems whenever such facilities are available. The systems should complement each other.

To users of MINOS/AUGMENTED, the most apparent extensions are a scaling option (for linear constraints and variables only), and the ability to estimate some or all gradients numerically, if they are not computed by the user. On a more mundane level, the names of the user subroutines for computing nonlinearities have been changed from **CALCFG** and **CALCON** to **FUNOBJ** and **FUNCON**, and two new parameters allow access to the workspace used by MINOS.

Internally, one of the major improvements has been the development of a new basis-handling package, which forms the foundation of LUSOL (Gill, *et al.*, 1986), a set of routines for computing and updating a sparse LU factorization. This package draws much from the work of Reid (1976, 1982). It replaces the P^4-based procedures in MINOS/AUGMENTED (Saunders, 1976) and is

substantially more efficient on problems whose basis matrices are not close to triangular. As before, column updates are performed by the method of Bartels and Golub (1969, 1971), but the implementation is more efficient and there is no severe degradation arising from large numbers of "spikes". We venture to say that LUSOL is the first *truly stable basis package* that has been implemented for production use.

A further vital improvement has been the development of two new linesearch procedures (Gill, *et al.*, 1979) for finding a step length with and without the aid of derivatives. In particular they cater for function values that are somewhat "noisy"—a common practical circumstance.

From a software engineering viewpoint, the source code has been restructured to ease the problems of maintenance and future development. *MINOS* still stands for *Modular In-core Nonlinear Optimization System*, and we have done our best to respect the implications of the "M". Nevertheless, MINOS 5.0 remains a parameter-driven system. It is a speeding train on a railroad that has parallel tracks and many switches but few closed circuits. Its various modules *cannot* be called upon in an arbitrary order. In fact, there are *80 parameters* that can be set if necessary—these are the switching points along the railroad. Fortunately, only a handful need be set for any particular application. In most cases, the default values are appropriate for large and small problems alike.

For interactive users, a new feature is the SUMMARY file, which provides at the terminal a brief commentary on the progress of a run. Unfortunately, a two-way conversation is not possible. The only *input* engendered by this feature is an occasional dive for the Break key to abort an errant run. While rarely called upon, such a facility can be crucial to the security of one's computer funds.

Throughout the development of MINOS, we have received a great deal of assistance from many kind people. Most especially, our thanks go to Philip Gill, Walter Murray and Margaret Wright, whose knowledge and advice have made much of this work possible. They are largely responsible for the linesearch procedures noted above (which are as vital to nonlinear optimization as basis factors are to linear programming), and they are authorities on all of the algorithms employed within MINOS. Their patience has been called upon continually as other important work at SOL either languished or fell unfairly on their shoulders.

Further to basis factors, we acknowledge the pioneering work of John Reid in implementing the Markowitz-based *LU* factorization and the Bartels-Golub update. The LUSOL procedures in MINOS 5.0 owe much to the ingenuity embodied in his LA05 package.

Users have naturally provided an essential guiding influence. In some cases they are algorithm developers themselves. At home, we have had constant encouragement from George Dantzig and the benefit of his modeling activity within SOL, notably on the energy-economic model PILOT. We thank him warmly for bringing the Systems Optimization Laboratory into existence. We also thank Patrick McAllister, John Stone and Wesley Winkler for the feedback they have provided by running various versions of MINOS during their work on PILOT. (We note that PILOT has grown to 1500 constraints and 4000 variables, and now has a quadratic objective. From our perspective, it is a nontrivial test problem!) Likewise, Alan Manne has provided encouragement and assistance from the beginning. Two of his nonlinear economic models have been invaluable as test problems (and are included on the MINOS distribution tape). We also thank him and Paul Preckel for the development of procedures for solving sequences of related problems (Preckel, 1980). The main ingredients of these procedures are now an integral part of MINOS.

From industry, we have received immense benefit from the working relationship between SOL and Robert Burchett of the General Electric Company (Electric Utility Systems Engineering Department) in Schenectady, New York. Many algorithmic and user-oriented details have resulted

from his experience and from his interest in the fine points of optimization. Three years ago we did not envisage that problems involving thousands of nonlinear constraints would soon be solved successfully. Rob constantly pushed test versions of MINOS to their limits, and inspired the development of techniques to extend those limits. We thank him for his tireless contributions.

We are also grateful to Zenon Fortuna, Steven Gorelick, Marc Hellman, Thomas McCormick, Larry Nazareth, Scott Rogers, John Rowse and John Tomlin for their helpful suggestions and/or assistance in tracking down bugs. Finally, we thank the staff of the Office of Technology Licensing and the Information Technology Services at Stanford University for undertaking the task of distributing MINOS.

Most of the software development was carried out at the Stanford Linear Accelerator Center with the aid of the Wylbur text editor and the University of Waterloo's WATFIV compiler. This User's Guide was typeset using TEX*, with editorial assistance from Philip Gill and Margaret Wright.

Bruce Murtagh
University of New South Wales

Michael Saunders
Stanford University

December, *1983*

Preface to MINOS 5.5

This manual is a revision of the 1983 MINOS 5.0 User's Guide. The main changes implemented in MINOS 5.5 are summarized in the Appendices. A significant change is that MINOS is now callable as a subroutine.

Certain parts of this Guide are no longer relevant, but Chapter 7, for example, still conveys the main implementation philosophy. For exact details, please see `minos.doc` in the distribution files.

MINOS is licensed by Stanford University. Fortran 77 source code for all common machines (mainframes, workstations and PCs) is available from SBSI:

Stanford Business Software, Inc.	sales@SBSI-SOL-Optimize.com
2672 Bayshore Parkway, *Suite 304*	*(650) 962-8719*
Mountain View, CA 94043	(650) 962-1869 Fax
Contact: Ms Radhika Thapa	Manager, Software Distribution

SBSI handles individual and site licenses, both non-profit and commercial. It should be noted that SBSI is separate and independent from Stanford University.

*D. E. Knuth, *TEX and METAFONT, New Directions in Typesetting*, American Mathematical Society and Digital Press, Bedford, Massachusetts (1979).

Special commercial licenses, such as those involving the re-sale of MINOS as part of a larger package, are negotiated by

Hans Wiesendanger	Hans@OTLmail.stanford.edu
Office of Technology Licensing	http://www.stanford.edu/group/OTL/
Stanford University	(650) 723-0651
900 Welch Road, Suite 350	(650) 725-7295 Fax
Palo Alto, CA 94304-1850	

For many applications involving linear and nonlinear models, we recommend the use of algebraic modeling languages. Two of the most widely used systems are GAMS and AMPL. They provide a convenient interface to MINOS and to several other linear, integer and nonlinear programming systems (notably CPLEX, OSL and CONOPT). Implementations are available for PCs, workstations and mainframes.

GAMS Development Corporation	http://www.gams.com/
1217 Potomac Street NW	sales@gams.com
Washington, DC 20007	(202) 342-0180
AMPL development	http://www.ampl.com/
	info@ampl.com
AMPL sales and AMPL Plus	http://www.modeling.com
	info@modeling.com

CONTENTS

1. INTRODUCTION

MINOS is a Fortran-based computer system designed to solve large-scale optimization problems expressed in the following standard form:

$$\underset{x,y}{\text{minimize}} \quad F(x) + c^T x + d^T y \tag{1}$$

$$\text{subject to} \quad f(x) + A_1 y = b_1, \tag{2}$$

$$A_2 x + A_3 y = b_2, \tag{3}$$

$$l \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u, \tag{4}$$

where the vectors c, d, b_1, b_2, l, u and the matrices A_1, A_2, A_3 are constant, $F(x)$ is a smooth scalar function, and $f(x)$ is a vector of smooth functions $\{f^i(x)\}$. Ideally the first derivatives (gradients) of $F(x)$ and $f^i(x)$ should be known and coded by the user. (If only *some* gradients are known, MINOS will estimate the missing ones using finite differences.)

The n_1 components of x are called the *nonlinear variables*, and the n_2 components of y are the *linear variables*. Similarly, the m_1 equations (2) are called the *nonlinear constraints*, and the m_2 equations (3) are the *linear constraints*. Equations (2) and (3) together are called the *general constraints*. We define $m = m_1 + m_2$ and $n = n_1 + n_2$.

The constraints (4) specify *upper and lower bounds* on all variables. These are fundamental to many problem formulations and are treated specially by the solution algorithms in MINOS. Some of the components of l and u may be $-\infty$ or $+\infty$ if desired.

Similar bounds may be defined for the general constraints (2), (3). These constraints may therefore be thought of as taking the form

$$l_1 \leq f(x) + A_1 y \leq u_1,$$

$$l_2 \leq A_2 x + A_3 y \leq u_2,$$

though for historical reasons the bounds are specified in terms of a *right-hand side* b_i and a *range* $u_i - l_i$.

In the following sections we introduce some of the terminology required, and give an overview of the algorithms used in MINOS and the main system features.

1.1 Linear Programming

If the functions $F(x)$ and $f(x)$ are absent, the problem becomes a *linear program*. Since there is no need to distinguish between linear and nonlinear variables, we prefer to use x rather than y. It is also convenient computationally to convert all general constraints into equalities, with the only inequalities being simple bounds on the variables. Thus, we will write linear programs in the form

$$\underset{x,s}{\text{minimize}} \; c^T x \quad \text{subject to} \quad Ax + Is = 0, \qquad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u,$$

where the elements of x are called *structural variables* (or *column variables*) and s is a set of *slack variables* (called *logical variables* by some authors). The bounds l and u are suitably redefined.

MINOS solves linear programs using a reliable implementation of the *primal simplex method* (Dantzig, 1963). The simplex method partitions the constraints $Ax + Is = 0$ into the form

$$Bx_B + Nx_N = 0,$$

where the *basis matrix* B is square and nonsingular. The elements of x_B and x_N are called the *basic* and *nonbasic variables* respectively; they are a permutation of the elements of x and s. At any given stage, each nonbasic variable is equal to its upper or lower bound, and the basic variables take on whatever values are needed to satisfy the general constraints. (Clearly they may be computed by solving the linear equation $Bx_B = -Nx_N$.) It can be shown that if an optimal solution to a linear program exists, then it has this form. The simplex method reaches such a solution by performing a sequence of *iterations*, in which one column of B is replaced by one column of N (and *vice versa*), until no such interchange can be found that will reduce the value of c^Tx.

If the components of x_B do not satisfy their upper and lower bounds, we say that the current point is *infeasible*. In this case, the simplex method first aims to reduce the sum of infeasibilities to zero.

MINOS maintains a sparse LU factorization of the basis matrix B, using a Markowitz ordering scheme and Bartels-Golub updates, as implemented in the LUSOL package of Gill, Murray, Saunders and Wright (1986). (For a description of the concepts involved, see Reid, 1976, 1982.) The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.

1.2 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 algorithm* (Wolfe, 1962) in conjunction with a *quasi-Newton algorithm* (Davidon, 1959). The implementation follows that described in Murtagh and Saunders (1978).

In this case, the constraints $Ax + Is = 0$ are partitioned into the form

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

where x_S is a set of *superbasic variables*. At a solution, the basic and superbasic variables will lie somewhere between their bounds, while the nonbasic variables will again be equal to one of their bounds. In broad terms, the number of superbasic variables (the number of columns in S) is a measure of *how nonlinear* the problem is. Let this number be s. (The context will always distinguish s from the vector of slack variables.) In many practical cases we have found that s remains reasonably small, say 200 or less, regardless of the size of the problem.

In the reduced-gradient algorithm, x_S is regarded as a set of independent variables that are free to move in any desirable direction, namely one that will improve the value of the objective function (or reduce the sum of infeasibilities). The basic variables can then be 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, some of the nonbasic variables are selected to be added to S, and the process is repeated with an increased value of s. At all stages, if a basic or superbasic variable encounters one of its bounds, that variable is made nonbasic and the value of s is reduced by one.

Users familiar with linear programs may interpret the simplex method as being exactly the above process, with s oscillating between 0 and 1. (Later, one step of the simplex method *or* the reduced-gradient method will be called a *minor iteration*.)

A certain operator Z will frequently be useful for descriptive purposes. In the reduced-gradient algorithm it takes the form

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

though it is never computed explicitly. Since it has full column rank and satisfies $(B\ S\ N)Z = 0$, we say that Z *spans the null space* of the constraint matrix $(A\ \ I)$. Given an LU factorization of the basis matrix B, Z allows us to work within a region defined by the linear constraints.

An important part of MINOS is a stable implementation of the quasi-Newton algorithm for optimizing the superbasic variables. This can achieve superlinear convergence within each relevant subspace (defined by the current B, S and N). It obtains a *search direction* p_s for the superbasic variables by solving a system of the form

$$R^T R p_s = -Z^T g,$$

where g is the gradient of $F(x)$, $Z^T g$ is the *reduced gradient*, and R is a dense upper triangular matrix that is updated in various ways in order to approximate the *reduced Hessian* according to $R^T R \approx Z^T H Z$, where H is the matrix of second derivatives of $F(x)$ (i.e., the *Hessian*).

Once p_s is available, the search direction for all variables is defined by $p = Zp_s$. A *line search* is then performed to find an approximate solution to the one-dimensional problem

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

where α_{max} is determined by the bounds on the variables. Another important part of MINOS is the *step-length procedure* used in the line search to determine the step-length α. Two different procedures are used, depending on whether all gradients are known. (See Gill, Murray, Saunders and Wright, 1979.) Interested users can influence the amount of work involved by setting a parameter called the **LINESEARCH TOLERANCE**.

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 may be specified by the user). MINOS ensures that the bounds on the variables 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, or x_0 should be re-specified.

For details of the matters mentioned here and many other essential aspects of numerical optimization, see Gill, Murray and Wright (1981).

1.3 Nonlinear Constraints

When the problem contains nonlinear constraints, MINOS uses a *projected augmented Lagrangian algorithm*, based on a method due to Robinson (1972); see Murtagh and Saunders (1982). MINOS treats linear constraints and bounds specially, but the nonlinear constraints may not be satisfied until an optimal point is reached. Thus, $f(x)$ and its gradients (the *Jacobian matrix* $J(x) = [\partial f'(x)/\partial x_j]$) may need to be defined outside the region of interest.

In fact, the constraint functions will *almost never* be evaluated unless the linear constraints are satisfied. Again, the starting point is an exception; it will satisfy its bounds, but $f(x)$ and $J(x)$ will be evaluated at x_0 *regardless of the general linear and nonlinear constraints.* This matter must be borne in mind during the formulation of a nonlinear program.

The nature of the solution process can be summarized as follows. A sequence of *major* iterations is performed, each one requiring the solution of a *linearly constrained subproblem*. The subproblems contain the original linear constraints and bounds, as well as linearized versions of the nonlinear constraints. This just means that $f(x)$ in equation (2) is replaced by Lf, its linear approximation at the current point. We shall write this approximation as

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

or more briefly

$$\tilde{f} = f_k + J_k(x - x_k), \tag{5}$$

where x_k is the estimate of the nonlinear variables at the start of the k-th major iteration. The subproblem to be solved takes the form

$$\underset{x,y}{\text{minimize}} \quad F(x) + c^T x + d^T y - \lambda_k^T(f - \tilde{f}) + \frac{1}{2}\rho(f - \tilde{f})^T(f - \tilde{f}) \tag{6}$$

$$\text{subject to} \quad \tilde{f} + A_1 y = b_1, \tag{7}$$

$$A_2 x + A_3 y = b_2, \tag{8}$$

$$l \le \begin{pmatrix} x \\ y \end{pmatrix} \le u. \tag{9}$$

The objective function (6) is called an *augmented Lagrangian*. The vector λ_k is an estimate of λ, the *Lagrange multipliers* for the nonlinear constraints. The scalar ρ is a *penalty parameter*, and the term involving ρ is a modified *quadratic penalty function*.

Using (5) we see that the linear constraints (7) and (8) take the form

$$\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} J_k x_k - f_k \\ 0 \end{pmatrix}. \tag{10}$$

MINOS uses the reduced-gradient algorithm to minimize (6) subject to (10), with the original bounds on x and y, and suitable bounds on the slack variables s_1 and s_2. The Jacobian J_k is treated as a sparse matrix, the same as the matrices A_i.

Unfortunately, there is no guarantee that the algorithm just described will converge from an arbitrary starting point. The concerned user can influence the *likelihood of convergence* in several ways:

1. By specifying x_0 as carefully as possible.
2. By including sensible upper and lower bounds on all variables.
3. By specifying a **PENALTY PARAMETER** ρ that is higher than the default value, if the problem is suspected of being highly nonlinear.
4. By specifying a **DAMPING PARAMETER** that is lower than the default value, again if the problem is highly nonlinear.

In rare cases it may be safe to use $\lambda_k = 0$ and $\rho = 0$ for all subproblems, by specifying **LAGRANGIAN = NO**. However, convergence is much more likely with the default setting, **LAGRANGIAN = YES**. The initial estimate of the Lagrange multipliers is then $\lambda_0 = 0$, but for later subproblems, λ_k is taken to be the Lagrange multipliers associated with the (linearized) nonlinear constraints at the end of the previous major iteration.

The penalty parameter is initially $100.0/m_1$ by default, and it is reduced in stages for later subproblems when it appears that the sequence $\{x_k, \lambda_k\}$ is converging. In many cases it is safe to specify $\rho = 0$ from the beginning, particularly if the problem is only mildly nonlinear. This may improve the overall efficiency.

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 re-defining 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 important not to move nonlinearities from the objective function into the constraints. Thus, we would *not* replace minimize $F(x)$ by

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

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 an option to scale constraints and variables automatically.

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 problem functions. For safety when such singularities exist, the initial point x_0 discussed above should lie within the bounds.

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). The functions need not be separable. Integer restrictions cannot be imposed directly.

A certain region is defined by the linear constraints in a problem and by the bounds on the variables. If the nonlinear objective and constraint functions are convex within this region, 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.

MINOS uses one large array of main storage for most of its workspace. The length of this array may need to be adjusted to suit a particular problem, but otherwise the implementation places no fixed limitation on the size of a problem or on its shape (many constraints and relatively few variables, or *vice versa*). In general, the limiting factor will be the amount of main storage

available on a particular machine, and the amount of computation time that one's budget can stand.

Some *a priori* knowledge of a particular application will usually indicate whether the solution procedure is likely to be efficient. An important quantity is $m = m_1 + m_2$, the total number of general constraints in (2) and (3). We note that $m \leq 100$ is considered "small", $m = 1000$ or 2000 is "medium", and $m \geq 5000$ would be "large". On machines that use 16-bit integers (INTEGER*2 on IBM and DEC VAX systems), the normal implementation of MINOS requires that $m \leq 32767$.

The amount of workspace required by MINOS is roughly $100m$ words, where one "word" is the relevant storage unit for the floating-point arithmetic being used (REAL*8 on IBM and DEC VAX, REAL on Burroughs and most CDC machines). On IBM and VAX systems, this means about $800m$ bytes for workspace. A further 300K bytes, approximately, are needed for the program itself, along with buffer space for several files.

Another important quantity is $n = n_1 + n_2$, the total number of variables in x and y. For nonlinear problems, if m_1 and n_1 are small compared to m and n, the total storage required should not be much greater than just described. If n_1 is "large" (say $n_1 \geq 200$), the amount of storage required may or may not be substantial, depending on whether $F(x)$ or $f(x)$ are highly nonlinear or not.

In this context, the efficiency of MINOS depends on s, the number of superbasic variables. Recall that $m + s$ variables lie between their upper and lower bounds, where s is zero for purely linear problems. We know that s need never be larger than $n_1 + 1$. In practice, s is often very much less than this upper limit.

In the quasi-Newton algorithm, the dense triangular matrix R has dimension s and requires about $\frac{1}{2}s^2$ words of storage. If it seems likely that s will be very large, some aggregation or reformulation of the problem should be considered.

1.6 Files

MINOS operates primarily within central memory, and is well suited to a virtual storage environment. Certain disk files are accessed as follows.

Input file	*Status*	*Record Length* (characters)
READ file	see below	
SPECS file	required	80
MPS file	required	81
BASIS files	optional	80
Output file	*Status*	*Record Length* (characters)
PRINT file	required	129
SUMMARY file	optional	80
BASIS files	optional	80
SOLUTION file	optional	111

Fixed-length, blocked records may be used in all cases, and the files are always accessed sequentially. The logical record length must be at least that shown. For efficiency, the physical block size should be several hundred characters in most cases.

Unit numbers for the READ, SPECS, PRINT files are defined at compile time; typically they will be 5, 5, 6 , but they may depend on the installation. The remaining unit numbers are specified at run time in the SPECS file.

Unit numbers for the READ, PRINT and SUMMARY files are stored in the following COMMON block:

```
COMMON    /M1FILE/ IREAD,IPRINT,ISUMM
```

It may be convenient to reference these in the user subroutines FUNOBJ, FUNCON and MATMOD.

System Note: The READ file is not used explicitly by MINOS, but its unit number is used to test if a file should be rewound. (Thus, input files are subject to a Fortran REWIND as long as they are not the same as the READ file.) The PRINT file is used frequently. Other output files are rewound if they are not the same as the PRINT file.

1.7 Input Data Flow

Some or all of the following items are supplied by the user:

- Subroutine FUNOBJ
- Subroutine FUNCON
- Subroutine MATMOD
- A SPECS file
- An MPS file
- A BASIS file
- Data read by FUNCON on its first entry
- Data read by FUNOBJ on its first entry
- Data read by FUNCON on its last entry
- Data read by FUNOBJ on its last entry

The order of the files and data is important if all are stored in the same input stream.

Subroutines FUNOBJ and FUNCON define the nonlinear objective and constraint functions respectively (if any); they are not needed if the functions are purely linear and are defined in the MPS file.

Subroutine MATMOD is occasionally needed, for applications involving a sequence of closely related problems.

The SPECS file defines various run-time parameters (ITERATION LIMIT, SAVE FREQUENCY, etc.). Its file number is defined at compile time. It will normally be the first data set in the system card input stream.

The MPS file specifies names for the constraints and variables, and defines all the linear constraints and bounds. It may follow the SPECS file in the card input stream, but will often reside in a file of its own (as specified in the SPECS file). The data format is similar to that used in commercial mathematical programming systems (hence the name). The format has been generalized slightly for nonlinear problems.

If desired, a BASIS file may be loaded at the beginning of a run. This will normally have been saved at the end of an earlier run. Three kinds of basis file are available; they are used to restart the solution of a problem that was interrupted, or to provide a good starting point for some slightly modified problem.

1.8 Multiple SPECS Files

One or more problems may be processed during a run. The parameters for a particular problem are delimited by BEGIN and END in the SPECS file. While scanning for the keyword BEGIN, MINOS recognizes the keywords SKIP and ENDRUN. Thus in the following example:

```
BEGIN CASE 1
  .
  .
END CASE 1
SKIP CASE 2
  .
  .
END CASE 2
BEGIN CASE 3
  .
  .
END CASE 3
ENDRUN
BEGIN CASE 4
  .
  .
END CASE 4
```

only the first and third problem will be processed.

1.9 Internal Modifications

A sequence of closely related problems may be specified within a single SPECS file, via the CYCLE parameter; for example,

```
BEGIN CYCLING EXAMPLE
  .
CYCLE LIMIT        10
  .
END EXAMPLE
```

indicates that up to 10 problems are to be processed. This is intended for cases where the solution of one problem P_k is needed to *define* the next problem P_{k+1}.

The actual method for defining the next problem in a cycle depends on the application. Sometimes it can be done by changing the output from the function subroutines FUNOBJ and/or FUNCON. Alternatively, the user may provide a third subroutine MATMOD to perform some modifications to the problem data. MATMOD is called by MINOS at the beginning of every cycle.

If necessary, the number of linear variables can be *increased* when a problem P_{k+1} is defined. We think of this as *adding new columns to* P_k. The new columns are not included in the MPS file, and their sparsity pattern need not be known until P_k has been solved. Instead, an appropriate number of PHANTOM COLUMNS and PHANTOM ELEMENTS are defined in the SPECS file (to reserve a pool of storage), and the user's subroutine MATMOD generates each new column by calling the MINOS subroutine MATCOL.

2. USER-WRITTEN SUBROUTINES

To solve a purely linear problem, only a SPECS file and an MPS file (and possibly a BASIS file) need be supplied.

For nonlinear problems, one must also provide some appropriate Fortran code. Nonlinearities in the objective function are defined by subroutine FUNOBJ. Those in the constraints are defined separately by subroutine FUNCON. *On every entry* except perhaps the last, these subroutines must return appropriate function values F. Wherever possible, they should also return *all* gradient *components* in the array G. This provides maximum reliability and corresponds to the default setting, DERIVATIVE LEVEL = 3.

In practice it is often convenient not to code gradients. MINOS is able to estimate gradients by finite differences, by making a call to FUNOBJ or FUNCON for each variable x_j whose partial derivatives need to be estimated. *However*, this reduces the reliability of the optimization algorithms, and it can be very expensive if there are many such variables x_j.

As a compromise, MINOS allows you to code as many gradients as you like. This option is implemented as follows: just before a function routine is called, each element of the gradient array G is initialized to a specific value. On exit, any element retaining that value must be estimated by finite differences.

Some rules of thumb follow:

1. For maximum simplicity and reliability, compute F and all components of G.
2. If not all gradients are known, compute as many of them as you can. (It often happens that some of them are constant or even zero.)
3. If *some* gradients are known (but not all), it may be convenient to compute them each time the function routines are called, even though they will be ignored if MODE = 0.
4. If the known gradients are *expensive* to compute, use the parameter MODE to avoid computing them on certain entries.
5. While the function routines are being developed, use the VERIFY parameter to check the computation of any gradient elements that are supposedly known.

2.1 Subroutine FUNOBJ

This subroutine is provided by the user to calculate the objective function $F(x)$ and as much of its gradient $g(x)$ as possible. (It is not needed if the objective function is entirely linear.)

Specification:

```
SUBROUTINE FUNOBJ( MODE, N, X, F, G, NSTATE, NPROB, Z, NWCORE )
IMPLICIT           REAL*8(A-H,O-Z)
DIMENSION          X(N), G(N), Z(NWCORE)
```

(The IMPLICIT statement should not be used on machines for which single-precision floating-point is adequate; e.g., Burroughs and CDC.)

Parameters:

MODE (Input) This parameter can be ignored if DERIVATIVE LEVEL = 1 or 3 (i.e., if all elements of G are computed). In this case, MODE will always have the value 2.

Otherwise, you must specify DERIVATIVE LEVEL = 0 or 2 in the SPECS file to indicate that FUNOBJ will not compute all of G. MINOS will then call FUNOBJ sometimes with MODE = 2 and sometimes with MODE = 0. You may test MODE to decide what to do:

If MODE = 2, compute F and as many components of G as possible.

If MODE = 0, compute F but set G only if you wish. (On return, the contents of G will be ignored.)

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

N (Input) The number of variables involved in $F(x)$. These must be the first N variables in the problem.

X(*) (Input) An array of dimension N containing the current values of the nonlinear variables x.

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

G(*) (Output) The computed gradient vector $g(x)$. In general, G(j) should be set to the partial derivative $\partial F/\partial x_j$ for as many j as possible (except perhaps if MODE = 0—see above).

NSTATE (Input) If NSTATE = 0, there is nothing special about the current call to FUNOBJ.

If NSTATE = 1, MINOS is calling your subroutine for the first time. Some data may need to be input or computed and saved in local or COMMON storage. Note that if there are nonlinear constraints, the first call to FUNCON will occur *before* the first call to FUNOBJ.

If NSTATE ≥ 2, MINOS is calling your subroutine for the last time. You may wish to perform some additional computation on the final solution. (If CYCLE LIMIT is specified, this call occurs at the end of each cycle.) Note again that if there are nonlinear constraints, the last call to FUNCON will occur *before* the last call to FUNOBJ.

In general, the last call is made with NSTATE = 2 + IERR, where IERR indicates the status of the final solution. In particular, if NSTATE = 2, the current X is *optimal*; if NSTATE = 3, the problem appears to be infeasible; if NSTATE = 4, the problem appears to be unbounded; and if NSTATE = 5, the iterations limit was reached. In some cases, the solution may be *nearly* optimal if NSTATE = 11; this value occurs if the linesearch procedure was unable to find an improved point.

If the nonlinear functions are expensive to evaluate, it may be desirable to do nothing on the last call, by including a statement of the form `IF (NSTATE .GE. 2) RETURN` at the start of the subroutine.

NPROB (Input) An integer that can be set by a card of the form PROBLEM NUMBER n in the SPECS file.

Z(*) (Input) The primary work array used by MINOS. In certain applications it may be desirable to access parts of this array, using various COMMON blocks to pinpoint the required locations. (For example, the dual variables are stored in Z(LPI) onward, where LPI is the first integer in the COMMON block M5LOC.) Otherwise, Z and NWCORE can be ignored.

NWCORE (Input) The dimension of Z.

2.2 Subroutine FUNCON

This subroutine is provided by the user to compute the nonlinear constraint functions $f(x)$ and as many of their gradients as possible. (It is not needed if the constraints are entirely linear.) *Note that the gradients of the vector $f(x)$ define the Jacobian matrix $J(x)$. The j-th column of $J(x)$* is the vector $\partial f/\partial x_j$.

FUNCON may be coded in two different ways, depending on the method used for storing the Jacobian, as specified in the SPECS file.

JACOBIAN = DENSE

Specification:

```
SUBROUTINE FUNCON( MODE, M, N, NJAC, X, F, G, NSTATE, NPROB, Z, NWCORE )
IMPLICIT           REAL*8(A-H,O-Z)
DIMENSION          X(N), F(M), G(M,N), Z(NWCORE)
```

Parameters:

MODE (Input) This parameter can be ignored if DERIVATIVE LEVEL = 2 or 3 (i.e., if all elements of G are computed). In this case, MODE will always have the value 2.

Otherwise, you must specify DERIVATIVE LEVEL = 0 or 1 in the SPECS file to indicate that FUNCON will not compute all of G. You may then test MODE to decide what to do:

If MODE = 2, compute F and as many components of G as possible.

If MODE = 0, compute F but set G only if you wish. (On return, the contents of G will be ignored.)

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

M (Input) The number of nonlinear constraints (not counting the objective function). These must be the first M constraints in the problem.

N (Input) The number of variables involved in $f(x)$. These must be the first N variables in the problem.

NJAC (Input) The value M*N.

X(*) (Input) An array of dimension N containing the current values of the nonlinear variables x.

F(*) (Output) The computed values of the functions in the constraint vector $f(x)$.

G(*,*) (Output) The computed Jacobian matrix $J(x)$. The j-th column of $J(x)$ should be stored in the j-th column of the 2-dimensional array G (except perhaps if MODE = 0—see above). Equivalently, the gradient of the i-th constraint should be stored in the i-th row of G.

The other parameters are the same as for subroutine FUNOBJ.

JACOBIAN = SPARSE

Specification:

```
SUBROUTINE FUNCON( MODE, M, N, NJAC, X, F, G, NSTATE, NPROB, Z, NWCORE )
IMPLICIT           REAL*8(A-H,O-Z)
DIMENSION          X(N), F(M), G(NJAC), Z(NWCORE)
```

This is the same as for JACOBIAN = DENSE, except for the declaration of G(NJAC).

Parameters:

NJAC (Input) The number of nonzero elements in the Jacobian matrix $J(x)$. This is exactly the number of entries in the MPS file that referred to nonlinear rows and nonlinear Jacobian columns (the first M rows in the ROWS section and the first N columns in the COLUMNS section).

Usually NJAC will be less than M*N. The actual value of NJAC may not be of any use when coding FUNCON, but in all cases, any expression involving G(*l*) should have the subscript *l* between 1 and NJAC.

G(*) (Output) The computed elements of the Jacobian matrix (except perhaps if MODE = 0—see previous page). These elements must be stored into G in exactly the same positions as implied by the MPS file. There is no internal check for consistency (except indirectly via the VERIFY parameter), so great care is essential.

The other parameters are the same as for JACOBIAN = DENSE.

2.3 Constant Jacobian Elements

If all constraint gradients (Jacobian elements) are known (DERIVATIVE LEVEL = 2 or 3), any *constant* elements may be specified in the MPS file if desired. An element of G that is not computed in FUNCON will retain the value implied by the MPS file. (The value is taken to be *zero* if not given explicitly in the MPS file.)

This feature is useful when JACOBIAN = DENSE and many Jacobian elements are identically zero. Such elements need not be specified in the MPS file, nor set in FUNCON.

Note that constant *nonzero* elements do affect F. Thus, if J_{ij} is defined in the MPS file and is constant, the array element G(i,j) need not be set in FUNCON, but the value G(i,j)*X(j) must be added to F(i).

When JACOBIAN = SPARSE, constant Jacobian elements will normally not be listed in the MPS file unless they are nonzero. If the correct value *is* entered in the MPS file, the corresponding element G(*l*) need not be reassigned, but a term of the form G(*l*)*X(j) must be added to one of the elements of F. (This feature allows a matrix generator to output constant data to the MPS file; FUNCON *does not need to know* that data at compile time, but can use it at run time to compute the elements of F.)

Remember, if DERIVATIVE LEVEL $<$ 2, unassigned elements of G are *not* treated as constant; they are estimated by finite differences, at significant expense.

2.4 SUBROUTINE MATMOD

For stand-alone MINOS, **matmod** allows you to define a sequence of related problems and have them solved one by one. It is used in conjunction with the **Cycle** and **Phantom** options. If the **Cycle limit** = 1 (the default), **matmod** is never called. If **Cycle limit** > 1, **matmod** is called *before* the original problem is solved (cycle 0), and also after each problem is solved (cycle 1, 2, 3, ...).

Within **matmod** you might alter some bounds on the variables or revise some of the constraint coefficients. You may also communicate with subroutines **funobj** and **funcon** to alter their behavior (e.g., by setting variables in your own **common** blocks). Finally, **matmod** may specify whether a Cold, Warm or Hot start should be used when MINOS starts solving the new problem.

Specification:

```
      subroutine matmod( ncycle, nprob, finish,
     $                   m, n, nb, ne, nka, ns, nscl, nname,
     $                   a, ha, ka, bl, bu,
     $                   ascale, hs, name1, name2,
     $                   x, pi, rc, z, nwcore )

      implicit           double precision (a-h,o-z)
      integer            ncycle, nprob,
     $                   m, n, nb, ne, nka, ns, nscl, nname, nwcore
      logical            finish
      integer*4          ha(ne), hs(nb)
      integer            ka(nka), name1(nname), name2(nname)
      double precision   a(ne), ascale(nscl), bl(nb), bu(nb),
     $                   x(nb), pi(m), rc(nb), z(nwcore)
```

On entry:

ncycle says how many problems have been solved.

If **ncycle** = 0, **matmod** is being called for the first time. MINOS has read the MPS file, but the problem has not yet been scaled or solved. If a BASIS file *was specified, it has been read* and **hs** is defined. Otherwise, Crash has not yet been called and **hs** does not define a basis.

This entry allows **matmod** to initialize problem-dependent quantities. To do nothing before the first problem is solved, put "**if (ncycle .eq. 0) return**" at the beginning of **matmod**.

nprob is the **Problem number** specified in the SPECS file.

finish is **.false.**.

m, n, nb, ne are the problem dimensions m, n, $nb = n + m$, ne (see Appendix B).

nka is $n + 1$ (used to dimension ka).

ns is the number of superbasic variables.

nscl says if the problems are being scaled prior to each solve. If nscl = 1, scaling has not been specified. Otherwise, nscl = nb and ascale contains the scales used for the problem just finished (assuming ncycle > 0). However, the problem itself has been unscaled.

nname is normally the same as nb, assuming MINOS read an MPS file. If matmod is for some reason being used with minoss, nname is the same as the minoss parameter: it may be nb or 1, depending on whether names exist.

a(*), ha(*), ka(*) contain the constraint matrix (see Appendix B).

bl(*), bu(*) are the lower and upper bounds on all column and slack variables (x, s).

ascale(*) contains scale factors for columns and rows (if ncycle > 0 and nscl > 1).

hs(*) is the state vector for all variables.. See Appendix B.

name1(*), name2(*) contain the first and second halves of the names of the columns and rows in a4 format. For example, if the 9th variable were named 'Capital ', we would have name1(9) = 'Capi' and name2(9) = 'tal '.

x(*) contains (unscaled) values for all variables (x, s).

pi(*) contains the values of the dual variables π. The first m_1 components are current estimates of λ, the Lagrange multipliers for the nonlinear constraints. Good values for λ can sometimes assist convergence of the projected Lagrangian algorithm. They may be provided to MINOS by the MPS file, but it may be more convenient to define them in matmod on the first entry (ncycle = 0).

rc(*) contains reduced costs for the variables and slacks (x, s), as printed in the COLUMNS and ROWS sections of the solution.

z(nwcore) is the primary work array used by MINOS. As in funobj or funcon, it may be desirable to access parts of this array via common blocks.

On exit:

Set finish = .true. if you wish the cycles to be terminated; e.g., if some convergence criterion has been satisfied. The following common blocks may be useful:

```
double precision   cnvtol
common    /cyclcm/ cnvtol,jnew,materr,maxcy,nephnt,nphant,nprint
logical            gotbas,gotfac,gothes,gotscl
common    /cycle1/ gotbas,gotfac,gothes,gotscl
```

Cycle tolerance may be used to specify a numerical value for cnvtol. The four logical variables may be set to .true. to request various Warm or Hot starts (see Page 121).

2.5 Subroutine MATCOL

If PHANTOM COLUMNS c and PHANTOM ELEMENTS e are defined in the SPECS file (along with CYCLE LIMIT k), this subroutine may be called by MATMOD up to c times throughout cycles 2 through k. The aim is to turn at most c "phantom columns" into normal columns containing a total of at most e nonzero elements. MATMOD must provide an array COL(*) and a zero tolerance ZTOL for each call. The significant elements of COL will be packed into the matrix data structure, to form a new column. The associated variable will be given the default LOWER and UPPER bounds, and a scale factor of 1.0.

Specification:

```
      SUBROUTINE MATCOL( M, N, NB, NE, NKA,
     *                   A, HA, KA, BL, BU, COL, ZTOL )
      IMPLICIT           REAL*8(A-H,O-Z)
      INTEGER*2          HA(NE)
      INTEGER            KA(NKA)
      DIMENSION          A(NE), BL(NB), BU(NB), COL(M)
```

Parameters:

M (Input) The length of the array COL. Usually this will be m, the number of rows in the constraint matrix. In general, it may be anywhere in the range $1 \leq M \leq m$, if the new column is known to be zero beyond position M.

COL(*) (Input) The dense vector that is to become a new matrix column.

ZTOL (Input) A zero tolerance for deleting negligible elements from COL when it is packed into A and HA. On most machines, a reasonable value is ZTOL = 1.0E-8.

The other parameters come directly from MATMOD. For further details, see the CYCLE options in section 3.3, and the example in section 8.5.

2.6 Matrix Data Structure

In the MINOS source code, the constraint matrix A is stored column-wise in sparse format in the arrays A, HA, KA, as defined in the specifications of subroutine MATMOD (section 2.4). The matrix I associated with the slack variables is represented implicitly. If the objective function contains linear terms $c^Tx + d^Ty$, then $(c^T\ d^T)$ is included as the IOBJ-th row of A (see the COMMON block M5LOBJ below).

If there are nonlinear constraints, the top left-hand corner of A is loaded with the current Jacobian matrix at the start of each major iteration.

The following COMMON blocks contain dimensions and other items relating to the storage of A.

```
      COMMON    /M3LEN / M      ,N      ,NB     ,NSCL
```

M m, the number of rows in A, including the linear objective row (if any).

N n, the number of columns in A, possibly including c "phantom columns".

NB $n + m =$ N+M, the total number of variables in the problem, including the slacks.

NSCL Either NB or 1, depending on whether SCALE has been specified or not.

```
COMMON    /M2MAPA/ NE     ,NKA    ,LA     ,LHA    ,LKA
```

NE The number of nonzero elements in A, possibly including e "phantom elements".

NKA $n + 1 =$ N+1, the number of pointers in the array KA.

LA The address of A(*) in the work array Z(*).

LHA The address of HA(*) in the work array Z(*).

LKA The address of KA(*) in the work array Z(*).

```
COMMON    /M5LEN / MAXR  ,MAXS  ,MBS    ,NN     ,NNO    ,NR     ,NX
```

MAXR The HESSIAN DIMENSION.

MAXS The *SUPERBASICS* LIMIT.

MBS M+MAXS, the maximum number of basic and superbasic variables.

NN $n_1 =$ max{NNOBJ, NNJAC}, the number of NONLINEAR VARIABLES.

NNO max{1, NN}.

NR The dimension of the array R that is used to approximate the reduced Hessian, R.

NX max{MBS, NN}.

```
COMMON    /M5LOBJ/ SINF   ,WTOBJ ,MINIMZ,NINF   ,IOBJ
```

SINF The current sum of infeasibilities.

WTOBJ The scalar w used in the composite objective technique.

MINIMZ +1 if the objective is to be minimized; −1 if it is to be maximized.

NINF The current number of *infeasibilities*.

IOBJ The row number for the linear objective. (If IOBJ is zero, there is no such row.)

```
COMMON    /M7LEN / FOBJ   ,FOBJ2 ,NNOBJ ,NNOBJO
```

FOBJ The current value of the function value F returned by FUNOBJ.

FOBJ2 A temporary value of FOBJ.

NNOBJ n_1', the number of NONLINEAR OBJECTIVE VARIABLES.

NNOBJO max{1, NNOBJ}.

```
COMMON    /M8LEN / NJAC   ,NNCON ,NNCONO,NNJAC
```

NJAC The number of elements in the Jacobian.

NNCON m_1, the number of NONLINEAR CONSTRAINTS.

NNCONO max{1, NNCON}.

NNJAC n_1'', the number of NONLINEAR JACOBIAN VARIABLES.

3. THE SPECS FILE

The SPECS file sets various run-time parameters that describe the nature of the problem being solved and the manner in which a solution is to be obtained. The file consists of a sequence of card images, each of which contains a *keyword* and certain associated *values*.

The first keyword is **BEGIN** and the last keyword is **END**. If the problem could be solved using default values for all parameters, the SPECS file could consist of just those two keywords (on separate cards). Normally, however, at least some of the parameters must be specified; for example, the number of nonlinear variables if there are any.

3.1 SPECS File Format

Each card in the SPECS file contains a sequence of items in free format (they may appear anywhere in columns 1 to 72). The items are separated by spaces or equal signs (' ' or '='). Those selected from each card are:

1. The first word (the *keyword*). Only the first 3 characters are significant.
2. The second word (if any). Sometimes this is the keyword's associated *name value*, an 8-character name. More often it qualifies the keyword, and its first 4 characters are significant.
3. The first number (if any). This may be an *integer value* or a *real value*; up to 8 characters in Fortran's I, F, E or D format.

In the following examples the significant characters are underlined:

OBJECTIVE	PROFIT
SOLUTION FILE	12
ROWS	500
ROW TOLERANCE	0.0001
LOWER BOUND	-1.0
AIJ TOL	1.0E-6

If the first character of an item is one of the following *numeric characters*

1 2 3 4 5 6 7 8 9 0 + - .

then the item is taken to be a *number*. The number may be from 1 to 8 contiguous numeric characters, including an E or a D if need be. It is terminated by a non-numeric character such as a space.

(An exception is made for the keywords **OBJECTIVE**, **RHS**, **RANGE** and **BOUND**, which specify names to be extracted from the MPS file. For these keywords the second item is taken to be the required name value even if it begins with a numeric character. Thus,

AIJ TOLERANCE	.00001
OBJECTIVE	.00001
RHS	...2E001
BOUND	+1000

are all allowed. However, names like **OBJECTIVE = COST** or **RHS = DEMAND02** will be more common.)

Blank cards and comments may be used to improve readability. A *comment* begins with an asterisk ('*') and includes all subsequent characters on the same card; these are ignored. The '*' may be the first non-blank character on the card, or the first non-blank after a space or an equal sign. For example:

```
*
*  MPS file parameters
*
   ROWS            1000       * (or less)
   COLUMNS         2000       * (or less)
   ELEMENTS        8000       * (or less)
   OBJECTIVE  =    PROFIT02   * (the 2nd N row)
```

Scanning terminates once a number has been recognized. An asterisk is therefore not essential following a number:

```
WEIGHT ON OBJECTIVE = 10.0  DURING PHASE 1
```

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 IBM 360 and 370 systems and their successors ($\epsilon \approx 2.22 \times 10^{-16}$). Similar values would apply to any machine having about 15 decimal digits of precision.

```
BEGIN checklist of SPECS file parameters and their default values
*
*  Keywords for the MPS file
*
MINIMIZE                                  * (opposite of MAXIMIZE)
OBJECTIVE  =                  ?           * the first name encountered
RHS        =                  ?           * the first name encountered
RANGE      =                  ?           * the first name encountered
BOUNDS     =                  ?           * the first name encountered
ROWS                          100         *
COLUMNS                       300         * or 3*ROWS
ELEMENTS (or COEFFICIENTS)    1500        * or 5*COLUMNS
AIJ TOLERANCE                 1.0E-10     *
LOWER BOUND                   0.0         *
UPPER BOUND                   1.0E+20     * plus infinity
MPS FILE                      ?           * depends on installation
LIST LIMIT                    0           * for printing MPS data
ERROR MESSAGE LIMIT           10          * during MPS input
*
*  Keywords for the simplex method
*
CRASH OPTION                  1           * all variables eligible for initial basis
ITERATIONS LIMIT              300         * or 3*ROWS + 10*NONLINEAR VARIABLES
PARTIAL PRICE                 1           * or COLS/(2*ROWS) if COLS is large
MULTIPLE PRICE                1           * BEWARE - not like commercial LP
WEIGHT ON LINEAR OBJECTIVE    0.0         * during phase 1
```

```
SUMMARY FILE                    0        * > 0 for occasional output to terminal
SUMMARY FREQUENCY               100      * iteration log on SUMMARY file
LOG     FREQUENCY               1        * iteration log on PRINT file
CHECK FREQUENCY                 30       * numerical test on row residuals
FACTORIZATION FREQUENCY         50       * refactorize the basis matrix
SAVE FREQUENCY                  100      * basis map
SCALE                           NO       * linear constraints and variables
SOLUTION                        YES      * on PRINT file
*
*  BASIS files
*
OLD BASIS FILE                  0        * input basis map
NEW BASIS FILE                  0        * output basis map
BACKUP BASIS FILE               0        * output basis map
INSERT FILE                     0        * input in industry format
PUNCH FILE                      0        * output INSERT data
LOAD FILE                       0        * input names and values
DUMP FILE                       0        * output LOAD data
SOLUTION FILE                   0        * separate from printed solution
*
*  Convergence and stability tolerances
*
FEASIBILITY TOLERANCE           1.0E-6   * for satisfying bounds
OPTIMALITY TOLERANCE            1.0E-6   * for reduced gradients
PIVOT TOLERANCE                 3.7E-11  * ε^(2/3)
LU FACTOR TOLERANCE             10.0     * limits size of multipliers in L
LU UPDATE TOLERANCE             10.0     * the same during updates
*
*  Parameters for nonlinear problems
*
NONLINEAR CONSTRAINTS           0        * must be the exact number, m1
NONLINEAR VARIABLES             0        * must be the exact number, n1
NONLINEAR OBJECTIVE VARIABLES   0        * use if different from Jacobian variables
NONLINEAR JACOBIAN  VARIABLES   0        * use if different from objective variables
SUPERBASICS LIMIT               1        * or HESSIAN DIMENSION
HESSIAN DIMENSION               1        * or SUPERBASICS LIMIT
*
PROBLEM NUMBER                  0        * sets subroutine parameter NPROB
DERIVATIVE LEVEL                3        * assumes all gradients are known
VERIFY LEVEL                    0        * gives cheap check on gradients
EMERGENCY VERIFY LEVEL          0        * cheap check before stopping
*
START OBJECTIVE  CHECK AT COL   1        *
STOP  OBJECTIVE  CHECK AT COL   n1       *
START CONSTRAINT CHECK AT COL   1        *
STOP  CONSTRAINT CHECK AT COL   n1       *
```

```
LINESEARCH TOLERANCE              0.1        * smaller for more accurate search
SUBSPACE TOLERANCE                0.5        * affects when to PRICE
FUNCTION PRECISION                3.0E-13    * ε^0.8 (almost full accuracy)
DIFFERENCE INTERVAL               5.5E-7     * (FUNCTION PRECISION)^½
CENTRAL DIFFERENCE INTERVAL       6.7E-5     * (FUNCTION PRECISION)^⅓
*
*  Further parameters for nonlinear constraints
*
JACOBIAN                          DENSE      *
LAGRANGIAN                        YES        *
MAJOR ITERATIONS                  20         *
MINOR ITERATIONS                  40         *
PENALTY PARAMETER                 100.0/m1   * may need to be larger if very nonlinear
DAMPING PARAMETER                 2.0        * affects step-size between subproblems
*
COMPLETION                        PARTIAL    * FULL if no nonlinear constraints
ROW TOLERANCE                     1.0E-6     * allowable nonlinear constraint violation
RADIUS OF CONVERGENCE             0.01       * for reducing the penalty parameter
PRINT LEVEL  (JFLXB)              00001      * J(xk), f(xk), λk, xk, Basis statistics
*
*  Sequences of related problems
*
CYCLE LIMIT                       1          *
CYCLE PRINT                       1          *
CYCLE TOLERANCE                   0.0        *
PHANTOM COLUMNS                   0          *
PHANTOM ELEMENTS                  0          *
*
*  Miscellaneous
*
DEBUG LEVEL                       0          *
LINESEARCH DEBUG AFTER ITN        999999     *
WORKSPACE (USER)                  0          *
WORKSPACE (TOTAL)                 ?          * depends on installation
*  SUPPRESS PARAMETER LISTING
END of SPECS file checklist
```

3.3 SPECS File Definitions

The following is an alphabetical list of recognized SPECS file keywords. A typical use of each keyword is given, along with a definition of the quantities involved and comments on usage. In many cases the value associated with a keyword is denoted by a letter such as k, and allowable values for k are subsequently defined.

AIJ TOLERANCE t (default $t = 1.0E\text{-}10$)

During input of the MPS file, matrix coefficients a_{ij} will be ignored if $|a_{ij}| < t$.

If a_{ij} is a Jacobian element, it is *not* ignored. (Its position is recorded, and it will retain the value t if DERIVATIVE LEVEL = 2 or 3 and FUNCON does not reset the corresponding element of G.)

If CYCLE LIMIT > 1 and a_{ij} is to be changed from zero to a value greater than t during a later cycle, set $t = 0.0$ to retain all entries in the MPS file.

BACKUP BASIS FILE k (default $k = 0$)

This is intended as a safeguard against losing the results of a long run. Suppose that a NEW BASIS FILE is being saved every 100 iterations, and that MINOS is about to save such a basis at iteration 2000. It is conceivable that the run may time-out during the next few milliseconds (i.e., in the middle of the save), or the host computer could unexpectedly crash. In this case the basis file will be corrupted and the run will have been essentially wasted.

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

```
OLD BASIS FILE      11        (or 0)
BACKUP BASIS FILE   11
NEW BASIS FILE      12
SAVE FREQUENCY     100
```

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

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

BOUNDS BOUND01

This specifies the 8-character name of the bound set to be selected from the MPS file.

1. BNDS is a valid alternative keyword.
2. If BOUNDS is not specified, or if the name is blank, the *first* bound set in the MPS file will be selected.
3. If the MPS file contains one or more bound sets but you do not want any of them to be used, specify a dummy name such as BOUND = NONE.

CENTRAL DIFFERENCE INTERVAL h_2 (default $h_2 = (\text{FUNCTION PRECISION})^{\frac{1}{3}}$)
When DERIVATIVE LEVEL < 3, the central-difference interval h_2 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 j-th variable is $h_j = h_2(1 + |x_j|)$. The resulting gradient estimates should be accurate to $O(h_j^2)$, unless the functions are badly scaled.

CHECK FREQUENCY k (default $k = 30$)
Every k-th 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 any linearized nonlinear constraints, if any). If these are $Ax + s = 0$ where s is the set of slack variables, the residual vector $r = Ax + s$ 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.

COEFFICIENTS 5000
See ELEMENTS.

COLUMNS n (default n = 3*ROWS)
This must specify an *over-estimate* of the number of columns in the constraint matrix (excluding slack variables, but including any PHANTOM COLUMNS). If n proves to be too small, MINOS will continue reading the MPS file to determine the true value of n, and an appropriate warning message will be issued. If the MPS file number is the same as the system card reader, the problem will then be terminated; otherwise the MPS file will be re-read.

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 effects 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 k (default $k = 3$)
If a basis file is not specified, a triangular basis will be selected from certain rows and columns of the constraint matrix (A I). Free rows and variables are given priority. Slack columns (from I) are added where necessary. Please see Page 120 for further details.

CRASH tolerance t (default $t = 0.1$)
This tolerance allows CRASH to ignore certain "small" nonzeros in the constraint matrix while searching for a triangular basis. For each column of A, if a_{max} is the largest element in the column, other nonzeros in that column are ignored if they are less than or equal to $a_{max} \times t$.

When $t > 0.0$, 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 with more structural variables and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

CYCLE LIMIT	l	(default $l = 1$)
CYCLE PRINT	p	(default $p = 1$)
CYCLE TOLERANCE	t	(default $t = 0.0$)
PHANTOM COLUMNS	c	(default $c = 0$)
PHANTOM ELEMENTS	e	(default $e = 0$)

These keywords refer to a facility for constructing and solving a sequence of related problems, as described in sections 1.9, 2.4 and 2.5. The COMMON block

```
COMMON    /CYCLCM/ CNVTOL,JNEW,MATERR,MAXCY,NEPHNT,NPHANT,NPRINT
```

contains certain relevant variables.

1. l = MAXCY is the maximum number of problems to be solved.
2. p = NPRINT controls the printing of intermediate solutions. At most, the last p solutions will be output.
3. t = CNVTOL is a real number for possible use in a user-specified convergence test within subroutine MATMOD.
4. c = NPHANT is the number of columns that can be added to the constraint matrix beyond those specified in the MPS file. Each column must be added by means of a call to subroutine MATCOL. If an error occurs, MATCOL increments MATERR (which is initially zero). Otherwise, JNEW records the index of the new column.
5. e = NEPHNT is the number of nonzero elements that are allocated to the "phantom columns" beyond those specified in the MPS file.

DAMPING PARAMETER* d (default $d = 2.0$)
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.

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

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

for some step-length $\sigma < 1$. (In the case of nonlinear equations, this gives a *damped Newton method*.)

*Now called Major damping parameter.

1. 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 ρ (say 10 or 100 times the default ρ). (Skip this re-run in the case of nonlinear equations. There are no degrees of freedom and the value of ρ is irrelevant.)
2. If the subproblem solutions continue to change violently, try reducing d to 0.2 or 0.1 (say).
3. For implementation reasons, the shortened step σ applies to the nonlinear variables x, but not to the linear variables y or the slack variables s. This may reduce the efficiency of the control.

DEBUG LEVEL d (default $d = 0$)
This causes various amounts of information to be output to the PRINT file.

k	*Meaning*
0	No debug output.
2	(or more) Output from M5SETX showing the maximum residual after a row check.
40	Output from LU8RPC showing the position of the last nonzero in the transformed incoming column.
50	Output from LU2FAC showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.
100	Output from M2BFAC and M5LOG listing the basic and superbasic variables and their values at every iteration.

DERIVATIVE LEVEL d (default $d = 3$)
This specifies which nonlinear function gradients are known analytically and will be supplied to MINOS by the user subroutines FUNOBJ and FUNCON.

d	*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 $d = 3$ should be used whenever possible. It is the most reliable and will usually be the most efficient.

If $d = 0$ or 2, MINOS will *estimate* the missing components of the objective gradient, using finite differences. This may simplify the coding of subroutine FUNOBJ. However, it could increase the total run-time substantially (since a special call to FUNOBJ is required for each missing element), 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 $d = 0$ or 1, MINOS will estimate missing elements of the Jacobian. For each column of the Jacobian, one call to FUNCON 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, MINOS will use one call to FUNCON 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. Twice as many calls to FUNOBJ and FUNCON are then needed. (This is not under the user's control.)

Remember: when analytic derivatives are not provided, the attainable accuracy in locating an optimal solution is usually less than when all gradients are available. DERIVATIVE LEVEL 3 is strongly recommended.

DIFFERENCE INTERVAL h_1 (default $h_1 = (\text{FUNCTION PRECISION})^{\frac{1}{2}}$)

This alters the interval h_1 that is used to estimate gradients by forward differences in the following circumstances:

1. In the initial ("cheap") phase of verifying the objective gradients.
2. For verifying the constraint gradients.
3. For estimating missing objective gradients.
4. For estimating missing Jacobian elements.

In the last three 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 $F(x)$ 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.

DUMP FILE f (default $f = 0$)

If $f > 0$, the last solution obtained will be output to file f in the format described in section 5.3. The file will usually have been output previously as a LOAD file.

ELEMENTS e (default $e = 5*\text{COLUMNS}$)

This must specify an *over-estimate* of the number of nonzero elements (coefficients a_{ij}) in the constraint matrix, including all entries in a DENSE or SPARSE Jacobian, and all nonzeros in the matrices A_1, A_2, A_3. (It should also include the number of PHANTOM ELEMENTS, if any.)

1. COEFFICIENTS is a valid alternative keyword.
2. If e proves to be too small, MINOS continues in the manner described under COLUMNS.

EMERGENCY VERIFY LEVEL
See VERIFY LEVEL.

ERROR MESSAGE LIMIT e (default $e = 10$)
This is the maximum number of error messages to be printed for each type of error occurring when the MPS file is read. The default value is reasonable for early runs on a particular MPS file. If the same file is used repeatedly, e can be reduced to suppress warning of non-fatal errors.

FACTORIZATION FREQUENCY k (default $k = 50$)
At most k basis changes will occur between factorizations of the basis matrix.

1. 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.
2. 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.
3. When the constraints are nonlinear, the MINOR ITERATIONS limit will probably preempt k.

FEASIBILITY TOLERANCE t (default $t = 1.0E\text{-}6$)
A *feasible solution* 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 linear constraints are also satisfied to within t.)

1. MINOS attempts to find a feasible point before optimizing the objective function. 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.
2. *Note*: if SINF is not small, there may be other points that have a *significantly smaller sum of infeasibilities*. MINOS does not attempt to find the solution that minimizes the sum.
3. If SCALE is used, feasibility is defined in terms of the scaled problem (since it is then more likely to be meaningful).
4. A nonlinear objective function $F(x)$ will be evaluated only at feasible points. If there are regions where $F(x)$ 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 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 x as far away from singularities as possible.)
5. Bounds should also be used to keep x more than t away from singularities in $f(x)$.
6. If there are any nonlinear constraints, each major iteration attempts to satisfy their linearization to within the tolerance t. If this is not possible, the bounds on the nonlinear constraints are relaxed temporarily (in several stages).
7. Feasibility with respect to the nonlinear constraints themselves is measured against the ROW TOLERANCE (not against t). The relevant test is made at the *start* of a major iteration.

FUNCTION PRECISION ϵ_R (default $\epsilon_R = \epsilon^{0.8}$)

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 $f'(x)$ 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.)

1. The default value of ϵ_R is appropriate for simple analytic functions.
2. 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 linesearch 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 r = SUPERBASICS LIMIT or 30)

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.

1. If $s \leq r$, the first s columns of R will be 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 will ultimately be superlinear.
2. If $s > r$, a matrix of the form
$$R = \begin{pmatrix} R_r & 0 \\ & D \end{pmatrix}$$
will be 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 will no longer be superlinear.
3. The storage required is of order $\frac{1}{2}r^2$, which is substantial if r is as large as 200 (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.
4. If SUPERBASICS LIMIT s is specified, the default value of r is the same number, s (and conversely). This is a safeguard to ensure superlinear convergence wherever possible. If neither r nor s is specified, both default to the value 30.

INSERT FILE f (default $f = 0$)

If $f > 0$, this references a file containing basis information in the format of section 5.2.

1. The file will usually have been output previously as a PUNCH file.
2. The file will not be accessed if an OLD BASIS file is specified.

INVERT FREQUENCY

See FACTORIZATION FREQUENCY.

ITERATIONS LIMIT k (default k = 3*ROWS + 10*NONLINEAR VARS)

This is the maximum number of minor iterations allowed (i.e., iterations of the simplex method or the reduced-gradient algorithm).

1. ITNS is an alternative keyword.
2. $k = 0$ is valid. Both feasibility and optimality are checked.
3. If CYCLE LIMIT > 1, the limit of k minor iterations applies to each cycle separately.

JACOBIAN DENSE (default)

JACOBIAN SPARSE

This determines the manner in which the constraint gradients are evaluated and stored. It affects the MPS file and subroutine FUNCON.

1. The DENSE option is convenient if there are not many nonlinear constraints or variables. It requires storage for three dense matrices of order $m_1 \times n_1$.
2. The MPS file may then contain any number of Jacobian entries. Usually this means no entries at all.
3. For efficiency, the SPARSE option is preferable in all nontrivial cases. (*Beware—it must be specifically requested.*) The MPS file must then specify the position of all Jacobian elements (that are not identically zero), and subroutine FUNCON must store the elements of the Jacobian array G in exactly *the same order.*
4. In both cases, if DERIVATIVE LEVEL = 2 or 3 the MPS file may specify Jacobian elements that are constant for all values of the nonlinear variables. The corresponding elements of G need not be reset in FUNCON.

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, subroutine FUNCON will be called only twice per major iteration. Hence this option may be useful if the nonlinear constraint functions are very expensive to evaluate. However, in general there is a great risk that convergence may not occur. (*Note*: FUNCON will be called more often to estimate $J(x)$ if DERIVATIVE LEVEL < 2.)

LINESEARCH DEBUG AFTER ITERATION i (default i = 999999)

This causes considerable information to be output by the linesearch procedures every iteration, once iteration i has been completed. Its principal purpose is to assist the authors of the linesearch procedures to determine if the procedures are functioning correctly. In some cases it may confirm that the function values are very "noisy", or that the gradients computed in FUNOBJ or FUNCON are incorrect.

LINESEARCH TOLERANCE t (default $t = 0.1$)
For nonlinear problems, this controls the accuracy with which an optimum of the merit function will be located along the direction of search each iteration.

1. t must be a real value in the range $0.0 \leq t \leq 1.0$.
2. The default value $t = 0.1$ requests a moderately accurate search. It should be satisfactory for many problems.
3. 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.
4. If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. *If all gradients are known,* 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.)
5. If not all gradients are known, a reasonably accurate search remains appropriate. Each search will require only 2–5 function values (typically), but many function calls will then be needed to estimate missing gradients for the next iteration.

LIST LIMIT k (default $k = 0$)
This limits the number of lines of the MPS file to be listed on the PRINT file during input. The header cards (NAME, ROWS, COLUMNS, RHS, RANGE, BOUNDS, ENDATA) and comment cards will always be listed, along with their position in the file.

LOAD FILE f (default $f = 0$)
If $f > 0$, this references a file containing basis information in the format of section 5.3.

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

LOG FREQUENCY k (default $k = 1$)
One line of the iteration log will be printed every k-th minor iteration. A value such as $k = 10$ is suggested for those interested only in the final solution.

LOWER BOUND l (default $l = 0.0$)
Before the BOUNDS section of the MPS file is read, all structural variables are given the default lower bound l. (Individual variables may subsequently have their lower bound altered by a BOUND set in the MPS file.)

1. LOWER BOUND = 1.0E-5 (say) is a useful method for bounding all variables away from singularities at zero. (Explicit bounds may also be necessary in the MPS file.)
2. If all or most variables are to be *FREE*, use LOWER BOUND = -1.0E+20 to specify "minus infinity". (The default upper bound is already 1.0E+20, which is treated as "plus infinity".)

LU FACTOR TOLERANCE t_1 (default $t_1 = 10.0$)
LU UPDATE TOLERANCE t_2 (default $t_2 = 10.0$)
These tolerances affect the stability and sparsity of the basis factorization $B = LU$, during refactorization and updates respectively. Both tolerances must satisfy $t_i \geq 1.0$. The matrix L is a product of matrices of the form

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

where the multipliers μ will satisfy $|\mu| \leq t_i$.

1. The default values $t_i = 10.0$ usually strike a good compromise between stability and sparsity.
2. For large and relatively dense problems, $t_i = 25.0$ (say) may give a marked improvement in sparsity without impairing stability to a serious degree.
3. For certain very special structures (e.g., band matrices) it may be necessary to set t_1 in the range $1.0 \leq t_1 < 2.0$ to achieve stability.

MAJOR ITERATIONS k (default $k = 20$)
This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints, since in some cases the sequence of major iterations may not converge.

For preliminary runs on a new problem, a fairly low MAJOR ITERATIONS limit should be specified (e.g., 10 or 20). See the advice given under PENALTY PARAMETER.

MAXIMIZE
MINIMIZE (default)
This specifies the required direction of optimization. It applies to both linear and nonlinear terms in the objective.

MINOR ITERATIONS k (default $k = 40$)
This is the maximum number of iterations allowed between successive linearizations of the nonlinear constraints. A moderate value (e.g., $10 \leq k \leq 50$) prevents excessive effort being expended on early major iterations, but allows later subproblems to be solved to completion.

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.)

Note that an independent limit on total iterations should be specified by the ITERATIONS keyword as usual. If the problem is linearly constrained, this is the *only* limit (i.e., the MINOR ITERATIONS keyword is ignored).

MPS FILE f (default $f = ?$)
This is the file number for the MPS file. The default value is the system card reader IREAD, which is often $f = 5$.

1. INPUT FILE is a valid alternative keyword.
2. For nontrivial problems it is usually best to store the MPS file separately from the SPECS file. If the ROWS, COLUMNS or ELEMENTS estimates prove to be too low, MINOS will be able to rewind the MPS file and try again.

MULTIPLE PRICE k (default $k = 1$)

Whenever a PRICE operation is performed, the k best nonbasic variables will be selected for admission to the superbasic set. ("Best" means the variables with largest reduced gradients of appropriate sign. If partial pricing is in effect, up to k variables are selected from the current partition of A and I.)

1. The default value $k = 1$ is best for linear programs, since an optimal solution will have zero superbasic variables.
2. Warning: if $k > 1$, MINOS will go into reduced-gradient mode even on purely linear problems. The subsequent iterations do *not* correspond to the very efficient suboptimization ("minor iterations") carried out by standard linear programming systems using multiple pricing. (MINOS varies all superbasic variables simultaneously. However, its storage requirements are essentially independent of k on linear problems. Thus, k need not be limited to 5 or 6 as it is in standard systems, which require storage for k dense vectors of dimension m.)
3. On large nonlinear problems it may be important to set $k > 1$, if the starting point does not contain many superbasic variables. For example, if a problem has 3000 variables and 500 of them are nonlinear, the optimal solution may well have 200 variables superbasic. If the problem is solved in several runs, it may be beneficial to use $k = 10$ (say) for early runs, until it seems that the number of superbasics has levelled off.

NEW BASIS FILE f (default $f = 0$)

If $f > 0$, a basis map will be saved on file f every k-th iteration, where k is the SAVE FREQUENCY.

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

NONLINEAR CONSTRAINTS	m_1	(default $m_1 = 0$)
NONLINEAR VARIABLES	n_1	(default $n_1 = 0$)
NONLINEAR OBJECTIVE VARIABLES	n_1'	(default $n_1' = 0$)
NONLINEAR JACOBIAN VARIABLES	n_1''	(default $n_1'' = 0$)

These keywords define the parameters M and N in subroutines FUNOBJ and FUNCON. For example, M in FUNCON will take the value m_1, if $m_1 > 0$.

1. If the objective function and the constraints involve the same set of nonlinear variables x, then NONLINEAR VARIABLES n_1 is the simplest way to set N to be the same value for both subroutines.
2. Otherwise, the NONLINEAR OBJECTIVE and NONLINEAR JACOBIAN keywords should be used to specify n_1' and n_1'' separately.
3. If $m_1 = 0$, the value $n_1'' = 0$ is assumed regardless of n_1 or n_1'.
4. Remember that the nonlinear constraints and variables must always be the first ones in the problem. It is usually best to place Jacobian variables before objective variables, so that $n_1'' \leq n_1'$ (unless $n_1' = 0$). This affects the way the function subroutines should be programmed, and the order in which variables should be placed in the COLUMNS section of the MPS file.

OBJECTIVE COST

This specifies the 8-character name of the type N row in the MPS file to be selected as the linear part of the objective function (i.e., the objective function for linear programs).

1. If OBJECTIVE is not specified, or if the name is blank, the *first* N row in the ROWS section of the MPS file will be selected. (*Warning*: objective rows must be listed *after* nonlinear constraint rows in the ROWS section of the MPS file.)

2. If the ROWS section contains *one or more* N rows but you do not want *any* of them to be used in the objective function, specify a dummy name. If the objective is defined entirely by subroutine FUNOBJ it may be helpful to specify OBJECTIVE = FUNOBJ. (However, don't expect a different name to invoke a different subroutine!)

OLD BASIS FILE f (default $f = 0$)

If $f > 0$, the starting point will be obtained from this file in the format of section 5.1.

1. The file will usually have been output previously as a NEW BASIS FILE.

2. The file will not be acceptable if the number of rows or columns in the problem has been altered.

OPTIMALITY TOLERANCE t (default $t = 1.0E\text{-}6$)

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.

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

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

respectively, and if

$$|d_j|/\|\pi\| \leq t$$

for superbasic variables.

2. 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.

3. The quantity actually used is defined by

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

so that only large scale factors are allowed for. If the objective is scaled *down* substantially, the test for optimality reduces to comparing just d_j against t.

PARTIAL PRICE p (default $p = 1$ or c (see below))

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).

1. When $p = 1$, all columns of the constraint matrix $(A \quad I)$ are searched.
2. 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-1}, I_{j-1}, the next search begins on the segments A_j, I_j. (All subscripts here 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 PRICE has been specified.) If nothing is found, the search continues on the next segments A_{j+1}, I_{j+1}, and so on.
3. The default value of p is 1 for moderate-sized problems, but may be greater than 1 otherwise. A quantity
$$c = \max\{1000, 4*\text{ROWS}\}$$
is defined. If COLUMNS $\geq c$ and PARTIAL PRICE has not been specified, p will take the value COLUMNS/2*ROWS}.
4. PARTIAL PRICE p is recommended for time-stage models having p time periods.

PENALTY PARAMETER ρ (default $\rho = 100.0/m_1$)

This is the value of ρ in the modified augmented Lagrangian. It is used only when LAGRANGIAN = YES.

For early runs on a problem with unknown characteristics, something like the default value should be specified. If the problem is known to be highly nonlinear, specify a *larger* value, such as 10 times the default. In general, a positive value of ρ may be necessary to ensure convergence, *even for convex programs.*

On the other hand, if ρ 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 will often be 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 of ρ, such as the default, and a reasonably low ITERATIONS and/or MAJOR ITERATIONS limit.
2. If successive major iterations appear to be terminating with radically different solutions, the penalty parameter should be increased. (See also the DAMPING PARAMETER.)
3. If there appears to be little progress between major iterations, the penalty parameter could be reduced.

PHANTOM COLUMNS c (default $c = 0$)
PHANTOM ELEMENTS e (default $e = 0$)

See the CYCLE parameters.

PIVOT TOLERANCE t (default $t = \epsilon^{\frac{2}{3}}$)
This allows the pivot tolerance to be altered if necessary. (The tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular.) The default value of t is roughly 10^{-11} for double precision on IBM systems. This should be satisfactory in most circumstances.

PRINT LEVEL (JFLXB) p (default $p = 00001$)
This varies the amount of information that will be output to the printer file. It is independent of the LOG FREQUENCY. Typical values are

PRINT LEVEL 1

which gives normal output for linear and nonlinear problems, and

PRINT LEVEL 11

which in addition gives the values of the nonlinear variables x_k at the start of each major iteration, for problems with nonlinear constraints.

In general, the value being specified 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, i.e., information relating to the basis matrix whenever it is refactorized.
- 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 the option LAGRANGIAN = NO is specified, since $\lambda_k = 0$ then.)
- 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.

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.

PRINT LEVEL 0 may be used to suppress most output, including page ejects between major iterations. (Error messages will not be suppressed.) This print level should be used only for production runs on well understood models. A high LOG FREQUENCY may also be appropriate for such cases, e.g. 100 or 1000. (For convenience, LOG FREQUENCY 0 may be used as shorthand for LOG FREQUENCY 99999.)

PROBLEM NUMBER n (default $n = 0$)
For nonlinear problems, this assigns a value to the parameter NPROB in the user subroutines FUNOBJ, FUNCON and MATMOD.

PUNCH FILE f (default $f = 0$)
If $f > 0$, the final solution obtained will be output to file f in the format described in section 5.2. For linear programs, this format is compatible with various commercial systems.

RADIUS OF CONVERGENCE r (default $r = 0.01$)
This determines when the penalty parameter ρ will be reduced (if initialized to a positive value). Both the nonlinear constraint violation (see *ROWERR* below) and the relative change in consecutive Lagrange multipler estimates must be less than r at the start of a major iteration before ρ is reduced or set to zero. Once ρ is zero, the sequence of major iterations should converge quadratically to an optimum.

RANGES RANGE001
This specifies the 8-character name of the range set to be selected from the MPS file.

1. RNGS is a valid alternative keyword.
2. If RANGES is not specified, or if the name is blank, the first range set in the MPS file will be selected.
3. If the MPS file contains one or more range sets but you do not want *any* of them to be used, specify a dummy name such as RANGES = NONE.

RHS RHSIDE3
This specifies the 8-character name of the righthand side to be selected from the MPS file.

1. If RHS is not specified, or if the name is blank, the *first* righthand side in the MPS file will be selected.
2. If the MPS file contains one or more righthand sides but you do not want *any* of them to be used, specify a dummy name such as RHS = NONE.

ROWS m (default $m = 100$)
This must specify an *over-estimate* of the number of rows in the constraint matrix. It includes the number of nonlinear constraints and the number of general linear constraints.

If m proves to be too small, MINOS continues in the manner described under COLUMNS.

ROW TOLERANCE ϵ_r (default $\epsilon_r = 1.0E\text{-}6$)
This specifies how accurately the nonlinear constraints should be satisfied. (Both "ROW" and "TOLE" are significant on this data card.) The default value of 1.0E-6 is often appropriate, since the MPS file contains data to about that accuracy.

Let *ROWERR* be defined as the maximum component of the residual vector $f(x) + A_1 y - b_1$, normalized by the size of the solution. Thus,

$$ROWERR = \|f(x) + A_1 y - b_1\|_\infty / XNORM,$$

where *XNORM* is a measure of the size of the basic and superbasic variables. The solution (x, y) is regarded as acceptably feasible if $ROWERR \leq \epsilon_r$.

If some of the problem functions are known to be of low accuracy, a larger ROW TOLERANCE may be appropriate.

SAVE FREQUENCY k (default $k = 100$)
If a NEW BASIS file has been specified, a basis map describing the current solution will be saved on the appropriate file every k-th iteration. A BACKUP BASIS file will also be saved if specified.

SCALE NO (default)
SCALE OPTION 0

SCALE
SCALE YES
SCALE LINEAR VARIABLES
SCALE OPTION 1

SCALE NONLINEAR VARIABLES
SCALE ALL VARIABLES
SCALE OPTION 2

SCALE, PRINT
SCALE TOLERANCE t (default $t = 0.9$)
Three scale options are available, with equivalent definitions as shown. The default is: No scaling. Otherwise, the constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1 (see Fourer, 1982). This will sometimes improve the performance of the solution procedures. SCALE OPTION 1 scales only the linear constraints and variables.

If the constraints are linear, SCALE OPTION 1 scales all rows of the constraint matrix A, but only the columns associated with linear variables. SCALE OPTION 2 performs an additional scaling that may be helpful if the solution x is large; it takes into account columns of $(\ A \quad I\ )$ that are fixed or have positive lower bounds or negative upper bounds. SCALE OPTION 2 is suitable for linear programs and for problems with nonlinear objectives.

If nonlinear constraints are present, SCALE OPTION 0 or 1 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 good starting point is provided (by the INITIAL bounds set or a basis file).

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 $0.0 < t < 1.0$ (for example: SCALE, PRINT, TOLERANCE = 0.99). Raising t from 0.9 to 0.99 (say) will probably increase the number of scaling passes through A. At most 10 passes will be made.

If a SCALE OPTION has not already been specified, SCALE PRINT or SCALE TOLERANCE both set SCALE OPTION 1.

SOLUTION YES (default)
SOLUTION NO
SOLUTION IF OPTIMAL, INFEASIBLE, or UNBOUNDED
SOLUTION IF ERROR CONDITION
SOLUTION FILE f (default $f = 0$)
The first four options determine whether the final solution obtained is to be output to the PRINT file. The FILE option operates independently; if $f > 0$, the final solution will be output to file f (whether optimal or not).

1. For the YES, IF OPTIMAL, and IF ERROR options, floating-point numbers are printed in F18.5 format, and "infinite" bounds are denoted by the word NONE.
2. For the FILE option, all numbers are printed in 1PE18.8 format, including "infinite" bounds which will have magnitude 1.000000E+20.
3. To see more significant digits in the printed solution, it will sometimes be useful to make f refer to the system PRINT file.

START OBJECTIVE CHECK AT COLUMN k (default $k = 1$)
START CONSTRAINT CHECK AT COLUMN k (default $k = 1$)
STOP OBJECTIVE CHECK AT COLUMN l (default $l = n_1'$)
STOP CONSTRAINT CHECK AT COLUMN l (default $l = n_1''$)

These keywords may be used to abbreviate the verification of individual gradient elements computed by subroutines FUNOBJ and FUNCON. For example:

1. If the first 100 objective gradients appeared to be correct in an earlier run, and if you have just found a bug in FUNOBJ 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 matrix.
2. If the first 100 variables occur nonlinearly in the constraints, and the remaining variables are nonlinear only in the objective, then FUNOBJ must set the first 100 components of G(*) to zero, but these hardly need to be verified. The above data card would again be appropriate.

These keywords are effective if VERIFY LEVEL > 0.

SUBSPACE TOLERANCE t (default $t = 0.5$)

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).

1. t must be a real number in the range $0.0 < t \leq 1.0$. It is used as follows.
2. When a nonbasic variable x_j is made superbasic, the resulting norm of the reduced-gradient vector (for all superbasics) is recorded. Let this be $\|Z^T g_0\|$. (In fact, the norm will be $|d_j|$, the size of the reduced gradient for x_j.)
3. Subsequent Phase 4 iterations will 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 component among the superbasic variables.)
4. 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 has to increase substantially between the starting point and an optimal solution.
5. 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.

SUMMARY FILE f (default $f = 0$)
SUMMARY FREQUENCY k (default $k = 100$)
If $f > 0$, a brief log will be output to file f, including one line of information every k-th iteration. In an interactive environment, it is useful to direct this output to the terminal, to allow a run to be monitored on-line. (If something looks wrong, the run can be manually terminated.) Further details are given in section 6.6.

SUPERBASICS LIMIT s (default s = HESSIAN DIMENSION, 30, or 1)
This specifies "how nonlinear" you expect a problem to be.

1. Normally, s need not be greater than $n_1 + 1$, where n_1 is the specified number of nonlinear variables.
2. For many problems (that are not highly nonlinear), s may be considerably smaller than n_1. This will save storage if n_1 is very large.
3. This parameter also sets the HESSIAN DIMENSION, unless the latter is specified explicitly (and conversely). If neither parameter is specified, both default to the value 30 (except if there are no nonlinear variables, in which case both default to 1).

SUPPRESS PARAMETERS
Normally MINOS prints the SPECS file as it is being read, and then prints a complete list of the available keywords and their final values. The SUPPRESS PARAMETERS option tells MINOS not to print the full list. (Both "SUP" and "PARA" are significant.)

UNBOUNDED OBJECTIVE VALUE F_{max} (default $F_{max} =$ 1.0E+20)
UNBOUNDED STEP SIZE α_{max} (default $\alpha_{max} =$ 1.0E+10)
These parameters are intended to detect unboundedness in nonlinear problems. (They may or may not achieve that purpose!) During a linesearch of the form

$$\min_{\alpha} F(x + \alpha p),$$

if $|F|$ exceeds F_{max} or α exceeds α_{max}, iterations are terminated with the exit message PROBLEM IS UNBOUNDED (OR BADLY SCALED).

1. 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 F_{max} can be made.
2. Unboundedness in x is best avoided by placing finite upper and lower bounds on the variables. (For convenience, this can be accomplished in the SPECS file; see the LOWER and UPPER BOUND parameters.)

UPPER BOUND u (default $u =$ 1.0E+20)
Before the BOUNDS section of the MPS file is read, all structural variables are given the default upper bound u. (Individual variables may subsequently have their upper bound altered by the BOUNDS section in the MPS file.)

VERIFY LEVEL	l	(default $l = 0$)

VERIFY	NO
VERIFY LEVEL	0
VERIFY OBJECTIVE GRADIENTS	
VERIFY LEVEL	1
VERIFY CONSTRAINT GRADIENTS	
VERIFY LEVEL	2
VERIFY	
VERIFY	YES
VERIFY GRADIENTS	
VERIFY LEVEL	3

These keywords refer to finite-difference checks on the gradient elements computed by the user subroutines FUNOBJ and FUNCON. It is possible to specify VERIFY LEVELs 0–3 in several ways, as indicated above. For example, the nonlinear objective gradients (if any) will be verified if either VERIFY OBJECTIVE or VERIFY LEVEL 1 is specified. Similarly, both the objective and the constraint gradients will be verified if VERIFY YES or VERIFY LEVEL 3 or just VERIFY is specified.

If $0 \leq l \leq 3$, gradients will be verified at the starting point. If $l = 0$, only a "cheap" test will be performed, requiring 3 calls to FUNOBJ or 2 calls to FUNCON. If $1 \leq l \leq 3$, a more reliable check will be made on individual gradient components, within the ranges specified by the START and STOP keywords. A key of the form "OK" or "BAD?" indicates whether or not each component appears to be correct.

Gradient checking occurs before the problem is scaled and before the first basis is factorized. (Hence, it occurs before the basic variables are reset to satisfy $Ax + Is = 0$.)

EMERGENCY gradient checking (at the end of an abortive run) is no longer performed.

1. VERIFY LEVEL 3 should be specified whenever a new function routine is being developed.
2. Missing gradients are not checked; i.e., they result in no overhead.
3. The default action is to perform a cheap check on the gradients at the first feasible point. Even on debugged function routines, the message "GRADIENTS SEEM TO BE OK" will provide certain comfort at nominal expense.
4. If necessary, checking can be suppressed by specifying VERIFY LEVEL -1.

WEIGHT ON LINEAR OBJECTIVE w (default $w = 0.0$)
This keyword invokes the so-called *composite objective* technique, if the first solution obtained is infeasible, and if linear terms for the objective function are specified in the MPS file. While trying to reduce the sum of infeasibilities, the method also attempts to optimize the linear objective.

1. At each infeasible iteration, the objective function is defined to be

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

   where $\sigma = 1$ for **MINIMIZE**, $\sigma = -1$ for **MAXIMIZE**, and c is the linear objective row.
2. 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 the sum of infeasibilities is tending towards zero.
3. The effect of w is disabled after 5 such reductions, or if a feasible solution is obtained.

WORKSPACE (USER) **maxw** (default **maxw** = 0)
WORKSPACE (TOTAL) **maxz** (default **maxz** = **NWCORE**)
These keywords define the limits of the region of storage that MINOS may use in solving the current problem. The main work array is declared in the main program, along with its length, by statements of the form

```
DOUBLE PRECISION   Z(25000)
DATA               NWCORE/25000/
```

where the actual length of Z must be specified at compile time. The values specified by the WORKSPACE keywords are stored in

```
COMMON     /M2MAPZ/ MAXW,MAXZ
```

and workspace may be shared according to the following rules:

1. Z(1) through Z(MAXW) is available to the user.
2. Z(MAXW+1) through Z(MAXZ) is available to MINOS, and should not be altered by the user.
3. Z(MAXZ+1) through Z(NWCORE) is unused (or available to the user).

The arrays LEN and LOC are not used by MINOS.

The WORKSPACE parameters are most useful on machines with a virtual (paged) store. Some systems will allow NWCORE to be set to a very large number (say 500000) with no overhead in saving the resulting object code. At run time, when various problems of different size are to be solved, it may be sensible to confine MINOS to a portion of Z to reduce paging activity slightly. (However, MINOS accesses storage contiguously wherever possible, so the benefit may be slight. In general it is far better to have too much storage than not enough.)

4. THE MPS FILE

An MPS file is required for all problems to specify names for the variables and constraints, and to define the constraints themselves. In contrast to the relatively free format allowed in the SPECS file, a very *fixed* format must be used for the MPS file. (This means that each item of data must appear in specific columns.)

Various "header cards" divide the MPS file into several sections as follows:

```
NAME
ROWS
 .
COLUMNS
 .
RHS
 .
RANGES    (optional)
 .
BOUNDS    (optional)
 .
ENDATA
```

Each header card must begin in column 1. The intervening card images (indicated by "." above) all have the following data format:

Columns	2–3	5–12	15–22	25–36	40–47	50–61
Contents	*Key*	*Name0*	*Name1*	*Value1*	*Name2*	*Value2*

In addition, "comment" cards are allowed; these have an asterisk "*" in column 1 and any characters in columns 2–22.

MPS format has become the industry standard. Files of this kind are recognized by all commercial mathematical programming systems (including MPS/360, MPSX, MPSX/370 and MPS III on IBM systems; APEX III and IV on CDC machines; FMPS on Univac systems; TEMPO on Burroughs systems). They may be created by hand, by your own special-purpose program, or by various commercial "matrix generators", such as GAMMA, MAGEN and OMNI.

Beware that variations are inevitable in almost any "standard" format. Some restrictions in the format accepted by MINOS are listed later. Some extensions are also needed for nonlinear problems.

4.1 The NAME Card

```
NAME          MODEL001    (for example)
```

This card contains the word NAME in columns 1–4, and a name for the problem in columns 15–22. (The name may be from 1 to 8 characters of any kind, or it may be blank.) The name is used to label the solution output, and it appears on the first card of each basis file.

The NAME card is normally the first card in the MPS file, but it may be preceded or followed by comment cards.

4.2 The ROWS Section

```
ROWS
 E  FUNO1
 G  FUNO2        (for example)
 L  CAPITAL1
 N  COST
```

The general constraints are commonly referred to as *rows*. The ROWS section contains one card for each constraint (i.e., for each row). *Key* defines what type the constraint is, and *Name0* gives the constraint an 8-character name. The various row-types are as follows:

Key	*Row-type*
E	$=$
G	$\geq$
L	$\leq$
N	Objective
N	Free

(*The 1-character Key* may be in column 2 or column 3.)

Row-types E, G and L are easily understood in terms of a linear function a^Tx and a right-hand side β. They would be used to specify constraints of the form

$$a^Tx = \beta, \qquad a^Tx \geq \beta \qquad \text{and} \qquad a^Tx \leq \beta$$

respectively. (Nonzero elements of the row-vector a will appear in appropriate parts of the COLUMNS section, and if β is nonzero it will appear in the RHS section.)

Row-type N stands for "Not binding", also known as "Free". It is used to define the *objective row*, and also to prevent a constraint from actually being a constraint. (Note that $-\infty \leq a^Tx \leq +\infty$ is not really a constraint at all. Type N rows are implemented by giving them infinite bounds of this kind.)

The *objective row* is a free row that specifies the vectors c and d in the objective function $F(x) + c^Tx + d^Ty$. It is taken to be the *first* free row, unless some other free row is specified by the OBJECTIVE keyword in the SPECS file.

The ROWS section need not contain any free rows if $c = d = 0$. If there are some nonlinear objective variables, the objective function will then be $F(x)$ as defined by subroutine FUNOBJ. Otherwise, *no* objective function exists and MINOS will terminate at the first point that satisfies the constraints.

If the ROWS section does contain free rows but none of them is intended to be an objective row, then some dummy name such as *OBJECTIVE = NONE* should be specified in the SPECS file to prevent the first free row from being selected. (If the objective function is $F(x)$ with no linear terms, *OBJECTIVE = FUNOBJ* would be a mnemonic reminder.)

Row-names for Nonlinear Constraints

The names of nonlinear constraints must be listed *first* in the ROWS section, and their order must be consistent with the computation of the array F(*) in subroutine *FUNCON*.

In particular, the objective row (if any) must appear after the list of nonlinear row names. For simplicity we suggest that potential objective rows be placed *last:*

```
ROWS
 G  FUNO1      nonlinear constraints first
 G  FUNO2
 .
 E  LINO1      now linear constraints
 E  LINO2
 .
 N  COSTO1     objective rows last
 N  COSTO2
```

4.3 The COLUMNS Section

```
1    5.....12  15....22  25........36   40....47  50........61  (fields)
COLUMNS
    XO1       FUNO6         1.0        ROW09        -3.0
    XO1       ROW08         2.5        ROW12         1.123456  (example)
    XO1       ROW03      -11.111111
    XO2       FUNO2         1.0
    XO2       COSTO1        5.0
```

For each variable x_j (say), the COLUMNS section defines a name for x_j and lists the nonzero entries a_{ij} in the corresponding column of the constraint matrix. The nonzeros for the first column must be grouped together before those for the second column, and so on. If a column has several nonzeros, it does not matter what order they appear in (as long as they all appear before the next column).

In general, *Key* is blank (except for comments), *Name0* is the column name, and *Name1*, *Value1* give a row name and value for some coefficient in that column. If there is another row name and value for the same column, they may appear as *Name2*, *Value2* on the same card, or they may be on the next card.

If either *Name1* or *Name2* is blank, the corresponding value is ignored.

Values are read by MINOS using Fortran format E12.0. This allows values to be entered in several forms; for example, 1.2345678, 1.2345678E+0, 123.45678E-2 and 12345678E-07 all represent the same number. It is usually best to include an explicit decimal point.

Beware that spaces within the value fields are the same as 0's (on most computer systems). In particular, this means that if an exponent like E-2 appears then it must be *right-justified* in the value field. For example, the two values

```
1.23E-02
1.23E-2
```

are *not* the same if the decimal point is in column 30 in both cases. The second value is actually 1.23E-20.

In the example above, the variable called XO1 has 5 nonzero coefficients in the constraints named FUNO6, ROW09, ROW08, ROW12 and ROW03. The row names and values may be in an arbitrary order, but they must all appear before the entries for column XO2.

There is no need to specify columns for the slack variables; they are incorporated implicitly.

Nonlinear Variables

Nonlinear variables must appear first in the COLUMNS section, ordered in a manner that is consistent with the array X(*) in the user subroutines FUNOBJ and/or FUNCON. In the example

$$\begin{array}{lll} \text{minimize} & (x+y+z)^2 + 3x + 5w & \\ \text{subject to} & x^2 + y^2 \quad + z \quad = 2 & \\ & x^4 + y^4 \quad\quad + w = 4 & \\ & 2x + 4y \quad\quad\quad \geq 0 & \\ & z \geq 0, \quad w \geq 0 & \end{array}$$

we have three nonlinear objective variables (x, y, z), two nonlinear Jacobian variables (x, y), one linear variable w, two nonlinear constraints, one linear constraint, and some simple bounds. The nonlinear constraints and variables should always be ordered in a similar way, at the *top left-hand corner* of the constraint matrix. The latter is therefore of the form

$$A = \begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix}$$

where J_k is the Jacobian matrix. The variables associated with J_k and A_2 must appear *first* in the COLUMNS section, and their order must be consistent with the array X(*) in subroutine FUNCON. Similarly, entries belonging to J_k must appear in an order that is consistent with the array G(*) in subroutine FUNCON.

For convenience, let the first n_1 columns of the constraint matrix be

$$\begin{pmatrix} J_k \\ A_2 \end{pmatrix} = \begin{pmatrix} j_1 \; j_2 \cdots j_{n_1} \\ a_1 \; a_2 \ldots a_{n_1} \end{pmatrix},$$

where j_1 is the first column of J_k and a_1 is the first column of A_2. The coefficients of j_1 and a_1 must appear before the coefficients of j_2 and a_2 (and so on for all columns). Usually, those belonging to j_1 will appear before any in a_1, but this is not essential. (If certain linear constraints are made nonlinear at a later date, this means that entries in the COLUMNS section will not have to be reordered. However, the corresponding row names will need be moved towards the top of the ROWS section.)

If JACOBIAN = DENSE, the elements of J_k need not be specified in the MPS file. If JACOBIAN = SPARSE, *all* nonzero elements of J_k must be specified. Any variable coefficients should be given a dummy value, such as zero. These dummy entries identify the location of the elements; their actual values will be computed later by subroutine FUNCON or by finite differences.

If all constraint gradients are known (DERIVATIVE LEVEL = 2 or 3), any Jacobian elements that are constant may be given their correct values in the COLUMNS section, and then they need not be reset by subroutine FUNCON. This includes values that are identically zero--such elements do not have to be specified anywhere (in the MPS file or in FUNCON). In other words, Jacobian elements are assumed to be zero unless specified otherwise.

Note that X(*) need not have the same dimension in subroutines FUNOBJ and FUNCON (i.e., the parameter N may differ), in the event that different numbers are specified by the NONLINEAR OBJECTIVE and NONLINEAR JACOBIAN keywords. However the shorter set of nonlinear variables must occur at the beginning of the longer set, and the ordering of variables in the COLUMNS section must match both sets.

A nonlinear objective function will often involve variables that occur only linearly in the constraints. In such cases we recommend that the objective variables be placed *after* the Jacobian variables in the COLUMNS section, since the Jacobian will then be as small as possible. (See the variable z in the example above.)

4.4 The RHS Section

```
1    5.....12  15....22  25........36   40....47  50........61
RHS
    RHS01     FUN01         1.0        ROW09        -3.0
    RHS01     ROW08         2.5        ROW12         1.123456
    RHS01     ROW03      -11.111111
    RHS02     FUN02         1.0
    RHS02     FUN04         5.0
```

This section specifies the elements of b_1 and b_2 in (2)–(3). Together these vectors comprise what is called the right-hand side. Only the nonzero coefficients need to be specified. They may appear in any order. The format is exactly the same as in the *COLUMNS* section, with *Name0* giving a name to the right-hand side.

If $b_1 = 0$ and $b_2 = 0$, the RHS header card must appear as usual, but no rhs coefficients need follow.

The RHS section may contain more than one right-hand side. The *first* one will be used unless some other name is specified in the SPECS file.

4.5 The RANGES Section (Optional)

```
1    5.....12  15....22  25........36   40....47  50........61
ROWS
 E  FUN01
 E  FUN02
 G  CAPITAL1
 L  CAPITAL2
 .
COLUMNS
 .
RHS
    RHS01     FUN01         4.0        FUN02         4.0
 .
RANGES
    RANGE01   FUN01         1.0        FUN02        -1.0
    RANGE01   CAPITAL1      1.0        CAPITAL2      1.0
```

Ranges are used for constraints of the form

$$l \leq a^T x \leq u,$$

where both l and u are finite. The range of the constraint is $r = u - l$. Either l or u is specified in the RHS section (as b say), and r is defined in the RANGES section. The resulting l and u depend on the row-type of the constraint and the sign of r as follows:

Row-type	*Sign of r*	*Lower limit, l*	*Upper limit, u*
E	+	b	$b + \|r\|$
E	−	$b - \|r\|$	b
G	+ or −	b	$b + \|r\|$
L	+ or −	$b - \|r\|$	b

The format is exactly the same as in the COLUMNS section, with *Name0* giving a name to the range set. The constraints listed above will have the following limits:

$$4.0 \leq \texttt{FUNO1} \leq 5.0,$$
$$3.0 \leq \texttt{FUNO2} \leq 4.0,$$
$$4.0 \leq \texttt{CAPITAL1} \leq 5.0,$$
$$3.0 \leq \texttt{CAPITAL2} \leq 4.0.$$

The RANGES section may contain more than one set of ranges. The *first* set will be used unless some other name is specified in the SPECS file.

4.6 The BOUNDS Section (Optional)

```
1    5.....12  15....22  25........36
BOUNDS
 UP BOUND01    X01           4.0
 UP BOUND01    X02           4.0
 .
 LO BOUND01    X04          -1.0
 UP BOUND01    X04           4.0
 .
 FR BOUND01    X06
 UP BOUND01    X06           4.0
 .
```

The default bounds on all variables x_j (excluding slacks) are $0 \leq x_j \leq \infty$. If necessary, the default values 0 and ∞ can be changed in the SPECS file to $l \leq x_j \leq u$ by the LOWER and UPPER keywords respectively.

If uniform bounds of this kind are not suitable, any number of alternative values may be specified in the BOUNDS section. As usual, several sets of bounds may be given, and the first set will be used unless some other name is specified in the SPECS file.

In this section, *Key* gives the type of bound required, *Name0* is the name of the bound set, and *Name1* and *Value1* are the column name and bound value. (*Name2* and *Value2* are ignored.)

Let l and u be the default bounds just mentioned, and let x and b be the column and value specified. The various bound-types allowed are as follows:

Key	*Bound-type*	*Resulting bounds*
LO	Lower bound	$b \leq x \leq u$
UP	Upper bound	$l \leq x \leq b$
FX	Fixed variable	$b \leq x \leq b$ (i.e., $x = b$)
FR	Free variable	$-\infty \leq x \leq +\infty$
MI	Minus infinity	$-\infty \leq x \leq u$
PL	Plus infinity	$l \leq x \leq +\infty$

The effect of the examples above is to give the following bounds:

$$l \leq \texttt{X01} \leq 4.0$$
$$l \leq \texttt{X02} \leq 4.0$$
$$-1.0 \leq \texttt{X04} \leq 4.0$$
$$-\infty \leq \texttt{X06} \leq 4.0$$

Note that types FR, MI, or PL should always be used to specify "infinite" bounds; they imply values of $\pm 10^{20}$, which are treated specially at certain times.

Nonlinear Problems

It is often essential to use bounds to avoid singularities in the nonlinear functions. For example, if an objective function involves $\log x_j$, a bound of the form $x_j \geq 10^{-4}$ may be necessary to avoid evaluating the objective function at zero or negative values of x_j. (Subroutine FUNOBJ is usually not called until a feasible point has been found. Note that x is regarded as feasible if it satisfies its bounds to within the FEASIBILITY TOLERANCE t. Thus, it would not be safe to specify the bound $x_j \geq 10^{-8}$ if t retained its default value $t = 10^{-6}$.)

Beware that subroutine FUNCON sometimes *will* be called before the nonlinear variables satisfy their bounds. If this causes difficulty, one approach is to specify feasible values for the offending variables in the INITIAL bounds set described next.

The INITIAL *Bounds Set*

In general, variables will initially have the value zero, if zero lies between the associated upper and lower bounds. Otherwise, the initial value will be the bound closest to zero.

The name INITIAL is reserved for a special bounds set that may be used to assign other initial values. The INITIAL bounds set must appear after any normal bound sets (if any); a warning is given if it is the first set encountered after the BOUNDS card.

The INITIAL bounds set also influences CRASH during construction of an initial basis. Broadly speaking, CRASH favors certain variables, ignores certain others, and treats the remainder as neutral. The following example illustrates the various cases:

```
FR INITIAL   X1          1.0
FX INITIAL   X2          2.0
LO INITIAL   X3
UP INITIAL   X4
MI INITIAL   X5          5.0
PL INITIAL   X6          6.0
```

1. During gradient checking and evaluation of the initial Jacobian, the value of X1 will be 1.0. X1 will then be favored by CRASH for inclusion in the initial basis. (Free rows and columns will also be favored.)

2. X2 will initially be superbasic at the value 2.0. (If the number of FX INITIALs has already reached the SUPERBASICS LIMIT, X2 will initially be nonbasic at the same value 2.0.)

3. X3 and X4 will initially be nonbasic at their respective lower and upper bounds (or at value zero if both bounds are infinite).

4. X5 and X6 will initially be nonbasic at the specified values 5.0 and 6.0.

The last five bound types (FX, LO, UP, MI, PL) prevent the associated variables from being included in the initial basis.

FR INITIAL or FX INITIAL should be used if good values are known for variables that are likely to lie between their bounds in an optimal solution. (Type FR is preferred if many such values are to be specified; however, the values may change when the basic variables are reset to satisfy $Ax + Is = 0$. Type FX guarantees the specified starting value, but should not be used excessively if the optimal solution is likely to be close to a vertex.)

LO INITIAL or UP INITIAL should be used for variables that are likely to be on their lower or upper bound at a solution.

MI INITIAL and PL INITIAL are included for completeness.

As with normal bound sets, variables may be listed in any order. (For each entry a linear search is made through the column names, starting at the name on the previous entry. Thus, for large problems it helps to follow the order of the variables in the COLUMNS section, at least to some extent.)

The INITIAL bounds set is ignored if a basis file is supplied.

4.7 Comment Cards

Any card in the MPS file may contain an asterisk "*" in column 1 and arbitrary data in columns 2–61. Such cards will be treated as comments. They will appear in the printer listing but will otherwise be ignored.

4.8 Restrictions and Extensions in MPS Format

1. Blanks are significant in the 8-character name fields. We recommend that all names be left-justified with no imbedded blanks. In particular, names referred to in the SPECS file *must be left-justified in the MPS file; for example, OBJECTIVE = COST02 specifies an 8-character* name whose last two characters are blank.
2. Comments ideally should use only columns 1–61 as noted above.
3. Scale factors cannot be entered in the ROWS section.
4. It does not matter if there is no row of type N.
5. There must be at least one row in the ROWS section, even for problems with no general constraints. (It may have row-type N.)
6. Nonlinear constraints must appear before linear constraints in the ROWS section.
7. Markers such as INTORG and INTEND are not recognized in the COLUMNS section.
8. Numerical values may be entered in E or F format. Spaces within the 12-character fields are treated as if they were 0's.
9. Nonlinear variables must appear before linear variables in the COLUMNS section.
10. If RANGES and BOUNDS sections are both present, the RANGES section must appear first.
11. In the BOUNDS section, if an UP entry specifies a zero upper bound, the corresponding lower bound is *not* affected. (Beware—in some MP systems, the lower bound is converted to $-\infty$.)
12. The bounds name INITIAL has a special meaning.

5. BASIS Files

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

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

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

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

NEW BASIS and BACKUP files are saved every k-th iteration, in that order, where k is the SAVE FREQUENCY.

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

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

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

Figures 5.1–5.3 illustrate the data formats used for BASIS files. 80-character fixed-length records are suitable in all cases. (36-character records would be adequate for PUNCH and DUMP files.) The files shown correspond to the optimal solution for the economic-growth model MANNE, described in section 8.4. Selected column numbers are included to define significant data fields. The problem has 10 nonlinear constraints, 10 linear constraints, and 30 variables.

5.1 NEW and OLD BASIS Files

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

As illustrated in Figure 5.1, the following information is recorded in a NEW BASIS file.

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

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

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

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

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

4. A set of cards of the form

$$j \qquad x_j$$

written with FORMAT(I8, 1PE24.14) and terminated by an entry with $j = 0$, where j denotes the j-th variable and x_j is a real value. The j-th variable is either the j-th column or the $(j-N)$-th slack, if $j > N$. Typically, HS(j) = 2 (superbasic). When nonlinear constraints are present, this list of superbasic variables is extended to include all basic nonlinear variables. The *Jacobian matrix can then be reconstructed exactly for a restart.*

Loading a NEW BASIS *file*

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

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

```
1.....8       15.....23      29........40  43....50       57....................80

MANNE10        ITN    11      OPTIMAL SOLN  NINF   0      OBJ  -2.6700976576430 00
OBJ=FUNOBJ     RHS=RHS        RNG=RANGE1    BND=BOUND1     M=   20  N=   30 SB=   7
03222222230333333333333333333111111111110000000000
        3    3.214430306846170 00
        4    3.304004540903450 00
        5    3.395219987011400 00
        6    3.487878208733720 00
        7    3.581722961684240 00
        8    3.676428591145790 00
        9    3.771582587441020 00
        1    3.050000000000000 00
        2    3.126650351567880 00
       10    3.866666666666670 00
       11    9.500000000000000-01
       12    9.684180638592470-01
       13    9.978010109641690-01
       14    1.028200569133170 00
       15    1.059670152206730 00
       16    1.092272224137000 00
       17    1.126076354918100 00
       18    1.161163958088100 00
       19    1.197628149454330 00
       20    1.213943080245590 00
        0

1.....8   12.................32
```

Figure 5.1. Format of NEW and OLD BASIS files

Warning: This format is not quite compatible with MINOS 4.0 in the following respects.

1. On the second card, `M` is the number of constraints (m, as before) but `N` is now the number of variables excluding slacks (i.e., n, the number of columns in the MPS file plus the number of phantom columns, if any). Previously, `N` had the value $n + 1 + m$; this included 1 for the right-hand side and m for the slacks.
2. The basis map starting at card 3 does not contain an entry for the right-hand side, which was previously in position $n + 1$. The length of the map is now $n + m$, not $n + 1 + m$.
3. In the list of (j x_j) entries, the values of j referring to slacks are now one less than before. (These are entries for which $j > n$.)

A basis map from MINOS 4.0 can therefore be converted to the present format with reasonable ease. PUNCH and DUMP files from MINOS 4.0 should be acceptable as INSERT and LOAD files without change.

5.2 PUNCH and INSERT Files

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

The standard MPS format has been slightly generalized to allow the saving and reloading of nonbasic solutions. It is illustrated in Figure 5.2. Apart from the first and last card, each entry has the following form:

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

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

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

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

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

Notes on PUNCH *Data*

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

Notes on INSERT *Data*

1. Before an INSERT file is read, column variables are made nonbasic at their smallest bound in absolute magnitude, and the slack variables are made basic.
2. Preferably an INSERT file should be an unmodified PUNCH file from an earlier run on the same problem. If some rows have been added to the problem, the INSERT file need not be altered. (The slacks for the new rows will be in the basis.)

3. Entries will be ignored if *Name1* is already basic or superbasic. XL and XU cards will be ignored if *Name2* is not basic.
4. SB cards may be added before the ENDATA card, to specify additional superbasic columns or slacks.
5. An SB card will not alter the status of *Name1* if the *SUPERBASICS LIMIT has been reached.* However, the associated *Value* will be retained if *Name1* is a Jacobian variable.

5.3 DUMP and LOAD Files

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

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

as illustrated in Figure 5.3. The keys LL, UL, BS and SB mean *Lower Limit, Upper Limit, Basic* and Superbasic respectively.

Notes on DUMP Data

1. A card is output for every variable, columns followed by slacks.
2. Nonbasic free variables will be output with either LL or UL keys and with *Value* zero.

Notes on LOAD *Data*

1. Before a LOAD file is read, all columns and slacks are made nonbasic at their smallest bound in absolute magnitude. The basis is initially empty.
2. Each LL, UL or BS card causes *Name* to adopt the specified status. The associated *Value* will be retained if *Name* is a Jacobian variable.
3. An SB card causes *Name* to become superbasic at the specified *Value.*
4. An entry will be ignored if *Name* is already basic or superbasic. (Thus, only the first BS or SB card takes effect for any given *Name.*)
5. An SB card will not alter the status of *Name* if the SUPERBASICS LIMIT has been reached, but the associated *Value* will be retained if *Name* is a Jacobian variable.
6. (*Partial basis*) Let M be the number of rows in the problem. If fewer than M variables are specified to be basic, a tentative basis list will be constructed by adding the requisite number of slacks, starting from the first row and taking those that were not previously specified to be basic or superbasic. (If the resulting basis proves to be singular, the basis factorization routine will replace a number of basic variables by other slacks.) The starting point obtained in this way will not necessarily be "good".
7. (*Too many basics*) If M variables have already been specified as basic, any further BS keys will be treated as though they were SB. This feature may be useful for combining solutions to smaller problems.

```
1   5.....12  15....22  25........36

NAME          MNEMIC    PUNCH/INSERT
 LL KAP001              3.050000 00
 XU KAP002    MON001    3.126650 00
 SB KAP003              3.214430 00
 SB KAP004              3.304000 00
 SB KAP005              3.395220 00
 SB KAP006              3.487860 00
 SB KAP007              3.581720 00
 SB KAP008              3.676430 00
 SB KAP009              3.771580 00
 XU KAP010    MON002    3.866670 00
 LL CON001              9.500000-01
 XU CON002    MON003    9.684180-01
 XU CON003    MON004    9.978010-01
 XU CON004    MON005    1.028200 00
 XU CON005    MON006    1.059670 00
 XU CON006    MON007    1.092270 00
 XU CON007    MON008    1.126480 00
 XU CON008    MON009    1.161160 00
 XU CON009    MON010    1.197630 00
 XL CON010    CAP002    1.213940 00
 XL INV001    CAP003    7.665840-02
 XL INV002    CAP004    8.778800-02
 XL INV003    CAP005    8.957420-02
 XL INV004    CAP006    9.121540-02
 XL INV005    CAP007    9.265820-02
 XL INV006    CAP008    9.384480-02
 XL INV007    CAP009    9.478560-02
 XL INV008    CAP010    9.515400-02
 XL INV009    TERMINV   9.508410-02
 UL INV010              1.160000-01
ENDATA
```

Figure 5.2. Format of PUNCH and INSERT files

```
1   5.....12  15....22  25........36

NAME          MNEMIC      DUMP/LOAD
 LL KAP001                3.050000 00
 XU KAP002                3.126650 00
 SB KAP003                3.214430 00
 SB KAP004                3.304000 00
 SB KAP005                3.395220 00
 SB KAP006                3.487860 00
 SB KAP007                3.581720 00
 SB KAP008                3.676430 00
 SB KAP009                3.771580 00
 XU KAP010                3.866670 00
 LL CON001                9.500000-01
 XU CON002                9.684180-01
 XU CON003                9.978010-01
 XU CON004                1.028200 00
 XU CON005                1.059670 00
 XU CON006                1.092270 00
 XU CON007                1.126480 00
 XU CON008                1.161160 00
 XU CON009                1.197630 00
 XL CON010                1.213940 00
 XL INV001                7.665840-02
 XL INV002                8.778800-02
 XL INV003                8.957420-02
 XL INV004                9.121540-02
 XL INV005                9.265820-02
 XL INV006                9.384480-02
 XL INV007                9.478560-02
 XL INV008                9.515400-02
 XL INV009                9.508410-02
 UL INV010                1.160000-01
 UL MON001                0.000000-01
 UL MON002                0.000000-01
 UL MON003                0.000000-01
 UL MON004                0.000000-01
 UL MON005                0.000000-01
 UL MON006                0.000000-01
 UL MON007                0.000000-01
 UL MON008                0.000000-01
 UL MON009                0.000000-01
 UL MON010                0.000000-01
 LL CAP002                0.000000-01
 LL CAP003                0.000000-01
 LL CAP004                0.000000-01
 LL CAP005                0.000000-01
 LL CAP006                0.000000-01
 LL CAP007                0.000000-01
 LL CAP008                0.000000-01
 LL CAP009                0.000000-01
 LL CAP010                0.000000-01
 LL TERMINV               0.000000-01
ENDATA
```

Figure 5.3. Format of DUMP and LOAD files

5.4 Restarting Modified Problems

Sections 5.1–5.3 document three distinct starting methods (OLD BASIS, INSERT and LOAD files), which may be preferable to any of the cold start (CRASH) options. The best choice depends on the extent to which a problem has been modified, and whether it is more convenient to specify variables by number or by name. The following notes offer some rules of thumb.

Protection

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

Default Status

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

Restarting with Different Bounds

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

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

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

In contrast, if X is nonbasic at its *upper* bound and if that bound is altered, the starting values for an arbitrary number of basic variables could be changed (since they will be recomputed from the nonbasic and superbasic variables). This may not be of great consequence, but sometimes it may be worthwhile to retain the old solution precisely. To do this, one must make X superbasic at the original bound value.

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

```
 j                        5.0
```

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

```
 SB X                     5.0
```

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

Sequences of Problems

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

Altering Bounds with the CYCLE *Option*

Sequences of problems will sometimes be defined in conjunction with the CYCLE facilities. Various alterations can be made to each problem from within your own subroutine **MATMOD**. In particular, it is straightforward to alter the bounds on any of the columns or slacks.

Unfortunately, the present implementation of MINOS does not make it easy to alter the set of superbasic variables from within **MATMOD**. If the bound on a nonbasic variable is altered, it is simplest to accept the resulting perturbation to the values of the basic variables (rather than making the variable superbasic as suggested above).

6. OUTPUT

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

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

The last four items are described in the following sections. Further brief output may be directed to the SUMMARY file, as discussed in section 6.6.

6.1 Iteration Log

One line of information is output to the PRINT file every k-th minor iteration, where k is the specified LOG 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). Normally just one variable is selected, which we will denote 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_{PP} or I_{PP}, the PP-th segments of the constraint matrix $(A \quad I)$. If MULTIPLE PRICE is in effect, *several* variables may be selected from A_{PP} or I_{PP}. In this case, JQ refers to the variable with the largest favorable reduced cost, DJ.

Label	*Description*
ITN	The current iteration number. For problems with nonlinear constraints, this is the cumulative number of minor iterations.
PH	The current phase of the solution procedure, as follows:
1	Phase 1 of the simplex method is being used to find a feasible point.
2	Phase 2 of the simplex method is being used to optimize the linear objective.
	Normally, Phase 1 and 2 are used for purely linear problems. They may also be used at the start of a run even for nonlinear problems, if the initial basis contains only linear variables. Any superbasic variables will be temporarily held at their initial values.

3 Phase 3 of the reduced-gradient procedure is being used. This is the same as Phase 4 except that a PRICE operation is performed prior to the iteration, adding one or more nonbasic variables to the superbasic set.

4 Phase 4 of the reduced-gradient procedure is being used. Optimization is performed on the basic and superbasic variables (ignoring the nonbasics).

PP The Partial Price indicator. The variable(s) 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. It is reset during Phase 1, 2 or 3.

NOPT The number of "non-optimal" variables present in the set of nonbasic variables that were scanned during the last PRICE operation. It is reset during Phase 1, 2 or 3.

DJ,RG In Phase 1, 2 or 3, 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 j-th column of the constraint matrix $(A \;\; I)$.

In Phase 4, this quantity is RG, the norm of the reduced-gradient vector *after* the present iteration. (It is the largest value of $|d_j|$ for variables j in the superbasic set.)

Note that for Phase 3 iterations, 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. (This is zero in Phase 4.)

-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 basic and superbasic variables x_{BS} have just been changed to $x_{BS} + \alpha p$.

PIVOT If column a_q replaces the r-th column of the basis B, PIVOT is the r-th 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 workspace available to MINOS should be increased by a significant amount. As a suggestion, the work array Z(*) should be extended by $L + U$ elements.

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 will usually decrease 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. Note that "current objective function" can mean different things when NINF $= 0$. 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 function for the current major iteration (which tends to the true objective function as convergence is approached).

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 nonbasic).

Label	*Description*

NCON The number of times subroutine FUNCON has been called to evaluate the nonlinear constraint functions.

NOBJ The number of times subroutine FUNOBJ has been called to evaluate the nonlinear objective function.

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.)

H-CONDN 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 matrix $R^T R$ that approximates the reduced Hessian. H-CONDN 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 reduced-gradient algorithm will make slow progress if H-CONDN becomes as large as $\epsilon^{-1/2}$, and will probably fail to find a better solution if H-CONDN reaches $\epsilon^{-3/4}$ or larger. (On IBM-like machines, these values are about 10^8 and 10^{12}.)

To guard against high values of H-CONDN, 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.*

In the present implementation, 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 (see section 3.3).

6.2 Basis Factorization Statistics

The following items are output whenever the basis matrix B is factored. Gaussian elimination is used to compute an LU factorization of the form

$$B = LU,$$

where L is unit lower triangular and PUQ is upper triangular for some permutation matrices P and Q. This factorization is stabilized in the manner described under LU FACTOR TOLERANCE in section 3.3.

Label	*Description*
FACTORIZE	The number of factorizations since the start of the run.
DEMAND	A code giving the reason for the present factorization. (Since this is not important to the user we omit details.)
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. If there are nonlinear constraints, it is the value of the augmented Lagrangian for the present subproblem.
NONLINEAR	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.
ELEMS	The number of nonzero matrix elements in B.
DENSITY	The percentage nonzero density of B, 100 $\times$ ELEMS/(M $\times$ M), where M is the number of rows in the problem (M $=$ NONLINEAR + 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 MINOS 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 IBM-like machines, each nonzero is represented by one REAL*8 and two INTEGER*2 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., $100 \times (LENL + LENU - ELEMS)/ELEMS$.

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.)

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.)

6.3 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.

System Note: A number is associated with each message below. It is the final value assigned to the integer variables **INFORM** and **IERR**, for possible use within subroutines **MINOS1** and **MINOS2**. The variables appear in the declarations

```
SUBROUTINE MINOS2( Z,NWCORE,NCALLS,INFORM )
```

and

```
COMMON     /M5LOG1/ IDEBUG,IERR,LPRINT
```

If a problem is infeasible, for example, their final values will be **INFORM = IERR = 1**.

> 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 ON UNIT nn

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

Otherwise, the SPECS file may be empty, or cards containing the keywords **SKIP** or **ENDRUN** may imply that all problems should be ignored (see section 1.8).

-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 optimization terminates gracefully. A solution exists, any of the BASIS files may be saved, and the solution will be printed and/or saved on the *SOLUTION file* if requested.

0. EXIT -- OPTIMAL SOLUTION FOUND

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 astonishingly new high (or low). Or perhaps it will signal the end of a strenuous series of runs that have iterated far into the night, depleting one's patience and computing funds to an equally alarming degree. (We hope not!)

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 or is entered in the wrong columns of an input record. There may be thousands of items of data in the MPS file, and the nonlinear functions (if any) could depend on input files and other

data in innumerable ways. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder. For early runs, we suggest that the LIST LIMIT be set to a suitably large number to allow the MPS file to be printed for visual checking. It is also good practice in the function subroutines to print any data that is read in on 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 modellers to *know something about their problem*, and to make use of that knowledge as they themselves know best.

One other caution about "OPTIMAL SOLUTION"s. When nonlinearities are present, the final size of the reduced-gradient norm (NORM RG) should be examined to see if it is reasonably small compared to the norm of the dual variables (NORM PI). These quantities are printed following the EXIT message. MINOS attempts to ensure that

$$\texttt{NORM RG} \;/\; \texttt{NORM PI} \leq \texttt{OPTIMALITY TOLERANCE}.$$

However, if messages of the form XXX SEARCH TERMINATED occur at the end of the run, this condition will probably not have been satisfied. The final solution may or may not be acceptably close to optimal. Broadly speaking, if

$$\texttt{NORM RG} \;/\; \texttt{NORM PI} = 10^{-d},$$

then the objective function would probably change in the d-th significant digit if optimization could be continued. One must judge whether or not d is sufficiently large.

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. 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 MINOS recognizes the situation and exits. There may exist other points that have a significantly lower sum of infeasibilities.

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, MINOS is prepared to relax the bounds on the slacks associated with nonlinear rows. In the current implementation, the bounds are relaxed by increasingly large amounts up to 5 times per major iteration. Normally a feasible point

will be obtained to the perturbed constraints, and optimization can continue on the subproblem. However, if 5 consecutive subproblems require such perturbation, the problem is terminated and declared **INFEASIBLE**. Clearly this is an ad hoc procedure. 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)

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 – see section 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.

3. 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 a basis file that was saved (*or should* have been saved!) at the end of the run.

4. EXIT -- THE OBJECTIVE HAS NOT CHANGED FOR THE LAST nnn ITERATIONS

This is an emergency measure for the rare occasions when the solution procedure appears to be *cycling*. Suppose that a zero step is taken for several consecutive iterations, with a basis change occurring each time. It is theoretically possible for the set of basic variables to become the same as they were one or more iterations earlier. The same sequence of iterations would then occur *ad infinitum*.

No direct attempt is made to recognise such cycling. The method used for determining the step size tends to guard against it happening, but nothing is guaranteed. Furthermore, on so-called degenerate models (in which many basic variables are equal in value to their upper or lower bounds), a great number of consecutive zero steps may *have* to occur before any progress can be made. A generous limit is therefore used on the number of consecutive zero steps allowed before this exit is taken. For small problems, the limit nnn is the maximum of 200 and $2(m+n)$. For large problems ($m+n \geq 1000$) it is 1000.

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 $1/2s^2$ 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 -- REQUESTED BY USER IN SUBROUTINE FUNOBJ (or FUNCON)
AFTER nnn CALLS

This exit occurs if the subroutine parameter **MODE** is set to a negative number 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. For example, you may have asked the operating system to

LINK MINOS, FUNOBJ, FUNCON

when in fact you should have said

LINK FUNOBJ, FUNCON, MINOS

(or something similar) to give your own subroutines priority. Most linkers or loaders retain the *first* version of any subprogram that they see.

7. EXIT -- *SUBROUTINE FUNOBJ SEEMS TO BE GIVING INCORRECT GRADIENTS*

A check has been made on some individual elements of the gradient array, and at least one component **G**(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*. A simple omission (such as *forgetting to divide* **F** *by 2) could explain everything. If* **F** *or* **G**(j) *is very large, then give serious* thought to scaling the function or the nonlinear variables.

If you feel *certain* that the computed **G**(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 is likely to terminate unsuccessfully.

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 **F** and **G** are being set correctly in subroutine **FUNCON**.

9. EXIT -- THE CURRENT POINT CANNOT BE IMPROVED UPON

Several circumstances could lead to this exit.

1. Subroutine **FUNOBJ** and/or subroutine **FUNCON** could be returning accurate function values but inaccurate gradients (or *vice versa*). This is the most likely cause. Study the comments given for exits 7 and 8, and do your utmost to ensure that the subroutines are coded correctly.

2. The function and gradient values could be consistent, but their precision could be too low. For example, accidental use of a single-precision data type when double-precision was intended throughout, 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
   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 -- NUMERICAL ERROR. GENERAL CONSTRAINTS CANNOT BE SATISFIED ACCURATELY**
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 section 6.2, and set the **LU FACTOR TOLERANCE** to 2.0 (or possibly even smaller, but not less than 1.0).

11. **EXIT -- CANNOT FIND SUPERBASIC TO REPLACE BASIC VARIABLE**
If this exit occurs, the problem must be very badly scaled. A basic variable has reached a bound and must be replaced, but none of the superbasic columns has a pivot element exceeding the **PIVOT TOLERANCE**. The latter could be reduced (at great risk). You should first try specifying **SCALE**.

12. **EXIT -- BASIS FACTORIZATION REQUESTED TWICE IN A ROW**
This exit may occur after the linesearch has terminated unsuccessfully several times in a row. It is a safeguard to prevent the various recovery measures from being repeated endlessly. It should probably be treated as if it were exit 9.

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

20. EXIT -- NOT ENOUGH STORAGE FOR THE BASIS FACTORIZATION

The main storage array Z(*) is apparently not large enough for this problem. The routine declaring Z is probably the main program. It should be recompiled with a larger dimension for Z. The new value should also be assigned to NWCORE.

In some cases it may be sufficient to increase the specified WORKSPACE (USER), if it is currently less than WORKSPACE (TOTAL).

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

21. EXIT -- ERROR IN BASIS PACKAGE

A preceding message will describe the error in more detail. One such message says that the current basis has more than one element in row i and column j. This could be caused by a corresponding error in the MPS file. (MINOS does not check for duplicate row names within each column.) Determine the name of row i (e.g., by consulting the i-th entry in the rows section of the printed solution), and scan the COLUMNS section of the MPS file for that name. Alternatively, check the $(j-l)$-th variable in the COLUMNS section of the MPS file, where l is the number of slack variables in the basis.

22. EXIT -- THE BASIS IS STRUCTURALLY SINGULAR AFTER TWO FACTORIZATION ATTEMPTS

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

If the following messages arise, the MPS file was read successfully. However, 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 card of the OLD BASIS file, the dimensions labelled M and N are different from those associated with the MPS file that has just been read. You have probably loaded a file that belongs to some other problem.

Remember, if you have added rows or columns to the MPS file, you will have to alter M and N and the map beginning on the third card (a hazardous operation). It may be easier to restart with a PUNCH or DUMP file from the earlier version of the problem.

31. EXIT -- THE BASIS FILE STATE VECTOR DOES NOT MATCH THIS PROBLEM
For some reason, the OLD BASIS file is incompatible with the present problem, or is not consistent within itself. The number of basic entries in the state vector (i.e., the number of 3's in the map) is not the same as M on the first card, or else some of the 2's in the map did not have a corresponding j x_j entry following the map.

32. EXIT -- SYSTEM ERROR. WRONG NO. OF BASIC VARIABLES... nnn
This exit should never happen. If it does, something is seriously awry in the MINOS source code. Perhaps the single- and double-precision files have been mixed up.

The following messages arise if the MPS file is seriously deficient, or if additional storage is needed to allow the MPS file to be input or to allow optimization to begin. The problem is abandoned.

40. EXIT -- FATAL ERRORS IN THE MPS FILE
One of the following conditions exists:

1. There are no entries in the ROWS section.
2. There are no entries in the COLUMNS section.
3. A type N row has been selected to be the linear objective row, but it is one of the first m_1 rows, where m_1 is the number of NONLINEAR CONSTRAINTS.

The first two conditions speak for themselves. If condition 3 occurs, the N row may be have been selected by default (if you did not specify any OBJECTIVE name in the SPECS file). To prevent this, specify some other (possibly fictitious) row name. Otherwise, you must put the N row *after* the nonlinear row names in the ROWS section.

41. EXIT -- NOT ENOUGH STORAGE TO READ THE MPS FILE
One of the ROWS, COLUMNS, or ELEMENTS estimates in the SPECS file proved to be too small. The minimum (exact) values are shown in earlier messages. You must specify these values, or higher values, and re-run the problem.

If the MPS data had been on a file of its own (rather than in the card input stream), MINOS would have been able to continue by rewinding the MPS file and trying again.

42. EXIT -- NOT ENOUGH STORAGE TO START SOLVING THE PROBLEM
The MPS file was read successfully, but the main storage array Z(*) is not large enough to provide workspace for the optimization procedure. Be sure that the SUPERBASICS LIMIT and HESSIAN DIMENSION are not unreasonably large. Otherwise, see the advice given for exit 20.

6.4 Solution Output

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

ROWS Section

The general constraints take the form $l \leq f(x) + Ay \leq u$, where x and y are the nonlinear and linear variables respectively. The i-th constraint is therefore of the form

$$\alpha \leq f^i(x) + a^T y \leq \beta,$$

and we define the i-th "row" to be the linearization of $f^i(x) + a^T y$. For linear constraints, the i-th row is just $a^T y$.

Internally, the constraints take the form $Lf(x) + Ay + s = 0$ where $Lf(x)$ is the current linearization of $f(x)$, and s is the set of slack variables (which happen to satisfy the bounds $-u \leq s \leq -l$). For the i-th constraint it is the slack variable s_i that is directly available, and it is sometimes convenient to refer to its state.

Label	*Description*
NUMBER	The value $n + i$. This is the internal number used to refer to the i-th slack in the iteration log.
ROW	The name of the i-th row.
STATE	The state of the i-th 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 row is equal to the RHS element, $\alpha = \beta$.
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 $a^T y$ for linear constraints, or the value of the linearization $Lf'(x) + a^T y$ if the constraint is nonlinear.

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 i-th 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.

If the solution is feasible, the first m_1 components of π are used at the start of the k-th major iteration to define λ_k, the estimate of the Lagrange multipliers for the nonlinear constraints.

I The constraint number, i.

COLUMNS Section

Here we talk about the "column variables" (x, y). For convenience we let the j-th component of (x, y) be the variable x_j and assume that it satisfies the bounds $\alpha \le x_j \le \beta$. Linear and nonlinear variables are treated the same.

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 j-th component of the combined linear and nonlinear objective function $F(x) + c^T x + d^T y$. (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 j-th column of the constraint matrix (or the j-th 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 chapter 8. 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 $F(x)$ 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.

6.5 SOLUTION File

If a positive SOLUTION FILE is specified, the information contained in a printed solution may also be output to the relevant file (which may be the PRINT file if so desired). Infinite UPPER and LOWER LIMITS appear as $\pm 10^{20}$ rather than NONE. Other real values are output with format 1PE16.6. Again, the maximum record length is 111 characters, including what would be the carriage-control character if the file were printed.

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

```
FORMAT(I8, 2X, 2A4, 1X, A1, 1X, A3, 5E16.6, I7)
```

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

6.6 SUMMARY File

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

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

The following information is included:

1. The **BEGIN** card from the SPECS file.
2. The actual number of rows, columns and elements in the MPS file.
3. The basis file loaded, if any.
4. The status of the solution after each basis factorization (whether feasible; the objective value; the number of function calls so far).
5. The same information every k-th iteration, where k is the specified **SUMMARY FREQUENCY** (default $k = 100$).
6. Warnings and error messages.
7. For nonlinear constraints, $\|x_{k+1} - x_k\|$, $\|\lambda_{k+1} - \lambda_k\|$ and the norm of the nonlinear constraint violation at the start of each major iteration.
8. 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 in Figure 6.1, which shows the SUMMARY file for the test problem MANNE, using **SUMMARY FREQUENCY** 1.

```
--------------------
MINOS 5.1 (Jan 1987)
--------------------

BEGIN MANNE10

SCALE OPTION  0

NAME    MANNE10
ROWS          20
XXXX  Warning - no linear objective selected
COLUMNS       30
ELEMENTS      59
XXXX  Warning - the RHS is zero

XXXX  Total no. of errors in MPS file     2

***  FUNCON  sets        7   out of       10   constraint gradients.

***  FUNOBJ  sets       17   out of       20   objective  gradients.

------------------------------------------------------------
START OF MAJOR ITN   1           PENALTY PARAMETER =    1.00E-01
Constraint violation  =  0.0000E+00

  Itn  Nopt  Ninf  Sinf/Objective  Nobj  Ncon  NSB
    1    -1     1  1.00000000E-03    28    13    8
    2    -1     0  2.66982754E+00    38    23    8
Optimal subproblem at minor itn   2  -  Total itns =     2
```

```
------------------------------------------------------------
START OF MAJOR ITN    2          PENALTY PARAMETER =    1.00E-01
Change in Jacobn vars =  3.3333E-02
Change in multipliers =  9.8643E+00
Constraint violation  =  9.1735E-06

Multiplier estimates
--------------------
  9.9838772E-01   9.1861799E-01   8.6971638E-01   7.9949979E-01   7.3521971E-01

  Itn  Nopt  Ninf  Sinf.Objective  Nobj  Ncon  NSB
    3    -1     0  2.6698282E+00    41    34    7
    4    -1     0  2.6700226E+00    48    41    7
Optimal subproblem at minor itn   2  -  Total itns =     4

------------------------------------------------------------
START OF MAJOR ITN    3          PENALTY PARAMETER =    1.00E-01
Change in Jacobn vars =  1.6781E-02
Change in multipliers =  1.4206E-02
Constraint violation  =  2.7679E-06

Multiplier estimates
--------------------
  1.0116027E+00   9.3802418E-01   8.6184468E-01   7.8993208E-01   7.3893361E-01


COMPLETION  FULL     requested as from now.

  Itn  Nopt  Ninf  Sinf.Objective  Nobj  Ncon  NSB
    5    -1     0  2.6700636E+00    61    54    7
    6     0     0  2.6700636E+00    66    59    7
    7     0     0  2.6700680E+00    71    64    7
    8     0     0  2.6700980E+00    79    72    7
    9     0     0  2.6700976E+00    86    79    7

  Itn  Nopt  Ninf  Sinf.Objective  Nobj  Ncon  NSB
   10     0     0  2.6700976E+00    96    87    7
   11     0     0  2.6700976E+00   100    93    7
Optimal subproblem at minor itn   7  -  Total itns =    11

------------------------------------------------------------
START OF MAJOR ITN    4          PENALTY PARAMETER =    0.00E+00
Change in Jacobn vars =  1.5231E-02
Change in multipliers =  5.7251E-03
Constraint violation  =  2.8170E-06

Multiplier estimates
--------------------
  1.0106344E+00   9.3193151E-01   8.5986301E-01   7.9816423E-01   7.3818655E-01

  Itn  Nopt  Ninf  Sinf.Objective  Nobj  Ncon  NSB
   12    -1     0  2.6700976E+00   110   103    7
Optimal subproblem at minor itn   1  -  Total itns =    12

------------------------------------------------------------
START OF MAJOR ITN    5          PENALTY PARAMETER =    0.00E+00
Change in Jacobn vars =  4.0111E-06
Change in multipliers =  9.6670E-07
Constraint violation  =  1.4360E-13

Multiplier estimates
--------------------
  1.0106358E+00   9.3193822E-01   8.5986440E-01   7.9816476E-01   7.3820902E-01
1
EXIT -- OPTIMAL SOLUTION FOUND

NEW BASIS FILE saved on file  11    Itn =   12

Major, Minor Itns              5          12
Objective function      2.6700976576430E+00
FUNOBJ, FUNCON calls          110         103
Superbasics, Norm RG            7    2.42E-09
Norm X,      Norm PI    6.53E+00    7.61E+00
Constraint violation    1.43E-13    1.99E-14

Solution printed

FUNCON called with NSTATE =   2

FUNOBJ called with NSTATE =   2
```

7. SYSTEM INFORMATION

7.1. Distribution Files

The MINOS source code and test problems are distributed as a set of Fortran and data files.

- For installation instructions, please see file **miminos.doc**.
- Certain other *.doc files give information for specific machines.
- File **readme** lists changes not documented elsewhere.

Troubleshooting

If you encounter difficulty with compiling or linking, please check the following items. The Fortran files are referred to here by names of the form *.for. (On Unix systems, they are renamed *.f.)

1. Most current machines require double-precision arithmetic. Check that the Fortran files use appropriate declarations. For example, file **mi00main.for** should contain the line
   ```
   implicit          double precision (a-h,o-z)
   ```
   Single precision is correct on a few machines (notably Cray and Convex). These use
   ```
   implicit          real (a-h,o-z)
   ```
   throughout.

2. File **mi00main.for** declares an array **z(nwcore)** for MINOS to use as workspace. Make **nwcore** as large as possible, bearing in mind the maximum problem size that is likely to be encountered. Very roughly, linear programs with m rows may require **nwcore** $\geq 100m$.

3. File **mi05funs.for** contains nonlinear function routines for the supplied test problems. Use this file initially to run the test cases, but replace it later with your own functions.

4. On most machines, use file **mi10unix.for**. Check a few machine-dependencies in the following subroutines. The requirements are described in the source code.
   miopen opens files.
   miinit sets the machine precision, **eps**. Typically 2^{-52} = 2.22d-16 in IEEE arithmetic.
   micpu calls the system timer. On some Unix systems, the timer is **etime**. If the name is unknown, set **time** = -1.0 as shown in the source code.

5. For DEC OpenVMS systems, use file **mi10vms.for**. All machine-dependent subroutines are ready to go. In addition, **minos2** uses dynamic memory allocation.

6. In file **mi35inpt.for**, subroutine **m3hash** is suitable for most machines. In rare cases it may need to be altered if MPS data files are not input correctly. Again, the requirements are described in the source code.

7.2. Source Files

The Fortran source code is divided into several files, each containing several subroutines or functions. The naming convention used should minimize the risk of a clash with user-written routines.

`mi00main.for` Main program for Stand-alone MINOS.

```
Program MINOS
```

`mi05funs.for` Function routines for test problems.

```
funobj   funcon   matmod
t2obj    t3obj    t4obj    t4con    t5obj    t6con    t7obj
```

`mi10unix.for` Machine-dependent routines. (Use `mi10vms.for` for OpenVMS.)

```
minoss   minos1   minos2   minos3
mifile   mispec   misolv
miclos   mienvt   miinit
miopen   mipage   mitime   mitimp   micpu
```

`mi15blas.for` Basic Linear Algebra Subprograms (a subset).

```
dasum    daxpy    dcopy    ddot     dnrm2    dscal    idamax
```

These routines are members of the Level 1 BLAS (Lawson, *et al.*, 1979). It may be possible to replace them by versions that have been tuned to your particular machine.

Single-precision versions of MINOS use `sasum`, `saxpy`, etc.

```
dddiv    ddscl    dload    dnorm1
hcopy    hload    icopy    iload    iload1
```

These are additional utility routines that could be tuned to your machine. `dload` is used the most, to set a vector to zero.

`mi20amat.for` Core allocation and constraint matrix routines.

```
m2core   m2amat   m2aprd   m2apr1   m2apr5
m2crsh   m2scal   m2scla   m2unpk   matcol
```

`mi25bfac.for` Basis factorization routines.

```
m2bfac   m2bmap   m2belm   m2bsol   m2sing
lu1fac   lu1fad   lu1gau   lu1mar   lu1pen
lu1max   lu1or1   lu1or2   lu1or3   lu1or4
lu1pq1   lu1pq2   lu1pq3   lu1rec
lu6chk   lu6sol   lu7add   lu7elm   lu7for   lu7zap   lu8rpc
```

`mi30spec.for` SPECS file input.

```
miopt    miopti   mioptr   m3char   m3dflt   m3key
m3file   oplook   opnumb   opscan   optokn   opuppr
```

`mi35inpt.for` MPS file input.

```
m3getp   m3hash   m3imov
m3inpt   m3mpsa   m3mpsb   m3mpsc   m3read
```

`mi40bfil.for` BASIS file input/output and SOLUTION printing.

```
m4getb   m4chek   m4id     m4name   m4inst   m4load   m4oldb
m4savb   m4dump   m4newb   m4pnch   m4rc     m4infs
m4rept   m4soln   m4solp   m4stat
```

`mi50lp.for` Primal simplex method.

```
m5bsx    m5chzr   m5dgen   m5frmc   m5hs     m5log    m5lpit
m5pric   m5rc     m5setp   m5setx   m5solv
```

mi60srch.for Linesearch and function evaluation.
m6dmmy m6fcon m6fobj m6fun m6fun1 m6grd m6grd1
m6dobj m6dcon m6srch srchc srchq

mi65rmod.for Maintaining the quasi-Newton factor R
m6bfgs m6bswp m6radd m6rcnd m6rdel
m6rmod m6rset m6rsol m6swap

mi70nobj.for Nonlinear objective; reduced-gradient algorithm.
m7bsg m7chkd m7chkg m7chzq m7fixb
m7rg m7rgit m7sdir m7sscv

mi80ncon.for Nonlinear constraints; projected Lagrangian algorithm.
m8ajac m8augl m8augi m8chkj m8prtj m8sclj
m8setj m8viol

minosl.for *For installations solving linear programs only.*
Program MINOSL
funobj funcon etc. (dummy entries)

The last file **minosl.for** is included as a substitute for files **mi00main.for**, **mi60srch.for**, **mi65rmod.for**, **mi70nobj.for**, **mi80ncon.for**, if MINOS is to be used to solve linear programs only. It reduces the compiled code size by about 100K bytes. It is recommended for use on microcomputers and machines that do not have virtual memory.

7.3. COMMON Blocks

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

m1env	m1eps	m1file	m1savz	m1tim	m1word		
m2file	m2len	m2lu1	m2lu2	m2lu3	m2lu4	m2mapa	m2mapz
m2parm							
m3len	m3loc	m3mps1	m3mps2	m3mps3	m3mps4	m3mps5	m3scal
m5len	m5loc	m5freq	m5inf	m5lobj	m5log1	m5log2	m5log3
m5log4	m5lp1	m5lp2	m5prc	m5step	m5tols		
m7len	m7loc	m7cg1	m7cg2	m7conv	m7phes	m7tols	
m8len	m8loc	m8all	m8al2	m8diff	m8func	m8save	m8veri
cycle1	cycle2	cyclem					

A complete listing of the **COMMON** blocks and their contents appears in subroutine **minos3**. (Also see Section 2.6.) It may be convenient to make use of these occasionally; for example,

```
common    /m1file/ iread,iprint,isumm
```

gives the unit numbers for the PRINT file and the SUMMARY file.

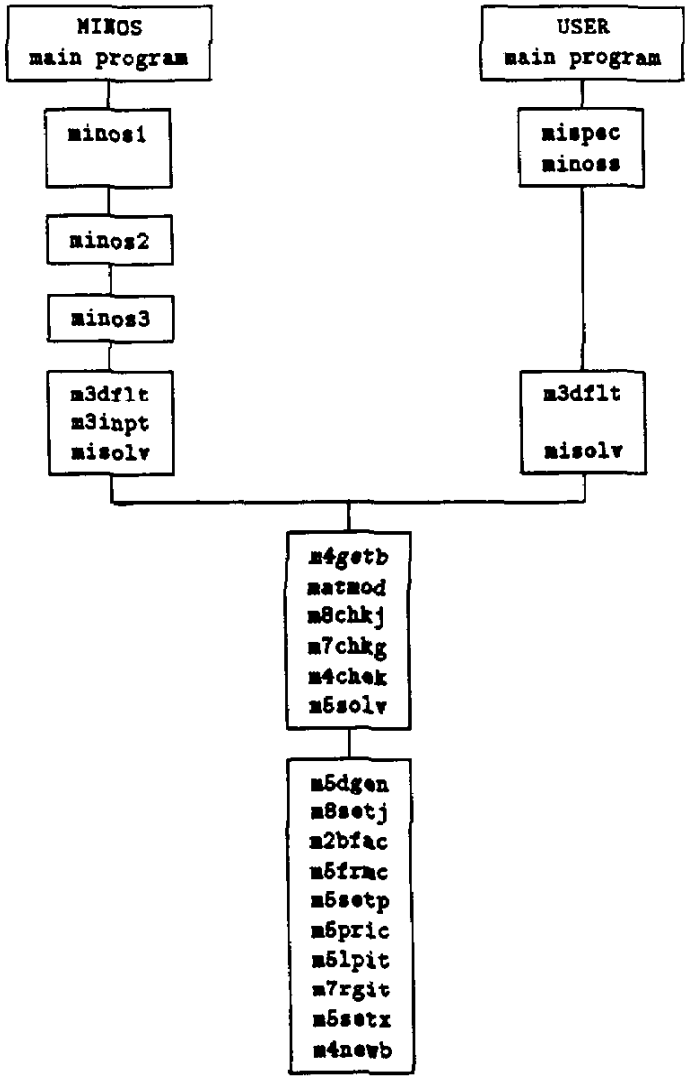
As supplied, MINOS does not use blank **COMMON**. However, in some installations it may be desirable to store the workspace array Z there.

Revision

Pages 78–81 are intentionally omitted in this version of the manual.

7.5. Subroutine Structure

The following picture illustrates the top levels of the subroutine hierarchy for Stand-alone MINOS and for user programs that call subroutine **minoss**.



1. For Stand-alone MINOS, **minos1** reads the SPECS file. For each **begin**–**end** sequence found, it allocates storage and calls **minos2**.

2. In some implementations (e.g. file `mi10vms.for`), **minos2** expands the work array `z(*)` if necessary. It then calls **minos3** to finish processing the current problem.

3. **minos3** reads an MPS file, loads a basis file (if any), and checks gradients. According to the `Cycle limit`, it then solves one or more related problems.

4. For User programs, **mispec** reads a SPECS file (if any). It must be called before **minoss**, even if no SPECS file is provided.

7.6 Test Problems

Test Problem MANNE

This is a small example of an economic model due to Manne (1979). It has a nonlinear objective function, 10 nonlinear constraints, 10 linear constraints, and 30 variables. The nonlinearities are defined by the default function routines *FUNOBJ* and *FUNCON* in the *MINOS source code*. The starting point given in the MPS file is intentionally close to the optimum solution, to make this an inexpensive test problem. Other values in the INITIAL bounds set can be tried.

As supplied, FUNOBJ and FUNCON compute all gradients analytically if the SPECS file specifies DERIVATIVE LEVEL 3. For test purposes, the first three nonzero gradients in each routine are *not* computed if DERIVATIVE LEVEL = 0. We give a summary of the output produced by MINOS for the latter case. A full listing appears in section 8.4.

For this and later examples, the results were obtained on an IBM 3081 using the Fortran H Extended (Enhanced) compiler with optimization level OPT=3.

Maximum objective value:	2.67009603
Iterations to get feasible:	1
Total iterations:	14
Major iterations:	3
Evaluations of $F(x)$ and its gradient:	21
Evaluations of $f(x)$ and its Jacobian:	24
Number of superbasics at optimum:	7
CPU time (IBM 3081):	0.3 seconds

The Weapon Assignment Problem, WEAPON

This problem has a nonlinear objective function and linear constraints. It is described by Bracken and McCormick (1969) and Himmelblau (1972). The constraint matrix is 12×100 and all 100 variables occur nonlinearly in the objective function $F(x)$. The latter depends on 12 data cards which are read during the first entry to subroutine FUNOBJ.

The following are some solution statistics, obtained by MINOS on an IBM 3081 as noted above. They give an indication of the effort required to solve the problem. However, one should not expect to obtain identical results on some other machine.

Minimum objective value:	−1735.56958
Iterations to get feasible:	3
Total iterations:	120
Evaluations of $F(x)$ and its gradient:	270
Number of superbasics at optimum:	18
CPU time (IBM 3081):	2 seconds

Test Problem ETAMACRO (linear version)

This is one example of the energy model developed by Manne (1977). The constraint matrix is 401×689. To obtain a linear problem, we have included one linear objective row OPTIMALG in the MPS file. The latter also contains one right-hand-side vector RHS00001, and one bounds set BOUNDS01.

The objective row OPTIMALG contains the optimal gradient values for the 80 nonlinear variables in the original (nonlinear) form of ETAMACRO. Hence the linear version of the problem has the same optimal dual variables π as the nonlinear version (but rather different primal variables x).

The file ETAMACRO SPECS is set up to solve this linear program first. It asks for the linear variables and constraints to be scaled. (Note that it also asks for a BASIS map to be saved on unit 11 every 100 iterations. This may be used as a starting basis for the nonlinear version of the problem.)

Typical solution statistics follow.

Maximum objective value:	755.715213
Iterations to get feasible:	240
Total iterations:	904
CPU time (IBM 3081):	15 seconds

Test Problem ETAMACRO (nonlinear version)

The objective function for the original form of the energy model is entirely nonlinear, and involves the first 80 variables. It is defined by subroutine FUNOBJ in file ETAMACRO FORTRAN. It depends on 3 data cards which are included at the end of file ETAMACRO SPECS and are read during the first entry to FUNOBJ.

The MPS file does not initialize any of the nonlinear variables. When started from the optimal solution to the preceding linear problem, typical solution statistics (with scaling requested) are as follows.

Maximum objective value:	1337.72468
Iterations to get feasible:	0
Total iterations:	235
Evaluations of $F(x)$ and its gradient:	444
Number of superbasics at optimum:	28
CPU time (IBM 3081):	7 seconds

From a cold start, with and without scaling, typical statistics are as follows.

	SCALE YES	SCALE NO
Maximum objective value:	1337.72468	1337.72468
Iterations to get feasible:	235	213
Total iterations:	1022	1267
Evaluations of $F(x)$ and its gradient:	1271	1554
Number of superbasics at optimum:	28	28
CPU time (IBM 3081):	21 seconds	26 seconds

8. EXAMPLES

The following sections define some example problems and show the input required to solve them using MINOS. The last example in section 8.4 is test problem MANNE as supplied on the distribution tape. For this example we also give the output produced by MINOS.

As the examples show, certain Fortran routines may be required to run a particular problem, depending on the problem and on the Fortran installation:

- A main program to allocate workspace
- Subroutine **FUNOBJ** to define a nonlinear objective function (if any)
- Subroutine **FUNCON** to define nonlinear constraint functions (if any)
- Subroutine **MATMOD** for special applications

The following input items are *always* required:

- A SPECS file
- An MPS file

Additional input may include a BASIS file and data read by the Fortran routines.

Load modules and file specifications are inevitably machine-dependent. A resident expert will be needed to install MINOS on your particular machine and to recommend job control or operating system commands. On some machines it will be possible to run *linear programs* through MINOS without compiling any routines or linking them to the MINOS code file. For nonlinear problems, some compilation and linking is unavoidable.

For some installations it may also be convenient to have your own copy of subroutine **MIFILE**, to define certain file attributes in (non-standard) Fortran, rather than via operating system commands. The resident expert will know best.

Good luck! We hope the examples that follow are general enough to set you on the right track.

8.1 Linear Programming

One of the classical applications of the simplex method was to the so-called *diet problem*. Given the nutritional content of a selection of foods, the cost of each food, and the consumer's minimum daily requirements, the problem is to find the combination that is least expensive. The following example is taken from Chvátal (1983).

$$\text{minimize } c^T x \quad \text{subject to} \quad Ax \geq b, \quad 0 \leq x \leq u,$$

where

$$A = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 4 & 32 & 13 & 8 & 4 & 14 \\ 2 & 12 & 54 & 285 & 22 & 80 \end{pmatrix}, \qquad b = \begin{pmatrix} 2000 \\ 55 \\ 800 \end{pmatrix},$$

and

$$c = (3 \quad 24 \quad 13 \quad 9 \quad 20 \quad 19)^T, \qquad u = (4 \quad 3 \quad 2 \quad 8 \quad 2 \quad 2)^T.$$

Main program (not needed for some installations)

```
      DOUBLE PRECISION   Z(10000)
      DATA           NWCORE/10000/
C
      CALL MINOS1( Z,NWCORE )
      STOP
      END
```

Dummy user routines (not needed for some installations)

```
      SUBROUTINE FUNOBJ
      ENTRY FUNCON
      ENTRY MATMOD
      RETURN
      END
```

SPECS File

```
BEGIN DIET PROBLEM
   MINIMIZE
   ROWS                 20
   COLUMNS              30
   ELEMENTS             50

   SUMMARY FILE          9
   SUMMARY FREQUENCY     1  * (for small problems only)
   NEW BASIS FILE       11
END DIET PROBLEM
```

MPS File

```
NAME          DIET
ROWS
 G  ENERGY
 G  PROTEIN
 G  CALCIUM
 N  COST
COLUMNS
    OATMEAL   ENERGY     110.0      PROTEIN      4.0
    OATMEAL   CALCIUM      2.0      COST         3.0
    CHICKEN   ENERGY     205.0      PROTEIN     32.0
    CHICKEN   CALCIUM     12.0      COST        24.0
    EGGS      ENERGY     160.0      PROTEIN     13.0
    EGGS      CALCIUM     54.0      COST        13.0
    MILK      ENERGY     160.0      PROTEIN      8.0
    MILK      CALCIUM    284.0      COST         9.0
    PIE       ENERGY     420.0      PROTEIN      4.0
    PIE       CALCIUM     22.0      COST        20.0
    PORKBEAN  ENERGY     260.0      PROTEIN     14.0
    PORKBEAN  CALCIUM     80.0      COST        19.0
RHS
    DEMANDS   ENERGY    2000.0      PROTEIN     55.0
    DEMANDS   CALCIUM    800.0
BOUNDS
 UP SERVINGS  OATMEAL      4.0
 UP SERVINGS  CHICKEN      3.0
 UP SERVINGS  EGGS         2.0
 UP SERVINGS  MILK         8.0
 UP SERVINGS  PIE          2.0
 UP SERVINGS  PORKBEAN     2.0
ENDATA
```

Notes on the Diet Problem

1. For small problems such as this, the SPECS file does not really need to specify certain parameters, because the default values are large enough. However, we include them as a reminder for more substantial models.
2. In the MPS file we put the objective row last. Looking ahead, this is one way of ensuring that it does not get mixed up with nonlinear constraints, whose names must appear *first* in the ROWS section.
3. The constraint matrix is unusual in being 100% *dense*. Most models have at least a few zeros in each column and in b. They would not need to appear in the COLUMNS and RHS sections.
4. MINOS takes three iterations to solve the problem. The optimal objective is $c^T x = 92.5$. The optimal solution is $x = (1, 0, 0, 4.5, 2, 0)^T$ and $s = (0, -5, -534.5)^T$. The optimal dual variables are $\pi = (0.05625, 0, 0)^T$.

8.2 Unconstrained Optimization

The following is a classical unconstrained problem, due to Rosenbrock (1960):

$$\text{minimize } F(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

We use it to illustrate the data required to minimize a function with no general constraints. Bounds on the variables are easily included; we specify $-10 \leq x_1 \leq 5$ and $-10 \leq x_2 \leq 10$.

Calculation of $F(x)$ and its gradients

```
      SUBROUTINE FUNOBJ( MODE, N, X, F, G, NSTATE, NPROB, Z, NWCORE )
      IMPLICIT           REAL*8(A-H,O-Z)
      DOUBLE PRECISION   X(N), G(N), Z(NWCORE)
C
C     ROSENBROCK'S BANANA FUNCTION.
C
      X1     =   X(1)
      X2     =   X(2)
      T1     =   X2 - X1**2
      T2     =   1.0 - X1
      F      =   100.0 * T1**2  +  T2**2
      G(1)   = - 400.0 * T1 * X1  -  2.0 * T2
      G(2)   =   200.0 * T1
      RETURN
C
C     END OF FUNOBJ FOR ROSENBROCK
      END
```

SPECS File

```
BEGIN ROSENBROCK
   OBJECTIVE  =  FUNOBJ
   NONLINEAR VARIABLES   2
   SUPERBASICS LIMIT     3

   LOWER BOUND        -10.0
   UPPER BOUND         10.0

   SUMMARY FILE           9
   SUMMARY FREQUENCY      1
   ITERATIONS LIMIT      50
END ROSENBROCK
```

MPS File

```
NAME          ROSENBROCK
ROWS
 N  DUMMYROW
COLUMNS
    X1
    X2
RHS
BOUNDS
 UP BOUND1    X1             5.0
 FX INITIAL   X1            -1.2
 FX INITIAL   X2             1.0
ENDATA
```

Notes on Rosenbrock's function

1. There is nothing special about subroutine FUNOBJ. It returns the function value $F(x)$ and two gradient values $g_j = \partial F/\partial x_j$ on every entry. If G(1) or G(2) were not assigned values, MINOS would "notice" and proceed to estimate either or both by finite differences.
2. The SPECS file apparently does not need to estimate the dimensions of the constraint matrix A, which is supposed to be void anyway. But in fact, MINOS will represent A as a $1 \times n_1$ matrix containing n_1 elements that are all zero. For very large unconstrained problems, the COLUMNS and ELEMENTS keywords must be specified accordingly.
3. The SPECS file must specify the exact number of nonlinear variables, n_1. To allow a little elbow room, the SUPERBASICS LIMIT must be set to $n_1 + 1$, unless it is known that some of the bounds will be active at the solution.
4. The MPS file must specify at least one row. Here it is a dummy free row (type N = non-binding constraint). The basis matrix will remain $B = 1$ throughout, corresponding to the slack variable on the free row.
5. The COLUMNS section contains just a list of the variable names. The RHS header card must appear, but a free row has no right-hand-side entry.
6. Uniform bounds $-10 \leq x_j \leq 10$ are specified in the SPECS file as a matter of good practice. Their presence does not imply additional work. If the LOWER and UPPER BOUND keywords did *not* appear, the variables would implicitly have the bounds $0 \leq x_j \leq \infty$, which will not always be appropriate.
7. With the uniform bounds specified, only one additional card is needed in the BOUNDS section to impose the restriction $x_1 \leq 5$.
8. The INITIAL bound set illustrates how the starting point $(x_1, x_2) = (-1.2, 1.0)$ is specified. These cards must appear at the end of the BOUNDS section. Since the SUPERBASICS LIMIT is sufficiently high, both variables will initially be superbasic at the indicated values.
9. If the INITIAL bound set were absent (and if no BASIS file were loaded), x_1 and x_2 would initially be nonbasic at the bound that is smaller in absolute value (with ties broken in favor of lower bounds); in this case, $x_1 = u_1 = 5$ and $x_2 = l_2 = -10$.
10. From the standard starting point shown, a quasi-Newton method with a moderately accurate linesearch takes about 20 iterations and 80 function and gradient evaluations to reach the unique solution $x_1 = x_2 = 1.0$.

8.3 Linearly Constrained Optimization

Quadratic programming (QP) is a particular case of linearly constrained optimization, in which the objective function $F(x)$ includes linear and quadratic terms. There is no special way of informing MINOS that $F(x)$ is quadratic, but the algorithms in MINOS will tend to perform more efficiently on quadratics than on other nonlinear functions. The following items are required to solve the quadratic program

$$\text{minimize } F(x) = \frac{1}{2}x^TQx + c^Tx \qquad \text{subject to} \qquad Ax \le b, \quad x \ge 0$$

for the particular data

$$Q = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 4 & 0 \\ 2 & 0 & 2 \end{pmatrix}, \qquad c = \begin{pmatrix} -8 \\ -6 \\ -4 \end{pmatrix}, \qquad A = (1 \quad 1 \quad 2), \qquad b = 3.$$

Calculation of quadratic term and its gradients

```
      SUBROUTINE FUNOBJ( MODE, N, X, F, G, NSTATE, NPROB, Z, NWCORE )
      IMPLICIT           REAL*8(A-H,O-Z)
      DOUBLE PRECISION   X(N), G(N), Z(NWCORE)
      COMMON    /QPCOMM/ Q(50,50)
C
C     Computation of   F = 1/2 x'Qx,   g = Qx.
C     The COMMON statement and subroutine SETQ are problem dependent.
C
C
      IF (NSTATE .EQ. 1) CALL SETQ( Q, 50, N )
      F      = 0.0
C
      DO 200 I = 1, N
         GRAD  = 0.0
         DO 100 J = 1, N
            GRAD  = GRAD + Q(I,J)*X(J)
  100    CONTINUE
         F     = F + X(I)*GRAD
         G(I)  = GRAD
  200 CONTINUE
C
      F      = 0.5*F
      RETURN
C
C     END OF FUNOBJ FOR QP
      END
```

SPECS File

```
BEGIN QP
   NONLINEAR VARIABLES    3
   SUPERBASICS LIMIT      3

   SUMMARY FILE           9
   SUMMARY FREQUENCY      1
   ITERATIONS LIMIT      50
END QP
```

MPS File

```
NAME          QP
ROWS
 L  A
 N  C
COLUMNS
    X1        A         1.0         C        -8.0
    X2        A         1.0         C        -6.0
    X3        A         2.0         C        -4.0
RHS
    B         A         3.0
ENDATA
```

Notes on the QP example

1. In subroutine **FUNOBJ** we assume that the array **Q(*,*)** is initialized during the first entry by another subroutine **SETQ**, which is problem-dependent. The **COMMON** statement is also problem-dependent and is included to ensure that **Q** will retain its values for later entries. (In some Fortran implementations, local variables are not retained between entries.)
2. The quadratic form will often involve only some of the variables. In such cases the variables should be ordered so that the nonzero rows and columns of Q come first, thus:

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

3. The parameter **N** and the number of **NONLINEAR VARIABLES** would then be the dimension of $\bar{Q}$.
4. **FUNOBJ** could have computed the linear term $c^T x$ (and its gradient c). However we have included c as an objective row in the MPS file, in the same manner as for linear programs. This is more general, because c could contain entries for all variables, not just those associated with $\bar{Q}$.
5. Beware—if $c \neq 0$, the factor $\frac{1}{2}$ makes a vital difference to the function being minimized.
6. The optimal solution to the QP problem as stated is

$$x = (1.3333, 0.77777, 0.44444), \qquad \frac{1}{2}x^T Q x = 8.2222, \qquad c^T x = -17.111 \qquad F(x) = -8.8888.$$

Test Problems WEAPON and ETAMACRO

The MINOS distribution tape contains data for these two linearly constrained problems. The SPECS file for ETAMACRO is as follows. It is set up to solve a linear form of the problem first, and then use the optimal basis as a starting point for the nonlinear form.

```
BEGIN  ETAMACRO AS AN LP PROBLEM.
   MAXIMIZE
   OBJECTIVE  =  OPTIMALS
   ROWS                          500
   COLUMNS                       700
   ELEMENTS                     2600

   SUMMARY FILE                   9
   MPS FILE                      10
   NEW BASIS FILE                11

   SCALE                        YES
   ITERATIONS                  1000
END
BEGIN  ALAN MANNE'S ENERGY MODEL ETAMACRO
   MAXIMIZE
   OBJECTIVE  =  FUNOBJ
   ROWS                          500
   COLUMNS                       700
   ELEMENTS                     2600

   SUMMARY FILE                   9
   MPS FILE                      10
   OLD BASIS FILE                11
   NEW BASIS FILE                12

   NONLINEAR VARIABLES           60
   SUPERBASICS LIMIT             40

   SCALE                        YES
   ITERATIONS                  2000
*
*  NOTE -- AFTER THIS SPECS FILE THERE ARE 3 CARDS OF DATA,
*  TO BE READ ON THE FIRST ENTRY TO SUBROUTINE FUNOBJ.
END
  1.160     1.446     1.717     2.039     2.364     2.740     3.191     3.588
  3.873     4.276     4.721     5.213     5.755     6.354     7.016     7.746
 10.000     0.200     0.400     0.33330   0.600
```

Linear Least Squares

Data-fitting can give rise to a *constrained linear least-squares problem* of the form

$$\text{minimize } \|Xx - y\|_2 \qquad \text{subject to} \qquad Ax \geq b, \quad l \leq x \leq u.$$

This problem may be solved with MINOS as it stands, by coding subroutine FUNOBJ to compute the objective function $F(x) = \frac{1}{2}\|Xx - y\|_2^2$ and its gradient $g(x) = X^T(Xx - y)$. If X is a sparse matrix, it may be more convenient to express the problem in the form

$$\text{minimize } F(r) = \frac{1}{2}r^T r \qquad \text{subject to} \qquad \begin{pmatrix} I & X \\ & A \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix} \begin{matrix} = \\ \geq \end{matrix} \begin{pmatrix} y \\ b \end{pmatrix}, \qquad r \text{ free}, \quad l \leq x \leq u.$$

Notes on the least-squares problem

1. As usual, the constraints in $Ax \geq b$ may include all types of inequality.
2. $r = y - Xx$ is the *residual vector* and $r^T r$ is the *sum of squares*.
3. The objective function is easily programmed as $F(r) = \frac{1}{2} r^T r$ and $g(r) = r$.
4. More stable methods are known for the least-squares problem. If there are no constraints at all, several codes are available for minimizing $\|Xx - y\|_2$ when X is either dense or sparse. When there are equality constraints only ($Ax = b$), we know of one specialized method that can treat X and A as sparse matrices (Björck and Duff, 1980). For the more general case with inequalities and bounds, MINOS is one of very few systems that could attempt to solve problems in which X and A are sparse. *However*, if n (the dimension of x) is very large, MINOS will not be efficient unless almost n constraints and bounds are active at the solution.
5. If it is expected that most of the elements of x will be away from their bounds, it will be worthwhile to specify **MULTIPLE PRICE 10** (say). This will allow up to 10 variables at a time to be added to the set currently being optimized, instead of the usual 1.

The Discrete ℓ_1 Problem

An apparently similar data-fitting problem is

$$\text{minimize } \|Xx - y\|_1 \qquad \text{subject to} \qquad Ax \geq b$$

where $\|r\|_1 \equiv \sum |r_i|$. However, this problem is best solved by means of the following purely *linear* program:

$$\underset{\lambda,\,\mu}{\text{maximize}} \qquad y^T\lambda + b^T\mu$$

$$\text{subject to} \qquad X^T\lambda + A^T\mu = 0, \qquad -1 \leq \lambda_i \leq 1, \qquad \mu \geq 0.$$

Notes on the ℓ_1 problem

1. The solution x is recovered as the dual variables, i.e., the Lagrange multipliers associated with the general constraints.
2. The optimal value of $\|Xx - y\|_1$ is the sum of the absolute values of the reduced costs associated with λ. (It is also the maximal value of $y^T\lambda + b^T\mu$.)
3. If a particular row in $Ax \geq b$ is required to be an equality constraint, the corresponding component of μ should be a free variable.
4. It does not appear simple to include the bounds $l \leq x \leq u$ except as part of $Ax \geq b$. If there are many finite bounds, it may be best to solve the original problem directly as a linear program, thus:

$$\underset{r,\,s}{\text{minimize}} \qquad e^T r + e^T s$$

$$\text{subject to} \qquad \begin{pmatrix} & & A \\ I & -I & X \end{pmatrix} \begin{pmatrix} r \\ s \\ x \end{pmatrix} \geq = \begin{pmatrix} b \\ y \end{pmatrix}, \qquad r,\, s \geq 0, \qquad l \leq x \leq u,$$

where $e^T = (1\ 1 \ldots 1)$.

8.4 Nonlinearly Constrained Optimization

Two example problems are described here to illustrate the subroutines and data required to specify a problem with nonlinear constraints. The first example is small, dense and highly nonlinear; it shows how the Jacobian matrix may be handled most simply (as a dense matrix) when there are very few nonlinear constraints or variables. The second example has both linear and nonlinear constraints, and illustrates most of the features that will be present in large-scale applications where it is essential to treat the Jacobian as a sparse matrix.

Problem MHW4D (Wright (1976), example 4, starting point D)

$$\text{minimize} \quad (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^3 + (x_3 - x_4)^4 + (x_4 - x_5)^4$$

$$\text{subject to} \quad x_1 + x_2^2 + x_3^3 = 3\sqrt{2} + 2,$$
$$x_2 - x_3^2 + x_4 = 2\sqrt{2} - 2,$$
$$x_1 x_5 = 2.$$

Starting point: $x_0 = (-1, 2, 1, -2, -2)$

Notes for problem MHW4D

1. The function subroutines include code for a second problem (Wright, 1976, example 9). The parameter NPROB is used to branch to the appropriate calculation.
2. In subroutine FUNOBJ, F is the value of the objective function $F(x)$ and G contains the corresponding 5 partial derivatives.
3. In subroutine FUNCON, F is an array containing the vector of constraint functions $f(x)$, and G holds the Jacobian matrix; thus, the i-th row of G contains the partial derivatives for the i-th constraint. In this example the Jacobian is best treated as a dense matrix, so G is a two-dimensional array. Note that several elements of G are zero; they do not need to be explicitly set.
4. Subroutine FUNCON will be called before subroutine FUNOBJ. The parameter NSTATE is used to print a message on the very first entry to FUNCON. This is just a matter of good practice, since it is often convenient to compile MINOS and the function routines into an executable code file, and one can easily forget which particular function routines were used.
5. The SPECS file shown contains keywords that should in general be specified for small, dense problems (i.e., ones whose default values would not be ideal).
6. The COLUMNS section of the MPS file contains only the names of the variables, since they are all "nonlinear", and because there are no linear constraints.
7. The BOUNDS section specifies only the initial point. Uniform bounds on the variables are given in the SPECS file.
8. Since FX indicators are used for the INITIAL bounds, the SUPERBASICS LIMIT needs to be at least 5 in this case, plus 1 for elbow room during the optimization.
9. This example has several local minima, and the performance of MINOS is very dependent on the initial point x_0. See Wright (1976) or Murtagh and Saunders (1982) for computational details.

Problem MHW4D; computation of the objective function

```
      SUBROUTINE FUNOBJ( MODE,N,X,F,G,NSTATE,NPROB,Z,NWCORE )
      IMPLICIT           REAL*8(A-H,O-Z)
      DOUBLE PRECISION   X(N),G(N),Z(NWCORE)
C
C     MHW 4
C
      IF (NPROB .NE. 4) GO TO 500
      T1 = X(1) - 1.0
      T2 = X(1) - X(2)
      T3 = X(2) - X(3)
      T4 = X(3) - X(4)
      T5 = X(4) - X(5)
C
      F    =  T1**2 + T2**2 + T3**3 + T4**4 + T5**4
      G(1) =  2.0*(T1 + T2)
      G(2) = -2.0*T2 + 3.0*T3**2
      G(3) = -3.0*T3**2 + 4.0*T4**3
      G(4) = -4.0*T4**3 + 4.0*T5**3
      G(5) = -4.0*T5**3
      RETURN
C
C     MHW 9
C
  500 T1   = DSIN(X(5) - X(3))
      T2   = DCOS(X(5) - X(3))
      F    = 10.0*X(1)*X(4) + X(1)**3 * X(2) - 6.0*X(2)**2 * X(3)
     1       + 9.0*T1 + X(2)**3 * X(4)**2 * X(5)**4
      G(1) = 10.0*X(4)  +  3.0*X(1)**2 * X(2)
      G(2) = X(1)**3  -  12.0*X(2)*X(3)
     1       +  3.0*X(2)**2 * X(4)**2 * X(5)**4
      G(3) = -6.0*X(2)**2 - 9.0*T2
      G(4) = 10.0*X(1)  +  2.0*X(2)**3 * X(4) * X(5)**4
      G(5) =  9.0*T2  +  4.0*X(2)**3 * X(4)**2 * X(5)**3
      RETURN
C
C     END OF FUNOBJ FOR MHWA AND 9
      END
```

Problem MHW4D; computation of the constraint functions

```
      SUBROUTINE FUNCON( MODE,M,N,NJAC,X,F,G,NSTATE,NPROB,Z,NWCORE )
      IMPLICIT           REAL*8(A-H,O-Z)
      DOUBLE PRECISION   X(N),F(M),G(M,N),Z(NWCORE)
C
C     MHW 4
C
      IF (NSTATE .EQ. 1) WRITE(6, 1000) NPROB
      IF (NPROB .NE. 4) GO TO 500
      F(1)   = X(1) + X(2)**2 + X(3)**3
      G(1,1) = 1.0
      G(1,2) = 2.0*X(2)
      G(1,3) = 3.0*X(3)**2
C
      F(2)   = X(2) - X(3)**2 + X(4)
      G(2,2) = 1.0
      G(2,3) = -2.0*X(3)
      G(2,4) = 1.0
C
      F(3)   = X(1)*X(5)
      G(3,1) = X(5)
      G(3,5) = X(1)
      RETURN
C
C     MHW 9
C
  500 F(1)   = X(1)**2 + X(2)**2 + X(3)**2 + X(4)**2 + X(5)**2
      G(1,1) = 2.0*X(1)
      G(1,2) = 2.0*X(2)
      G(1,3) = 2.0*X(3)
      G(1,4) = 2.0*X(4)
      G(1,5) = 2.0*X(5)
C
      F(2)   = X(1)**2*X(3) + X(4)*X(5)
      G(2,1) = 2.0*X(1)*X(3)
      G(2,3) = X(1)**2
      G(2,4) = X(5)
      G(2,5) = X(4)
C
      F(3)   = X(2)**2*X(4) + 10.0*X(1)*X(5)
      G(3,1) = 10.0*X(5)
      G(3,2) =  2.0*X(2)*X(4)
      G(3,4) = X(2)**2
      G(3,5) = 10.0*X(1)
      RETURN
C
 1000 FORMAT(/ 34H THIS IS PROBLEM  MHW4AND9.  NPROB =, I3)
C     END OF FUNCON FOR MHW4AND9
      END
```

Problem MHW4D; the SPECS file

```
BEGIN MHW 4
   PROBLEM NUMBER              4
   NONLINEAR CONSTRAINTS       3
   NONLINEAR VARIABLES         5
   JACOBIAN                DENSE
   UPPER BOUND               5.0
   LOWER BOUND              -5.0

   SUMMARY FILE                9
   ITERATIONS                100
   MAJOR ITERATIONS           15
   MINOR ITERATIONS           10
   PENALTY PARAMETER         1.0

   SUPERBASICS LIMIT           6
   PRINT LEVEL  (JFLXB)      101
   VERIFY LEVEL                0
END MHW 4
```

Problem MHW4D; the MPS file

```
NAME          MHW 4D
ROWS
 E  CON1
 E  CON2
 E  CON3
COLUMNS
    X1
    X2
    X3
    X4
    X5
RHS
    RHS       CON1          6.24263
    RHS       CON2          0.82842
    RHS       CON3          2.0
BOUNDS
 FX INITIAL   X1           -1.0
 FX INITIAL   X2            2.0
 FX INITIAL   X3            1.0
 FX INITIAL   X4           -2.0
 FX INITIAL   X5           -2.0
ENDATA
```

Problem MANNE (Manne, 1979)

$$\text{maximize} \quad \sum_{t=1}^{T} \beta_t \log C_t$$

$$\begin{aligned}
\text{subject to} \quad & \alpha_t K_t^b \geq C_t + I_t, && 1 \leq t \leq T, && \text{(nonlinear constraints)} \\
& K_{t+1} \leq K_t + I_t, && 1 \leq t < T-1, && \text{(linear constraints)} \\
& gK_T \leq I_T,
\end{aligned}$$

with various ranges and bounds.

The variables here are K_t, C_t and I_t, representing capital, consumption and investment during T time periods. The first T constraints are nonlinear because the variables K_t are raised to the power $b = 0.25$. The problem is described more fully in Murtagh and Saunders (1982), where results are given for the case $T = 100$.

The main program and subroutines shown on the following pages are part of the file **HEAD1** on the MINOS distribution tape (see sections 7.1 and 7.4). The SPECS data and MPS data are contained in the file **MANNE DATA**; they apply to the case $T = 10$.

Notes for problem MANNE

1. For efficiency, the Jacobian variables K_t are made the first T components of x, followed by the objective variables C_t. Since the objective does not involve K_t, subroutine **FUNOBJ** must set the first T components of the objective gradient to zero. The parameter **N** will have the value $2T$. Verification of the objective gradients may as well start at variable $T + 1$.
2. For subroutine **FUNCON**, **N** will be T. The Jacobian matrix is particularly simple in this example; in fact $J(x)$ has only one nonzero element per column (i.e., it is diagonal). The parameter **NJAC** will therefore be T also. It is used only to dimension the array **G**.
3. **NSTATE** enables **B**, **AT** and **BT** to be initialized on the first entry to **FUNCON**, for subsequent use in both of the function subroutines. (Remember that the first call to **FUNCON** occurs before the first call to **FUNOBJ**.) The name chosen for the labeled **COMMON** block holding these quantities must be different from the other **COMMON** names used by MINOS, as listed in section 7.3.
4. **NSTATE** is also used to produce some output on the final call to **FUNCON**.
5. The **COMMON** block **M1FILE** is one of those used by MINOS; see section 1.6. For test purposes we also use **COMMON** block **M8DIFF** to access the variable **LDERIV**.
6. The SPECS file uses keywords that you should become familiar with before running large problems. Other values will be appropriate for other applications.
7. The MPS file specifies a sparse $T \times T$ Jacobian in the top left corner of the constraint matrix. An arbitrary value of **0.1** has been used for the nonzero variable coefficients. A zero or blank numeric field would be equally good.

Problem MANNE; main program and calculation of the objective function

```
*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      program           MINOS

      implicit          double precision (a-h,o-z)

*     ------------------------------------------------------------------
*     This is the default main program for MINOS.
*     It provides all of the necessary workspace.
*     If your compiler wants all common blocks to be in the main program
*     (e.g. MACFORTRAN), grab them from subroutine misolv in file mil0..
*     ------------------------------------------------------------------

      parameter          (nwcore = 50000)
      double precision   z(nwcore)

      call minosl( z, nwcore )

*     end of main program for stand-alone MINOS.
      end



*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      subroutine funobj( mode, n, x, f, g, nstate, nprob, z, nwcore )

      implicit          double precision (a-h,o-z)
      double precision  x(n), g(n), z(nwcore)

*     ------------------------------------------------------------------
*     This is funobj for problem t4manne.
*
*     The data bt(*) is computed by t4con  on its first entry.
*
*     For test purposes, we look at    Derivative level
*     and sometimes pretend that we don't know the first
*     three elements of the gradient.
*     ------------------------------------------------------------------

      common    /slfile/ iread,iprint,isumm
      common    /s8diff/ difint(2),gdummy,lderiv,lvldif,knowng(2)
      common    /manne / b,at(100),bt(100)

      intrinsic          log
      logical            gknown
      parameter        ( zero = 0.0d+0 )

      gknown = lderiv .eq. 1  .or.  lderiv .eq. 3
      nt     = n/2
      f      = zero

      do 50 j = 1, nt
         xcon = x(nt+j)
         f    = f  +  bt(j) * log(xcon)
         if (mode .eq. 2) then
            g(j) = zero
            if (gknown  .or.  j .gt. 3) g(nt+j) = bt(j) / xcon
         end if
   50 continue

*     end of funobj for t4manne
      end
```

Problem MANNE; calculation of the constraint functions

```
      subroutine funcon( mode, m, n, njac, x, f, g,
     $                   nstate, nprob, z, nwcore )

      implicit           double precision (a-h,o-z)
      double precision   x(n), f(m), g(njac), z(nwcore)

*     ------------------------------------------------------------------
*     This is funcon for problem t4manne.
*
*     For test purposes, we look at    Derivative level
*     and sometimes pretend that we don't know the first
*     three elements of the gradient.
*     ------------------------------------------------------------------

      common    /mlfile/ iread,iprint,isumm
      common    /m8diff/ difint(2),gdummy,lderiv,lvldif,knowng(2)
      common    /manne / b,at(100),bt(100)

      logical            gknown
      parameter        ( one = 1.0d+0 )

      gknown = lderiv .ge. 2
      nt     = n

*     ---------------------------------------
*     First entry.  Define b, at(*) and bt(*)
*     for this and all subsequent entries.
*     ---------------------------------------
      if (nstate .eq. 1) then
         grow   = 0.03
         beta   = 0.95
         xk0    = 3.0
         xc0    = 0.95
         xi0    = 0.05
         b      = 0.25
         if (iprint .gt. 0) write(iprint, 1000) nt, b

         a      = (xc0 + xi0) / xk0**b
         gfac   = (one + grow)**(one - b)
         at(1)  = a*gfac
         bt(1)  = beta

         do 10 j  = 2, nt
            at(j) = at(j-1)*gfac
            bt(j) = bt(j-1)*beta
   10    continue

         bt(nt) = bt(nt) / (one - beta)
      end if

*     -------------
*     Normal entry.
*     -------------
      do 150 j = 1, nt
         xkap  = x(j)
         f(j)  = at(j) * xkap**b
         if (mode .eq. 2) then
            if (gknown  .or.  j .gt. 3) g(j) = b*f(j) / xkap
         end if
  150 continue

*     ------------
*     Final entry.
*     ------------
      if (nstate .ge. 2) then
         if (iprint .gt. 0) write(iprint, 2000) (f(j), j = 1, nt)
      end if
      return

 1000 format(// ' This is problem  t4manne.   nt =', i4, '   b =', f8.3)
 2000 format(// ' Final nonlinear function values' / (5f12.5))

*     end of funcon for t4manne
      end
```

Problem MANNE; the SPECS file

```
Begin t4manne  (10-period economic growth model)

   Problem number          1114
   Maximize
   Rows                     100
   Columns                  100
   Elements                 100
   Upper bound            100.0
   Objective =           funobj

   Nonlinear constraints     10
   Nonlinear Jacobian  vars  10
   Nonlinear objective vars  20

   MPS file                  10
*  New Basis file            11

   Jacobian              Sparse
   Major iterations           8
   Minor iterations          20
   Penalty parameter        0.1

   Hessian dimension         10
*  Derivative level           0
*  Verify gradients

   Scale option               2
   Iterations                50
   Print level (jflxb)    00000
   Print frequency            1
   Summary level              0
   Summary frequency          1
End Manne10
```

Problem *MANNE*; the MPS file

```
NAME          MANNE10
ROWS
 G  MON001
 G  MON002
 G  MON003
 G  MON004
 G  MON005
 G  MON006
 G  MON007
 G  MON008
 G  MON009
 G  MON010
 L  CAP002
 L  CAP003
 L  CAP004
 L  CAP005
 L  CAP006
 L  CAP007
 L  CAP008
 L  CAP009
 L  CAP010
 L  TERMINV
COLUMNS
    KAP001    MON001      .1          CAP001    1.0
    KAP001    CAP002    -1.0
    KAP002    MON002      .1          CAP002    1.0
    KAP002    CAP003    -1.0
    KAP003    MON003      .1          CAP003    1.0
    KAP003    CAP004    -1.0
    KAP004    MON004      .1          CAP004    1.0
    KAP004    CAP005    -1.0
    KAP005    MON005      .1          CAP005    1.0
    KAP005    CAP006    -1.0
    KAP006    MON006      .1          CAP006    1.0
    KAP006    CAP007    -1.0
    KAP007    MON007      .1          CAP007    1.0
    KAP007    CAP008    -1.0
    KAP008    MON008      .1          CAP008    1.0
    KAP008    CAP009    -1.0
    KAP009    MON009      .1          CAP009    1.0
    KAP009    CAP010    -1.0
    KAP010    MON010      .1          CAP010    1.0
    KAP010    TERMINV     .03
    CON001    MON001    -1.0
    CON002    MON002    -1.0
    CON003    MON003    -1.0
    CON004    MON004    -1.0
    CON005    MON005    -1.0
    CON006    MON006    -1.0
    CON007    MON007    -1.0
    CON008    MON008    -1.0
    CON009    MON009    -1.0
    CON010    MON010    -1.0
    INV001    MON001    -1.0          CAP002   -1.0
    INV002    MON002    -1.0          CAP003   -1.0
    INV003    MON003    -1.0          CAP004   -1.0
    INV004    MON004    -1.0          CAP005   -1.0
    INV005    MON005    -1.0          CAP006   -1.0
    INV006    MON006    -1.0          CAP007   -1.0
    INV007    MON007    -1.0          CAP008   -1.0
    INV008    MON008    -1.0          CAP009   -1.0
    INV009    MON009    -1.0          CAP010   -1.0
    INV010    MON010    -1.0          CAP011   -1.0
    INV010    TERMINV   -1.0
```

Problem MANNE; the MPS file, continued

```
RHS
*
*   The RHS is zero
*
    LAGRANGE  MON002      -0.9          MON003      -0.8
    LAGRANGE  MON010     -10.0
RANGES
    RANGE1    MON010      10.0          TERMINV     20.0
BOUNDS
 FX BOUND1    KAP001      3.05
 LO BOUND1    KAP002      3.05
 LO BOUND1    KAP003      3.05
 LO BOUND1    KAP004      3.05
 LO BOUND1    KAP005      3.05
 LO BOUND1    KAP006      3.05
 LO BOUND1    KAP007      3.05
 LO BOUND1    KAP008      3.05
 LO BOUND1    KAP009      3.05
 LO BOUND1    KAP010      3.05
 LO BOUND1    CON001       .95
 LO BOUND1    CON002       .95
 LO BOUND1    CON003       .95
 LO BOUND1    CON004       .95
 LO BOUND1    CON005       .95
 LO BOUND1    CON006       .95
 LO BOUND1    CON007       .95
 LO BOUND1    CON008       .95
 LO BOUND1    CON009       .95
 LO BOUND1    CON010       .95
 LO BOUND1    INV001       .05
 LO BOUND1    INV002       .05
 LO BOUND1    INV003       .05
 LO BOUND1    INV004       .05
 LO BOUND1    INV005       .05
 LO BOUND1    INV006       .05
 LO BOUND1    INV007       .05
 LO BOUND1    INV008       .05
 LO BOUND1    INV009       .05
 LO BOUND1    INV010       .05
 UP BOUND1    INV008       .112
 UP BOUND1    INV009       .114
 UP BOUND1    INV010       .116
 FX INITIAL   KAP002      3.1
 FX INITIAL   KAP003      3.2
 FX INITIAL   KAP004      3.3
 FX INITIAL   KAP005      3.4
 FX INITIAL   KAP006      3.5
 FX INITIAL   KAP007      3.6
 FX INITIAL   KAP008      3.7
 FX INITIAL   KAP009      3.8
 FX INITIAL   KAP010      3.9
ENDATA
```

Problem MANNE; output from MINOS

```
      ==============================
      M I N O S    5.5    (May 1998)
      ==============================

    Begin t4manne  (10-period economic growth model)

       Problem number          1114
       Maximize
       Rows                     100
       Columns                  100
       Elements                 100
       Upper bound            100.0
       Objective =           funobj

       Nonlinear constraints     10
       Nonlinear Jacobian vars   10
       Nonlinear objective vars  20

       MPS file                  10
    *  New Basis file            11

       Jacobian              Sparse
       Major iterations           8
       Minor iterations          20
       Penalty parameter        0.1

       Hessian dimension         10
    *  Derivative level           0
    *  Verify gradients

       Scale option               2
       Iterations                50
       Print level (jflxb)    00000
       Print frequency            1
       Summary level              0
       Summary frequency          1
    End Manne10

 Reasonable Workspace limits are         0 ...    6994
 Actual     Workspace limits are         0 ...  100000 ...  100000  words of  z.
1

 MPS file
 --------
      1    NAME          MANNE10
      2    ROWS
     23    COLUMNS
 XXXX  Warning - no linear objective selected
 XXXX  Non-existent row    specified -- CAP001   -- entry ignored in line     24
 XXXX  Non-existent row    specified -- CAP011   -- entry ignored in line     63
     65    RHS
     66    *
     67    *   The RHS is zero
     68    *
 XXXX  Warning - first RHS is LAGRANGE.   Other RHS's will be ignored.
     71    RANGES
 XXXX  Warning - the RHS is zero
     73    BOUNDS
    116    ENDATA

 XXXX  Total no. of errors in MPS file     2
```

```
Names selected
--------------
Objective      FUNOBJ   (Max)       0
RHS                                 0
RANGES         RANGE1               3
BOUNDS         BOUND1              33


No. of Jacobian entries specified        10
No  of LAGRANGE entries specified         3
No. of INITIAL  bounds  specified         9
No. of superbasics specified              9

Nonzeros allowed for in LU factors     49396

Scale option  2,      Partial price       1
Partial price section size (A)           30
Partial price section size (I)           20


Matrix Statistics
-----------------
              Total      Normal        Free       Fixed     Bounded
Rows             20          18           0           0           2
Columns          30           0           0           1          29

No. of matrix elements                   59     Density      9.833
Biggest                         1.4110E+00  (excludieg fixed columns,
Smallest                        3.0000E-02   free rows, and RHS)

No. of objective coefficients             0


Nonlinear constraints      10     Linear constraints      10
Nonlinear variables        20     Linear variables        10
Jacobian  variables        10     Objective variables     20

Initial basis
-------------
No basis file supplied


Scaling
-------
            Min elem    Max elem      Max col ratio
After  0    3.00E-02    1.41E+00              33.33
After  1    4.16E-01    2.40E+00               5.77
After  2    4.42E-01    2.26E+00               5.12
After  3    4.42E-01    2.26E+00               5.12

            Min scale                    Max scale      Between 0.5 and 2.0
Col     10  4.0E-01          Col     30  2.6E+00              28     93.3
Row     20  1.7E-01          Row     19  1.7E+00              19     95.0

Norm of fixed columns and slacks          4.3E+00
(before and after row scaling)            4.2E+00


Crash option  3

Crash on linear E  rows:
```

```
Iterations
----------

Crash on linear LG rows:
Slacks     0  Free cols     0  Preferred     0
Unit      10  Double        0  Triangle      0  Pad      0
Itn       1 -- linear constraints satisfied.


This is problem  t4manne.   nt =  10   b =   0.250
funcon  sets      10   out of      10   constraint gradients.
funobj  sets      20   out of      20   objective  gradients.

Cheap test on funcon...
The Jacobian seems to be OK.
The largest discrepancy was    6.67E-10  in constraint     2

Cheap test on funobj...
The objective gradients seem to be OK.
Gradient projected in two directions   4.00258220426E+00   1.00000000000E+00
Difference approximations              4.00257400842E+00   9.99999843622E-01


Scaling
-------
            Min elem    Max elem       Max col ratio
After  0    3.00E-02    1.00E+00               33.33
After  1    4.16E-01    2.40E+00                5.77
After  2    5.06E-01    1.98E+00                3.90
After  3    5.17E-01    1.93E+00                3.74

           Min scale                      Max scale      Between 0.5 and 2.0
Col     10   5.0E-01            Col     30   6.7E+00              19    63.3
Row     20   1.1E-01            Row     11   1.5E+00              10    50.0

Norm of fixed columns and slacks             1.8E+00
(before and after row scaling)               5.7E+00

Major minor   total ninf step     objective     Feasible Optimal  nsb   ncon     LU penalty BSsup
    1     1T      1    0 0.0E+00 0.00000000E+00 0.0E+00 1.2E+01    8      4     31 1.0E-01     0

Crash on nonlinear rows:
Slacks     0  Free cols     0  Preferred     0
Unit      10  Double        0  Triangle      0  Pad      0
    2     1       2    0 1.0E+00 2.66979778E+00 4.4E-06 7.9E-04    7      6     40 1.0E-01     8
Completion Full    nor requested
    3     7       9    0 1.0E+00 2.67011960E+00 4.7E-06 7.9E-05    7     21     40 1.0E-01     0
    4     2      11    0 1.0E+00 2.67009863E+00 1.1E-12 1.6E-08    7     26     40 1.0E-02     0
1

EXIT -- optimal solution found

Problem name                 MANNE10
No. of iterations                 11   Objective value      2.67009862?2E+00
No. of major iterations            4   Linear objective     0.0000000000E+00
Penalty parameter           0.001000   Nonlinear objective  2.67009862?2E+00
No. of calls to funobj            27   No. of calls to funcon           26
No. of superbasics                 7   No. of basic nonlinears          18
No. of degenerate steps            0   Percentage                     0.00
Norm of x   (scaled)         1.5E+00   Norm of pi  (scaled)          1.7E+01
Norm of x                    6.5E+00   Norm of pi                    7.6E+00
Max Prim inf(scaled)       0 0.0E+00   Max Dual inf(scaled)      22 1.6E-08
Max Primal infeas          0 0.0E+00   Max Dual infeas           22 5.4E-09
Nonlinear constraint violn   1.9E-12
```

1

NAME PANNE10 OBJECTIVE VALUE 2.6700986272E+00

STATUS OPTIMAL SOLN ITERATION 11 SUPERBASICS 7

OBJECTIVE FUNOBJ (Max)
RHS
RANGES RANGE1
BOUNDS BOUND1

SECTION 1 - ROWS

NUMBER	...ROW..	STATE	...ACTIVITY...	SLACK ACTIVITY	..LOWER LIMIT.	..UPPER LIMIT.	.DUAL ACTIVITY	..I
31	MON001	LL	0.00000	0.00000	.	None	-1.01064	1
32	MON002	LL	0.00000	0.00000	.	None	-0.93193	2
33	MON003	LL	0.00000	0.00000	.	None	-0.85926	3
34	MON004	LL	0.00000	0.00000	.	None	-0.79217	4
35	MON005	LL	0.00000	0.00000	.	None	-0.73021	5
36	MON006	LL	0.00000	0.00000	.	None	-0.67299	6
37	MON007	LL	0.00000	0.00000	.	None	-0.62015	7
38	MON008	LL	0.00000	0.00000	.	None	-0.57134	8
39	MON009	LL	0.00000	0.00000	.	None	-0.52625	9
40	MON010	LL	0.00000	0.00000	.	10.00000	-9.86433	10
41	CAP002	UL	.	.	None	.	1.01064	11
42	CAP003	UL	.	.	None	.	0.93193	12
43	CAP004	UL	.	.	None	.	0.85926	13
44	CAP005	UL	.	.	None	.	0.79217	14
45	CAP006	UL	.	.	None	.	0.73021	15
46	CAP007	UL	.	.	None	.	0.67299	16
47	CAP008	UL	.	.	None	.	0.62015	17
48	CAP009	UL	.	.	None	.	0.57134	18
49	CAP010	UL	.	.	None	.	0.52625	19
50	TERMINV	UL	.	.	-20.00000	.	10.73212	20

1

SECTION 2 - COLUMNS

NUMBER	.COLUMN.	STATE	...ACTIVITY...	.OBJ GRADIENT.	..LOWER LIMIT.	..UPPER LIMIT.	REDUCED GRADNT	M+J
1	KAP001	EQ	3.05000	.	3.05000	3.05000	1.09568	21
2	KAP002	BS	3.12666	.	3.05000	100.00000		22
3	KAP003	BS	3.21443	.	3.05000	100.00000	0.00000	23
4	KAP004	BS	3.30400	.	3.05000	100.00000	0.00000	24
5	KAP005	BS	3.39522	.	3.05000	100.00000	0.00000	25
6	KAP006	BS	3.48788	.	3.05000	100.00000	0.00000	26
7	KAP007	BS	3.58172	.	3.05000	100.00000	0.00000	27
8	KAP008	BS	3.67643	.	3.05000	100.00000	0.00000	28
9	KAP009	BS	3.77158	.	3.05000	100.00000	0.00000	29
10	KAP010	BS	3.86667	.	3.05000	100.00000	.	30
11	CON001	LL	0.95000	1.0	0.95000	100.00000	-0.01064	31
12	CON002	BS	0.96842	0.93193	0.95000	100.00000	.	32
13	CON003	BS	0.99780	0.85926	0.95000	100.00000	.	33
14	CON004	BS	1.02820	0.79217	0.95000	100.00000	.	34
15	CON005	BS	1.05967	0.73021	0.95000	100.00000	.	35
16	CON006	BS	1.09217	0.67299	0.95000	100.00000	.	36
17	CON007	BS	1.12608	0.62015	0.95000	100.00000	.	37
18	CON008	BS	1.16116	0.57134	0.95000	100.00000	.	38
19	CON009	BS	1.19763	0.52625	0.95000	100.00000	.	39
20	CON010	BS	1.21394	9.86433	0.95000	100.00000	.	40
21	INV001	BS	0.07665	.	0.05000	100.00000	.	41
22	INV002	SBS	0.08778	.	0.05000	100.00000	0.00000	42
23	INV003	SBS	0.08967	.	0.05000	100.00000	0.00000	43

```
    24  INV004     SBS          0.09122          .          0.05000      100.00000      0.00000     44
    25  INV005     SBS          0.09266          .          0.05000      100.00000      0.00000     45
    26  INV006     SBS          0.09384          .          0.05000      100.00000      0.00000     46
    27  INV007      BS          0.09471          .          0.05000      100.00000            .     47
    28  INV008     SBS          0.09515                     0.05000        0.11200      0.00000     48
    29  INV009     SBS          0.09508                     0.05000        0.11400      0.00000     49
    30  INV010      UL          0.11600          .          0.05000        0.11600      0.86778     50

funcon called with nstate =    2


Final nonlinear function values
    1.02665     1.05620     1.08738     1.11942     1.15233
    1.18612     1.22078     1.25632     1.29271     1.32994

funobj called with nstate =    2

Time for MPS input                             0.05 seconds
Time for solving problem                       0.09 seconds
Time for solution output                       0.03 seconds
Time for constraint functions                  0.01 seconds
Time for objective function                    0.00 seconds
Endrun
```

8.5 Use of Subroutine MATMOD

The following example illustrates the construction of a sequence of problems, based on the Diet problem of Section 8.1. It assumes that the following cards have been added to the SPECS file:

```
CYCLE LIMIT          3
CYCLE PRINT          3
CYCLE TOLERANCE    2.0
PHANTOM COLUMNS      1  (or more)
PHANTOM ELEMENTS     3  (or more)
```

1. Solution of the original problem constitutes cycle 1.
2. After cycle 1, **MATMOD** will be called twice with **NCYCLE** = 2 and 3 respectively, denoting the beginning of cycles 2 and 3. The value of **N** will include the normal columns and the phantom columns; in this case, **N** = 6 + 1 = 7. Likewise, **NE** includes normal and phantom elements; in this case, **NE** = 24 + 3 = 27.
3. For cycle 2, we alter the cost coefficient on the variable called **CHICKEN**. This happens to be the second variable, but for illustrative purposes we use the MINOS subroutine **M4NAME** to search the list of column names to find the appropriate index. In this case, **M4NAME** will return the value JCHICK = 2.
4. Similarly, we use **M4NAME** to search the list of row names to find the index for the objective row, whose name is known to be **COST**. In this case, **M4NAME** will return the value JCOST = 11. Since rows are stored after the **N** columns, this means that the objective is row number JCOST − **N** = 4. (As it happens, this value is already available in the **COMMON** variable IOBJ.)
5. This example assumes that **CHICKEN** already had a nonzero cost coefficient, since it is not possible to increase the number of entries in existing columns. If the cost coefficient was previously zero, it would have to be entered as such in the MPS file, and the SPECS file would have to set **AIJ TOLERANCE** = 0.0 to prevent zero coefficients from being rejected.
6. For cycle 3, we generate one new column by calling upon the MINOS subroutine **MATCOL**. The **PHANTOM COLUMNS** and **PHANTOM ELEMENTS** keywords must define sufficient storage for this new column. (The estimates defined by the normal **COLUMNS** and **ELEMENTS** keywords must also allow for the phantom columns and elements.)
7. For illustrative purposes, we make use of the specified **CYCLE TOLERANCE** and the value of **X(1)** in the current solution, to decide whether to proceed with cycle 3.
8. After the call to **MATCOL**, the **COMMON** variable **JNEW** points to the new column. It allows us to set a finite upper bound on the associated variable. If there had been insufficient storage, or if **COL(*)** contained no significant elements, **MATERR** would have been increased from 0 to 1. Usually, this means that the sequence of cycles should be terminated (by setting **FINISH** = **.TRUE.**).

```
      subroutine matmod( ncycle, nprob, finish,
     $                   m, n, nb, ne, nka, ns, nscl, nname,
     $                   a, ha, ka, bl, bu,
     $                   ascale, hs, name1, name2,
     $                   x, pi, rc, z, nwcore )

      implicit           double precision (a-h,o-z)
      integer*4          ha(ne), hs(nb)
      integer            ka(nka), name1(nname), name2(nname)
      double precision   a(ne), ascale(nscl), bl(nb), bu(nb)
      double precision   x(nb), pi(m), rc(nb), z(nwcore)
      logical            finish
c
c     MINOS  COMMON BLOCKS (TO BE USED BUT NOT ALTERED).
c
      COMMON    /M1FILE/ IREAD,IPRINT,ISUMM
      COMMON    /MSLOBJ/ SINF,WTOBJ,MINIMZ,NINF,IOBJ
      COMMON    /CYCLCM/ CWTOL,JNEW,MATERR,MAXCY,NEPRNT,NPMAXT,NPRINT
c
c     LOCAL STORAGE.
c
      DOUBLE PRECISION   COL(10), ZTOL
      INTEGER            CHICK1, CHICK2, COST1, COST2
      DATA               CHICK1, CHICK2 /'CHIC', 'KEN '/
      DATA               COST1,  COST2  /'COST', '    '/
c     ------------------------------------------------------------------
c     THIS IS AN EXAMPLE OF A USER-WRITTEN SUBROUTINE  MATMOD,
c     WHICH DEFINES A SEQUENCE OF PROBLEMS BY PERFORMING INTERNAL
c     MODIFICATIONS TO THE DATA FOR THE DIET PROBLEM.
c
c     MATMOD  IS CALLED AT THE BEGINNING OF EACH CYCLE EXCEPT THE FIRST.
c     NCYCLE  WILL TAKE THE VALUES  2, 3, ... UP TO THE  CYCLE LIMIT.
c     ------------------------------------------------------------------
c
      IF (NCYCLE .GT. 2) GO TO 300
c
c     ------------------------------------------------------------------
c     CYCLE 2.  ALTER THE COST ON  CHICKEN.
c     ------------------------------------------------------------------
c
c     USE THE MINOS SUBROUTINE  MINAME  TO FIND THE COLUMN INDEX
c     FOR THE VARIABLE NAMED  CHICKEN.  COLUMN NAMES ARE CONTAINED
c     IN THE FIRST  N  LOCATIONS OF  ID1  AND  ID2.
c
c     CERTAIN QUANTITIES MUST BE INITIALIZED BEFORE THE CALL.
c     THE FIRST THREE SUPPRESS ERROR MESSAGES.  THE NEXT THREE
c     DEFINE THE RANGE OF NAMES TO BE SEARCHED AND WHERE TO START.
c
      NCARD  = 0
      NOTFND = 0
      MAXMSG = 0
      J1     = 1
      J2     = N
      JMARK  = J1
      CALL M4NAME( NB, ID1, ID2, CHICK1, CHICK2,
     $             NCARD, NOTFND, MAXMSG, J1, J2, JMARK, JCHICK )
      IF (JCHICK .EQ. 0) GO TO 900
c
```

```
C     NOW FIND THE INDEX OF THE OBJECTIVE ROW, WHICH IS NAMED  COST.
C     ROW NAMES ARE STORED IN THE LAST  M   OCATIONS OF  ID1  AND  ID2.
C
      J1     = M + 1
      J2     = MB
      JMARK  = J1
      CALL MANAME( MB, ID1, ID2, COST1, COST2,
     *             NCARD, NOTFND, MAXROW, J1, J2, JMARK, JCOST )
      IF (JCOST .EQ. 0) GO TO 900
C
C     THE ROW NUMBER IS NOW  JCOST - N.  IN FACT, THIS VALUE COULD HAVE
C     BEEN OBTAINED DIRECTLY FROM THE COMMON VARIABLE  IOBJ.
C
      ICOST  = JCOST - N
      IF (ICOST .NE. IOBJ) GO TO 900
C
C     NOW WE DIP INTO THE MATRIX DATA STRUCTURE TO FIND WHERE THE
C     COST COEFFICIENT IS IN THE MATRIX COLUMN ASSOCIATED WITH  CHICKEN.
C
      K1     = KA(JCHICK)
      K2     = KA(JCHICK + 1) - 1
      DO 220 K = K1, K2
         IF (HA(K) .EQ. ICOST) GO TO 250
  220 CONTINUE
      GO TO 900
C
C     WE FOUND IT.  NOW SUPPOSE  CHICKEN  IS SELLING AT A BARGAIN RATE.
C
  250 OLDC   = A(K)
      A(K)   = 10.0
      IF (ISUMM .GT. 0) WRITE(ISUMM, 2000) OLDC, A(K)
      RETURN
C
C     ------------------------------------------------------------------
C     CYCLE 3.  GENERATE A NEW COLUMN.
C
C     FOR ILLUSTRATIVE PURPOSES WE SET UP THE NEW PROBLEM ONLY IF
C     THE SOLUTION TO THE CURRENT PROBLEM CONTAINS MORE  OATMEAL  THAN
C     THE SPECIFIED  CYCLE TOLERANCE.  WE HAPPEN TO KNOW THAT  OATMEAL
C     IS THE FIRST VARIABLE,  X(1).
C     ------------------------------------------------------------------
  300 IF (NCYCLE .GT. 3) GO TO 900
      IF (ISUMM  .GT. 0) WRITE(ISUMM, 3000) X(1)
      IF (X(1)   .LE. CNVTOL) GO TO 900
      COL(1) = 500.0
      COL(2) =  20.0
      COL(3) =   5.0
      COL(4) =   5.0
      ZTOL   =   1.0E-6
      CALL MATCOL( M, N, MB, NE, NKA,
     *             A, HA, KA, BL, BU, COL, ZTOL )
C
C     THE COMMON VARIABLE  MATERR  IS INITIALIZED EARLIER TO ZERO.
C     MATCOL  WILL INCREMENT IT IN THE EVENT OF ERRORS.
C     MATCOL  ALSO INCREMENTS  JNEW  TO POINT TO THE NEW COLUMN.
C     WE USE  JNEW  TO GIVE THE ASSOCIATED VARIABLE AN UPPER BOUND.
C
      IF (MATERR .GT. 0) GO TO 900
      BU(JNEW) = 2.0
      RETURN
C
C     ------------------------------------------------------------------
C     TERMINATE CYCLES UNDER VARIOUS CONDITIONS.
C     ------------------------------------------------------------------
  900 FINISH = .TRUE.
      RETURN
C
 2000 FORMAT(/ ' *** COST OF  CHICKEN  CHANGED FROM', F8.2,
     *         '   TO', F8.2)
 3000 FORMAT(/ ' *** CURRENT AMOUNT OF  OATMEAL  IS', F8.2)
C     END OF MATMOD
      END
```

8.6 Things to Remember

Use the following space to record the fruits of your experience. They may be useful reminders the next time you come to run MINOS. (We suggest you use a pencil.)

REFERENCES

Bartels, R. H. (1971). A stabilization of the simplex method, *Num. Math.* 16, 414–434.

Bartels, R. H. and Golub, G. H. (1969). The simplex method of linear programming using the *LU* decomposition, *Comm. ACM* 12, 266–268.

Björck, Å. and Duff, I. S. (1980). A direct method for the solution of sparse linear least squares problems, *Linear Algebra and its Applics.* 34, 43–67.

Bracken, J. and McCormick, G. P. (1968). *Selected Applications of Nonlinear Programming*, John Wiley and Sons, New York and Toronto.

Brooke, A., Drud, A. and Meeraus, A. (1985). High level modeling systems and nonlinear programming, in P. T. Boggs, R. H. Byrd and R. B. Schnabel (eds.), *Numerical Optimization 1984*, SIAM, Philadelphia, 178–198.

Chvátal, V. (1983). *Linear Programming*, W. H. Freeman and Company, New York and San Francisco.

Dantzig, G. B. (1951). Maximization of a linear function of variables subject to linear inequalities, in T. C. Koopmans (ed.), *Activity Analysis of Production and Allocation*, Proceedings of Linear Programming Conference, June 20–24, 1949, John Wiley and Sons, New York, 359–373.

Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.

Davidon, W. C. (1959). Variable metric methods for minimization, A.E.C. Res. and Develop. Report ANL-5990, Argonne National Laboratory, Argonne, Illinois.

Fourer, R. (1982). Solving staircase linear programs by the simplex method, *Math. Prog.* 23, 274–313.

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

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1979). Two step-length algorithms for numerical optimization, Report SOL 79-25, Department of Operations Research, Stanford University.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1986). Maintaining *LU* factors of a general sparse matrix, Report SOL 86-8, Department of Operations Research, Stanford University. (To appear in *Linear Algebra and its Applics.*, 1987.)

Himmelblau, D. M. (1972). *Applied Nonlinear Programming*, McGraw-Hill.

Lawson, C. L., Hanson, R. J., Kincaid, D. R. and Krogh, F. T. (1979). Basic Linear Algebra Subprograms for Fortran usage, *ACM Trans. Math. Software* 5, 308–323 and (Algorithm) 324–325.

Manne, A. S. (1977). ETA-MACRO: A Model of Energy-Economy Interactions, in C. J. Hitch (ed.), *Modeling Energy-Economy Interactions*, Resources for the Future, Washington, DC. Also in R. Pindyck (ed.), *Advances in the Economics of Energy and Resources*, Vol. 2: The

Production and Pricing of Energy Resources, JAI Press, Inc., Greenwich, Connecticut, 1979, 205–233.

Manne, A. S. (1979). Private communication.

Murtagh, B. A. and Saunders, M. A. (1978). Large-scale linearly constrained optimization, *Math. Prog.* 14, 41–72.

Murtagh, B. A. and Saunders, M. A. (1982). A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints, *Math. Prog. Study* 16, *Algorithms for Constrained Minimization of Smooth Nonlinear Functions*, 84–117.

Preckel, P. V. (1980). Modules for use with MINOS/AUGMENTED in solving sequences of mathematical programs, Report SOL 80-15, Department of Operations Research, Stanford University.

Reid, J. K. (1976). Fortran subroutines for handling sparse linear programming bases, Report R8269, Atomic Energy Research Establishment, Harwell, England.

Reid, J. K. (1982). A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases, *Math. Prog.* 24, 55–69.

Robinson, S. M. (1972). A quadratically convergent algorithm for general nonlinear programming problems, *Math. Prog.* 3, 145–156.

Rosen, J. B. and Kreuser, J. (1972). A gradient projection algorithm for nonlinear constraints, in *Numerical Methods for Non-Linear Optimization* (F. A. Lootsma, ed.), Academic Press, London and New York, 297–300.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function, *Computer J.* 3, 175–184.

Saunders, M. A. (1976). A fast, stable implementation of the simplex method using Bartels-Golub updating, in *Sparse Matrix Computations* (J. R. Bunch and D. J. Rose, eds.), Academic Press, New York, 213–226.

Wolfe, P. (1962). The reduced-gradient method, unpublished manuscript, RAND Corporation.

Wright, M. H. (1976). Numerical methods for nonlinearly constrained optimization, Ph. D. thesis, Computer Science Department, Stanford University.

INDEX

Appendix A

MINOS 5.5

Most of the MINOS 5.0 User's Guide applies to all versions of MINOS since 1983. The Guide has been changed slightly to match MINOS 5.1. These appendices summarize further changes and new features in MINOS 5.5.

A.1 CHANGES BETWEEN MINOS 5.1 AND MINOS 5.5

1. MINOS is now callable as a subroutine (see Appendix B). The stand-alone form of MINOS reads constraint data from an MPS file, whereas subroutine **minoss** has the same information passed to it as parameters. In these notes the term MINOS usually refers to both cases, but occasionally we need to distinguish between them.

2. Upper and lower case may be used in the SPECS file. Numerical values may contain up to 16 characters. For example,

```
Iterations limit          2000
Lower bound        -1.23456E+07
```

3. The default values of some options have changed as follows:

```
Print level                0
Print frequency          100   (alias Log frequency)
Summary frequency        100
Hessian dimension         50
Superbasics limit         50
Crash option               3   (new default and new meaning)
Scale option               2   for LP,   1   for NLP
Factorize frequency      100   for LP,  50   for NLP
LU Factor tolerance    100.0   for LP,  5.0  for NLP
LU Update tolerance     10.0   for LP,  5.0  for NLP
Partial price             10   for LP,  1    for NLP
Check frequency           60
Penalty parameter        1.0   is equivalent to old default
```

4. **Derivative level** 0 requests a function-only search, even if **funobj** and **funcon** compute all gradients. The linesearch calls these routines with **mode** = 0, not **mode** = 2. An extra call with **mode** = 2 is needed after the search, but the net cost may be less if gradients are very expensive (e.g., if the user is estimating them by differences).

5. funobj and funcon may now return mode = −1 to mean "My nonlinear function is undefined here". During normal iterations, this signals the linesearch to try again with a shorter steplength.

   Previously, if funobj or funcon returned mode ¡ 0, it meant "Please terminate". To request termination now, set mode ≤ −2.

6. Crash option 2 and 3 have been altered. The Crash procedure chooses a triangular basis from various rows and columns of (A I). In some cases it is called more than once as follows:

Crash option 0	chooses the all-slack basis $B = I$.
Crash option 1	calls Crash once, looking at all rows and columns.
Crash option 2	calls Crash twice, looking at linear rows first. Nonlinear rows are treated at the start of Major 2.
Crash option 3	(default) calls Crash three times, looking at linear *equality* rows first, then linear inequalities, then nonlinear rows (if any) at the start of Major 2.

7. For problems with many degrees of freedom (lots of superbasic variables), experience suggests the following. Up to a certain point, it is best to provide a full triangular matrix R for the "reduced Hessian approximation" used by the quasi-Newton algorithm. For example,

   ```
   Hessian dimension     1000
   Superbasics limit     1000
   ```

   would be suitable for most practical models. However, if the number of superbasic variables does reach 1000, considerable computation is needed to update the 500,000 elements of the dense matrix R.

   For more extreme cases it may be better to work with a smaller matrix R:

   ```
   Hessian dimension      100 or 200
   Superbasics limit     5000
   ```

   (e.g., for optimization with many variables and few constraints). The number of iterations and function calls will increase substantially. The functions and gradients should therefore be cheap to evaluate.

   For general problems with many degrees of freedom, consider LANCELOT. For large problems with bound constraints only, consider LBFGS-B or LANCELOT. Both systems are available via NEOS: http://www.mcs.anl.gov/home/otc/

8. Jacobian = Dense or Sparse is still needed with MPS files, but need not be specified when subroutine minoss is used.

9. The Minor iterations limit now applies to the *feasible* iterations in each major iteration. Any number of (infeasible) minor iterations are allowed while MINOS iterates towards a "feasible subproblem".

The first major iteration is special—it stops as soon as the original linear constraints are satisfied.

For later major iterations, if the log says SOT and the **Minor iterations** limit is 40, we know that 10 minor iterations were needed to satisfy the linearized constraints of the subproblem, and a further 40 were spent optimizing the subproblem before it was terminated by the **Minor iterations** limit.

10. **Penalty parameter** 1.0 is now the default, and it is relative to the old default of $100/m_1$, where m_1 is the number of nonlinear constraints. **Penalty parameter** 2.0 means twice the default value. This makes it easier to experiment with.

11. It is possible to turn off all output to the PRINT and SUMMARY files. The Print and Summary options are as follows:

Print file	0	No output to PRINT file.
	> 0	Output to specified file.
Print level	0	One line per major iteration.
	> 0	Full output as before.
Print frequency	0	No minor iteration log.
	i	A minor iteration line every i itns.
Summary file	0	No output to SUMMARY file.
	> 0	Output to specified file.
Summary level	0	One line per major iteration.
	> 0	More output.
Summary frequency	0	No minor iteration log.
	i	A minor iteration line every i itns.

12. Cold, Warm and Hot starts may be used when solving a sequence of problems of the *same size*.

    For stand-alone MINOS, the sequence of problems is defined via the **Cycle** parameters and the user *routine* **matmod**, which may access the common block

```
logical             gotbas,gotfac,gothes,gotscl
common    /cycle1/  gotbas,gotfac,gothes,gotscl
```

    to say whether or not the existing basis, basic factorization, reduced Hessian, and/or scales should be used to initialize the next solve. If **gotbas** = **.false.**, Crash will be used to choose a starting basis. Otherwise, a basis is assumed to be specified by the array **hs(*)**, and some or all of the other three quantities may be preserved.

    For subroutine **minoss**, these logicals are set if the first parameter **start** is **'Hot xxx'**, where **xxx** is any of the letters **FHS**. See Appendix B.

13. Following the **EXIT** message, some information is output to the PRINT file and the SUMMARY file. Lines of the form

```
Primal inf (scaled)     444    4.6E-07   Dual inf   (scaled)     268    5.2E-06
Primal infeas           412    2.6E-06   Dual infeas             502    9.3E-07
Nonlinear constraint violn     2.5E-14
```

show the maximum primal and dual infeasibilities before and after scaling, and the associated variable number. (Variable j is a column x_j for $1 \leq j \leq n$ and slack s_{j-n} for $n+1 \leq j \leq n+m$.)

Note that "Primal infeasibility" is the amount by which x and s lie outside their bounds. In this example, variable **444** lies furthest outside its bounds *before the solution is unscaled.* More importantly, variable **412** is the most infeasible in the final solution—it lies outside its bounds by **2.6e-6**. If this seems too large, the **Feasibility tolerance** would need to be reduced below the maximum *scaled* infeasibility **4.6e-7** (or the unscaled value **2.6e-6** if scaling was not used).

Similarly, variable 502 is the one whose reduced gradient has the "wrong sign" by the largest amount. If this seems too large, the **Optimality tolerance** would need to be reduced below **5.2E-06*norm(pi)**, where the required norm of π is printed three or one lines above (depending on whether scaling was used).

Where relevant, the **Nonlinear constraint violn** line gives the maximum amount by which any nonlinear constraint value lies outside its bounds in the final unscaled solution.

14. The printed solution and SOLUTION file treat 0.0, 1.0, -1.0 specially. In particular, a dot (.) means 0.0, not "Same as the line above"!

15. In the Fortran source code, **integer*2** has been changed to **integer*4** throughout, to allow solution of arbitrarily large problems. This change is reversible. (The variable **nwordh** must be set appropriately in subroutine **m1init**.) If **integer*2** is used, the maximum number of rows is 16383.

16. In source file **mi10*.for**, subroutine **mifile** defines some "hard-wired" file numbers and opens most files by calling **m1open**. Some of the file numbers and **open** statements may need to be altered to suit your system.

17. The first two lines of OLD BASIS and NEW BASIS files accommodate larger problems than in MINOS 5.1.

A.2 NEW SPECS FILE KEYWORDS

All of the following keywords are new except the first. Crash options 2 and 3 now have a different effect and option 4 is not defined.

Crash option i Default = 3

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.

Defaults

When **minoss** is in use, **call miopt('Defaults')** causes all MINOS options to be set to their default values.

Expand frequency i Default = 10000

This option is part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems.[1]

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 **Feasibility tolerance** is δ. Over a period of i iterations, the tolerance actually used by MINOS 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 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**).

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

[1] The EXPAND procedure is described in "A practical anti-cycling procedure for linearly constrained optimization" P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, *Mathematical Programming* 45 (1989), pp. 437–474.

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 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 r_2 = `1.0e-5`, say, may help cause a judicious change of basis.

`Minor damping parameter` r 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.

1. A linesearch of the form 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 $\alpha_1 = min(1, \beta)$.

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 $\bar{\beta} = r(1 + \|x\|)/\|p\|$, 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 **`Minor damping parameter`** 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.

`Timing level` i Default = 2

$i = 0$ suppresses timing.
$i = 1$ times input, solve and output.
$i = 2$ times input, solve, output, **`funcon`** and **`funobj`**.

The values $i = -1$ and -2 are the same as 1 and 2, except the times are not printed at the end. If you are calling subroutine **`minoss`**, you may print the times in your own format by accessing the following common block:

```
parameter        ( ntime = 5 )
common     /mitim / tlast(ntime), tsum(ntime), numt(ntime), ltime
```

where

`numt(`k`)`	is the number of times clock k has been turned on.
`tlast(`k`)`	is the time at which clock k was last turned on.
`tsum(`k`)`	is the total time elapsed while clock k was on.
`ltime`	is the `Timing level` i.

For $k = 1$ to 5, clock k times input, solve, output, **funcon** and **funobj** respectively. See subroutines **mitime** and **mitimp** for further details. For **Timing level** 2, MINOS and **minoss** both call **mitime** at the end of a run. This prints the "total time" statistics using a loop of the form

```
do k = 1, ntime
   call mitimp( k, 'Time', tsum(k) )
end do
```

A.3 ALGORITHMIC CHANGES

1. The linesearch takes shorter steps if **funobj** or **funcon** return **mode** $= -1$ (mentioned above).

2. "*Basis repair*" *is sometimes* invoked at the start of a major iteration, or following a linesearch failure. A stable, sparse LU factorization of the combined basic/superbasic matrix $\begin{pmatrix} B & S \end{pmatrix}$ is computed by LUSOL in the form

$$P \begin{pmatrix} B^T \\ S^T \end{pmatrix} Q = LU,$$

   where P and Q are permutations and L is well-conditioned. Then P provides a *reordering of the columns of* $\begin{pmatrix} B & S \end{pmatrix}$ *that makes the condition of the new* B close to optimal.

   In the major iteration log, **BSswp** gives the number of variables that were swapped between B and S. (Zero means that the current basis was retained. The current reduced Hessian matrix R is then also retained, to help solve the subproblem more quickly.)

3. The triangular reduced-Hessian matrix R is now stored row-wise instead of column-wise in an array `r(*)`, because most updating operations traverse the rows of R. This reduces paging on a machine with virtual memory and improves the use of cache memory when there are many superbasics and **Hessian dimension** is large.

4. Nonlinear objective and constraint functions are not evaluated until the linear constraints have been satisfied (to within the **Feasibility tolerance**). Previously, any nonlinear constraints were evaluated at the starting point regardless of feasibility.

5. Gradient checking now takes place after the linear constraints have been satisfied. Previously, it occurred at the starting point.

Appendix B

Subroutine minoss

This appendix describes **minoss**, the subroutine version of MINOS. Later sections describe an auxiliary routine (**mispec**) for reading a SPECS file, and some additional routines for specifying individual lines of such a file as part of the calling program.

Note that subroutine **mispec** must be called before the first call to **minoss**, even if a SPECS file is not being read.

In the subroutine specifications, "double precision" entities are appropriate for most machines, but in some cases (e.g. on Cray and Convex systems) they should be changed to their "single precision" equivalents. In some installations, **integer*4** may have been changed to **integer*2** throughout the MINOS source code, to conserve storage. Otherwise, both **integer*4** and plain **integer** are intended to mean 4-byte words.

B.1 SUBROUTINE MINOSS

Problem data is passed to **minoss** as parameters, rather than from an MPS file. This is generally more efficient and convenient for applications that would normally use a "matrix generator".

Specification

```
      subroutine minoss( start, m, n, nb, ne, nname,
     $                   nncon, nnobj, nnjac,
     $                   iobj, objadd, names,
     $                   a, ha, ka, bl, bu, name1, name2,
     $                   hs, xn, pi, rc,
     $                   inform, mincor, ns, ninf, sinf, obj,
     $                   z, nwcore )

      implicit           double precision (a-h,o-z)
      character*(*)      start
      integer            m, n, nb, ne, nname,
     $                   nncon, nnobj, nnjac, iobj,
     $                   inform, mincor, ns, ninf, nwcore
      double precision   objadd, sinf, obj
      character*8        names(5)
      integer*4          ha(ne), hs(nb)
      integer            ka(n+1), name1(nname), name2(nname)
      double precision   a(ne), bl(nb), bu(nb)
      double precision   xn(nb), pi(m), rc(nb), z(nwcore)
```

On entry:

start specifies how a starting basis (and certain other items) are to be obtained.

start = 'Cold' means that Crash should be used to choose an initial basis (unless a basis file is provided).

start = 'Warm' means that a basis is already defined in **hs** (probably from an earlier call).

start = 'Hot' or 'Hot FHS' implies a Hot start. **hs** defines a basis and an earlier call has defined certain other things that should also be kept. The problem dimensions and the array **z(*)** must not have changed.

F refers to the LU factors of the basis.

H refers to the approximate reduced Hessian R.

S refers to column and row scales.

start = 'Hot H' (for example) means that only the Hessian is defined.

start = 'Basis file' is the same as **start** = Cold" (but is more meaningful if an OLD BASIS, INSERT or LOAD file is provided).

m is m, the number of general constraints. For LP problems this means the number of rows in the constraint matrix A. If **integer*4** has been replaced by **integer*2** throughout the Fortran source code, **m** should not exceed 16383. Otherwise there is essentially no upper limit.

In principle, **m** > 0, though sometimes **m** = 0 may be acceptable. (Strictly speaking, Fortran declarations of the form **double precision pi(m)** require **m** to be positive. In debug mode, compilers will probably enforce this, but optimized code may sometimes run successfully with **m** = 0.)

n is n, the number of variables (excluding slacks). For LP problems, this is the number of columns in A (> 0).

nb is $nb = n + m$ (the number of bounds in **bl** or **bu**).

ne is ne, the number of nonzero entries in A (including the Jacobian for any nonlinear constraints). In principle, **ne** > 0, though again **m** = 0, **ne** = 0 may work with some compilers.

nname is the number of column and row names provided in the arrays **name1** and **name2**. If **nname** = 1, there are *no* names. Generic names will be used in the printed solution. Otherwise, **nname** = nb and all names must be provided.

nncon is m_1, the number of nonlinear constraints (≥ 0).

nnobj is n_1', the number of nonlinear objective variables (≥ 0).

nnjac is n_1'', the number of nonlinear Jacobian variables (≥ 0). If **nncon** = 0, **nnjac** = 0. If **nncon** > 0, **nnjac** > 0.

`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` $\leq m$.

`objadd` is a constant that will be added to the objective. Typically `objadd` = `0.0d+0`.

`names(5)` is a set of 8-character names for the problem, the linear objective, the rhs, the ranges and bounds. (This is a hangover from MPS files. The names are used in the printed solution and in some of the basis files.)

`a(ne)` is the constraint matrix (Jacobian), stored column-wise.

`ha(ne)` is a list of row indices for each nonzero in `a(*)`.

`ka(n+1)` is a set of pointers to the beginning of each column of the constraint matrix within `a(*)` and `ha(*)`. It is essential that `ka(1)` = 1 and `ka(`$n+1$`)` = $ne+1$.

1. If the problem has a nonlinear objective, the first `nnobj` columns of `a` and `ha` belong to the nonlinear objective variables. Subroutine `funobj` deals with these variables.
2. If the problem has nonlinear constraints, the first `nnjac` columns of `a` and `ha` belong to the nonlinear Jacobian variables, and the first `nncon` rows of `a` and `ha` belong to the nonlinear constraints. Subroutine `funcon` deals with these variables and constraints.
3. If `nnobj` > 0 and `nnjac` > 0, the two sets of nonlinear variables overlap. The total number of nonlinear variables is `nn` = max(`nnobj`, `nnjac`).
4. The Jacobian forms the top left corner of `a` and `ha`. If a Jacobian column j ($1 \leq j \leq$ `nnjac`) contains any entries `a(`k`)`, `ha(`k`)` associated with nonlinear constraints ($1 \leq$ `ha(`k`)` $\leq$ `nncon`), those entries must come before any other (linear) entries.
5. The row indices `ha(`k`)` for a column may be in any order (subject to Jacobian entries appearing first). Subroutine `funcon` must define Jacobian entries in the same order.
6. Columns of A should contain at least one entry, so that `ka(`j`)` < `ka(`$j+1$`)` for every j. If a column has no meaningful entry, include a dummy entry `a(`k`)` = `0.0d+0`, `ha(`k`)` = 1.

`bl(nb)` is the lower bounds on the variables and slacks (x, s).

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

`bu(nb)` is the upper bounds on (x, s).

Beware: MINOS represents general constraints as $Ax + s = 0$. Constraints of the form $l \leq Ax \leq u$ therefore mean $l \leq -s \leq u$, so that $-u \leq s \leq -l$. The last m components of `bl` and `bu` are $-u$ and $-l$.

name1(nname), name2(nname) are integer arrays.

If **nname** = 1, **name1** and **name2** are not used. The printed solution will use generic names for the columns and rows. If **nname** = nb, **name1**(j) and **name2**(j) should contain the name of the j-th variable in **2a4** format (j = 1 to nb). If $j = n + i$, the j-th variable is the i-th row.

hs(nb) sometimes contains a set of initial states for each variable x, or for each variable and slack (x, s). See next lines.

xn(nb) sometimes contains a set of initial values for each variable x, or for each variable and slack (x, s).

1. For cold starts, you must define **hs**(j) and **xn**(j), j = 1 to n. (The values for $j = n + 1$ to nb need not be set.) If nothing special is known about the problem, or if there is no wish to provide special information, you may set **hs**(j) = 0, **xn**(j) = 0.0 for all j = 1 to n. All variables will be eligible for the initial basis.

   Less trivially, to say that variable j will probably be equal to one of its bounds, set **hs**(j) = 4 and **xn**(j) = **bl**(j) or **hs**(j) = 5 and **xn**(j) = **bu**(j) as appropriate.

2. For Cold starts with no basis file, a Crash procedure is used to select an initial basis. The initial basis matrix will be triangular (ignoring certain small entries in each column). The values **hs**(j) = 0, 1, 2, 3, 4, 5 have the following meaning:

   If **hs**(j) = 0, 1 or 3, Crash considers that column j is eligible for the basis, with preference given to 3.

   If **hs**(j) = 2, 4 or 5, Crash ignores column j.

   After Crash, columns for which **hs**(j) = 2 are made superbasic. Other columns not selected for the basis are made nonbasic at the value **xn**(j) if **bl**(j) $\le$ **xn**(j) $\le$ **bu**(j), or at the value **bl**(j) or **bu**(j) closest to **xn**(j).

3. For Warm or Hot starts, all of **hs(1:nb)** is assumed to be set to the values 0, 1, 2 or 3 (probably from some previous call) and all of **xn(1:nb)** must have values.

If **start** = 'Cold' or Basis file" and an OLD BASIS, INSERT or LOAD file is provided, **hs** and **xn** need not be set at all.

pi(m) contains an estimate of the vector of Lagrange multipliers (shadow prices) for the nonlinear constraints. The first **nncon** components must be defined. They will be used as λ_k in the subproblem objective function for the first major iteration. If nothing is known about λ_k, set **pi**(i) = 0.0d+0, i = 1 to nncon.

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

`z(nwcore)` is a (large) array that provides all workspace. Problems involving m general constraints typically need `nwcore` at least $100m$. See the output parameter `mincor` below.

On exit:

`hs(nb)` is the final state vector. If the solution is optimal or feasible, the entries of `hs` usually have the following meaning:

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

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).

`xn(nb)` is the final variables and slacks (x, s).

`pi(m)` is the vector of dual variables π (a set of Lagrange multipliers for the general constraints).

`rc(nb)` is a 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 $-\pi$.

`inform` says what happened, as described more fully in Chapter 6.3. The next page summarizes the possible values.

inform	*Meaning*
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	Subroutine **funobj** or **funcon** requested termination by returning **mode** < 0.
7	**funobj** seems to be giving incorrect gradients.
8	**funcon** seems to be giving incorrect gradients.
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.

inform	*Meaning*
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.

mincor says how much storage is needed to solve the problem. If **inform** = 42, the work array **z(nwcore)** was too small. **minoss** may be called again with **nwcore** suitably larger than **mincor**. (The bigger the better, since it is not certain how much storage the basis factors need.)

ns is the final number of superbasics.

ninf is the number of infeasibilities.

sinf is the sum of infeasibilities.

obj is the value of the objective function. If **ninf** = 0, **obj** includes the nonlinear objective if any. If **ninf** > 0, **obj** is just the linear objective if any.

B.2 SUBROUTINE MISPEC

This subroutine must be called before the first call to **minoss**. It opens the SPECS, PRINT and SUMMARY files (if they exist), sets all options to default values, and reads the SPECS file if any. File numbers must be in the range 1 to 99, or 0 if the associated file does not exist.

Specification

```
subroutine mispec( ispecs, iprint, isumm, nwcore, inform )

integer            ispecs, iprint, isumm, nwcore, inform
```

On entry:

ispecs says whether or not a SPECS file exists. If **ispecs** > 0, a file is read from the specified Fortran file number. Typically **ispecs** = 4.

iprint says if a PRINT file is to be created. Typically **iprint** = 9.

isumm says if a SUMMARY file is to be created. Typically **isumm** = 6. In an interactive environment, this value usually denotes the screen.

nwcore is the length of the workspace array **z(*)** that is later passed to **minoss**.

On exit:

inform is 0 if there was no SPECS file, or if the SPECS file was successfully read. Otherwise, it returns the number of errors encountered.

B.3 SUBROUTINES MIOPT, MIOPTI, MIOPTR

These subroutines may be called from the program that calls **minoss**. They specify a single option that might otherwise be defined in one line of a SPECS file.

Specification

```
subroutine miopt ( buffer,         iprint, isumm, inform )
subroutine miopti( buffer, ivalue, iprint, isumm, inform )
subroutine mioptr( buffer, rvalue, iprint, isumm, inform )

character*(*)      buffer
integer            ivalue
double precision   rvalue
integer            iprint, isumm, inform
```

On entry:

buffer is a string to be decoded as if it were a line of a SPECS file. For **miopt**, the maximum length of **buffer** is 72 characters. Use **miopt** if the string contains all of the data associated with a particular keyword. For example,

```
call miopt ( 'Iterations 1000',    iprint, isumm, inform )
```

is suitable if the value 1000 is known at compile time.

For **miopti** and **mioptr** the maximum length of **buffer** is 55 characters.

ivalue is an integer value associated with the keyword in **buffer**. Use **miopti** if it is convenient to define the value at run time. For example,

```
itnlim = 1000
if (m .gt. 500) itnlim = 8000
call miopti( 'Iterations', itnlim, iprint, isumm, inform )
```

allows the iteration limit to be computed.

rvalue is a floating-point value associated with the keyword in **buffer**. Use **mioptr** if it is convenient to define the value at run time. For example,

```
factol = 100.0d+0
if ( illcon ) factol = 5.0d+0
call mioptr( 'LU factor tol', factol, iprint, isumm, inform )
```

allows the LU stability tolerance to be computed.

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

isumm is a file number for printing any error messages. **isumm** = 0 suppresses this output.

inform should be 0.

On exit:

inform is the number of errors encountered so far.

B.4 EXAMPLE USE OF MINOSS

File minost.for contains a Fortran test program to illustrate the use of subroutines mispec, minoss, miopt, miopti and mioptr. The test program reads a SPECS file, generates test problem MANNE (see Pages 98–108 of the User's Guide), sets some options not specified in the SPECS file, then calls minoss to solve the problem.

The SPECS file is in minost.spc. The required function subroutines funobj and funcon are part of the MINOS source file mi05funs.for.

To use the test program, compile and link minost.for and all of the MINOS source files, excluding the stand-alone MINOS main program (mi00main.for). See file unix.mak or minost.mak.

To run the resulting binary file, see file unix.run or vminost.com.

Good luck with your own use of minoss!

File minost.for

```
*     ------------------------------------------------------------------
*     File minost.for
*     This is a main program to test subroutine minoss, which is
*     part of MINOS 5.5.  It generates the problem called MANNE on
*     Pages 98-108 of the MINOS 5.1 User's Guide, then asks minoss
*     to solve it.
*
*
*     11 Nov 1991: First version.
*     27 Nov 1991: miopt, miopti, mioptr used to alter some options
*                  for a second call to minoss.
*     10 Apr 1992: objadd added as input parameter to minoss.
*     26 Jun 1992: integer*2 changed to integer*4.
*     15 Oct 1993: t4data now outputs pi.
*     24 Jan 1995: MINOS inadvertently scales all of xn before solving,
*                  so t4data sets dummy values for the slacks after all.
*     05 Feb 1998: No longer have to set Jacobian = dense or sparse
*                  when MINOS is called as a subroutine.
*     ------------------------------------------------------------------

      program           minost

      implicit          double precision (a-h,o-z)

      parameter        ( maxm   = 100,
     $                   maxn   = 150,
     $                   maxnb  = maxm + maxn,
     $                   maxne  = 500,
     $                   nname  = 1 )

      character*8       names(5)
      integer*4         ha(maxne) , hs(maxnb)
      integer           ka(maxn+1), name1(nname), name2(nname)
      double precision  a(maxne)  , bl(maxnb)   , bu(maxnb),
```

```
     $                  xn(maxnb) , pi(maxm)     , rc(maxnb)

      parameter         ( nwcore = 50000 )
      double precision  z(nwcore)
*     ------------------------------------------------------------------

*     Give names to the Problem, Objective, Rhs, Ranges and Bounds.

      names(1) = 'manne10 '
      names(2) = 'funobj  '
      names(3) = 'zero    '
      names(4) = 'range1  '
      names(5) = 'bound1  '

*     Specify some of the MINOS files.
*     ispecs  is the Specs file   (0 if none).
*     iprint  is the Print file   (0 if none).
*     isumm   is the Summary file (0 if none).
*     (mispec opens these files via mifile and miopen.)
*     nout    is an output file used here by mitest.

      ispecs = 4
      iprint = 9
      isumm  = 6
      nout   = 6

*     ------------------------------------------------------------------
*     Set options to default values.
*     Read a Specs file (if ispecs > 0).
*     ------------------------------------------------------------------
      call mispec( ispecs, iprint, isumm, nwcore, inform )

      if (inform .ge. 2) then
         write(nout, *) 'ispecs > 0 but no Specs file found'
         stop
      end if

*     ------------------------------------------------------------------
*     Generate a 10-period problem (nt = 10).
*     Instead of hardwiring nt here, we could do the following:
*     1. Say    Nonlinear constraints  10      in the Specs file.
*     2. At the top of this program include the following common block:
*               common    /m8len / njac  ,nncon ,nncon0,nnjac
*     3. Say    nt = nncon    in the line below.
*     ------------------------------------------------------------------
      nt     = 10
      call t4data( nt, maxm, maxn, maxnb, maxne, inform,
     $             m, n, nb, ne, nncon, nnobj, nnjac,
     $             a, ha, ka, bl, bu, hs, xn, pi )

      if (inform .ge. 1) then
```

```
         write(nout, *) 'Not enough storage to generate a problem ',
     $                  'with  nt =', nt
         stop
       end if

*      ------------------------------------------------------------------
*      Specify options that were not set in the Specs file.
*      i1 and i2 may refer to the Print and Summary file respectively.
*      Setting them to 0 suppresses printing.
*      ------------------------------------------------------------------
       i1     = 0
       i2     = 0
       ltime  = 2
       call miopti( 'Timing level      ', ltime, i1, i2, inform )

*      ------------------------------------------------------------------
*      Go for it, using a Cold start.
*      iobj   = 0 means there is no linear objective row in a(*).
*      objadd = 0.0 means there is no constant to be added to the
*             objective.
*      hs     need not be set if a basis file is to be input.
*             Otherwise, each hs(1:n) should be 0, 1, 2, 3, 4, or 5.
*             The values are used by the Crash procedure m2crsh
*             to choose an initial basis B.
*             If hs(j) = 0 or 1, column j is eligible for B.
*             If hs(j) = 2, column j is initially superbasic (not in B).
*             If hs(j) = 3, column j is eligible for B and is given
*                           preference over columns with hs(j) = 0 or 1.
*             If hs(j) = 4 or 5, column j is initially nonbasic.
*      ------------------------------------------------------------------
       iobj   = 0
       objadd = 0.0

*      For straightforward applications we would call minoss just once,
*      giving it all of z(*) for workspace.
*      Here we call it twice to illustrate situations where z(*) can be
*      expanded to suit the problem size.
*
*      For the first call, set lenz foolishly small and let minoss
*      tell us (via mincor) how big it would like z(*) to be.

       lenz   = 2
       call minoss( 'Cold', m, n, nb, ne, nname,
     $              nncon, nnobj, nnjac,
     $              iobj, objadd, names,
     $              a, ha, ka, bl, bu, name1, name2,
     $              hs, xn, pi, rc,
     $              inform, mincor, ns, ninf, sinf, obj,
     $              z, lenz )

       write(nout, *) ' '
```

```
      write(nout, *) 'Estimate of required workspace: mincor =', mincor

*     Since nwcor2 was not big enough, we will now have inform = 42.
*     Make z(*) longer and try again.  mincor SHOULD be enough.
*     (In general we should allow more to give the LU factors
*     as much room as possible).  For example,
*     mincor = mincor + 5*m + 1000   might be enough.)
*
*     Note that we can't say z(*) is longer than nwcore here.
*     minoss will return inform = 42 again if mincor > nwcore.

      lenz   = min( mincor, nwcore )

      call minoss( 'Cold', m, n, nb, ne, nname,
     $             nncon, nnobj, nnjac,
     $             iobj, objadd, names,
     $             a, ha, ka, bl, bu, name1, name2,
     $             hs, xn, pi, rc,
     $             inform, mincor, ns, ninf, sinf, obj,
     $             z, lenz )

      write(nout, *) ' '
      write(nout, *) 'minoss finished.'
      write(nout, *) 'inform =', inform
      write(nout, *) 'ninf   =', ninf
      write(nout, *) 'sinf   =', sinf
      write(nout, *) 'obj    =', obj
      if (inform .ge. 20) go to 900

*     ------------------------------------------------------------------
*     Alter some options and test the Warm start.
*     ------------------------------------------------------------------

*     The following illustrates the use of miopt, miopti and mioptr
*     to set specific options.  If necessary, we could ensure that
*     all unspecified options take default values
*     by first calling miopt ( 'Defaults', ... ).
*     Beware that certain parameters would then need to be redefined.
      write(nout, *) ' '
      write(nout, *) 'Alter options and test Warm start:'

      inform = 0
      itnlim = 20
      penpar = 0.01
      call miopt ( '                    ',         iprint, isumm, inform )
*---  call miopt ( 'Defaults            ',         iprint, isumm, inform )
*---  call miopti( 'Problem number      ',   1114, iprint, isumm, inform )
*---  call miopt ( 'Maximize            ',         iprint, isumm, inform )
      call miopt ( 'Derivative level 3',         iprint, isumm, inform )
*---  call miopt ( 'Print  level     0',         iprint, isumm, inform )
      call miopt ( 'Verify level     0',         iprint, isumm, inform )
```

```
      call miopt ( 'Scale option     0',         iprint, isumm, inform )
      call miopti( 'Iterations        ', itnlim, iprint, isumm, inform )
      call mioptr( 'Penalty parameter ', penpar, iprint, isumm, inform )

      if (inform .gt. 0) then
         write(nout, *) 'NOTE: Some of the options were not recognized'
      end if

*     Test the Warm start.
*     hs(*) specifies a complete basis from the previous call.
*     A Warm start uses hs(*) directly, without calling Crash.
*
*     Warm and Hot starts are normally used after minoss has solved a
*     problem with the SAME DIMENSIONS but perhaps altered data.
*     Here we have not altered the data, so very few iterations
*     should be required.

      call minoss( 'Warm', m, n, nb, ne, nname,
     $             nncon, nnobj, nnjac,
     $             iobj, objadd, names,
     $             a, ha, ka, bl, bu, name1, name2,
     $             hs, xn, pi, rc,
     $             inform, mincor, ns, ninf, sinf, obj,
     $             z, nwcore )

      write(nout, *) ' '
      write(nout, *) 'minoss finished again.'
      write(nout, *) 'inform =', inform
      write(nout, *) 'obj    =', obj
      if (inform .ge. 20) go to 900

*     ------------------------------------------------------------------
*     Alter more options (perhaps) and test the Hot start.
*     As with a Warm start, hs(*) specifies a basis from the
*     previous call.  In addition, up to three items from the previous
*     call can be reused.  They are denoted by F, H and S as follows:
*     'Hot F'    means use the existing basis FACTORS (B = LU).
*     'Hot H'    means use the existing reduced HESSIAN approximation.
*     'Hot S'    means use the existing column and row SCALES.
*     'Hot FS'   means use the Factors and Scales but not the Hessian.
*     'Hot FHS'  means use all three items.
*     'Hot'      is equivalent to 'Hot FHS'.
*     The letters F,H,S may be in any order.
*     Note that 'Hot' keeps existing scales.  Must say
*     'Hot H' or 'Hot ...' or something longer than 4 characters
*     if new scales are wanted.
*     ------------------------------------------------------------------
      write(nout, *) ' '
      write(nout, *) 'Test Hot start:'
      call miopt ( '                ',         iprint, isumm, inform )
      call miopt ( 'Scale option    2',         iprint, isumm, inform )
```

```
      call minoss( 'Hot H', m, n, nb, ne, nname,
     $             nncon, nnobj, nnjac,
     $             iobj, objadd, names,
     $             a, ha, ka, bl, bu, name1, name2,
     $             hs, xn, pi, rc,
     $             inform, mincor, ns, ninf, sinf, obj,
     $             z, nwcore )

      write(nout, *) ' '
      write(nout, *) 'minoss finished again.'
      write(nout, *) 'inform =', inform
      write(nout, *) 'obj    =', obj
      if (inform .ge. 20) go to 900
      stop

*     ------------------------------------------------------------------
*     Error exit.
*     ------------------------------------------------------------------
  900 write(nout, *) ' '
      write(nout, *) 'STOPPING because of error condition'
      stop

*     end of main program to test subroutine minoss
      end

*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      subroutine t4data( nt, maxm, maxn, maxnb, maxne, inform,
     $                   m, n, nb, ne, nncon, nnobj, nnjac,
     $                   a, ha, ka, bl, bu, hs, xn, pi )

      implicit           double precision (a-h,o-z)
      integer*4          ha(maxne), hs(maxnb)
      integer            ka(maxn+1)
      double precision   a(maxne) , bl(maxnb), bu(maxnb),
     $                   xn(maxnb), pi(maxm)

*     ------------------------------------------------------------------
*     t4data  generates data for the test problem t4manne
*     (called problem MANNE in the MINOS 5.1 User's Guide).
*     The constraints take the form
*              f(x) + A*x + s = 0,
*     where the Jacobian for f(x) + Ax is stored in a(*), and any
*     terms coming from f(x) are in the TOP LEFT-HAND CORNER of a(*),
*     with dimensions  nncon x nnjac.
*     Note that the right-hand side is zero.
*     s is a set of slack variables whose bounds contain any constants
*     that might have formed a right-hand side.
*
*     The objective function is
```

```
*            F(x) + c'x
*     where c would be row iobj of A (but there is no such row in
*     this example).  F(x) involves only the FIRST nnobj variables.
*
*     On entry,
*     nt       is T, the number of time periods.
*     maxm, maxn, maxnb, maxne are upper limits on m, n, nb, ne.
*
*     On exit,
*     inform  is 0 if there is enough storage, 1 otherwise.
*     m       is the number of nonlinear and linear constraints.
*     n       is the number of variables.
*     nb      is n + m.
*     ne      is the number of nonzeros in a(*).
*     nncon   is the number of nonlinear constraints (they come first).
*     nnobj   is the number of nonlinear objective variables.
*     nnjac   is the number of nonlinear Jacobian variables.
*     a       is the constraint matrix (Jacobian), stored column-wise.
*     ha      is the list of row indices for each nonzero in a(*).
*     ka      is a set of pointers to the beginning of each column of a.
*     bl      is the lower bounds on x and s.
*     bu      is the upper bounds on x and s.
*     hs(1:n) is a set of initial states for each x  (0,1,2,3,4,5).
*     xn(1:n) is a set of initial values for x.
*     pi(1:m) is a set of initial values for the dual variables pi.
*
*     09 Jul 1992: No need to initialize xn and hs for the slacks.
*     15 Oct 1993: pi is now an output parameter.  (Should have been
*                  all along.)
*     24 Jan 1995: MINOS inadvertently scales all of xn before solving,
*                  so we set dummy values for the slacks after all.
*     ------------------------------------------------------------------

      parameter       ( zero   = 0.0d+0,   one    = 1.0d+0,
     $                  dummy  = 0.1d+0,   growth = .03d+0,
     $                  bplus  = 1.0d+20,  bminus = - bplus )

*     nt defines the dimension of the problem.

      m       = nt*2
      n       = nt*3
      nb      = n + m
      nncon   = nt
      nnobj   = nt*2
      nnjac   = nt
      ne      = nt*6 - 1

*     Check if there is enough storage.

      inform = 0
      if (m       .gt. maxm ) inform = 1
```

```
      if (n      .gt. maxn ) inform = 1
      if (nb     .gt. maxnb) inform = 1
      if (ne     .gt. maxne) inform = 1
      if (inform .gt.   0  ) return

*     Generate columns for Capital (Kt, t = 1 to nt).
*     The first nt rows are nonlinear, and the next nt are linear.
*     The Jacobian is an nt x nt diagonal.
*     We generate the sparsity pattern here.
*     We put in dummy numerical values of 0.1 for the gradients.
*     Real values for the gradients are computed by t4con.

      ne     = 0
      do 100  k = 1, nt

*        There is one Jacobian nonzero per column.

         ne     = ne + 1
         ka(k)  = ne
         ha(ne) = k
         a(ne)  = dummy

*        The linear constraints form an upper bidiagonal pattern.

         if (k .gt. 1) then
            ne     = ne + 1
            ha(ne) = nt + k - 1
            a(ne)  = one
         end if

         ne     = ne + 1
         ha(ne) = nt + k
         a(ne)  = - one
  100 continue

*     The last nonzero is special.

      a(ne)  = growth

*     Generate columns for Consumption (Ct for t = 1 to nt).
*     They form -I in the first nt rows.
*     jC and jI are base indices for the Ct and It variables.

      jC    = nt
      jI    = nt*2

      do 200 k = 1, nt
         ne       = ne + 1
         ka(jC+k) = ne
         ha(ne)   = k
         a(ne)    = - one
```

```
  200 continue

*     Generate columns for Investment (It for t = 1 to nt).
*     They form -I in the first nt rows and -I in the last nt rows.

      do 300 k = 1, nt
         ne       = ne + 1
         ka(jI+k) = ne
         ha(ne)   = k
         a(ne)    = - one
         ne       = ne + 1
         a(ne)    = - one
         ha(ne)   = nt + k
  300 continue

*     ka(*) has one extra element.

      ka(n+1) = ne + 1

*     Set lower and upper bounds for Kt, Ct, It.
*     Also initial values and initial states for all variables.
*     The Jacobian variables are the most important.
*     Set hs(k) = 2 to make them initially superbasic.
*     The others might as well be on their smallest bounds (hs(j) = 0).

      do 400  k = 1, nt
         bl(   k) = 3.05d+0
         bu(   k) = bplus
         bl(jC+k) = 0.95d+0
         bu(jC+k) = bplus
         bl(jI+k) = 0.05d+0
         bu(jI+k) = bplus

         xn(   k) = 3.0d+0 + (k - 1)/10.0d+0
         xn(jC+k) = bl(jC+k)
         xn(jI+k) = bl(jI+k)

         hs(   k) = 2
         hs(jC+k) = 0
         hs(jI+k) = 0
  400 continue

*     The first Capital is fixed.
*     The last three Investments are bounded.

      bu(1)       = bl(1)
      xn(1)       = bl(1)
      hs(1)       = 0
      bu(jI+nt-2) = 0.112d+0
      bu(jI+nt-1) = 0.114d+0
      bu(jI+nt  ) = 0.116d+0
```

```
*     Set bounds on the slacks.
*     The nt nonlinear (Money)    rows are >=.
*     The nt    linear (CapacitY) rows are <=.
*     We no longer need to set initial values and states for slacks.
*     24 Jan 1995: MINOS inadvertently scales all of xn before solving,
*                  so we set dummy values for the slacks after all.

      jM     = n
      jY     = n + nt

      do 500    k = 1, nt
         bl(jM+k) = bminus
         bu(jM+k) = zero
         bl(jY+k) = zero
         bu(jY+k) = bplus

         xn(jM+k) = zero
         xn(jY+k) = zero
*-       hs(jM+k) = 0
*-       hs(jY+k) = 0
  500 continue

*     The last Money and Capacity rows have a Range.

      bl(jM+nt) = - 10.0d+0
      bu(jY+nt) =   20.0d+0

*     Initialize pi.
*     5.4 requires only pi(1:nncon) to be initialized.
*     5.5 may want all of pi to be initialized (not yet sure).

      do 600 i = 1, nt
         pi(i)    = - one
         pi(nt+i) = + one
  600 continue

*     end of t4data
      end
```

File *minost.spc*

```
Begin manne10  (10-period economic growth model)

   Problem number          1114
   Maximize
   Major iterations           8
   Minor iterations          20
   Penalty parameter        0.1

   Hessian dimension         10
   Derivative level           3
*   Verify gradients
   Verify level               0

   Scale option               2
   Scale option               1
   Iterations                50
   Print level (jflxb)    00000
   Print frequency            1
   Summary level              0
   Summary frequency          1
End Manne10
```

B.5 MINOS(IIS): DEBUGGING INFEASIBLE MODELS

If the linear constraints in a model cannot be satisfied, MINOS will exit with the message "The problem is infeasible". This usually implies some formulation error in the model. The printed solution shows which variables or slacks lie outside their bounds, and by how much. However, the exact cause of infeasibility may be difficult to detect.

In such cases, further analysis is provided by MINOS(IIS), a modified version of MINOS available from John Chinneck at Carleton University:

> J. W. Chinneck (1993). MINOS(IIS) 4.2 User's Manual, Report SCE-93-17, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada K1S 5B6.
>
> Phone: (613)788-5733, Fax: (613)788-5727, Email: chinneck@sce.carleton.ca.