NAG Library Chapter Introduction

E04 – Minimizing or Maximizing a Function

Contents

1 Scope of the Chapter

This chapter provides routines for solving various mathematical optimization problems by solvers based on local stopping criteria. The main classes of problems covered in this chapter are:

Linear Programming (LP) – dense and sparse;

Quadratic Programming (QP) – convex and nonconvex, dense and sparse;

Nonlinear Programming (NLP) – dense and sparse, based on active-set SQP methods or interior point methods (IPM);

Semidefinite Programming (SDP) – both linear matrix inequalities (LMI) and bilinear matrix inequalities (BMI);

Derivative-free Optimization (DFO);

Least Squares (LSQ), data fitting – linear and nonlinear, constrained and unconstrained.

For a full overview of the functionality offered in this chapter, see Section 5 or the Chapter Contents (Chapter E04).

See also other chapters in the Library relevant to optimization:

Chapter E05 contains routines to solve **global optimization** problems;

Chapter H addresses problems arising in **operational research** and focuses on **Mixed Integer Programming (MIP)**;

Chapters F07 and F08 include routines for linear algebra and in particular unconstrained linear least squares;

Chapter E02 focuses on curve and surface fitting, in which linear data fitting in l_1 or l_∞ norm might be of interest.

This introduction is only a brief guide to the subject of optimization. It discusses a classification of the optimization problems and presents an overview of the algorithms and their stopping criteria to help with the choice of a correct solver for a particular problem. Anyone with a difficult or protracted problem to solve will find it beneficial to consult a more detailed text, see Gill *et al.* (1981), Fletcher (1987) or Nocedal and Wright (2006). If you are unfamiliar with the mathematics of the subject you may find Sections 2.1, 2.2, 2.3, 2.6 and 3 a useful starting point.

2 Background to the Problems

2.1 Introduction to Mathematical Optimization

Mathematical Optimization, also known as *Mathematical Programming*, refers to the problem of finding values of the inputs from a given set so that a function (called the **objective function**) is minimized or maximized. The inputs are called **decision variables**, *primal variables* or just *variables*. The given set from which the decision variables are selected is referred to as a **feasible set** and might be defined as a domain where **constraints** expressed as functions of the decision variables hold certain values. Each point of the feasible set is called a **feasible point**.

A general mathematical formulation of such a problem might be written as

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & x \in \mathcal{F} \end{aligned}$$

where x denotes the decision variables, $f(x)$ the objective function and $\mathcal{F}$ the feasibility set. In this chapter we assume that $\mathcal{F} \subset R^n$. Since **maximization** of the objective function $f(x)$ is equivalent to minimizing $-f(x)$, only minimization is considered further in the text. Some routines 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.

A point x^* is said to be a **local minimum** of a function f if it is feasible ($x^* \in \mathcal{F}$) and if $f(x) \geq f(x^*)$ for all $x \in \mathcal{F}$ near x^*. A point x^* is a **global minimum** if it is a local minimum and $f(x) \geq f(x^*)$ for all feasible x. The solvers in this chapter are based on algorithms which seek only a local minimum,

however, many problems (such as *convex optimization* problems) have only one local minimum. This is also the global minimum. In such cases the Chapter E04 solvers find the global minimum. See Chapter E05 for solvers which try to find a global solution even for nonconvex functions.

2.2   Classification of Optimization Problems

There is no single efficient solver for all optimization problems. Therefore it is important to choose a solver which matches the problem and any specific needs as closely as possible. A more generic solver might be applied, however the performance suffers in some cases, depending on the underlying algorithm.

There are various criteria to help to classify optimization problems into particular categories. The main criteria are as follows:

>   Type of objective function;

>   Type of constraints;

>   Size of the problem;

>   Smoothness of the data and available derivative information.

Each of the criteria is discussed below to give the necessary information to identify the class of the optimization problem. Section 2.5 presents the basic properties of the algorithms and Section 3 advises on the choice of particular routines in the chapter.

2.2.1 Types of objective functions

In general, if there is a structure in the problem the solver should benefit from it. For example, a solver for problems with the *sum of squares* objective should work better than when this objective is treated as a general *nonlinear objective*. Therefore it is important to recognize typical types of the objective functions.

An optimization problem which has **no objective** is equivalent to having a constant objective, i.e., $f(x) = 0$. It is usually called a **feasible point problem**. The task is to then find any point which satisfies the constraints.

A **linear objective function** is a function which is linear in all variables and therefore can be represented as

$$f(x) = c^{\mathrm{T}}x + c_0$$

where $c \in R^n$. Scalar c_0 has no influence on the choice of decision variables x and is usually omitted. It will not be used further in this text.

A **quadratic objective function** is an extension of a linear function with quadratic terms as follows:

$$f(x) = \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x.$$

Here H is a real symmetric $n \times n$ matrix. In addition, if H is positive semidefinite (all its eigenvalues are non-negative), the objective is **convex**.

A general **nonlinear objective function** is any $f : R^n \to R$ without a special structure.

Special consideration is given to the objective function in the form of a **sum of squares** of functions, such as

$$f(x) = \sum_{i=1}^{m} r_i^2(x)$$

where $r_i : R^n \to R$; often called *residual functions*. This form of the objective plays a key role in **data fitting** solved as a **least squares problem** as shown in Section 2.2.3.

2.2.2 Types of constraints

Not all optimization problems have to have constraints. If there are no restrictions on the choice of x except that $x \in \mathcal{F} = R^n$, the problem is called **unconstrained** and thus every point is a feasible point.

Simple bounds on decision variables $x \in R^n$ (also known as *box constraints* or *bound constraints*) restrict the value of the variables, e.g., $x_5 \le 10$. They might be written in a general form as

$$l_{x_i} \le x_i \le u_{x_i}, \quad \text{for } i = 1, \ldots, n$$

or in the vector notation as

$$l_x \le x \le u_x$$

where l_x and u_x are n-dimensional vectors. Note that lower and upper bounds are specified for all the variables. By conceptually allowing $l_{x_i} = -\infty$ and $u_{x_i} = +\infty$ or $l_{x_i} = u_{x_i}$ full generality in various types of constraints is allowed, such as unconstrained variables, one-sided inequalities, ranges or equalities (fixing the variable).

The same format of bounds is adopted to linear and nonlinear constraints in the whole chapter. Note that for the purpose of passing infinite bounds to the routines, all values above a certain threshold (typically 10^{20}) are treated as $+\infty$.

Linear constraints are defined as constraint functions that are linear in all of their variables, e.g., $3x_1 + 2x_2 \ge 4$. They can be stated in a matrix form as

$$l_B \le Bx \le u_B$$

where B is a general $m_B \times n$ rectangular matrix and l_B and u_B are m_B-dimensional vectors. Each row of B represents linear coefficients of one linear constraint. The same rules for bounds apply as in the simple bounds case.

Although the bounds on x_i could be included in the definition of linear constraints, we recommend you distinguish between them for reasons of computational efficiency as most of the solvers treat simple bounds explicitly.

A set of m_g **nonlinear constraints** may be defined in terms of a nonlinear function $g : R^n \to R^{m_g}$ and the bounds l_g and u_g which follow the same format as simple bounds and linear constraints:

$$l_g \le g(x) \le u_g.$$

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

A **matrix constraint** (or *matrix inequality*) is a constraint on eigenvalues of a matrix operator. More precisely, let S^m denote the space of real symmetric matrices m by m and let $\mathcal{A}$ be a matrix operator $\mathcal{A} : R^n \to S^m$, i.e., it assigns a symmetric matrix $\mathcal{A}(x)$ for each x. The matrix constraint can be expressed as

$$\mathcal{A}(x) \succeq 0$$

where the inequality $S \succeq 0$ for $S \in S^m$ is meant in the eigenvalue sense, namely all eigenvalues of the matrix S should be non-negative (the matrix should be positive semidefinite).

There are two types of matrix constraints allowed in the current mark of the Library. The first is **linear matrix inequality (LMI)** formulated as

$$\mathcal{A}(x) = \sum_{i=1}^{n} x_i A_i - A_0 \succeq 0$$

and the second one, **bilinear matrix inequality (BMI)**, stated as

$$\mathcal{A}(x) = \sum_{i,j=1}^{n} x_i x_j Q_{ij} + \sum_{i=1}^{n} x_i A_i - A_0 \succeq 0.$$

Here all matrices A_i, Q_{ij} are given real symmetric matrices of the same dimension. Note that the latter type is in fact quadratic in x, nevertheless, it is referred to as bilinear for historical reasons.

2.2.3 Typical classes of optimization problems

Specific combinations of the types of the objective functions and constraints give rise to various classes of optimization problems. The common ones are presented below. It is always advisable to consider the closest formulation which covers your problem when choosing the solver. For more information see classical texts such as Dantzig (1963), Gill *et al.* (1981), Fletcher (1987), Nocedal and Wright (2006) or Chvátal (1983).

A **Linear Programming (LP)** problem is a problem with a linear objective function, linear constraints and simple bounds. It can be written as follows:

$$
\begin{aligned}
&\underset{x \in R^n}{\text{minimize}} && c^{\mathrm{T}}x \\
&\text{subject to} && l_B \le Bx \le u_B \\
& && l_x \le x \le u_x
\end{aligned}
$$

Quadratic Programming (QP) problems optimize a quadratic objective function over a set given by linear constraints and simple bounds. Depending on the convexity of the objective function, we can distinguish between **convex** and **nonconvex** (or *general*) QP.

$$
\begin{aligned}
&\underset{x \in R^n}{\text{minimize}} && \tfrac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x \\
&\text{subject to} && l_B \le Bx \le u_B \\
& && l_x \le x \le u_x
\end{aligned}
$$

Nonlinear Programming (NLP) problems allow a general nonlinear objective function $f(x)$ and any of the nonlinear, linear or bound constraints. Special cases when some (or all) of the constraints are missing are termed as *unconstrained*, *bound-constrained* or *linearly-constrained nonlinear programming* and might have a specific solver as some algorithms take special provision for each of the constraint type. Problems with a linear or quadratic objective and nonlinear constraints should be still solved as general NLPs.

$$
\begin{aligned}
&\underset{x \in R^n}{\text{minimize}} && f(x) \\
&\text{subject to} && l_g \le g(x) \le u_g \\
& && l_B \le Bx \le u_B \\
& && l_x \le x \le u_x
\end{aligned}
$$

Semidefinite Programming (SDP) typically refers to *linear* semidefinite programming thus a problem with a linear objective function, linear constraints and linear matrix inequalities:

$$
\begin{aligned}
&\underset{x \in R^n}{\text{minimize}} && c^{\mathrm{T}}x \\
&\text{subject to} && \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1, \ldots, m_A \\
& && l_B \le Bx \le u_B \\
& && l_x \le x \le u_x
\end{aligned}
$$

This problem can be extended with a quadratic objective and bilinear (in fact quadratic) matrix inequalities. We refer to it as a semidefinite programming problem with **bilinear matrix inequalities (BMI-SDP)**:

$$
\begin{aligned}
&\underset{x \in R^n}{\text{minimize}} && \tfrac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x \\
&\text{subject to} && \sum_{i,j=1}^{n} x_i x_j Q_{ij}^k + \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1, \ldots, m_A \\
& && l_B \le Bx \le u_B \\
& && l_x \le x \le u_x
\end{aligned}
$$

A **least squares (LSQ)** problem is a problem where the objective function in the form of sum of squares is minimized subject to usual constraints. If the residual functions $r_i(x)$ are linear or nonlinear, the problem is known as **linear** or **nonlinear least squares**, respectively. Not all types of the constraints need to be present which brings up special cases of *unconstrained*, *bound-constrained* or *linearly-constrained least squares problems* as in NLP .

$$\begin{array}{ll} \underset{x \in R^n}{\text{minimize}} & \sum_{i=1}^{m} r_i^2(x) \\ \text{subject to} & l_g \leq g(x) \leq u_g \\ & l_B \leq Bx \leq u_B \\ & l_x \leq x \leq u_x \end{array}$$

This form of the problem is very common in **data fitting** as demonstrated on the following example. Let us consider a process that is observed at times t_i and measured with results y_i, for $i = 1, 2, \ldots, m$. Furthermore, the process is assumed to behave according to a model $\phi(t; x)$ where x are parameters of the model. Given the fact that the measurements might be inaccurate and the process might not exactly follow the model, it is beneficial to find model parameters x so that the error of the fit of the model to the measurements is minimized. This can be formulated as an optimization problem in which x are decision variables and the objective function is the sum of squared errors of the fit at each individual measurement, thus:

$$\underset{x \in R^n}{\text{minimize}} \quad \sum_{i=1}^{m} r_i^2(x) \quad \text{where} \quad r_i(x) = \phi(t_i; x) - y_i$$

2.2.4 Problem size, dense and sparse problems

The size of the optimization problem plays an important role in the choice of the solver. The size is usually understood to be the number of variables n and the number (and the type) of the constraints. Depending on the size of the problem we talk about *small-scale*, *medium-scale* or *large-scale* problems.

It is often more practical to look at the data and its structure rather than just the size of the problem. Typically in a large-scale problem not all variables interact with everything else. It is natural that only a small portion of the constraints (if any) involves all variables and the majority of the constraints depends only on small different subsets of the variables. This creates many explicit zeros in the data representation which it is beneficial to capture and pass to the solver. In such a case the problem is referred to as **sparse**. The data representation usually has the form of a sparse matrix which defines the linear constraint matrix B, Jacobian matrix of the nonlinear constraints g_i or the Hessian of the objective H. Common sparse matrix formats are used, such as *coordinate storage (CS)* and *compressed column storage (CCS)* (see Section 2.1 in the F11 Chapter Introduction).

The counterpart to a sparse problem is a **dense** problem in which the matrices are stored in general full format and no structure is assumed or exploited. Whereas passing a dense problem to a sparse solver presents typically only a small overhead, *calling a dense solver on a large-scale sparse problem is ill-advised; it leads to a significant performance degradation and memory overuse.*

2.2.5 Derivative information, smoothness, noise and derivative-free optimization (DFO)

Most of the classical optimization algorithms rely heavily on derivative information. It plays a key role in necessary and sufficient conditions (see Section 2.4) and in the computation of the search direction at each iteration (see Section 2.5). Therefore it is important that accurate derivatives of the nonlinear objective and nonlinear constraints are provided whenever possible.

Unless stated otherwise, it is assumed that the nonlinear functions are sufficiently *smooth*. The solvers will usually solve optimization problems even if there are isolated discontinuities away from the solution, however you should always consider whether an alternative smooth representation of the problem exists. A typical example is an absolute value $|x_i|$ which does not have a first derivative for $x_i = 0$. Nevertheless, if the model allows it can be transformed as

$$x_i = x_i^+ - x_i^-, \quad |x_i| = x_i^+ + x_i^-, \quad \text{where} \quad x_i^+, \, x_i^- \geq 0$$

which avoids the discontinuity of the first derivative. If many discontinuities are present, alternative methods need to be applied such as **e04cbf** or stochastic algorithms in Chapter E05, **e05saf** or **e05sbf**.

The vector of first partial derivatives of a function is called the **gradient vector**, i.e.,

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

the matrix of second partial derivatives is termed the **Hessian matrix**, i.e.,

$$\nabla^2 f(x) = \left[\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right]_{i,j=1,\ldots,n}$$

and the matrix of first partial derivatives of the vector-valued function $f : R^n \to R^m$ is known as the **Jacobian matrix**:

$$J(x) = \left[\frac{\partial f_i(x)}{\partial x_j} \right]_{i=1,\ldots,m,j=1,\ldots,n} .$$

If the function is smooth and the derivative is unavailable, it is possible to approximate it by **finite differences**, a change in function values in response to small perturbations of the variables. Many routines in the Library estimate missing elements of the gradients automatically this way. The choice of the size of the perturbations strongly affects the quality of the approximation. Too small perturbations might spoil the approximation due to the cancellation errors in floating-point arithmetic and too big reduce the match of the finite differences and the derivative (see **e04xaf/e04xaa** for optimal balance of the factors). In addition, finite differences are very sensitive to the accuracy of $f(x)$. They might be unreliable or fail completely if the function evaluation is inaccurate or *noisy* such as when $f(x)$ is a result of a stochastic simulation or an approximate solution of a PDE.

Derivative-free optimization (DFO) represents an alternative to derivative-based optimization algorithms. DFO solvers neither rely on derivative information nor approximate it by finite differences. They sample function evaluations across the domain to determine a new iteration point (for example, by a quadratic model through the sampled points). They are therefore less exposed to the relative error of the noise of the function because the sample points are never too close to each other to take the error into account. DFO might be useful even if the finite differences can be computed as the number of function evaluations is lower. This is particularly beneficial for problems where the evaluations of f are expensive. DFO solvers tend to exhibit a faster initial progress to the solution, however, they typically cannot achieve high-accurate solutions.

2.2.6 Minimization subject to bounds on the objective function

In all of the above problem categories it is assumed that

$$a \le f(x) \le b$$

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 3.7.

2.2.7 Multi-objective optimization

Sometimes a problem may have two or more objective functions which are to be optimized at the same time. Such problems are called **multi-objective**, *multi-criteria* or *multi-attribute* optimization. If the constraints are linear and the objectives are all linear then the terminology **goal programming** is also used.

Although there is no routine dealing with this type of problems explicitly in this mark of the Library, techniques used in this chapter and in Chapter E05 may be employed to address such problems, see Section 2.5.5.

2.3 Geometric Representation

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

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

(This function is used as the example function in the documentation for the unconstrained routines.)
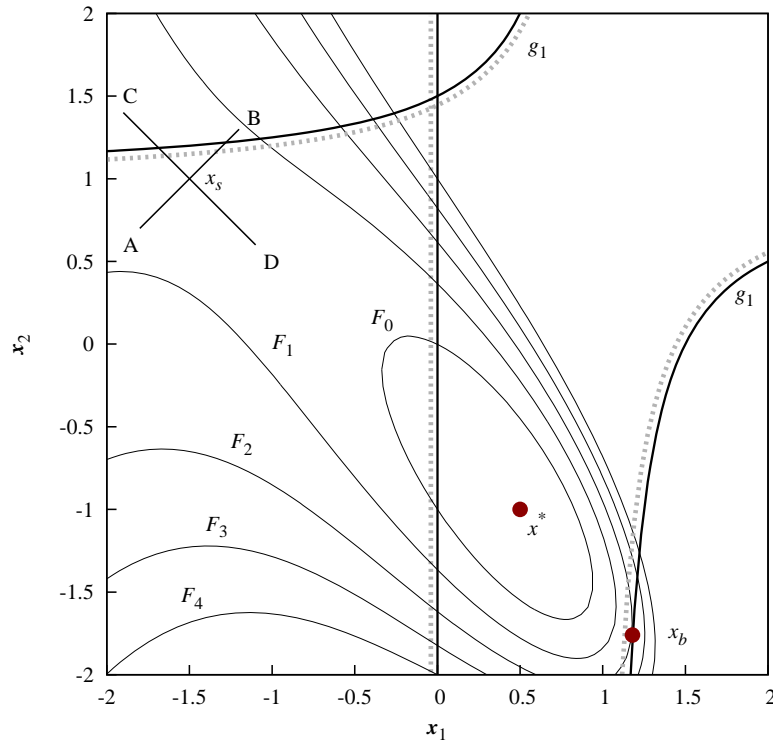
Figure 1

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 point x_s 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., $x_1 < 0$).

If we add the nonlinear constraint $g_1(x) : x_1 + x_2 - x_1 x_2 - \frac{3}{2} \geq 0$, represented by the curved shaded line in Figure 1, then x^* is not a feasible point because $g_1(x^*) < 0$. The solution of the new constrained problem is $x_b \simeq (1.1825, -1.7397)^{\mathrm{T}}$, the feasible point with the smallest function value (where $f(x_b) \simeq 3.0607$).

2.4 Sufficient Conditions for a Solution

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

2.4.1 Unconstrained minimization

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

(i) $\|\nabla f(x^*)\| = 0$ and

(ii) $\nabla^2 f(x^*)$ is positive definite,

where $\|\cdot\|$ denotes the Euclidean norm.

2.4.2 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) $\dfrac{\partial}{\partial x_j} f(x^*) < 0, x_j = u_j\,; \dfrac{\partial}{\partial x_j} f(x^*) > 0, x_j = l_j\,,$

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.

2.4.3 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

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

$\qquad G_Z(x) = Z^{\mathrm{T}} \nabla^2 f(x) Z$, 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 $\nabla f(x^*)$ can be written as a linear combination of the columns of $\hat{A}$, i.e.,

$\nabla f(x^*) = \sum\limits_{i=1}^{t} \lambda_i^* \hat{a}_i = \hat{A} \lambda^*$. 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:

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

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, $\nabla f(x^*) = \hat{A}\lambda^*$; and

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

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

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

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

2.4.4 Nonlinearly-constrained minimization

For nonlinearly-constrained problems, much of the terminology is defined exactly as in the linearly-constrained case. To simplify the notation, let us assume that all nonlinear constraints are in the form $c(x) \geq 0$. 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:

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

where the dependence on x has been suppressed for compactness.

The projected gradient vector $g_Z(x)$ is the vector $Z^T \nabla f(x)$. 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., $\nabla f(x^*) = \hat{A}(x^*)\lambda^*$.

The **Lagrangian function** is given by:

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

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., $\hat{G}_L = Z^T G_L Z$.

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, $\nabla f(x^*) = \hat{A}(x^*)\lambda^*$; 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 $\hat{A}(x^*)$.

2.5   Background to Optimization Methods

All the algorithms contained in this chapter generate an iterative sequence $\{x^{(k)}\}$ 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 2.7.

Most of the methods construct a sequence $\{x^{(k)}\}$ 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 $f(x^{(k+1)}) < f(x^{(k)})$ and is computed using one of the techniques for one-dimensional optimization referred to in Section 2.5.1.

2.5.1 One-dimensional optimization

The Library contains two special routines for minimizing a function of a single variable. Both routines are based on safeguarded polynomial approximation. One routine 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).

2.5.2 Methods for unconstrained optimization

The distinctions between 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 $\nabla^2 f(x^{(k)})$, or its finite difference approximation , to define the search direction. The routines in the Library either require a subroutine that computes the elements of the Hessian directly, or they approximate them 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 $\nabla^2 f(x^{(k)})$ 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 the Hessian is not computed directly, or approximated by finite differences. Quasi-Newton methods minimize a quadratic function in n iterations, where n is the number of variables. 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).

2.5.3 Methods for nonlinear least squares problems

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

Since

$$f(x) = \sum_{i=1}^{m} r_i^2(x)$$

the Hessian matrix is of the form

$$\nabla^2 f(x) = 2\left(J(x)^{\mathrm{T}} J(x) + \sum_{i=1}^{m} r_i(x) \nabla^2 r_i(x) \right),$$

where $J(x)$ is the Jacobian matrix of $r(x)$.

In the neighbourhood of the solution, $\|r(x)\|$ is often small compared to $\|J(x)^{\mathrm{T}} J(x)\|$ (for example, when $r(x)$ 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 $\nabla^2 f(x)$, thereby avoiding the need to compute or approximate second derivatives of $\{r_i(x)\}$. See Section 4.7 of Gill *et al.* (1981).

2.5.4 Methods for handling constraints

There are two main approaches for handling constraints in optimization algorithms – the **active-set sequential quadratic programming** method (or just SQP) and the **interior point method** (IPM). It is important to understand their very distinct features as both algorithms complement each other. The easiest method of comparison is to look at how the inequality constraints are treated and how the solver approaches the optimal solution (the progress of the KKT optimality measures: *optimality, feasibility, complementarity*).

Inequality constraints are the hard part of the optimization because of their 'twofold nature'. If the optimal solution strictly satisfies the inequality, i.e., the optimal point is in the interior of the constraint, the inequality constraint does not influence the result and could be removed from the model. On the other hand, if the inequality is satisfied as an equality (is *active* at the solution), the constraint must be present and could be treated as an equality from the very beginning. This is expressed by the complementarity in KKT conditions.

Solvers, based on the active-set method, solve at each iteration a quadratic approximation of the original problem; they try to estimate which constraints need to be kept (are active) and which can be ignored. A practical consequence is that the algorithm partly 'walks along the boundary' of the feasible region given by the constraints. The iterates are thus feasible early on with regard to all linear constraints (and a local linearization of the nonlinear constraints) which is preserved through the iterations. The complementarity is satisfied by default, and once the active set is determined correctly and optimality is within the tolerance, the solver finishes. The number of iterations might be high but each is relatively cheap. See Chapter 6 of Gill *et al.* (1981) for further details.

In contrast, an interior point method generates iterations that avoid the boundary defined by the inequality constraints. As the solver progresses the iterates are allowed to get closer and closer to the boundary and converge to the optimal solution which might lie on the boundary. From the practical point of view, IPM typically requires only tens of iterations. Each iteration consists of solving a large linear system of equations taking into account all variables and constraints, so each iteration is fairly expensive. All three optimality measures are reduced simultaneously.

In many cases it is difficult to predict which of the algorithms will behave better on a particular problem, however, some initial guidance can be given in the following table:

IPM advantages	**SQP advantages**
Can exploit second derivatives and its structure	Stay feasible with regard to linear constraints through most of the iterations
Efficient on unconstrained or loosely constrained problems	Very efficient for highly constrained problems
Efficient also for (both convex and nonconvex) quadratic problems (QP)	Better results on pathological problems in our experience
Better use of multi-core architecture (SMP library only)	Generally requires less function evaluations (efficient for problems with expensive function evaluations)
New interface, easier to use	Requires first derivatives but can work only with function values
	Can capitalize on a good initial point
	Allows warm starts (good for a sequence of similar problems)
	Infeasibility detection

Unless some of the specific features are required which are offered only by one algorithm, the initial decision should be based on the availability of the derivatives of the problem and the number of constraints (for example, expressed as a ratio between the numbers of variables and the sum of the number of linear and nonlinear constraints). Readiness of exact second derivatives is a clear advantage for IPM so unless the number of constraints is close to the number of variables, IPM will probably work better. Similarly, if a large-scale problem has relatively few constraints (e.g., less than 40%) IPM might be more successful, especially as the problem gets bigger. On the other hand, if no derivatives are available, either the SQP or a specialized algorithm from the Library (see Derivative Free Optimization, Section 2.2.5) needs to be used. With more and more constraints SQP might be faster. For problems

which do not fall in either of the categories, it is not easy to anticipate which solver will work better and some experimentation might be required.

2.5.5 Methods for handling multi-objective optimization

Suppose we have objective functions $f_i(x)$, $i > 1$, all of which we need to minimize at the same time. There are two main approaches to this problem:

(a) Combine the individual objectives into one composite objective. Typically this might be a weighted sum of the objectives, e.g.,

$$w_1 f_1(x) + w_2 f_2(x) + \cdots + w_n f_n(x)$$

Here you choose the weights to express the relative importance of the corresponding objective. Ideally each of the $f_i(x)$ should be of comparable size at a solution.

(b) Order the objectives in order of importance. Suppose f_i are ordered such that $f_i(x)$ is more important than $f_{i+1}(x)$, for $i = 1, 2, \ldots, n - 1$. Then in the lexicographical approach to multi-objective optimization a sequence of subproblems are solved. Firstly solve the problem for objective function $f_1(x)$ and denote by r_1 the value of this minimum. If $(i - 1)$ subproblems have been solved with results r_{i-1} then subproblem i becomes $\min(f_i(x))$ subject to $r_k \le f_k(x) \le r_k$, for $k = 1, 2, \ldots, i - 1$ plus the other constraints.

Clearly the bounds on f_k might be relaxed at your discretion.

In general, if NAG routines from the Chapter E04 are used then only local minima are found. This means that a better solution to an individual objective might be found without worsening the optimal solutions to the other objectives. Ideally you seek a Pareto solution; one in which an improvement in one objective can only be achieved by a worsening of another objective.

To obtain a Pareto solution routines from Chapter E05 might be used or, alternatively, a pragmatic attempt to derive a global minimum might be tried (see **e05ucf**). In this approach a variety of different minima are computed for each subproblem by starting from a range of different starting points. The best solution achieved is taken to be the global minimum. The more starting points chosen the greater confidence you might have in the computed global minimum.

2.6   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 routines in the Library, scaling can be performed automatically in routines which solve sparse LP, QP or NLP problems and in some dense solver routines. Such routines have an optional parameter 'Scale Option' which can be set by the user; see individual routine documents for details.

The following sections present some general comments on problem scaling.

2.6.1 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

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

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

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

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 routines 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.

2.6.2 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 routine. 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.

2.6.3 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 an active-set solver, 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, $\hat{g}_1(x^*) = 10^{-8}$, $\hat{g}_2(x^*) = -10^{-6}$, 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).

2.7 Analysis of Computed Results

2.7.1 Convergence criteria

The convergence criteria inevitably vary from routine to routine, 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 $\{x^{(k)}\}$ converging?

(ii) is the sequence $\{f^{(k)}\}$ 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 routines 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.

2.7.2 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 routine reports success. Frequently a 'solution' may have been found even when the routine does not report a success. The reason for this apparent contradiction is that the routine 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 routine 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 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 restarting the solver with a different initial approximation to the solution. See Section 8.3 of Gill *et al.* (1981) for further information.

2.7.3 Monitoring progress

Many of the routines 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 routine 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**.

2.7.4 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 the objective $f(x)$ at the solution.

In many least squares problems, the Hessian is adequately approximated at the solution by $G = 2J^{\mathrm{T}}J$ (see Section 2.5.3). The Jacobian, J, or a factorization of J is returned by all the comprehensive least squares routines and, in addition, a routine is available in the Library to estimate variances of the parameters following the use of most of the nonlinear least squares routines, 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

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

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 r_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 Routines

The choice of routine 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 routines); whether or not a problem is sparse; whether or not the routine 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 Routines

Many routines appear in the Library in two forms: a comprehensive form and an easy-to-use form. The purpose of the easy-to-use forms is to make the routine simple to use by including in the calling sequence only those arguments absolutely essential to the definition of the problem, as opposed to arguments relevant to the solution method. If you are an experienced user the comprehensive routines have additional arguments which enable 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 arguments.

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

For routines introduced since Mark 12 of the Library a different approach has been adopted between the choice of easy-to-use and comprehensive routines. The optimization routine has an easy-to-use argument list, but additional arguments may be changed from their default values by calling an 'option' setting routine before the call to the main optimization routine. 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   Thread Safe Routines

Many of the routines in this chapter come in pairs, with each routine in the pair having exactly the same functionality, except that one of them has additional arguments in order to make it safe for use in multithreaded applications. The routine that is safe for use in multithreaded applications has an 'a' as the last character in the name, in place of the usual 'f'.

An example of such a pair is **e04aba** and **e04abf**.

3.3   Reverse Communication Routines

Most of the routines 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 subroutines to a very rigid format described in the relevant routine document.

This chapter also contains a pair of **reverse communication** routines, **e04uff/e04ufa**, which solve dense NLP problems using a sequential quadratic programming method. These may be convenient to use when the minimization routine is being called from a computer language which does not fully support procedure arguments in a way that is compatible with the Library. These routines are also useful if a large amount of data needs to be transmitted into the routine. See Section 3.3.3 in How to Use the NAG Library and its Documentation for more information about reverse communication routines.

3.4   Choosing Between Variant Routines for Some Problems

As evidenced by the wide variety of routines available in Chapter E04, it is clear that no single algorithm can solve all optimization problems. It is important to try to match the problem to the most suitable routine, and that is what the decision trees in Section 4 help to do.

Sometimes in Chapter E04 more than one routine is available to solve precisely the same optimization problem. If their differences lay in the underlying method, refer to the sections above. Section 2.5.4 discusses key features of interior point methods (represented by **e04stf**) and active-set SQP methods (for example, **e04ugf/e04uga** or **e04vhf**). Alternatively, there are routines implementing slightly different variants of the same method (such as **e04ucf/e04uca** and **e04wdf**). Experience shows that in this case although both routines can usually solve the same problem and get similar results, sometimes one routine will be faster, sometimes one might find a different local minimum to the other, or, in difficult cases, one routine may obtain a solution when the other one fails.

After using one of these routines, if the results obtained are unacceptable for some reason, it may be worthwhile trying the other routine instead. In the absence of any other information, in the first instance you are recommended to try using **e04ucf/e04uca**, and if that proves unsatisfactory, try using **e04wdf**. Although the algorithms used are very similar, the two routines each have slightly different optional parameters which may allow the course of the computation to be altered in different ways.

Other pairs of routines which solve the same kind of problem are **e04nqf** (recommended first choice) or **e04nkf/e04nka**, for sparse quadratic or linear programming problems, and **e04vhf** (recommended) or **e04ugf/e04uga**, for sparse nonlinear programming. In these cases the argument lists are not so similar as **e04ucf/e04uca** or **e04wdf**, but the same considerations apply.

3.5   NAG Optimization Modelling Suite

Mark 26 of the Library introduced the *NAG optimization modelling suite*, a suite of routines which allows you to define and solve various optimization problems in a uniform manner. The first key feature of the suite is that the definition of the optimization problem and the call to the solver have been separated so it is possible to set up a problem in the same way for different solvers. The second feature is that the problem representation is built up from basic components (for example, a QP problem is composed of a quadratic objective, simple bounds and linear constraints), therefore different types of problems reuse the same routines for their common parts.

A connecting element to all routines in the suite is a *handle*, a pointer to an internal data structure, which is passed among the routines. It holds all information about the problem, the solution and the solver. Each handle should go through four stages in its life: *initialization*, *problem formulation*, *problem solution* and *deallocation*.

The initialization is performed by **e04raf** which creates an empty problem with n decision variables. A call to **e04rzf** marks the end of the life of the handle as it deallocates all the allocated memory and data within the handle and destroys the handle itself. After the initialization, the objective may be defined as one of the following:

> **e04ref** – a linear objective as a dense vector;
>
> **e04rff** – a quadratic objective or a sparse linear objective;
>
> **e04rgf** – a nonlinear objective function;
>
> **e04rmf** – a nonlinear least squares objective function.

The routines for constraint definition are

> **e04rhf** – simple bounds;
>
> **e04rjf** – linear constraints;
>
> **e04rkf** – nonlinear constraints;
>
> **e04rlf** – second derivatives for the objective and/or constraints;
>
> **e04rnf** – linear matrix inequalities;
>
> **e04rpf** – quadratic terms for bilinear matrix inequalities.

These routines may be called in an arbitrary order, however, a call to **e04rnf** must precede a call to **e04rpf** for the matrix inequalities with bilinear terms and the nonlinear objective or constraints (**e04rgf** or **e04rkf**) must precede the definition of the second derivatives by **e04rlf**. For further details please refer to the documentation of the individual routines.

The suite also includes the following service routines:

> **e04ryf** – query/printing routine;
>
> **e04zmf** – supply an optional parameter from a character string;
>
> **e04zpf** – supply one or more optional parameters from a file;
>
> **e04znf** – get the settings of an optional parameter;
>
> **e04rxf** – read or write information into the handle.

When the problem is fully formulated, the handle can be passed to a solver which is compatible with the defined problem. At the current mark of the Library the NAG optimization modelling suite comprises of **e04fff**, **e04mtf**, **e04stf** and **e04svf**. The solver indicates by an error flag if it cannot deal with the given formulation. A diagram of the life cycle of the handle is depicted in Figure 2.
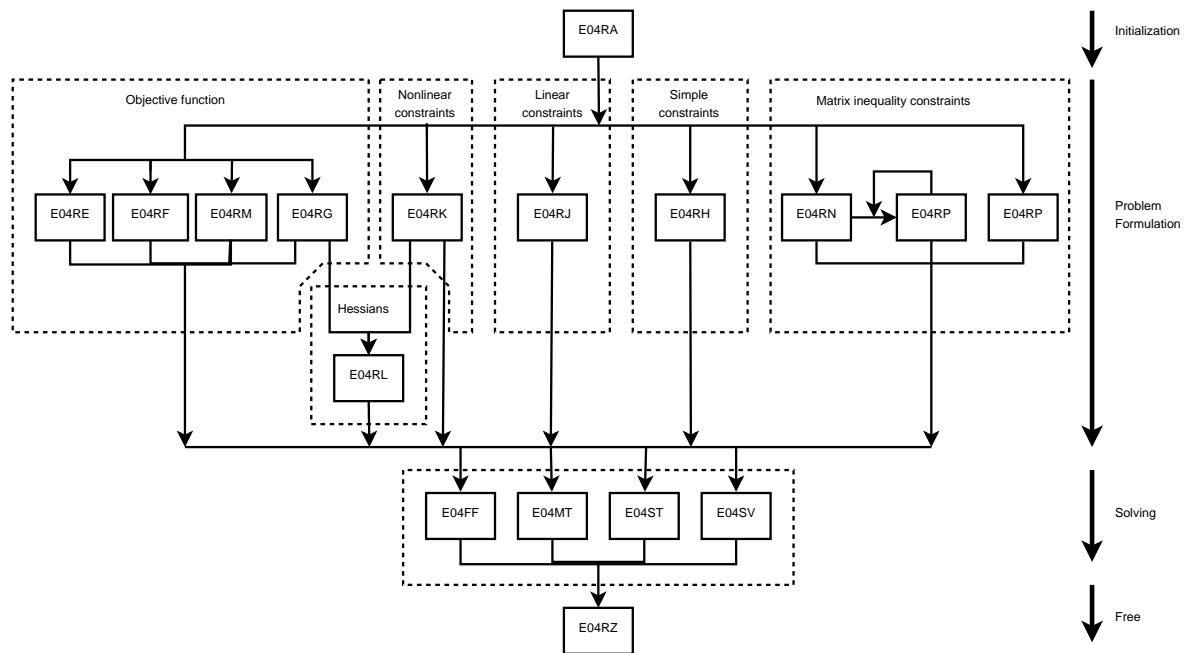
Figure 2

3.6 Service Routines

One of the most common errors in the use of optimization routines is that user-supplied subroutines do not evaluate the relevant partial derivatives correctly. Because exact gradient information normally enhances efficiency in all areas of optimization, you are 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 routines** are provided to perform an elementary check on the gradients you supplied. These routines are inexpensive to use in terms of the number of calls they require to user-supplied subroutines.

The appropriate checking routines are as follows:

Minimization routine	Checking routine(s)
e04kdf	e04hcf
e04lbf	e04hcf and e04hdf
e04gbf	e04yaf
e04gdf	e04yaf
e04hef	e04yaf and e04ybf

It should be noted that routines **e04stf**, **e04ucf/e04uca**, **e04uff/e04ufa**, **e04ugf/e04uga**, **e04usf/e04usa**, **e04vhf** and **e04wdf** each incorporate a check on the derivatives 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 routine computes a set of finite differences to be used when approximating first derivatives. Such differences are required as input arguments by some routines that use only function evaluations.

e04ycf estimates selected elements of the variance-covariance matrix for the computed regression parameters following the use of a nonlinear least squares routine.

e04xaf/e04xaa estimates the gradient and Hessian of a function at a point, given a routine to calculate function values only, or estimates the Hessian of a function at a point, given a routine to calculate function and gradient values.

3.7　Function Evaluations at Infeasible Points

All the solvers for constrained problems based on an active-set method will ensure that any evaluations of the objective function occur at points which **approximately** (up to the given tolerance) satisfy any **simple bounds** or **linear constraints**.

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 routines allow you to return the argument (**iflag**, **inform**, **mode** or **status**) with a negative value to indicate when the objective function (or nonlinear constraints where appropriate) cannot be evaluated. In case the routine cannot recover (e.g., cannot find a different trial point), it forces an immediate clean exit from the routine.

3.8　Related Problems

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

h02bbf solves **dense integer LP** problems, **h02cbf** solves **dense integer QP** problems, **h02cef** solves **sparse integer QP** problems, **h02daf** solves **dense mixed integer NLP** problems and **h03abf** solves a special type of such problem known as a **'transportation' problem**.

Several routines in Chapters F04 and F08 solve **linear least squares problems**, i.e., minimize $\sum\limits_{i=1}^{m} r_i(x)^2$

where $r_i(x) = b_i - \sum\limits_{j=1}^{n} a_{ij}x_j$.

e02gaf solves an overdetermined system of linear equations in the l_1 norm, i.e., minimizes $\sum\limits_{i=1}^{m} |r_i(x)|$, with r_i as above, and **e02gbf** solves the same problem subject to linear inequality constraints.

e02gcf solves an overdetermined system of linear equations in the l_∞ norm, i.e., minimizes $\max\limits_{i} |r_i(x)|$, with r_i as above.

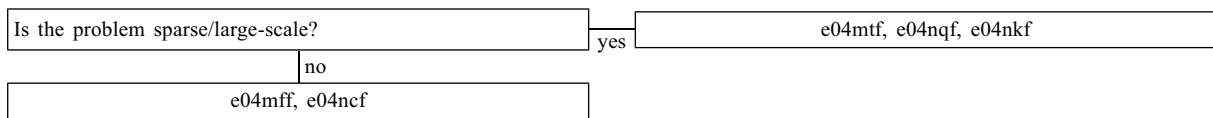Chapter E05 contains routines for global minimization.

Section 2.5.5 describes how a multi-objective optimization problem might be addressed using routines from this chapter and from Chapter E05.
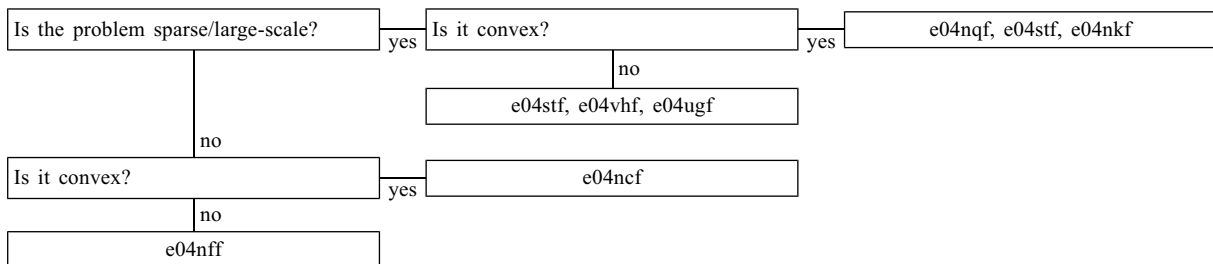
4   Decision Trees

	no objective	linear	quadratic	nonlinear	sum of squares
unconstrained			QP See Tree 2	NLP See Tree 3	LSQ See Tree 4
simple bounds	LP See Tree 1	LP See Tree 1	QP See Tree 2	NLP See Tree 3	LSQ See Tree 4
linear	LP See Tree 1	LP See Tree 1	QP See Tree 2	NLP See Tree 3	LSQ See Tree 4
nonlinear	NLP See Tree 3	NLP See Tree 3	NLP See Tree 3	NLP See Tree 3	LSQ See Tree 4
matrix inequalities	e04svf	e04svf	e04svf		

Table 1
Decision Matrix

Tree 1: Linear Programming (LP)

Is the problem sparse/large-scale? — yes → e04mtf, e04nqf, e04nkf

no → e04mff, e04ncf

Tree 2: Quadratic Programming (QP)

Is the problem sparse/large-scale? — yes → Is it convex? — yes → e04nqf, e04stf, e04nkf

no → e04stf, e04vhf, e04ugf

no → Is it convex? — yes → e04ncf

no → e04nff

Tree 3: Nonlinear Programming (NLP)

Is the problem sparse/large-scale? — **yes** → Is it unconstrained? — **yes** → Are first derivatives available? — **yes** → e04stf, e04dga, e04vhf, e04uga

Are first derivatives available? — **no** → e04vhf, e04uga

Is it unconstrained? — **no** → Are first derivatives available? — **yes** → Are second derivatives available? — **yes** → e04stf

Are second derivatives available? — **no** → e04vhf, e04stf, e04uga

Are first derivatives available? — **no** → e04vhf, e04uga

Is the problem sparse/large-scale? — **no** → Are there linear or nonlinear constraints? — **yes** → e04uca, e04ufa, e04wdf

Are there linear or nonlinear constraints? — **no** → Is there only one variable? — **yes** → Are first derivatives available? — **yes** → e04bba

Are first derivatives available? — **no** → e04aba

Is there only one variable? — **no** → Is it unconstrained with the objective with many discontinuities? — **yes** → e04cbf or e05saf

Is it unconstrained with the objective with many discontinuities? — **no** → Are first derivatives available? — **yes** → Are second derivatives available? — **yes** → Are you an experienced user? — **yes** → e04lbf

Are you an experienced user? — **no** → e04lyf

Are second derivatives available? — **no** → Are many function evaluations problematic? — **yes** → Are you an experienced user? — **yes** → e04uca, e04ufa, e04wdf

Are you an experienced user? — **no** → e04kyf

Are many function evaluations problematic? — **no** → Are you an experienced user? — **yes** → e04kdf

Are you an experienced user? — **no** → e04kzf

Are first derivatives available? — **no** → Is the objective expensive to evaluate or noisy? — **yes** → e04jcf

Is the objective expensive to evaluate or noisy? — **no** → Are you an experienced user? — **yes** → e04uca, e04ufa, e04wdf

Are you an experienced user? — **no** → e04jyf

Tree 4: Least squares problems (LSQ)

```
Is the objective sum of
squared linear functions and ──yes── Are there linear constraints? ──yes──        e04ncf
no nonlinear constraints?
                                            │ no
                                            │
                                       Are there simple bounds? ──yes──       e04pcf, e04ncf
                                            │ no
                                            │
                                       Chapters F04, F07 or F08 or
                                       e04pcf, e04ncf
    │ no
    │
Are there linear or nonlinear ──yes──        e04usf
constraints?
    │ no
    │
Are there bound constraints? ──yes── Are first derivatives ──yes──          e04usf
                                     available?
                                            │ no
                                            │
                                           e04fff
    │ no
    │
Are first derivatives ──yes── Are second derivatives ──yes── Are you an experienced ──yes──      e04hef
available?                     available?                     user?
                                     │                              │ no
                                     │                              │
                                     │                             e04hyf
                                     │ no
                                     │
                              Are many function ──yes── Are you an experienced ──yes──       e04gbf
                              evaluations problematic?  user?
                                     │                       │ no
                                     │                       │
                                     │                      e04gyf
                                     │ no
                                     │
                              Are you an experienced ──yes──      e04gdf
                              user?
                                     │ no
                                     │
                                    e04gzf
    │ no
    │
e04fff, e04fcf
```

5   Functionality Index

Linear programming (LP),
  dense,
    active-set method/primal simplex,
      alternative 1 .... e04mff
      alternative 2 .... e04ncf
  sparse,
    interior point method (IPM) e04mtf
    active-set method/primal simplex,
      recommended (see Section 3.4) e04nqf
      alternative.. e04nkf

Quadratic programming (QP),
  dense,
    active-set method for (possibly nonconvex) QP problem e04nff
    active-set method for convex QP problem... e04ncf
  sparse,
    active-set method sparse convex QP problem,
      recommended (see Section 3.4) e04nqf
      alternative.. e04nkf
    interior point method (IPM) for (possibly nonconvex) QP problems e04stf

6   Auxiliary Routines Associated with Library Routine Arguments

e04cbk   nagf_opt_uncon_simplex_dummy_monit
        See the description of the argument **monit** in e04cbf.
e04fcv   nagf_opt_lsq_uncon_quasi_deriv_comp_lsqlin_fun
        See the description of the argument **lsqlin** in e04gbf.
e04fdz   nagf_opt_lsq_dummy_lsqmon
        See the description of the argument **lsqmon** in e04fcf, e04gdf and e04hef.
e04ffu   nagf_opt_bobyqa_ls_dummy_monit
        See the description of the argument **mon** in e04fff.
e04hev   nagf_opt_lsq_uncon_quasi_deriv_comp_lsqlin_deriv
        See the description of the argument **lsqlin** in e04gbf.
e04jcp   nagf_opt_bounds_bobyqa_func_dummy_monfun
        See the description of the argument **monfun** in e04jcf.
e04mtu   nagf_opt_lp_imp_dummy_monit
        See the description of the argument **monit** in e04mtf.
e54nfu   nagf_opt_qp_dense_sample_qphess
        See the description of the argument **qphess** in e04nff/e04nfa and h02cbf.
e04nfu   nagf_opt_qp_dense_sample_qphess_old
        See the description of the argument **qphess** in e04nff/e04nfa and h02cbf.

e54nku   nagf_opt_qpconvex1_sparse_dummy_qphx
          See the description of the argument **qphx** in e04nkf/e04nka and h02cef.
e04nku   nagf_opt_qpconvex1_sparse_dummy_qphx_old
          See the description of the argument **qphx** in e04nkf/e04nka and h02cef.
e04nsh   nagf_opt_qpconvex2_sparse_dummy_qphx
          See the description of the argument **qphx** in e04nqf.
e04stu   nagf_opt_ipopt_dummy_mon
          See the description of the argument **mon** in e04stf.
e04stv   nagf_opt_ipopt_dummy_objfun
          See the description of the argument **objfun** in e04stf.
e04stw   nagf_opt_ipopt_dummy_objgrd
          See the description of the argument **objgrd** in e04stf.
e04stx   nagf_opt_ipopt_dummy_confun
          See the description of the argument **confun** in e04stf.
e04sty   nagf_opt_ipopt_dummy_congrd
          See the description of the argument **congrd** in e04stf.
e04stz   nagf_opt_ipopt_dummy_hess
          See the description of the argument **hess** in e04stf.
e04udm   nagf_opt_nlp1_dummy_confun
          See the description of the argument **confun** in e04ucf/e04uca and e04usf/e04usa.
e04ugm   nagf_opt_nlp1_sparse_dummy_confun
          See the description of the argument **confun** in e04ugf/e04uga.
e04ugn   nagf_opt_nlp1_sparse_dummy_objfun
          See the description of the argument **objfun** in e04ugf/e04uga.
e04wdp   nagf_opt_nlp2_dummy_confun
          See the description of the argument **confun** in e04wdf.

7 Routines Withdrawn or Scheduled for Withdrawal

The following lists all those routines that have been withdrawn since Mark 19 of the Library or are scheduled for withdrawal at one of the next two marks.

Withdrawn Routine	Mark of Withdrawal	Replacement Routine(s)
e04ccf/e04cca	24	e04cbf
e04fdf	19	e04fyf
e04gcf	19	e04gyf
e04gef	19	e04gzf
e04hff	19	e04hyf
e04jaf	19	e04jyf
e04kaf	19	e04kyf
e04kcf	19	e04kzf
e04laf	19	e04lyf
e04unf	22	e04usf/e04usa
e04upf	19	e04usf/e04usa
e04zcf/e04zca	24	No longer required

8 References

Bard Y (1974) *Nonlinear Parameter Estimation* Academic Press

Chvátal V (1983) *Linear Programming* W.H. Freeman

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

Nocedal J and Wright S J (2006) *Numerical Optimization* (2nd Edition) Springer Series in Operations Research, Springer, New York

Wolberg J R (1967) *Prediction Analysis* Van Nostrand