2 Some Tools of Optimization

This chapter discusses, in more detail, the tools of optimization that were introduced in Chapter 1. It begins with Lagrange multipliers, which see more use in optimization than any other device. It moves on then to discussions of the Kuhn–Tucker conditions and calculus of variations. It includes methods for solving optimization problems numerically such as sequential linear programming, which is the workhorse of this text.

2.1 The Lagrange Multiplier Rule

The Lagrange multiplier rule and the Kuhn–Tucker conditions come directly from topics of linear algebra. Gale (1960) offers a very elegant approach to these topics that is highly recommended by the authors. The approach used here is somewhat more intuitive.

The Lagrange multiplier rule can be used to convert an optimization problem with *equality* constraints into a nonlinear system of equations. It states that under appropriate circumstances there exist Lagrange multipliers, λ, such that

$$\text{minimize } f(x) \mid \boldsymbol{g}(x) = 0 \quad \Leftrightarrow \quad \nabla f + \lambda^{\mathrm{T}} \nabla g = 0\,,\ g = 0.$$

While the Lagrange multiplier rule itself skirts basic optimization ideas, its proof deals with them directly. The idea is that at an optimal point, the variation of f, $df = \nabla f \cdot dx$, must be zero or positive in any feasible direction dx which must also satisfy $dg = \nabla g \cdot dx = 0$. (Were this not the case it would be possible to reduce the objective function implying that the point in question is not optimal.)

One approach to Lagrange multipliers comes directly from the either/or theorems of linear algebra (Gale 1960),

Either the equation $A^T x = b$ has a solution $x \neq 0$ or the equations $Ay = 0$ and $b^T y = 1$ have a solution.

Obviously, both alternatives cannot hold since $A^T x = b \Leftrightarrow y^T A^T x = y^T b = 0$ if $Ay = 0$ also holds. In terms of optimization this either/or theorem reads

Either the equation $\nabla \mathrm{g}^T \lambda = -\nabla \mathrm{f}$ has a solution $\lambda \neq 0$ or the equations $\nabla g \cdot dx = 0$ and $\nabla f \cdot dx = -1$ have a solution.

W.R. Spillers, K.M. MacBain, *Structural Optimization*,
DOI 10.1007/978-0-387-95865-1_2, © Springer Science+Business Media, LLC 2009

Mechanically the Lagrange multiplier rule is sometimes described as follows. Given an optimization problem with equality constraints, the *Lagrangian, L*, is formed by combining f with each constraint g_i multiplied by a Lagrange multiplier λ_i as

$$L = f + \lambda^T g = f + \sum_{i=1}^{m} \lambda_i g_i$$

This allows a constrained minimization problem to be treated as an unconstrained problem that is then solved by taking the gradient

$$\nabla L = \nabla f + \lambda^T \nabla g = 0 \quad \text{with} \quad g = 0$$

also holding. In more detail this system can be written as

$$\frac{\partial f}{\partial x_j} + \frac{\partial}{\partial x_j} \sum_{i=1}^{m} \lambda_i g_i = 0, \qquad j = 1, \ldots, n$$

Another, rather physical, derivation of the Lagrange multiplier rule follows. Let x be an optimal point so that $df = \nabla f \cdot dx = 0$ for any $\boldsymbol{dg} = \boldsymbol{\nabla g} \cdot dx = 0$. Let $\boldsymbol{\nabla g}$ have rank m. Partition $\boldsymbol{\nabla g}$, perhaps after reordering, so that

$$\nabla g \cdot dx = 0 \qquad \Rightarrow \qquad [\nabla g_1, \nabla g_2] \begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix} = 0$$

with $\boldsymbol{\nabla} g_i$ nonsingular. It follows that

$$dx_1 = -\nabla g_1^{-1} \nabla g_2 dx_2$$

where dx_2 can be selected arbitrarily. Under these conditions, $\boldsymbol{\nabla g}^T \lambda = \nabla f$ can be written as

$$\begin{bmatrix} \nabla g_1^T \\ \nabla g_2^T \end{bmatrix} \lambda = \begin{bmatrix} \nabla f_1 \\ \nabla f_2 \end{bmatrix}$$

which can be solved for λ as

$$\lambda = (\nabla g_1^T)^{-1} \nabla f_1$$

It remains to show that

$$\nabla g_2^T \lambda = \nabla f_2 \qquad \Rightarrow \qquad \nabla g_2^T (\nabla g_1^T)^{-1} \nabla f_1 = \nabla f_2$$

Now in partitioned form

$$\nabla f \cdot dx = 0 \qquad \Rightarrow \qquad \nabla f_1 \cdot dx_1 + \nabla f_2 \cdot dx_2 = 0$$

or

$$-\nabla f_1^T \nabla g_1^{-1} \nabla g_2 dx_2 + \nabla f_2 \cdot dx_2 = 0$$

or

$$(\nabla g_2^T (\nabla g_1^T)^{-1} \nabla f_1 - \nabla f_2) \cdot dx_2 = 0$$

Since dx_2 is arbitrary, the term in parenthesis must be zero completing the proof of the theorem. Several examples of the use of Lagrange multipliers are included in Chapter 1.

2.2 The Kuhn–Tucker Conditions

The Kuhn–Tucker conditions, sometimes called the Karush–Kuhn–Tucker conditions, deal with optimization problems with *inequality* constraints. They do for inequality constraints what the Lagrange multiplier theorem does for equality constraints, but inequality constraints turn out to be much more difficult to deal with. The result is that the computational role of the Kuhn–Tucker conditions is nothing as pervasive as is the Lagrange multiplier rule.

Theorem: (Kuhn–Tucker) At an optimal point of the problem minimize f(x) subject to $\mathbf{g(x) \leq 0}$*, there exist Lagrange multipliers* $\lambda \geq 0$ *which satisfy* $\nabla f + \lambda^T \nabla g = 0$ *and* $\lambda_i g_i = 0$ *for* $i = 1,\ldots,m$.

The proof of this lemma follows directly from

Farkas' Lemma (Simonnard, 1966): The statement that $a \cdot x < 0$ *for all x such that* $Ax \geq 0$ *is equivalent to the statement that there exists a* $\lambda \geq 0$ *such that* $a + A^T \lambda = 0$.

Proof: The proof proceeds by induction on the number of rows of A. When A has one row it is only necessary to show that $a + A^T\lambda = 0 \Leftrightarrow a^T x < 0, Ax \geq 0$. This is shown in Fig. 2.1. In this case a, A, and x are vectors.

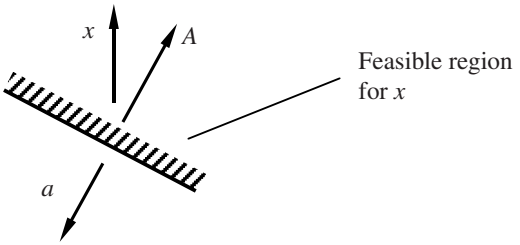

Fig. 2.1. Case $m = 1$

When m=2, the system can be put into the form

$$a + [\, A_1\, A_2\,] \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = 0 \qquad \Leftrightarrow \qquad a^T x < 0 \text{ and } \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} x \geq 0\lambda$$

In Fig. 2.2 this system is given geometric interpretation by regarding the rows of A to be n-vectors. Quite briefly, this figure attempts to show that if $-a$ can be expressed as a positive linear (convex) combination of A_1 and A_2 then there is no feasible x which corresponds to a decreasing value of the objective function.

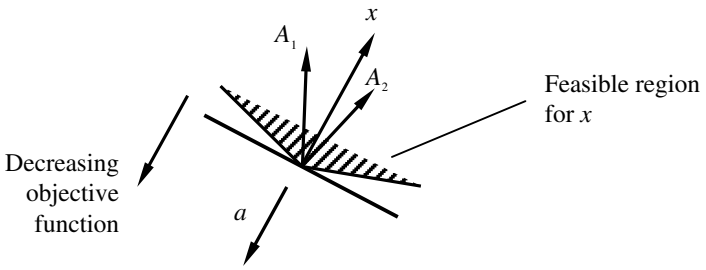

Fig. 2.2. Case $m = 2$

For an algebraic proof of Farkas' Lemma, the reader should consult Simonnard (1966).

With regard to the Kuhn–Tucker theorem, the statement $\lambda_i\, g_i = 0$ is added to handle the case in which some constraints may not be *tight.* In this case the variation *dx* is not required to lie within the corresponding tangent planes of these constraint surfaces and the associated Lagrange multipliers are taken to be zero.

As an example, consider the problem of Luenberger (1984, p. 315):

$$\begin{aligned} &\textit{minimize } 2x^2 + 2xy + y^2 - 10x - 10y \\ &\textit{subject to } x^2 + y^2 \le 5 \quad \textit{and} \quad 3x + y \le 6 \end{aligned}$$

The Kuhn–Tucker conditions for this problem are

$$\nabla f + \lambda^T \nabla g = 0 \qquad \lambda^T g = 0 \qquad \lambda \ge 0$$

or

$$\begin{aligned} 4x + 2y - 10 + \lambda_1(2x) + \lambda_2(3) &= 0 \\ 2x + 2y - 10 + \lambda_1(2y) + \lambda_2(1) &= 0 \\ \lambda_1\left(x^2 + y^2 - 5\right) &= 0 \\ \lambda_2(3x + y - 6) &= 0 \\ \lambda_1 &\ge 0 \\ \lambda_2 &\ge 0 \end{aligned}$$

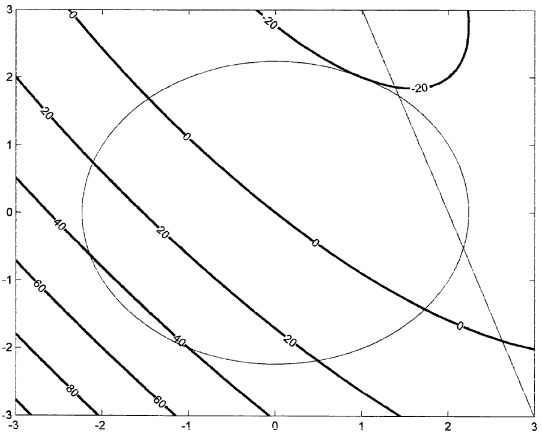

Fig. 2.3. Optimization Example

Rather than solving this system directly, it can be verified that there is a solution with x=1, y=2, λ_1 = 1, and λ_2 = 0. This corresponds to the first constraint being tight and the second not tight. (See Fig. 2.3. Optimization in this case attempts to move downhill, up and to the right, while staying inside the circle and to the left of the line.) Note that the Kuhn–Tucker conditions are satisfied at this point.

2.3 Calculus of Variations

When dealing with continuous systems, it is common to encounter integrals that are to be maximized or minimized subject to certain conditions. This is the realm of calculus of variations.

First of all there is the *fundamental lemma of calculus of variations*, which states that

$$\int_{x_0}^{x_1} \eta(x)\varphi(x)dx = 0 \quad \textit{for all functions } \eta(x) \quad \Rightarrow \quad \varphi(x) = 0$$

Proof: If the function η(x) is made sufficiently discontinuous (a delta function, for example) the proof becomes obvious. See Courant and Hilbert (1953) for another discussion of this lemma.

Calculus of variations deals with the following problem (Fig. 2.4):

$$\min_{y} \quad I = \int_{x_0}^{x_1} f(x, y, y', y'')dx$$

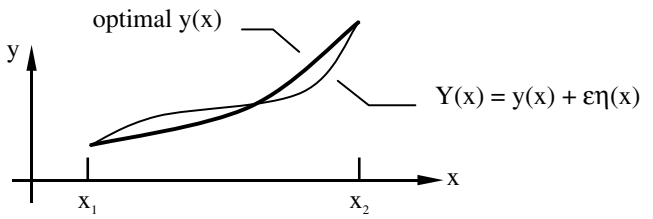

Fig. 2.4. Variation of y(x)

The technique used to solve this problem is to write the integral as a function of a single parameter ε as

$$I(\varepsilon) = \int_{x_1}^{x_2} f(x, Y, Y', Y'')dx$$

with

$Y = y + \varepsilon\eta(x)$ $\quad \partial Y/\partial\varepsilon = \eta$
$Y' = y' + \varepsilon\eta'(x)$ $\quad \partial Y'/\partial\varepsilon = \eta'$

The optimal I then satisfies

$dI/d\varepsilon = 0$ at $\varepsilon = 0$

But

$$dI / d\varepsilon \,|_{\varepsilon=0} = \int_{x_1}^{x_2} \left(\frac{\partial f}{\partial y}\eta + \frac{\partial f}{\partial y'}\eta + \frac{\partial f}{\partial y''}\eta'' \right) dx = 0$$

Integrating by parts gives

$$I'(0) = \int_{x_1}^{x_2} \left(\frac{\partial f}{\partial y} - \frac{d}{dx}\frac{\partial f}{\partial y'} + \frac{d^2}{dx^2}\frac{\partial f}{\partial y''} \right) \eta \, dx + boundary\, terms = 0$$

Using the lemma, Euler's equation follows directly as

$$\frac{\partial f}{\partial y} - \frac{d}{dx}\frac{\partial f}{\partial y'} + \frac{d^2}{dx^2}\frac{\partial f}{\partial y''} = 0$$

For a simple example of the use of Euler's equation, return to the beam problem from Chapter 1,

Find m to minimize $\int |m| \, dx$ subject to $m'' = -w$

Using Lagrange multipliers, form the Lagrangian

$L = \int (|m| + \lambda (m'' + w))dx$

Applying Euler's theorem directly gives the optimality condition

$\text{sgn}\, m + \lambda'' = 0$

We will return to the application of this optimality condition in Chapter 3.

2.4 Newton's Method

Given a vector function (an *n*-vector) **F(x)**, it is possible to *linearize* this function at some point $\mathbf{x}_0$ by making a Taylor series expansion and neglecting terms of order greater than 1 as

$$F(x) \cong F(x_0) + \nabla F(x_0) \cdot (x - x_0)$$

or simply

$$F(x) \cong F_0 + \nabla F_0 \cdot dx$$

Using Newton's method to solve a nonlinear system of n equations in n unknowns then requires iterating, at each step solving a system of linear equations,

$$F(x) = 0 \quad \Rightarrow \quad F(x) \cong F_0 + \nabla F_0 \cdot dx = 0 \quad \Rightarrow \quad dx = -(\nabla F_0)^{-1} F_0$$

The case (see below) for a single equation in a single variable x is most familiar.

Newton's method has the remarkable property of quadratic convergence, which will now be demonstrated for a scalar system. For a vector system the reader is referred to Isaacson and Keller (1966). If a function F(x) is expanded in a Taylor series about the point x_0 it follows that (Taylor series with remainder)

$$F(x) = F(x_0) + F'(x_0)(x - x_0) + \tfrac{1}{2} F''(\xi)(x - x_0)^2$$

where ξ is some point in the interval (x, x_0). In Newton's method an improved approximation x_1 is computed using only linear terms as

$$x_1 = x_0 - F(x_0)/F'(x_0)$$

The exact root x^* satisfies the equation

$$0 = F(x_0) + F'(x_0)(x^* - x_0) + \tfrac{1}{2} F''(\xi)(x^* - x_0)^2$$

or

$$0 = F(x_0)/F'(x_0) + (x^* - x_0) + \tfrac{1}{2} F''(\xi)(x^* - x_0)^2/F'(x_0)$$

Combining equations gives

$$0 = x_0 - x_1 + (x^* - x_0) + \tfrac{1}{2} F''(\xi) (x^* - x_0)^2 / F'(x_0)$$

finally

$$x_1 - x^* = (x^* - x_0)^2 \tfrac{1}{2} F''(\xi) / F'(x_0)$$

Now if $|\tfrac{1}{2} F''(\xi) / F'(x_0)| \leq 1$ it follows that

$$|x^* - x_1| \leq (x^* - x_0)^2$$

or that the convergence is quadratic in the neighborhood of the point x^*.

2.4.1 An Example of Newton's Method

Consider a beam with a rectangular section subjected to an axial load *P* and bending moment *M* (see Fig. 2.5). Given that the ratio of the base to the height of the cross section is fixed, we seek the height of the section that will support these forces and satisfy an allowable stress requirement.

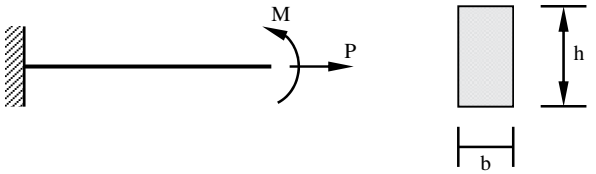

Fig. 2.5. A Beam Design Problem

For an allowable stress of σ_a, this constraint is given as

$$\frac{6M}{bh^2} + \frac{P}{bh} \leq \sigma_a$$

Anticipating optimization in the work that follows, this inequality will be treated as an equality. That is, we will assume that the optimal design will have the smallest value of h for which this allowable stress criterion is satisfied as an equality. Parameters for this example are tabulated below:

Parameter	Value
M	100 k-in.
P	20 k
α	0.3
σ_a	20 ksi

Introducing a parameter for the aspect ratio of the cross section as $\alpha = b/h$ the inequality can be written as

$$F(h) = 6M + hP - \alpha h^3 \sigma_a = 0$$

Here F is simply a cubic equation for h that will be solved as an equality using Newton's method. The required first derivative in this case is

$$F' = P - 3\alpha h^2 \sigma_a$$

Starting with an arbitrary initial value of h = 10 in., the first five iterations are

Table 2.1 Iterations of Newton's Method

Step	h (in.)	F	F'	dh
1	10	–5200	–1780	–2.92
2	7.08	–1387	–882	–1.57
3	5.51	–292	–526	–0.55
4	4.95	–29	–421	–0.07
5	4.88	0	–409	0.00

For this example, five iterations give a solution that is accurate to within two-tenths of an inch. One may also examine the solution space of this example graphically by plotting $F(h)$ (Fig. 2.6).

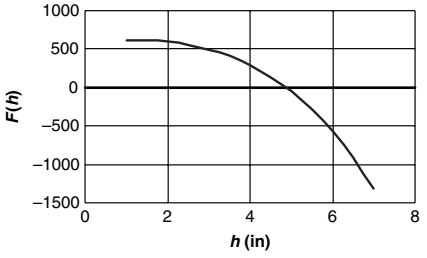

Fig. 2.6. Newton's Method

Referring to the equations of Newton's method for a single variable given above, one may now consider a graphical interpretation of a single step from x_i to x_{i+1} for a function $y = F(x)$. (see Fig. 2.7).

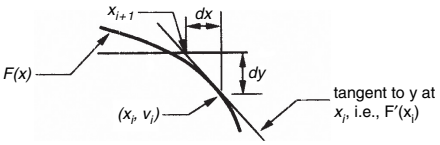

Fig. 2.7. A Typical Step of Newton's Method

Rearranging the identity $F' = dy/dx$ gives $dx = dy/F'$. In seeking a root $F(x) = 0$, dy is seen graphically to be equal to the magnitude of the function at x_i, noted here as $F(x_i)$. Newton's method is thus derived graphically as $x_{i+1} = x_i - F(x_i)/F'(x_i)$.

It can be instructive to examine some details of this example:

- One must start with an initial value. For this example the initial value was 10 in. although any non-zero value could be used because of the nature of the problem to be solved. It will be seen later that sometimes obtaining an initial value is non-trivial. Further, the solution may depend on the initial value if several solutions exist.

- The procedure is iterative and incremental. Throughout this book nonlinear problems will be solved using an iterative approach. In general, increments are sought which smoothly move toward the solution.

- Each step of Newton's method is linear. Each step (i.e., computing *dh*) is made using the tangent of the function (i.e., a straight line) at a given point. The function itself is, of course, nonlinear but a linear approximation made at the running solution *h*. As indicated in this example, the incrementally linear approach will often give excellent convergence.

- The transition from inequality to equality of the function *F* may be considered as what will be termed later a *tight* constraint. Tight constraints are defined as those that govern the solution. This means that although there may be several constraints, all of them may not be active. Generally it is not possible to determine which constraints are tight before a solution is developed.

- The solution is approximate. From a practical point of view one may consider the solution exact since the error may be made arbitrarily small by increasing the number of iterations. However, one should still keep in mind

that strictly speaking, the solution is approximate, as will normally be the case with any iterative solution.

- All solutions listed in Table 2.1 are within the *feasible space*, meaning that they all satisfy the constraint. This is not a general property of Newton's method.

2.5 Linear Programming

Historically, the workhorse tool of optimization has been linear programming (Damkilde et al., 1994). While more general tools have struggled with their applications, linear programming has continued to produce results for real (large) systems. In this section an introduction to linear programming will be sketched. Linear programming solvers are discussed in detail in Appendix A.

Linear programming is, of course, mathematical programming where both the objective function and the constraints are linear. The standard form of linear programming seems to be

Find x to minimize $c^T x$ subject to $Ax = b, \quad x \geq 0$

The so-called *simplex method* is frequently used to solve linear programming problems. It is based upon the following procedure. Partition the matrix A after some possible rearrangement of rows and columns as $A = [\, A_B \,,\, A_N \,]$ so that A_B is nonsingular. (Note that a partition of A implies a partition of x.) It follows that

$$Ax = b \quad \Rightarrow \quad A_B x_B + A_N x_N = b \quad \Rightarrow \quad x_B = A_B^{-1}(b - A_N x_N)$$

This implies that

$$\begin{aligned} c^T x &= c_B^{\,T} x_B + c_N^{\,T} x_N = c_B^{\,T} A_B^{-1}(b - A_N x_N) + c_N^{\,T} x_N \\ &= c_B^{\,T} A_B^{-1} b + (c_N^{\,T} - c_B^{\,T} A_B^{-1} A_N) x_N \end{aligned}$$

Note that this decomposition of the matrices A and x has allowed the objective function to be written in terms of the variables x_N. This decomposition is fundamental to the *simplex method.* The variables x_B and x_N are called basic and nonbasic variables, respectively. The simplex method starts with some set $x_N = 0$. (Starting may be nontrivial.) The coefficient of each $(x_N)_i$ in the expression $c^T x = 0$ is then examined. If any coefficient is negative, that variable can be increased to reduce the objective function. The limit of this increase is determined by some element of x_B going to zero. By this process there is one variable going into the basis and one variable coming out. This procedure gives rise to a theorem of linear

programming which states that "if there exists an optimal solution, there exists a basic optimal solution".

2.5.1 An Example of the Simplex Method

The simplex method will now be discussed by way of an example. Consider the following problem:

$$\begin{aligned} &\textit{maximize} \quad \phi = 5x_1 + 4x_2 + 3x_3 \\ &\textit{subject to} \quad 2x_1 + 3x_2 + x_3 \le 5 \\ &\qquad\qquad 4x_1 + x_2 + 2x_3 \le 11 \\ &\qquad\qquad 3x_1 + 4x_2 + 2x_3 \le 8 \\ &\qquad\qquad x_1, x_2, x_3 \ge 0 \end{aligned}$$

Slack variables are first added to convert this problem to one with *equality* constraints as

$$\begin{bmatrix} 2 & 3 & 1 & 1 & & \\ 4 & 1 & 2 & & 1 & \\ 3 & 4 & 2 & & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \\ 8 \end{bmatrix}$$

In this case the starting solution is obvious: $x = [0\ \ 0\ \ 0\ \ 5\ \ 11\ \ 8]$

Initial basis: 4,5,6

x_1 into basis

$$\begin{aligned} &2x_1 + z_1 = 5 \quad \Rightarrow \quad x_1 = 5/2 \quad (\textit{smallest}) \\ &4x_1 + z_2 = 11 \qquad\quad x_1 = 11/4 \\ &3x_1 + z_3 = 8 \qquad\quad x_1 = 8/3 \end{aligned}$$

$\therefore z_1$ *out of basis*

New basis

$$\begin{bmatrix} 1 & 1.5 & 0.5 & 0.5 & & \\ 0 & -5 & 0 & -2 & 1 & \\ 0 & -0.5 & 0.5 & -1.5 & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1 \\ 0.5 \end{bmatrix}$$

New objective function

$$x_1 = 2.5 - 1.5x_2 - 0.5x_3 - 0.5z_1 \quad \Rightarrow \quad \phi = 12.5 - 3.5x_2 + 0.5x_3 - 2.5z_1$$

x_3 into basis

$$x_1 + 0.5x_3 = 2.5 \quad \Rightarrow \quad x_3 = 5$$
$$z_3 + 0.5x_3 = 0.5 \quad \Rightarrow \quad x_3 = 1 \quad (smallest)$$

$$\therefore z_3 \quad out\ of\ basis$$

New basis

$$\begin{bmatrix} 1 & 2 & 0 & 2 & & -1 \\ 0 & -5 & 0 & -2 & 1 & \\ 0 & -1 & 1 & -3 & & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

New objective function

$$x_3 = 1 + x_2 + 3z_1 - 2z_3 \quad \Rightarrow \quad \phi = 13 - 3x_2 - z_1 - z_3$$

At this point there is no possible improvement. The final solution is $x_2 = z_1 = z_3 = 0$, $x_1 = 2$, $z_2 = 1$, $x_3 = 1$.

Program 30 on the CD discusses the simplex method and this example.

2.5.2 Interior Point Methods

The simplex method, the workhorse method of linear programming, is said to be an *exterior* method since it works on the exterior of the feasible region. Following the publication, in 1984, of a paper by Karmarkar, there was a flurry of activity over what have been called *interior* methods. Roughly, the idea is that it may be more efficient, when solving a linear programming problem, to move through the interior to find an optimal point rather than simply moving around the boundary as the simplex method does.

This section will give the reader a brief look at interior methods. For more information the reader is referred to the excellent book by Arbel (1993) and the more comprehensive work of Wright (1997).

The so-called *primal* method starts with the linear programming problem in the form

$$\textit{minimize } c^T x \quad \textit{subject to } Ax = b \textit{ and } x \geq 0$$

and looks for an incremental change dx, such that

$$c^T x_{new} \leq c^T x_o \quad \textit{and} \quad Ax_{new} = b$$

This implies that

$$c^T dx \leq 0 \quad \textit{and} \quad Adx = 0$$

It turns out that if dx is taken to be $-Pc$, with the *projection* $P = I - A^T(AA^T)^{-1}A$ then

$$c^T dx = -c^T Pc = -c^T P^2 c = -\left\|Pc\right\| \leq 0$$

since $P = P^T$ and $P^2 = P$.

There are two other points to be made. First, it is effective to work with variables that are *scaled.* This is done by transforming the problem as $x_1 = D^{-1}x$ with the matrix D diagonal having its nonzero values as the unscaled variables. Finally the increment dx is scaled so that x always remains within the feasible region. Some of the code outlined by Arbel is given in Programs 31 and 32 on the CD.

2.6 Sequential Linear Programming

Given an optimization problem such as minimize $f(x)$ subject to $\mathbf{g(x) \leq 0}$, there is a *Newtonian* approach in which the system is linearized at some point x_0 and the simplified system is solved as

minimize $f(x_0) + \nabla f(x_0) \cdot dx$ subject to $\mathbf{g(x_0) + \nabla g(x_0) \cdot dx \leq 0}$

This problem results in a new approximate solution as $x \Rightarrow x + dx$.

Repeated application of this step results in what is called *sequential linear programming* (El Hallabi 1994) since each step involves solving a linear programming problem. Since linear programming has the ability of dealing efficiently with large systems, sequential linear programming has considerable potential as a tool for structural optimization. (It should be noted here that sequential linear programming commonly requires the application of move limits. That point will be discussed in Chapter 3.)

It turns out that it is an easy matter to code this algorithm (see below). In the case described here the problem is to

$$\begin{aligned} \textit{minimize} \quad & f = -x_1^3 - 2x_2^2 + 10x_1 - 6 - 2x_2^3 \\ \textit{subject to} \quad & g_1 = 10 - x_1 x_2 \geq 0 \\ & g_2 = x_1 \geq 0 \\ & g_3 = 10 - x_2 \geq 0 \end{aligned}$$

There are several steps in this program:

- The objective function and the constraints must be described for the algorithm. That is done in the subroutine ROUTINES. (For each new problem this subroutine has to be replaced with an appropriate subroutine.)
- In the main program x, n, and m describe the starting values, the size of the x-vector, and the number of constraints.
- DO 4 ITER=1,NIT describes the iterative process. The linear programming solver DLPRS from the IMSL package is called at each iteration.
- Note the upper and lower bounds are specified as 10% of the running value of the variable under consideration.

Program03

```
c      SEQUENTIAL LP SOLVER
       USE IMSL
       DIMENSION X(50),G(50),DELG(50,50),B(50),DELF(50)
      & ,IRTYPE(50),XLB(50),XUB(50),XSOL(50),DSOL(50)
      & ,A(50,50)
       X(1)=1.
       X(2)=1.
```

```
      N=2
      M=3
      LDA=50
      NIT=100
      CFAC=.1
      DO 4 ITER=1,NIT
      CALL ROUTINES (X,F,G,DELG,N,ITER,DELF)
      DO 1 I=1,M
      B(I)=-G(I)
      IRTYPE(I)=2
    1 CONTINUE
      DO 2 I=1,N
      XLB(I)=-CFAC*X(I)
    2 XUB(I)= CFAC*X(I)
      CALL DLPRS(M,N,DELG,LDA,B,B,DELF,IRTYPE,XLB,XUB,OBJ,
     & XSOL,DSOL)
      DO 3 I=1,N
    3 X(I)=X(I)+XSOL(I)
    4 CONTINUE
      STOP
      END
C
      SUBROUTINE ROUTINES (X,F,G,DELG,N,ITER,DELF)
      DIMENSION X(50),G(50),DELG(50,50),DELF(50)
      F=-(X(1)**3)-2.*(X(2)**2)+10.*X(1)-6-2.*(X(2)**3)
      DELF(1)=-3.*(X(1)**2)+10.
      DELF(2)=-4.*X(2)-6.*(X(2)**2)
      G(1)=10.-X(1)*X(2)
      G(2)=X(1)
      G(3)=10.-X(2)
      DELG(1,1)=-X(2)
      DELG(1,2)=-X(1)
      DELG(2,1)=1.
      DELG(2,2)=0.
      DELG(3,1)=0.
      DELG(3,2)=-1.
      WRITE(60,*)ITER,F,(X(I),I=1,N)
      WRITE(61,*)ITER,F
      RETURN
      END
```

The performance is rather good in this case but this algorithm does not always work.

Output from Program 3

Iteration	Objective Function	x_1	x_2
1	-1.000000	1.000000	1.000000
2	-2.811000	9.000000E-01	1.100000
3	-4.902763	8.100000E-01	1.210000
4	-7.356440	7.290000E-01	1.331000
5	-10.285466	6.561000E-01	1.464100
6	-13.842975	5.904900E-01	1.610510
7	-18.232380	5.314410E-01	1.771561
8	-23.721951	4.782969E-01	1.948717
9	-30.664513	4.304672E-01	2.143589
10	-39.523773	3.874205E-01	2.357948
11	-50.909420	3.486784E-01	2.593743

12	-65.623672	3.138106E-01	2.853117
13	-84.723083	2.824295E-01	3.138429
14	-109.600487	2.541866E-01	3.452271
15	-142.093719	2.287679E-01	3.797499
16	-184.629623	2.058911E-01	4.177248
17	-240.415451	1.853020E-01	4.594974
18	-313.692261	1.667718E-01	5.054471
19	-410.071808	1.500946E-01	5.559918
20	-536.983582	1.350852E-01	6.115910
21	-704.267944	1.215767E-01	6.727500
22	-924.964844	1.094190E-01	7.400250
23	-1216.360352	9.847709E-02	8.140276
24	-1601.377686	8.862938E-02	8.954304
25	-2110.426025	7.976644E-02	9.849734
26	-2205.282471	7.178980E-02	10.000000
27	-2205.354248	6.461082E-02	10.000000
98	-2205.999756	3.643540E-05	10.000000
99	-2205.999756	3.279186E-05	10.000000
100	-2205.999756	2.951267E-05	10.000000

The disk includes an EXCEL Solver solution of this problem as Program04.

2.7 Other Methods of Mathematical Programming

There is a world of mathematical programming algorithms out there (see, for example, Ecker and Kupferschmid 1991 and Kumar, 2000). In particular, Ecker and Kupferschmid describe an "ellipsoidal algorithm" that is quite robust. Some of these will be returned to later in this text. For problems of structural optimization the reader may wish to consult Schittkowski et al., 1994. (Schittkowski has, over the years, made available computer code for structural optimization.)

Some of the more commonly used algorithms include sequential quadratic programming.

Since the Excel Solver program (Fylstra et al. 1998) is given a lot of use in this text, we will refer briefly to the generalized reduced gradient method of solving mathematical programming problems about which Solver is built. When discussing general approaches to solving mathematical programming problems, it is worthwhile to start with the fact that simple gradient methods just do not work. That is, if you try to move in the direction of the gradient of the objective function it is the conventional wisdom that you get hung up on the constraint surfaces.

Briefly, the generalized reduced gradient (GRG) method works something like this. Rather than working with the typical nonlinear programming formulation

minimize f (**x**) subject to $\mathbf{g(x)} \leq 0$

it is convenient in this case to start with a formulation of

minimize f(**x**) subject to **g(x)** = 0

This can be done on the basis of introducing slack variables or even just working with the tight constraints and ignoring the others. Now the system is linearized and the elements of **x** partitioned after possibly reordering them as

$$\min f_0 + \nabla f_0 dx \qquad \textit{subject to} \quad g_0 + \nabla g_0 dx = 0$$

or

$$\min f_0 + \nabla f_{01} dx_1 + \nabla f_{02} dx_2 \quad \textit{subject to} \quad g_0 + \nabla g_{01} dx_1 + \nabla g_{02} dx_2 = 0$$

Assuming that ∇g_{02} has the appropriate rank, it follows that

$$dx_2 = -\nabla g_{02}{}^{-1} \nabla g_{01} dx_1$$

and

$$\begin{aligned} \nabla f\, dx &= \nabla f_1 dx_1 + \nabla f_2 dx_2 \\ &= (\nabla f_1 - \nabla f_2 \nabla g_{02}{}^{-1} \nabla g_{01}) dx_1 \end{aligned}$$

This is the reduced gradient. A search can now be carried out in the direction of the reduced gradient projected on the working surface $\nabla g = 0$.

2.8 Genetic Algorithms

Genetic algorithms (GA) are a member of a class of *heuristic* algorithms that represent an active research area today (see, for example, Sivakumar et al. 2004). Broadly speaking, they model the natural selection process of biology. This section briefly describes some of the features of the GA algorithm but we leave it to the reader to pursue this area more fully. Genetic algorithms offer their users the potential of creative behavior and, in that sense, are part of an exciting field. They can be, at the same time, quite arbitrary and tend to neglect gradients, which the authors see as a serious shortcoming. But taken at their most general, it cannot be said for certain that they leave anything out.

In the following we will describe some features of a very simple genetic algorithm. In this case a design (chromosome) is represented by a binary, finite string. This string contains a description of the elements of the design (genes) that are under study. There must also be a metric (fitness) that can be used to determine that one design is better than another. In the paper cited above, one of their examples is a 10-bar truss whose design elements are the bar areas. The fitness in this case is the weight of the structure.

The question then is how to generate new designs given a starting design. Classically this involves *crossover* and *mutation.* With crossover, two designs are compared and parts taken from each to form a new design. With mutation, there is the potential for change within an individual design. For a sizeable design represented by a binary string, there are a large number of designs possible. It is this large number that creates problems for genetic algorithms. On the other hand, the algorithms used in this text work in the small using, for example, Taylor series methods. And if you work in the small you will not usually get radically different designs. This is what drives the designer to use heuristic methods.

2.9 Problems

1. Use the simplex method to solve the following problem:

   maximize $6x_1 + 8x_2 + 5x_3 + 9x_4$

   subject to

   $$2x_1 + x_2 + x_3 + 3x_4 \leq 5$$
   $$x_1 + 3x_2 + x_3 + 2x_4 \leq 3$$
   $$x_1, x_2, x_3, x_4 \geq 0$$

2. Discuss the solution of (Ecker and Kupferschmid 1991, pp. 315)

   minimize $(x_1 - 20)^4 + (x_2 - 12)^4$

   subject to
   $$8\, e^{(x_1 - 12)/9} - x_2 + 4 \leq 0$$
   $$6\,(x_1 - 12)^2 + 25\, x_2 - 600 \leq 0$$
   $$-x_1 + 12 \leq 0$$

   (They solve this problem using the so-called ellipsoid algorithm. Try to solve it using sequential linear programming.)

3. Discuss the paper (Koumousis and Georgiou 1994) “Genetic Algorithms in Discrete Optimization of Steel Truss Roofs”.

4. Discuss the paper (Fylstra et al. 1998) “Design and Use of the Microsoft Excel Solver”.

5. Discuss the use of *barrier* methods (Wright 1997) in linear programming.

6. Discuss the “Optimization Overview” in the Matlab Optimization Toolbox.

http://www.springer.com/978-0-387-95864-4

Structural Optimization
Spillers, W.R.; MacBain, K.M.
2009, XV, 304 p., Hardcover
ISBN: 978-0-387-95864-4