NAG Toolbox for MATLAB

Chapter Introduction

E04 – Minimizing or Maximizing a Function

Contents

1 Scope of the Chapter

An optimization problem involves minimizing a function (called the **objective function**) of several variables, possibly subject to restrictions on the values of the variables defined by a set of **constraint functions**. Most functions in the Library are concerned with function **minimization** only, since the problem of maximizing a given objective function *F(x)* is equivalent to minimizing $-F(x)$. Some functions allow you to specify whether you are solving a minimization or maximization problem, carrying out the required transformation of the objective function in the latter case.

This introduction is only a brief guide to the subject of optimization designed for the casual user. Anyone with a difficult or protracted problem to solve will find it beneficial to consult a more detailed text, such as Gill *et al.* 1981 or Fletcher 1987.

If you are unfamiliar with the mathematics of the subject you may find some sections difficult at first reading; if so, you should **concentrate** on Sections 2.1, 2.2, 2.5, 2.6 and 3.

2 Background to the Problems

2.1 Types of Optimization Problems

The solution of optimization problems by a single, all-purpose, method is cumbersome and inefficient. Optimization problems are therefore classified into particular categories, where each category is defined by the properties of the objective and constraint functions, as illustrated by some examples below.

Properties of Objective Function	**Properties of Constraints**
Nonlinear	Nonlinear
Sums of squares of nonlinear functions	Sparse linear
Quadratic	Linear
Sums of squares of linear functions	Bounds
Linear	None

For instance, a specific problem category involves the minimization of a nonlinear objective function subject to bounds on the variables. In the following sections we define the particular categories of problems that can be solved by functions contained in this chapter. Not every category is given special treatment in the current version of the Library; however, the long-term objective is to provide a comprehensive set of functions to solve problems in all such categories.

2.1.1 Unconstrained minimization

In unconstrained minimization problems there are no constraints on the variables. The problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}}\, F(x)$$

where $x \in R^n$, that is, $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$.

2.1.2 Nonlinear least-squares problems

Special consideration is given to the problem for which the function to be minimized can be expressed as a sum of squared functions. The least-squares problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}}\left\{f^{\mathrm{T}}f = \sum_{i=1}^{m} f_i^2(x)\right\}, \qquad x \in R^n$$

where the ith element of the m-vector f is the function $f_i(x)$.

2.1.3 Minimization subject to bounds on the variables

These problems differ from the unconstrained problem in that at least one of the variables is subject to a simple bound (or restriction) on its value, e.g., $x_5 \leq 10$, but no constraints of a more general form are present.

The problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}}\, F(x), \qquad x \in R^n$$

subject to $l_i \leq x_i \leq u_i$, for $i = 1, 2, \ldots, n$.

This format assumes that upper and lower bounds exist on all the variables. By conceptually allowing $u_i = +\infty$ and $l_i = -\infty$ all the variables need not be restricted.

2.1.4 Minimization subject to linear constraints

A general linear constraint is defined as a constraint function that is linear in more than one of the variables, e.g., $3x_1 + 2x_2 \geq 4$. The various types of linear constraint are reflected in the following mathematical statement of the problem:

$$\underset{x}{\text{minimize}}\, F(x), \qquad x \in R^n$$

subject to the

$$\begin{array}{lll}
\text{equality constraints:} & a_i^{\mathrm{T}}x = b_i & i = 1, 2, \ldots, m_1; \\
\text{inequality constraints:} & a_i^{\mathrm{T}}x \geq b_i & i = m_1 + 1, m_1 + 2, \ldots, m_2; \\
 & a_i^{\mathrm{T}}x \leq b_i & i = m_2 + 1, m_2 + 2, \ldots, m_3; \\
\text{range constraints:} & s_j \leq a_i^{\mathrm{T}}x \leq t_j & i = m_3 + 1, m_3 + 2, \ldots, m_4; \\
 & & j = 1, 2, \ldots, m_4 - m_3; \\
\text{bounds constraints:} & l_i \leq x_i \leq u_i & i = 1, 2, \ldots, n
\end{array}$$

where each a_i is a vector of length n; b_i, s_j and t_j are constant scalars; and any of the categories may be empty.

Although the bounds on x_i could be included in the definition of general linear constraints, we prefer to distinguish between them for reasons of computational efficiency.

If $F(x)$ is a linear function, the linearly-constrained problem is termed a linear programming problem (LP); if $F(x)$ is a quadratic function, the problem is termed a quadratic programming problem (QP). For further discussion of LP and QP problems, including the dual formulation of such problems, see Dantzig 1963.

2.1.5 Minimization subject to nonlinear constraints

A problem is included in this category if at least one constraint function is nonlinear, e.g., $x_1^2 + x_3 + x_4 - 2 \geq 0$. The mathematical statement of the problem is identical to that for the linearly-constrained case, except for the addition of the following constraints:

$$\begin{array}{lll}
\text{equality constraints:} & c_i(x) = 0 & i = 1, 2, \ldots, m_5; \\
\text{inequality constraints:} & c_i(x) \geq 0 & i = m_5 + 1, m_5 + 2, \ldots, m_6; \\
\text{range constraints:} & v_j \leq c_i(x) \leq w_j & i = m_6 + 1, m_6 + 2, \ldots, m_7, \\
 & & j = 1, 2, \ldots, m_7 - m_6
\end{array}$$

where each c_i is a nonlinear function; v_j and w_j are constant scalars; and any category may be empty. Note that we do not include a separate category for constraints of the form $c_i(x) \leq 0$, since this is equivalent to $-c_i(x) \geq 0$.

Although the general linear constraints could be included in the definition of nonlinear constraints, again we prefer to distinguish between them for reasons of computational efficiency.

If $F(x)$ is a nonlinear function, the nonlinearly-constrained problem is termed a nonlinear programming problem (NLP). For further discussion of NLP problems, see Gill *et al.* 1981 or Fletcher 1987.

2.1.6 Minimization subject to bounds on the objective function

In all of the above problem categories it is assumed that

$$a \leq F(x) \leq b$$

where $a = -\infty$ and $b = +\infty$. Problems in which a and/or b are finite can be solved by adding an extra constraint of the appropriate type (i.e., linear or nonlinear) depending on the form of $F(x)$. Further advice is given in Section 3.4.

2.2 Geometric Representation and Terminology

To illustrate the nature of optimization problems it is useful to consider the following example in two dimensions:

$$F(x) = e^{x_1}\left(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1\right).$$

(This function is used as the example function in the documentation for the unconstrained functions.)

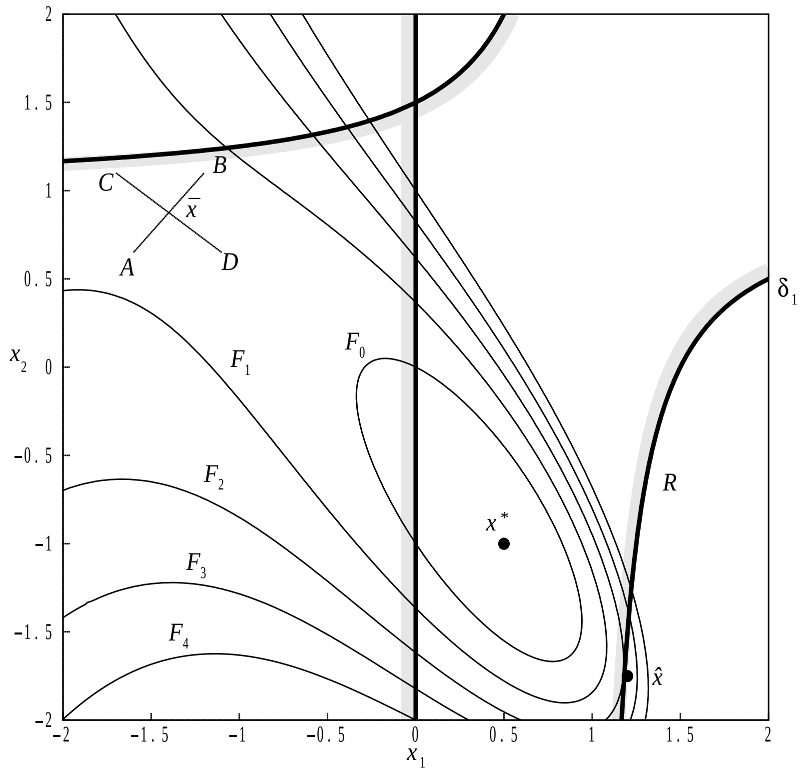


Figure 1

Figure 1 is a contour diagram of $F(x)$. The contours labelled $F_0, F_1, \ldots, F_4$ are isovalue contours, or lines along which the function $F(x)$ takes specific constant values. The point $x^* = \left(\frac{1}{2}, -1\right)^{\mathrm{T}}$ is a **local unconstrained minimum**, that is, the value of $F(x^*)$ ($= 0$) is less than at all the neighbouring points. A function may have several such minima. The lowest of the local minima is termed a **global minimum**. In the problem illustrated in Figure 1, x^* is the only local minimum. The point $\bar{x}$ is said to be a **saddle point** because it is a minimum along the line AB, but a maximum along CD.

If we add the constraint $x_1 \geq 0$ (a simple bound) to the problem of minimizing $F(x)$, the solution remains unaltered. In Figure 1 this constraint is represented by the straight line passing through $x_1 = 0$, and the shading on the line indicates the unacceptable region (i.e., $x_1 < 0$). The region in R^n satisfying the constraints of an optimization problem is termed the **feasible region**. A point satisfying the constraints is defined as a **feasible point**.

If we add the nonlinear constraint $c_1(x) : x_1 + x_2 - x_1x_2 - \frac{3}{2} \geq 0$, represented by the curved shaded line in Figure 1, then x^* is not a feasible point because $c_1(x^*) < 0$. The solution of the new constrained problem is $\hat{x} \simeq (1.1825, -1.7397)^{\mathrm{T}}$, the feasible point with the smallest function value (where $F(\hat{x}) \simeq 3.0607$).

2.2.1 Gradient vector

The vector of first partial derivatives of $F(x)$ is called the **gradient vector**, and is denoted by $g(x)$, i.e.,

$$g(x) = \left[\frac{\partial F(x)}{\partial x_1}, \frac{\partial F(x)}{\partial x_2}, \ldots, \frac{\partial F(x)}{\partial x_n}\right]^{\mathrm{T}}.$$

For the function illustrated in Figure 1,

$$g(x) = \begin{bmatrix} F(x) + e^{x_1}(8x_1 + 4x_2) \\ e^{x_1}(4x_2 + 4x_1 + 2) \end{bmatrix}.$$

The gradient vector is of importance in optimization because it must be zero at an unconstrained minimum of any function with continuous first derivatives.

2.2.2 Hessian matrix

The matrix of second partial derivatives of a function is termed its **Hessian matrix**. The Hessian matrix of $F(x)$ is denoted by $G(x)$, and its (i,j)th element is given by $\partial^2 F(x)/\partial x_i \partial x_j$. If $F(x)$ has continuous second derivatives, then $G(x)$ must be positive semi-definite at any unconstrained minimum of F.

2.2.3 Jacobian matrix; matrix of constraint normals

In nonlinear least-squares problems, the matrix of first partial derivatives of the vector-valued function $f(x)$ is termed the **Jacobian matrix** of $f(x)$ and its (i,j)th component is $\partial f_i/\partial x_j$.

The vector of first partial derivatives of the constraint $c_i(x)$ is denoted by

$$a_i(x) = \left[\frac{\partial c_i(x)}{\partial x_1}, \frac{\partial c_i(x)}{\partial x_2}, \ldots, \frac{\partial c_i(x)}{\partial x_n}\right]^{\mathrm{T}}.$$

The matrix whose columns are the vectors $\{a_i\}$ is termed the **matrix of constraint normals**. At a point $\hat{x}$, the vector $a_i(\hat{x})$ is orthogonal (normal) to the isovalue contour of $c_i(x)$ passing through $\hat{x}$; this relationship is illustrated for a two-dimensional function in Figure 2.

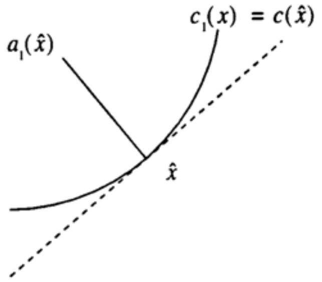


Figure 2

Note that if $c_i(x)$ is a linear constraint involving $a_i^{\mathrm{T}}x$, then its vector of first partial derivatives is simply the vector a_i.

2.3 Sufficient Conditions for a Solution

All nonlinear functions will be assumed to have continuous second derivatives in the neighbourhood of the solution.

2.3.1 Unconstrained minimization

The following conditions are sufficient for the point x^* to be an unconstrained local minimum of $F(x)$:

(i) $\|g(x^*)\| = 0$; and

(ii) $G(x^*)$ is positive-definite,

where $\|g\|$ denotes the Euclidean length of g.

2.3.2 Minimization subject to bounds on the variables

At the solution of a bounds-constrained problem, variables which are not on their bounds are termed **free variables**. If it is known in advance which variables are on their bounds at the solution, the problem can be solved as an unconstrained problem in just the free variables; thus, the sufficient conditions for a solution are similar to those for the unconstrained case, applied only to the free variables.

Sufficient conditions for a feasible point x^* to be the solution of a bounds-constrained problem are as follows:

(i) $\|\bar{g}(x^*)\| = 0$; and

(ii) $\bar{G}(x^*)$ is positive-definite; and

(iii) $g_j(x^*) < 0, x_j = u_j$; $g_j(x^*) > 0, x_j = l_j$,

where $\bar{g}(x)$ is the gradient of $F(x)$ with respect to the free variables, and $\bar{G}(x)$ is the Hessian matrix of $F(x)$ with respect to the free variables. The extra condition (iii) ensures that $F(x)$ cannot be reduced by moving off one or more of the bounds.

2.3.3 Linearly-constrained minimization

For the sake of simplicity, the following description does not include a specific treatment of bounds or range constraints, since the results for general linear inequality constraints can be applied directly to these cases.

At a solution x^*, of a linearly-constrained problem, the constraints which hold as equalities are called the **active** or **binding** constraints. Assume that there are t active constraints at the solution x^*, and let $\hat{A}$ denote the matrix whose columns are the columns of A corresponding to the active constraints, with $\hat{b}$ the vector similarly obtained from b; then

$$\hat{A}^{\mathrm{T}} x^* = \hat{b}.$$

The matrix Z is defined as an $n \times (n - t)$ matrix satisfying:

$$\hat{A}^{\mathrm{T}} Z = 0;$$
$$Z^{\mathrm{T}} Z = I.$$

The columns of Z form an orthogonal basis for the set of vectors orthogonal to the columns of $\hat{A}$.

Define

$g_Z(x) = Z^{\mathrm{T}} g(x)$, the **projected gradient vector** of $F(x)$;

$G_Z(x) = Z^{\mathrm{T}} G(x) Z$, the **projected Hessian matrix** of $F(x)$.

At the solution of a linearly-constrained problem, the projected gradient vector must be zero, which implies that the gradient vector $g(x^*)$ can be written as a linear combination of the columns of $\hat{A}$, i.e., $g(x^*) = \sum_{i=1}^{t} \lambda_i^* \hat{a}_i = \hat{A}\lambda^*$. The scalar λ_i^* is defined as the **Lagrange multiplier** corresponding to the ith active constraint. A simple interpretation of the ith Lagrange multiplier is that it gives the gradient of $F(x)$ along the ith active constraint normal; a convenient definition of the Lagrange multiplier vector (although not a recommended method for computation) is:

$$\lambda^* = \left(\hat{A}^{\mathrm{T}} \hat{A}\right)^{-1} \hat{A}^{\mathrm{T}} g(x^*).$$

Sufficient conditions for x^* to be the solution of a linearly-constrained problem are:

(i) x^* is feasible, and $\hat{A}^{\mathrm{T}} x^* = \hat{b}$; and

(ii) $\|g_Z(x^*)\| = 0$, or equivalently, $g(x^*) = \hat{A}\lambda^*$; and

(iii) $G_Z(x^*)$ is positive-definite; and

(iv) $\lambda_i^* > 0$ if λ_i^* corresponds to a constraint $\hat{a}_i^{\mathrm{T}} x^* \geq \hat{b}_i$;

$\lambda_i^* < 0$ if λ_i^* corresponds to a constraint $\hat{a}_i^{\mathrm{T}} x^* \leq \hat{b}_i$.

The sign of λ_i^* is immaterial for equality constraints, which by definition are always active.

2.3.4 Nonlinearly-constrained minimization

For nonlinearly-constrained problems, much of the terminology is defined exactly as in the linearly-constrained case. The set of active constraints at x again means the set of constraints that hold as equalities at x, with corresponding definitions of $\hat{c}$ and $\hat{A}$: the vector $\hat{c}(x)$ contains the active constraint functions, and the columns of $\hat{A}(x)$ are the gradient vectors of the active constraints. As before, Z is defined in terms of $\hat{A}(x)$ as a matrix such that:

$$\hat{A}^{\mathrm{T}}Z = 0;$$
$$Z^{\mathrm{T}}Z = I$$

where the dependence on x has been suppressed for compactness.

The projected gradient vector $g_Z(x)$ is the vector $Z^{\mathrm{T}}g(x)$. At the solution x^* of a nonlinearly-constrained problem, the projected gradient must be zero, which implies the existence of Lagrange multipliers corresponding to the active constraints, i.e., $g(x^*) = \hat{A}(x^*)\lambda^*$.

The **Lagrangian function** is given by:

$$L(x,\lambda) = F(x) - \lambda^{\mathrm{T}}\hat{c}(x).$$

We define $g_L(x)$ as the gradient of the Lagrangian function; $G_L(x)$ as its Hessian matrix, and $\hat{G}_L(x)$ as its projected Hessian matrix, i.e., $\hat{G}_L = Z^{\mathrm{T}}G_LZ$.

Sufficient conditions for x^* to be the solution of a nonlinearly-constrained problem are:

(i) x^* is feasible, and $\hat{c}(x^*) = 0$; and

(ii) $\|g_Z(x^*)\| = 0$, or, equivalently, $g(x^*) = \hat{A}(x^*)\lambda^*$; and

(iii) $\hat{G}_L(x^*)$ is positive-definite; and

(iv) $\lambda_i^* > 0$ if λ_i^* corresponds to a constraint of the form $\hat{c}_i \geq 0$.

The sign of λ_i^* is immaterial for equality constraints, which by definition are always active.

Note that condition (ii) implies that the projected gradient of the Lagrangian function must also be zero at x^*, since the application of Z^{T} annihilates the matrix $\hat{A}(x^*)$.

2.4 Background to Optimization Methods

All the algorithms contained in this chapter generate an iterative sequence $\left\{x^{(k)}\right\}$ that converges to the solution x^* in the limit, except for some special problem categories (i.e., linear and quadratic programming). To terminate computation of the sequence, a convergence test is performed to determine whether the current estimate of the solution is an adequate approximation. The convergence tests are discussed in Section 2.6.

Most of the methods construct a sequence $\left\{x^{(k)}\right\}$ satisfying:

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)}p^{(k)},$$

where the vector $p^{(k)}$ is termed the **direction of search**, and $\alpha^{(k)}$ is the **steplength**. The steplength $\alpha^{(k)}$ is chosen so that $F\left(x^{(k+1)}\right) < F\left(x^{(k)}\right)$ and is computed using one of the techniques for one-dimensional optimization referred to in Section 2.4.1.

2.4.1 One-dimensional optimization

The Library contains two special functions for minimizing a function of a single variable. Both functions are based on safeguarded polynomial approximation. One function requires function evaluations only and fits a quadratic polynomial whilst the other requires function and gradient evaluations and fits a cubic polynomial. See Section 4.1 of Gill *et al.* 1981.

2.4.2 Methods for unconstrained optimization

The distinctions among methods arise primarily from the need to use varying levels of information about derivatives of $F(x)$ in defining the search direction. We describe three basic approaches to unconstrained problems, which may be extended to other problem categories. Since a full description of the methods would fill several volumes, the discussion here can do little more than allude to the processes involved, and direct you to other sources for a full explanation.

(a) **Newton-type** Methods (Modified Newton Methods)

Newton-type methods use the Hessian matrix $G\left(x^{(k)}\right)$, or a finite-difference approximation to $G\left(x^{(k)}\right)$, to define the search direction. The functions in the Library either require a (sub)program that computes the elements of $G\left(x^{(k)}\right)$ directly, or they approximate $G\left(x^{(k)}\right)$ by finite-differences.

Newton-type methods are the most powerful methods available for general problems and will find the minimum of a quadratic function in one iteration. See Sections 4.4 and 4.5.1 of Gill *et al.* 1981.

(b) **Quasi-Newton** Methods

Quasi-Newton methods approximate the Hessian $G\left(x^{(k)}\right)$ by a matrix $B^{(k)}$ which is modified at each iteration to include information obtained about the curvature of F along the current search direction $p^{(k)}$. Although not as robust as Newton-type methods, quasi-Newton methods can be more efficient because $G\left(x^{(k)}\right)$ is not computed directly, or approximated by finite-differences. Quasi-Newton methods minimize a quadratic function in n iterations. See Section 4.5.2 of Gill *et al.* 1981.

(c) **Conjugate-Gradient** Methods

Unlike Newton-type and quasi-Newton methods, conjugate-gradient methods do not require the storage of an n by n matrix and so are ideally suited to solve large problems. Conjugate-gradient type methods are not usually as reliable or efficient as Newton-type, or quasi-Newton methods. See Section 4.8.3 of Gill *et al.* 1981.

2.4.3 Methods for nonlinear least-squares problems

These methods are similar to those for unconstrained optimization, but exploit the special structure of the Hessian matrix to give improved computational efficiency.

Since

$$F(x) = \sum_{i=1}^{m} f_i^2(x)$$

the Hessian matrix $G(x)$ is of the form

$$G(x) = 2\left(J(x)^{\mathrm{T}}J(x) + \sum_{i=1}^{m} f_i(x)G_i(x)\right),$$

where $J(x)$ is the Jacobian matrix of $f(x)$, and $G_i(x)$ is the Hessian matrix of $f_i(x)$.

In the neighbourhood of the solution, $\|f(x)\|$ is often small compared to $\left\|J(x)^{\mathrm{T}}J(x)\right\|$ (for example, when $f(x)$ represents the goodness-of-fit of a nonlinear model to observed data). In such cases, $2J(x)^{\mathrm{T}}J(x)$ may be an adequate approximation to $G(x)$, thereby avoiding the need to compute or approximate second derivatives of $\{f_i(x)\}$. See Section 4.7 of Gill *et al.* 1981.

2.4.4 Methods for handling constraints

Bounds on the variables are dealt with by fixing some of the variables on their bounds and adjusting the remaining free variables to minimize the function. By examining estimates of the Lagrange multipliers it is possible to adjust the set of variables fixed on their bounds so that eventually the bounds active at the solution should be correctly identified. This type of method is called an **active set method**. One feature of such methods is that, given an initial feasible point, all approximations $x^{(k)}$ are feasible. This approach can be extended to general linear constraints. At a point, x, the set of constraints which hold as equalities being used to predict, or approximate, the set of active constraints is called the **working set**.

Nonlinear constraints are more difficult to handle. If at all possible, it is usually beneficial to avoid including nonlinear constraints during the formulation of the problem. The methods currently implemented in the Library handle nonlinearly constrained problems by transforming them into a sequence of quadratic programming problems. A feature of such methods is that $x^{(k)}$ is not guaranteed to be feasible except in the limit, and this is certainly true of the functions currently in the Library. See Chapter 6, particularly Sections 6.4 and 6.5, of Gill *et al.* 1981.

Anyone interested in a detailed description of methods for optimization should consult the references.

2.5 Scaling

Scaling (in a broadly defined sense) often has a significant influence on the performance of optimization methods. Since convergence tolerances and other criteria are necessarily based on an implicit definition of 'small' and 'large', problems with unusual or unbalanced scaling may cause difficulties for some algorithms. Although there are currently no user-callable scaling functions in the Library, scaling is automatically performed by default in the functions which solve sparse LP, QP or NLP problems and in some newer dense solver functions. The following sections present some general comments on problem scaling.

2.5.1 Transformation of variables

One method of scaling is to transform the variables from their original representation, which may reflect the physical nature of the problem, to variables that have certain desirable properties in terms of optimization. It is generally helpful for the following conditions to be satisfied:

(i) the variables are all of similar magnitude in the region of interest;

(ii) a fixed change in any of the variables results in similar changes in $F(x)$. Ideally, a unit change in any variable produces a unit change in $F(x)$;

(iii) the variables are transformed so as to avoid cancellation error in the evaluation of $F(x)$.

Normally, you should restrict yourself to linear transformations of variables, although occasionally nonlinear transformations are possible. The most common such transformation (and often the most appropriate) is of the form

$$x_{\mathrm{new}} = Dx_{\mathrm{old}},$$

where D is a diagonal matrix with constant coefficients. Our experience suggests that more use should be made of the transformation

$$x_{\mathrm{new}} = Dx_{\mathrm{old}} + v,$$

where v is a constant vector.

Consider, for example, a problem in which the variable x_3 represents the position of the peak of a Gaussian curve to be fitted to data for which the extreme values are 150 and 170; therefore x_3 is known to lie in the range $150--170$. One possible scaling would be to define a new variable $\bar{x}_3$, given by

$$\bar{x}_3 = \frac{x_3}{170}.$$

A better transformation, however, is given by defining $\bar{x}_3$ as

$$\bar{x}_3 = \frac{x_3 - 160}{10}.$$

Frequently, an improvement in the accuracy of evaluation of $F(x)$ can result if the variables are scaled before the functions to evaluate $F(x)$ are coded. For instance, in the above problem just mentioned of Gaussian curve-fitting, x_3 may always occur in terms of the form $(x_3 - x_m)$, where x_m is a constant representing the mean peak position.

2.5.2 Scaling the objective function

The objective function has already been mentioned in the discussion of scaling the variables. The solution of a given problem is unaltered if $F(x)$ is multiplied by a positive constant, or if a constant value is added to $F(x)$. It is generally preferable for the objective function to be of the order of unity in the region of interest; thus, if in the original formulation $F(x)$ is always of the order of 10^{+5} (say), then the value of $F(x)$ should be multiplied by 10^{-5} when evaluating the function within an optimization function. If a constant is added or subtracted in the computation of $F(x)$, usually it should be omitted, i.e., it is better to formulate $F(x)$ as $x_1^2 + x_2^2$ rather than as $x_1^2 + x_2^2 + 1000$ or even $x_1^2 + x_2^2 + 1$. The inclusion of such a constant in the calculation of $F(x)$ can result in a loss of significant figures.

2.5.3 Scaling the constraints

A 'well scaled' set of constraints has two main properties. Firstly, each constraint should be well-conditioned with respect to perturbations of the variables. Secondly, the constraints should be balanced with respect to each other, i.e., all the constraints should have 'equal weight' in the solution process.

The solution of a linearly- or nonlinearly-constrained problem is unaltered if the ith constraint is multiplied by a positive weight w_i. At the approximation of the solution determined by a Library function, any active linear constraints will (in general) be satisfied 'exactly' (i.e., to within the tolerance defined by ***machine precision***) if they have been properly scaled. This is in contrast to any active nonlinear constraints, which will not (in general) be satisfied 'exactly' but will have 'small' values (for example, $\hat{c}_1(x^*) = 10^{-8}$, $\hat{c}_2(x^*) = -10^{-6}$, and so on). In general, this discrepancy will be minimized if the constraints are weighted so that a unit change in x produces a similar change in each constraint.

A second reason for introducing weights is related to the effect of the size of the constraints on the Lagrange multiplier estimates and, consequently, on the active set strategy. This means that different sets of weights may cause an algorithm to produce different sequences of iterates. Additional discussion is given in Gill *et al.* 1981.

2.6 Analysis of Computed Results

2.6.1 Convergence criteria

The convergence criteria inevitably vary from function to function, since in some cases more information is available to be checked (for example, is the Hessian matrix positive-definite?), and different checks need to be made for different problem categories (for example, in constrained minimization it is necessary to verify whether a trial solution is feasible). Nonetheless, the underlying principles of the various criteria are the same; in non-mathematical terms, they are:

(i) is the sequence $\left\{x^{(k)}\right\}$ converging?

(ii) is the sequence $\left\{F^{(k)}\right\}$ converging?

(iii) are the necessary and sufficient conditions for the solution satisfied?

The decision as to whether a sequence is converging is necessarily speculative. The criterion used in the present functions is to assume convergence if the relative change occurring between two successive iterations is less than some prescribed quantity. Criterion (iii) is the most reliable but often the conditions cannot be checked fully because not all the required information may be available.

2.6.2 Checking results

Little *a priori* guidance can be given as to the quality of the solution found by a nonlinear optimization algorithm, since no guarantees can be given that the methods will not fail. Therefore, you should always check the computed solution even if the function reports success. Frequently a 'solution' may have been found even when the function does not report a success. The reason for this apparent contradiction is that the function needs to assess the accuracy of the solution. This assessment is not an exact process and consequently may be unduly pessimistic. Any 'solution' is in general only an approximation to the exact solution, and it is possible that the accuracy you have specified is too stringent.

Further confirmation can be sought by trying to check whether or not convergence tests are almost satisfied, or whether or not some of the sufficient conditions are nearly satisfied. When it is thought that a function has returned a nonzero value of **ifail** only because the requirements for 'success' were too stringent it may be worth restarting with increased convergence tolerances.

For nonlinearly-constrained problems, check whether the solution returned is feasible, or nearly feasible; if not, the solution returned is not an adequate solution.

Confidence in a solution may be increased by re-solving the problem with a different initial approximation to the solution. See Section 8.3 of Gill *et al.* 1981 for further information.

2.6.3 Monitoring progress

Many of the functions in the chapter have facilities to allow you to monitor the progress of the minimization process, and you are encouraged to make use of these facilities. Monitoring information can be a great aid in assessing whether or not a satisfactory solution has been obtained, and in indicating difficulties in the minimization problem or in the ability of the function to cope with the problem.

The behaviour of the function, the estimated solution and first derivatives can help in deciding whether a solution is acceptable and what to do in the event of a return with a nonzero value of **ifail**.

2.6.4 Confidence intervals for least-squares solutions

When estimates of the parameters in a nonlinear least-squares problem have been found, it may be necessary to estimate the variances of the parameters and the fitted function. These can be calculated from the Hessian of $F(x)$ at the solution.

In many least-squares problems, the Hessian is adequately approximated at the solution by $G = 2J^{\mathrm{T}}J$ (see Section 2.4.3). The Jacobian, J, or a factorization of J is returned by all the comprehensive least-squares functions and, in addition, a function is available in the Library to estimate variances of the parameters following the use of most of the nonlinear least-squares functions, in the case that $G = 2J^{\mathrm{T}}J$ is an adequate approximation.

Let H be the inverse of G, and S be the sum of squares, both calculated at the solution $\bar{x}$; an unbiased estimate of the **variance** of the ith parameter x_i is

$$\operatorname{var}\bar{x}_i = \frac{2S}{m-n}H_{ii}$$

and an unbiased estimate of the covariance of $\bar{x}_i$ and $\bar{x}_j$ is

$$\operatorname{covar}(\bar{x}_i,\bar{x}_j) = \frac{2S}{m-n}H_{ij}.$$

If x^* is the true solution, then the $100(1-\beta)\%$ **confidence interval** on $\bar{x}$ is

$$\bar{x}_i - \sqrt{\operatorname{var}\bar{x}_i}.t_{(1-\beta/2,m-n)} < x_i^* < \bar{x}_i + \sqrt{\operatorname{var}\bar{x}_i}.t_{(1-\beta/2,m-n)}, \qquad i = 1,2,\ldots,n$$

where $t_{(1-\beta/2,m-n)}$ is the $100(1-\beta)/2$ percentage point of the t-distribution with $m-n$ degrees of freedom.

In the majority of problems, the residuals f_i, for $i = 1,2,\ldots,m$, contain the difference between the values of a model function $\phi(z,x)$ calculated for m different values of the independent variable z, and the corresponding observed values at these points. The minimization process determines the parameters, or constants x, of the fitted function $\phi(z,x)$. For any value, $\bar{z}$, of the independent variable z, an unbiased estimate of the **variance** of ϕ is

$$\operatorname{var}\phi = \frac{2S}{m-n}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[\frac{\partial\phi}{\partial x_i}\right]_{\bar{z}}\left[\frac{\partial\phi}{\partial x_j}\right]_{\bar{z}}H_{ij}.$$

The $100(1-\beta)\%$ **confidence interval** on F at the point $\bar{z}$ is

$$\phi(\bar{z},\bar{x}) - \sqrt{\operatorname{var}\phi}.t_{(\beta/2,m-n)} < \phi(\bar{z},x^*) < \phi(\bar{z},\bar{x}) + \sqrt{\operatorname{var}\phi}.t_{(\beta/2,m-n)}.$$

For further details on the analysis of least-squares solutions see Bard 1974 and Wolberg 1967.

3 Recommendations on Choice and Use of Available Functions

The choice of function depends on several factors: the type of problem (unconstrained, etc.); the level of derivative information available (function values only, etc.); your experience (there are easy-to-use versions of some functions); whether or not storage is a problem; whether or not the function is to be used in a multithreaded environment; and whether computational time has a high priority. Not all choices are catered for in the current version of the Library.

3.1 Easy-to-use and Comprehensive Functions

Many functions appear in the Library in two forms: a comprehensive form and an easy-to-use form. The objective in the easy-to-use forms is to make the function simple to use by including in the calling sequence only those parameters absolutely essential to the definition of the problem, as opposed to parameters relevant to the solution method. If you are an experienced user the comprehensive functions have additional parameters which you to improve their efficiency by 'tuning' the method to a particular problem. If you are a casual or inexperienced user, this feature is of little value and may in some cases cause a failure because of a poor choice of some parameters.

In the easy-to-use functions, these extra parameters are determined either by fixing them at a known safe and reasonably efficient value, or by an auxiliary function which generates a 'good' value automatically.

For functions introduced since Mark 12 of the Library a different approach has been adopted towards the choice of easy-to-use and comprehensive functions. The optimization function has an easy-to-use parameter list, but additional parameters may be changed from their default values by calling an 'option' setting function before the call to the main optimization function. This approach has the advantages of allowing the options to be given in the form of keywords and requiring only those options that are to be different from their default values to be set.

3.2 Reverse Communication Functions

Most of the functions in this chapter are called just once in order to compute the minimum of a given objective function subject to a set of constraints on the variables. The objective function and nonlinear constraints (if any) are specified by you and written as (sub)programs to a very rigid format described in the relevant function document. Such (sub)programs usually appear in the argument list of the minimization function.

For the majority of applications this is the simplest and most convenient usage. Sometimes however this approach can be restrictive:

(i) when the required format of the does not allow useful information to be passed conveniently to and from your calling program;

(ii) when the minimization function is being called from another computer language, such as Visual Basic, which does not fully support procedure arguments in a way that is compatible with the Library.

A way around these problems is to supply **reverse communication** functions. Instead of performing complete optimizations, these functions perform one step in the solution process before returning to the calling program with an appropriate flag (**irevcm**) set. The value of **irevcm** determines whether the minimization process has finished or whether fresh information is required. In the latter case the you calculate this information (in the form of a vector or as a scalar, as appropriate) and re-enter the reverse communication function with the information contained in appropriate arguments. Thus you have the responsibility for providing the iterative loop in the minimization process, but as compensation, has an extremely flexible and basic user-interface to the reverse communication function.

The only reverse communication functions in this chapter are e04uf, which solve dense NLP problems using a sequential quadratic programming method.

3.3 Service Functions

One of the most common errors in the use of optimization functions is that user-supplied (sub)programs do not evaluate the relevant partial derivatives correctly. Because exact gradient information normally enhances efficiency in all areas of optimization, you should be encouraged to provide analytical derivatives whenever possible. However, mistakes in the computation of derivatives can result in serious and obscure run-time errors. Consequently, **service functions** are provided to perform an elementary check on the user-supplied gradients. These functions are inexpensive to use in terms of the number of calls they require to user-supplied (sub)programs .

The appropriate checking functions are as follows:

Minimization function	**Checking function(s)**
e04kd	e04hc
e04lb	e04hc and e04hd
e04gb	e04ya
e04gd	e04ya
e04he	e04ya and e04yb

It should be noted that functions e04uf, e04us, e04vh and e04wd each incorporate a check on the gradients being supplied. This involves verifying the gradients at the first point that satisfies the linear constraints and bounds. There is also an option to perform a more reliable (but more expensive) check on the individual gradient elements being supplied. Note that the checks are not infallible.

A second type of service function computes a set of finite-differences to be used when approximating first derivatives. Such differences are required as input parameters by some functions that use only function evaluations.

e04yc estimates selected elements of the variance-covariance matrix for the computed regression parameters following the use of a nonlinear least-squares function.

e04xa estimates the gradient and Hessian of a function at a point, given a function to calculate function values only, or estimates the Hessian of a function at a point, given a function to calculate function and gradient values.

e04zc checks that user-supplied (sub)programs for evaluating an objective function, constraint functions and their first derivatives produce derivative values which are consistent with the function and constraint values calculated.

3.4 Function Evaluations at Infeasible Points

All the functions for constrained problems will ensure that any evaluations of the objective function occur at points which **approximately** satisfy any **simple bounds** or **linear constraints**. Satisfaction of such constraints is only approximate because functions which estimate derivatives by finite-differences may require function evaluations at points which just violate such constraints even though the current iteration just satisfies them.

There is no attempt to ensure that the current iteration satisfies any nonlinear constraints. If you wish to prevent your objective function being evaluated outside some known region (where it may be undefined or not practically computable), you may try to confine the iteration within this region by imposing suitable simple bounds or linear constraints (but beware as this may create new local minima where these constraints are active).

Note also that some functions allow the to return the parameter (**iflag** or **mode**) with a negative value to force an immediate clean exit from the minimization function when the objective function (or nonlinear constraints where appropriate) cannot be evaluated.

3.5 Related Problems

Apart from the standard types of optimization problem, there are other related problems which can be solved by functions in this or other chapters of the Library.

h02bb solves **dense integer LP** problems, h02cb solves **dense integer QP** problems, h02ce solves **sparse integer QP** problems and h03ab solves a special type of such problem known as a **'transportation' problem**.

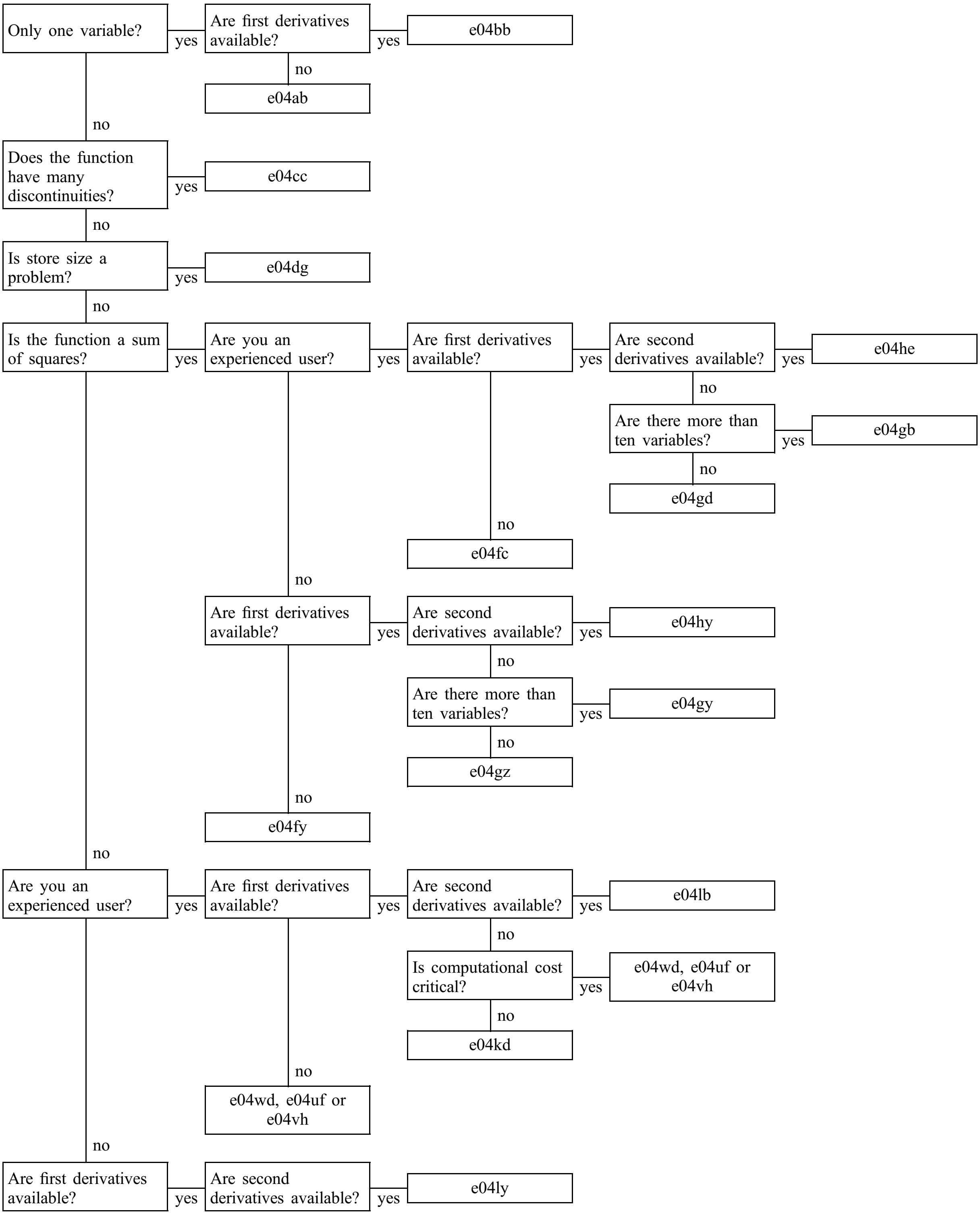
Several functions in Chapter F04 solve **linear least-squares problems**, i.e., minimize $\sum_{i=1}^{m} r_i(x)^2$ where $r_i(x) = b_i - \sum_{j=1}^{n} a_{ij}x_j$.

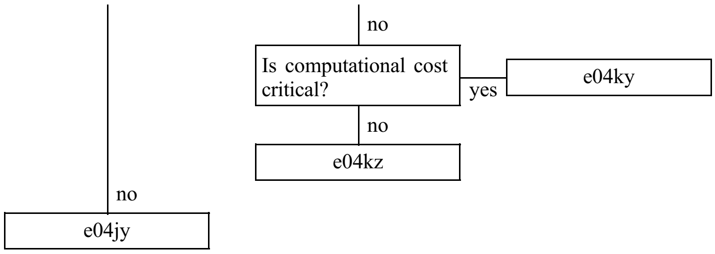
e02ga solves an overdetermined system of linear equations in the l_1 norm, i.e., minimizes $\sum_{i=1}^{m} |r_i(x)|$, with r_i as above, and e02gb solves the same problem subject to linear inequality constraints.

e02gc solves an overdetermined system of linear equations in the l_∞ norm, i.e., minimizes $\max_i |r_i(x)|$, with r_i as above.

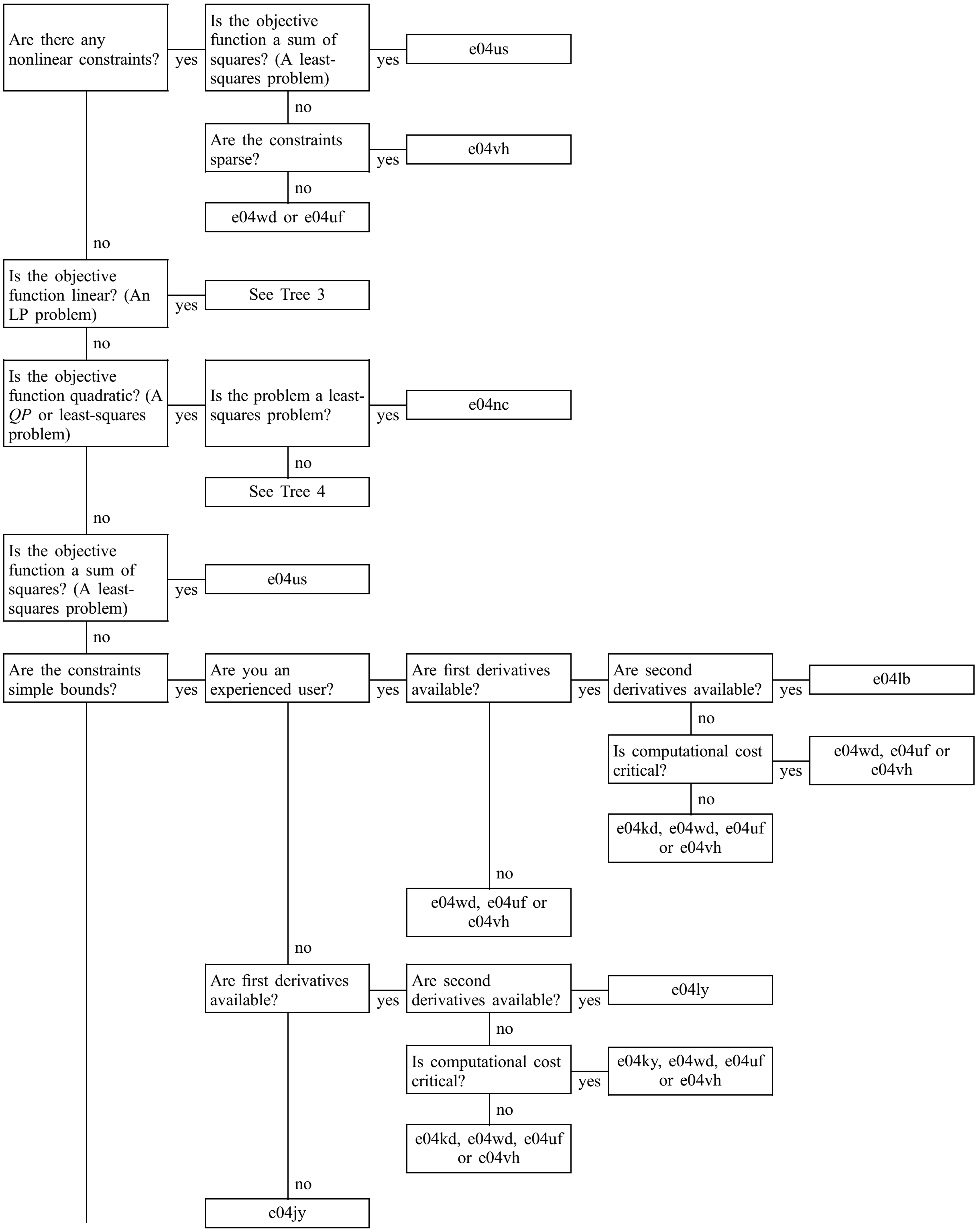
4 Decision Trees

Tree 1: Selection chart for unconstrained problems





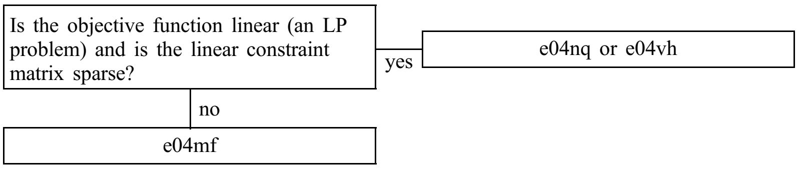
Tree 2: Selection chart for bound-constrained, linearly-constrained and nonlinearly-constrained problems



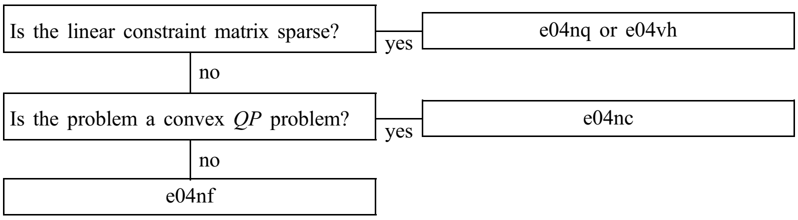
no

e04wd, e04uf or/
e04vh

Tree 3: Linear programming



Tree 4: Quadratic programming



5 Index

6 References

Bard Y 1974 *Nonlinear Parameter Estimation* Academic Press

Dantzig G B 1963 *Linear Programming and Extensions* Princeton University Press

Fletcher R 1987 *Practical Methods of Optimization* (2nd Edition) Wiley

Gill P E and Murray W (ed.) 1974 *Numerical Methods for Constrained Optimization* Academic Press

Gill P E, Murray W and Wright M H 1981 *Practical Optimization* Academic Press

Murray W (ed.) 1972 *Numerical Methods for Unconstrained Optimization* Academic Press

Wolberg J R 1967 *Prediction Analysis* Van Nostrand