NAG C Library 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 *constraints*. The functions in the NAG C Library are concerned with *minimization* only, since the problem of maximizing a given objective function $F(x)$ is equivalent to minimizing $-F(x)$.

This introduction is only a brief guide to the subject of optimization. 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).

Users who are unfamiliar with the mathematics of the subject may find some sections difficult at first reading; if so, they should *concentrate* on Section 2.1, Section 2.2, Section 2.5, Section 2.6, Section 3 and Section 4.

2 Background

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 function and the constraints, 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:

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

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

2.1.2 Nonlinear least-squares problems

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

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

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 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:

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

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

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:

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

subject to the

equality constraints:	$a_i^T x = b_i$	$i = 1, 2, \ldots, m_1;$
inequality constraints:	$a_i^T x \geq b_i$	$i = m_1 + 1, m_1 + 2, \ldots, m_2;$
	$a_i^T x \leq b_i$	$i = m_2 + 1, m_2 + 2, \ldots, m_3;$
range constraints:	$s_j \leq a_i^T x \leq t_j$	$i = m_3 + 1, m_3 + 2, \ldots, m_4;$
		$j = 1, 2, \ldots, m_4 - m_3;$
bounds constraints:	$l_i \leq x_i \leq u_i$	$i = 1, 2, \ldots, n$

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 problem); if $F(x)$ is a quadratic function, the problem is termed a *quadratic programming* problem (QP problem). 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:

equality constraints:	$c_i(x) = 0$	$i = 1, 2, \ldots, m_5;$
inequality constraints:	$c_i(x) \geq 0$	$i = m_5 + 1, m_5 + 2, \ldots, m_6;$
range constraints:	$v_j \leq c_i(x) \leq w_j$	$i = m_6 + 1, m_6 + 2, \ldots, m_7,$
		$j = 1, 2, \ldots, m_7 - m_6$

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 $-c_i(x) \geq 0$.

2.2 Geometric representation and Terminology

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

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

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

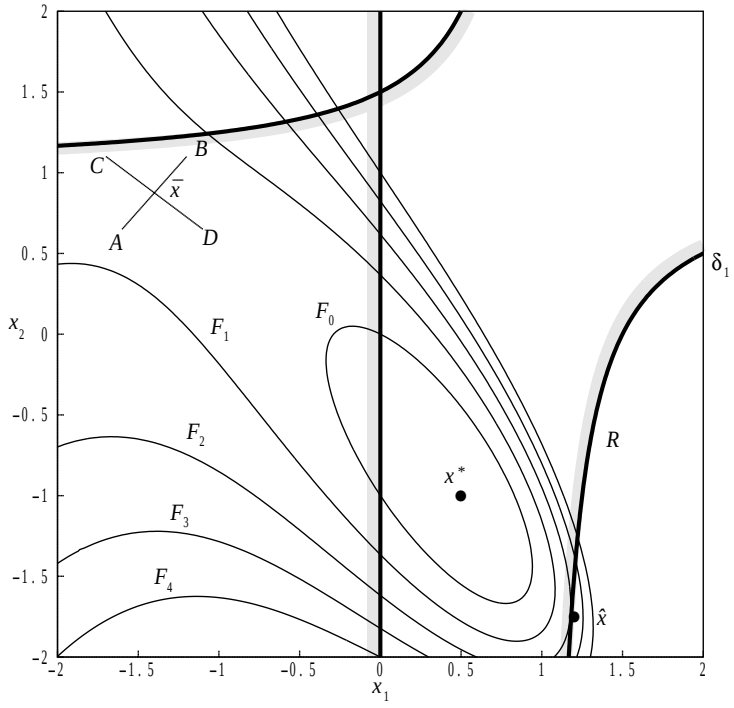

Figure 1

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 $F(x)$ takes specific constant values. The point $x^* = (\frac{1}{2}, -1)^T$ is a *local unconstrained minimum*, i.e., 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 $c_1(x) : x_1 + x_2 - x_1 x_2 - \frac{3}{2} \geq 0$, represented by the curved shaded line in Figure 1, then x^* is not a feasible point because $c_1(x^*) < 0$. The solution of the new constrained problem is $\hat{x} \simeq (1.1825, -1.7397)^T$, 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.,

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

For the function illustrated in Figure 1,

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

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 $F(x)$.

2.2.3 Jacobian matrix

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 $\partial f_i/\partial x_j$.

2.2.4 Matrix of constraint normals

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

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

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.

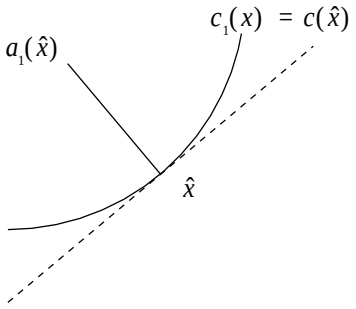

Figure 2

Note that if $c_i(x)$ is a linear constraint involving $a_i^T x$, 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 a solution.

2.3.1 Unconstrained minimization

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

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

(b) $G(x^*)$ 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 which 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:

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

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

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

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 (c) 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

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

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

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

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

Define

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

$G_Z(x) = Z^T G(x) Z$, 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:

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

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

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

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

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

(d) $\lambda_i^* > 0$ 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.

2.3.4 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:

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

where the dependence on x has been suppressed for compactness.

The projected gradient vector $g_Z(x)$ is the vector $Z^T g(x)$. 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:

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

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:

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

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

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

(d) $\lambda_i^* > 0$ 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 (b) 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 $\{x^{(k)}\}$ 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 2.6.

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 2.4.1.

2.4.1 One-dimensional optimization

The Library contains two special functions for minimizing a function of a single variable. Both are based on safeguarded polynomial approximation. One 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 the user to other sources for a full explanation.

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

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

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

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 $\|J(x)^T J(x)\|$ (for example, when $f(x)$ represents the goodness of fit of a nonlinear model to observed data). In such cases, $2J(x)^T J(x)$ 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 function which solves sparse LP, QP or NLP problems. 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:

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

(b) 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 $F(x)$;

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

Normally, users should restrict themselves 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 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 $x_1^2 + x_2^2 + 1000$ 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. You are here

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 nag_machine_precision (X02AJC)) 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:

(a) is the sequence $\{x^{(k)}\}$ converging?

(b) is the sequence $\{F^{(k)}\}$ converging?

(c) 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 (c) 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, the user 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 specified by the user 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 value of **fail.code** other than **NE_NOERROR** 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

The optimization functions in the chapter output information during the minimization process, which allows the user to monitor progress. This monitoring information can also 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 objective 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 function returning a warning.

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 $G = 2J^TJ$ (see Section 2.4.1). The Jacobian, J, or a factorization of J is returned by the unconstrained least-squares functions in the Library. In addition, a function is supplied in the Library to estimate variances of the parameters following the use of the nonlinear least-squares functions, in the case that $G = 2J^TJ$ 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

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

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

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

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

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

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

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

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

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

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

2.6.5 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 procedures 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.

No attempt is made to ensure that the current iteration satisfies any nonlinear constraints. Users who wish to prevent their objective function being evaluated outside some known region (where it may be undefined or not practically computable), 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).

3 Optional Facilities

The optimization functions of Chapter e04 provide a range of optional facilities: these offer the possibility of fine control over many of the algorithmic parameters and the means of adjusting the level and nature of the printed results.

Control of these optional facilities is exercised by a structure of type **Nag_E04_Opt**, the members of the structure being optional input or output parameters to the function. After declaring the structure variable, which is named **options** in this manual, the user must initialize the structure by passing its address in a call to the utility function nag_opt_init (e04xxc). Selected members of the structure may then be set to the user's required values and the address of the structure passed to the optimization function. Any member which has not been set by the user will indicate to the optimization function that the default value should be used for this parameter. A more detailed description of this process is given below in Section 3.4.

Examples of parameters to the algorithms which may be altered from their default value are **options.linesearch_tol** and **options.optim_tol** (these control the accuracy to which the linesearch and final solution are computed, respectively), and **options.max_iter** (which limits the number of iterations the algorithm will perform). Certain members of **options** supply further details concerning the final results, for example the member pointer **options.state** gives the status of the constraints, while in the LP and QP solvers the member pointers **options.lambda** and **options.ax** also give the final values of the Lagrange multipliers and the linear constraints respectively.

The optimization process may sometimes terminate before a satisfactory answer has been found, for instance when the limit on the number of iterations has been reached. In such cases the user may wish to re-enter the function making use of the information already obtained. Functions nag_opt_conj_grad (e04dgc), nag_opt_lsq_no_deriv (e04fcc) and nag_opt_lsq_deriv (e04gbc) can simply be re-entered but the functions nag_opt_lp (e04mfc), nag_opt_lin_lsq (e04ncc), nag_opt_qp (e04nfc), nag_opt_sparse_convex_qp (e04nkc), nag_opt_nlp (e04ucc) nag_opt_nlp_sparse (e04ugc) and nag_opt_nlin_lsq (e04unc) have the structure member **start** which needs to be set appropriately if the function is to make use of information from the previous call.

3.1 Control of Printed Output

Results from the optimization process are printed by default on the `stdout` (standard output) stream. These include the results after each iteration and the final results at termination of the search process. The amount of detail printed out may be increased or decreased by setting the optional parameter **print_level**, i.e., the structure member **options.print_level** . This member is an `enum` type, **Nag_PrintType**, and an example value is **Nag_Soln** which when assigned to **options.print_level** will cause the optimization function to print only the final result; all intermediate results printout is suppressed.

If the results printout is not in the desired form then it may be switched off, by setting **options.print_level** = **Nag_NoPrint**, or alternatively the user can supply his or her own function (except for nag_opt_nlp_sparse (e04ugc)) to printout or make use of both the intermediate and final results. Such a function would be assigned to the pointer to function member **options.print_fun**; the user defined function would then be called in preference to the NAG print function.

In addition to the results, the values of the parameters to the optimization function are printed out when the function is entered; the Boolean member **options.list** may be set to **FALSE** if this listing is not required.

Printing may be output to a named file rather than to `stdout` by providing the name of the file in the **options** character array member **outfile** . Error messages will still appear on `stderr`, if **fail.print** = **TRUE** or the **fail** parameter is not supplied (see the Essential Introduction to the NAG C Library for details of error handling within the library).

3.2 Memory Management

The **options** structure contains a number of pointers for the input of data and the output of results. The optimization functions will manage the allocation of memory to these pointers; when all calls to these functions have been completed then a utility function nag_opt_free (e04xzc) can be called by the user's program to free the NAG allocated memory which is no longer required.

If the calling function is part of a larger program then this utility function allows the user to conserve memory by freeing the NAG allocated memory before the **options** structure goes out of scope.
nag_opt_free (e04xzc) can free all NAG allocated memory in a single call, but it may also be used selectively. In this case the memory assigned to certain pointers may be freed leaving the remaining memory still available; pointers to this memory and the results it contains may then be passed to other functions in the user's program without passing the structure and all its associated memory.

Although the NAG C Library optimization functions will manage all memory allocation and deallocation, it may occasionally be necessary for the user to allocate memory to the **options** structure from within the calling program before entering the optimization function.

An example of this is where the user stores information in a file from an optimization run and at a later date wishes to use that information to solve a similar optimization problem or the same one under slightly changed conditions. The pointer **options.state** , for example, would need to be allocated memory by the user before the status of the constraints could be assigned from the values in the file. The member **options.start** would need to be appropriately set for functions nag_opt_lp (e04mfc) and nag_opt_qp (e04nfc).

If the user does assign memory to a pointer within the **options** structure then the deallocation of this memory must also be performed by the user; the utility function nag_opt_free (e04xzc) will only free memory allocated by NAG C Library optimization functions. When user allocated memory is freed using the standard C library function `free()` then the pointer should be set to NULL immediately afterwards; this will avoid possible confusion in the NAG memory management system if a NAG function is subsequently entered.

3.3 Reading Optional Parameter Values From a File

Optional parameter values may be placed in a file by the user and the function nag_opt_read (e04xyc) used to read the file and assign the values to the **options** structure. This utility function permits optional parameter values to be supplied in any order and altered without recompilation of the program. The values read are also checked before assignment to ensure they are in the correct range for the specified option. Pointers within the **options** structure cannot be assigned to using nag_opt_read (e04xyc).

3.4 Method of Setting Optional Parameters

The method of using and setting the optional parameters is:

step 1 Declare a structure of type **Nag_E04_Opt**.

step 2 Initialize the structure using nag_opt_init (e04xxc).

step 3 Assign values to the structure.

step 4 Pass the address of the structure to the optimization function.

step 5 Call nag_opt_free (e04xzc) to free any memory allocated by the optimization function.

If after step 4, it is wished to re-enter the optimization function, then step 3 can be returned to directly, i.e., step 5 need only be executed when all calls to the optimization function have been made.

At step 3, values can be assigned directly and/or by means of the option file reading function nag_opt_read (e04xyc). If values are only assigned from the options file then step 2 need not be performed as nag_opt_read (e04xyc) will automatically call nag_opt_init (e04xxc) if the structure has not been initialized.

4 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.

nag_ip_bb (h02bbc) solves *integer programming* problems, and nag_transport (h03abc) solves a special type of such problem known as a *transportation* problem.

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

5 References

Powell M J D (1974) Introduction to constrained optimization *Numerical Methods for Constrained Optimization* (ed P E Gill and W Murray) 1–28 Academic Press

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

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

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

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

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

6 Functions Withdrawn or Scheduled for Withdrawal

The following routines have been scheduled for withdrawal at Mark 8.

e04jbc e04kbc

7 Available Functions

Minimum, function of one variable

e04abc Minimizes a function of one variable, using function values only

e04bbc Minimizes a function of one variable, requires first derivatives

Unconstrained minimum

e04ccc Unconstrained minimization using simplex algorithm

e04dgc Unconstrained minimization using conjugate gradients

Unconstrained minimum, simple bounds

e04lbc Solves bound constrained problems. 1st and 2nd derivatives are required

Unconstrained minimum of a sum of squares

e04fcc Unconstrained nonlinear least squares (no derivatives required)

e04gbc Unconstrained nonlinear least squares (first derivatives required)

Linear programming problem (dense)

e04mfc Linear programming

Linearly-constrained linear least-squares problem or convex quadratic programming problem (dense)

e04ncc Solves linear least-squares and convex quadratic programming problems (non-sparse)

Quadratic programming problem (dense)

e04nfc Quadratic programming

Linear programming problem or convex quadratic programming problem (sparse)

e04nkc Solves sparse linear programming or convex quadratic programming problems

Minimum function of several variables, nonlinear constraints, using function values and optional first derivatives

e04ucc Minimization with nonlinear constraints using a sequential QP method

Nonlinear programming problem (sparse)

e04ugc NLP problem (sparse)

Utility functions

e04hcc Checks 1st derivatives of a user-defined function

e04hdc Checks 2nd derivatives of a user-defined function

e04myc Free memory allocated by nag_opt_sparse_mps_read (e04mzc)

e04mzc Read MPSX data for sparse LP or QP problem from a file

e04xac Computes an approximation to the gradient vector and/or the Hessian matrix for use with nag_opt_nlp (e04ucc) and other nonlinear optimization functions

e04xxc Initialisation function for option setting

e04xyc Read options from a text file

e04xzc Memory freeing function for use with option setting

e04yac Least-squares derivative checker for use with nag_opt_lsq_deriv (e04gbc)

e04ycc Covariance matrix for nonlinear least-squares