Optimisation Theory and Applications

Contents

Chapter 1: Introduction.

Chapter 2: Linear Programming

Chapter 3: Integer Programming

Chapter 4: Nonlinear Programming

Chapter 5: Dynamic Programming

Last revised: 17th September 2019. File: opt19.tex
Author: Colin Campbell (C.Campbell@bris.ac.uk)

1 Introduction.

Optimisation theory appears in a vast range of application domains. These could be engineering applications where we wish to maximise the efficiency of a given system. They could be company economics applications where we wish to maximise profitibility or minimise the number of employees. They could be applications in medicine where we wish to design an optimal classifier for predicting the future course of a disease, for example. Indeed, optimisation theory appears to have useful applications across nearly all the sciences, engineeering, economics, medicine and many other areas.
At its simplest level optimisation theory is concerned with the maximisation or minimisation of a function, called the *objective function*, which has one or more variables. This maximisation or minimisation task may be subject to possible constraints - we called this *constrained optimisation*. In this course we will cover the four principal areas of optimisation theory:

- *Linear Programming* (often denoted LP). Linear programming problems occur in a very wide range of application domains, including manufacturing, commerce and finance, industrial control, management, network flow analysis, etc. With linear programming the objective function and any constraints are assumed to be linear functions of the variables and these variables can take any value.
- *Integer Programming* (often denoted IP). In this case the variables are fixed to take integer values. Such *integer programming* problems occur in a range of contexts in engineering, science, investment, industry and commerce.
- *Nonlinear Programming*. In this case the objective function and possible constraints may be nonlinear functions of the variables.
- *Dynamic Programming* (often denoted DP and sometimes called multi-stage decision making). This is a different type of optimisation problem in which we have a number of subtasks and an eventual goal. An example might be game playing where we have a number of subtasks (e.g. moves in a chess game) and we can make good or bad choices - the eventual goal would be to optimise our chances of winning the game. A more practical example would be the loading of goods into the containers of a container ship. The goods we load at each step may be of variable value and size and our goal is the maximise the total value of the goods we transport. Dynamic programming has many applications from robotics and machine learning (e.g. reinforcement learning) to algorithms for sequence analysis of genetic code.

Examinable Material

Optimisation theory is a large subject with a variety of different approaches. In this course we aim to cover the essential elements of the subject. In some cases there are methods which are complex to explain or where it is worth knowing that the method exists but without invoking great detail.

In light of the above remarks, several subsections are marked **not examinable**. Depending on the time available I expect to cover these in some lectures but they will not appear in the exam. This also lightens the learning load in this course and enables us to focus on essentials. The subject is assessed *by exam only* and the following will be potentially covered in the exam:

- All the material in Chapter 2 (linear programming) unless indicated otherwise (the Two Phase Method Section).
- Integer programming (Chapter 3): you would be expected to be able to formulate an integer programming problem and to be able to use the *branch and bound method*.
- Nonlinear programming (Chapter 4): the steepest descent and conjugate gradient methods are examinable but the Nelder/Mead downhill simplex method *is not*. The Newton method is examinable but the Davidon-Fletcher-Powell method is not. You are expected to understand the other elements in this chapter on concavity/convexity, constrained optimisation and the use of barrier functions.
- Dynamic programming (Chapter 5): you will **not** be expected to formulate a novel functional equation but you would be expected to be able to remember and use the functional equations (84), (92) and (94) for the network route, capital budgeting and cargo loading problems respectively and to be able to find a solution to these types of problem. You could be presented with a new functional equation in an exam and expected use it appropriately.

2 Linear Programming

2.1 Introduction.

To introduce the subject of Linear Programming we shall consider a simple problem: you are being served tea at a restaurant and on your tray you have 8 lumps of sugar, a teapot and a jug containing about 10 spoonfuls of milk. There are 6 cups of tea in the teapot and you take 2 sugar lumps and a spoonful of milk per cup. Implicit in the problem statement is the aim or objective of the problem solver. We may presume you are trying to get as many cups of tea for your money as possible. However, suppose that you are sharing the tea with a friend who takes one lump of sugar and two spoonfuls of milk. The problem is now more complicated and we will find it helpful to formulate it in mathematical terms. Let us define: x as the number of cups of tea you have and let y be the number of cups of tea your friend consumes. If you wish to maximise the total number of cups of tea drunk then we would write the objective as

$$\max[x + y] \tag{1}$$

where $x + y$ is called the *objective function*. Alternatively you may value the cups you drink twice as much as the cups your friend drinks, in which case the objective is:

$$\max[2x + y] \tag{2}$$

Another objective would be to use up as much of the tea as possible. The tea left over would be $6 - x - y$ giving an objective of

$$\min[6 - x - y] \tag{3}$$

which, in practice, will give the same result as (1). We now come to formulate the constraints on the problem. The variables x and y cannot be negative so we have a *non-negativity constraint*:

$$\begin{aligned} x &\geq 0 \\ y &\geq 0 \end{aligned} \tag{4}$$

or more concisely,

$$x, y \geq 0 \tag{5}$$

There are limits on the availability of both milk and sugar giving

Sugar constraint:

$$2x + y \leq 8 \tag{6}$$

Milk constraint:

$$x + 2y \leq 10 \tag{7}$$

The *formulation* of the problem consists of an objective, say (1), where the objective function is $x + y$, and a set of constraints (6) and (7). The characteristic of a *Linear Programming* problem is that the objective function and all the constraints are linear functions of the variables, i.e. there are no products of variables and the

variables are not squared or raised to any power. For example, $3x + 4y + z$ is linear but $2xy + 4y + z$ is not and neither is $3x^2 + 2y^2 + z$

The general form for a linear programming (LP) problem is:

$$\max \text{or} \min \left[\sum_{i=1}^{n} c_i x_i \right] \tag{8}$$

subject to constraints of the form:

$$\left. \begin{matrix} \geq \\ = \\ \leq \end{matrix} \right\} b_i \qquad\qquad i = 1, 2, \ldots, m \tag{9}$$

$$\sum_{j=1}^{n} a_{ij} x_i$$

and non-negativity constraints

$$x_i \geq 0 \qquad\qquad i = 1, 2....n. \tag{10}$$

If some variables are negative we can usually re-formulate the problem in terms of non-negative variables.

We will use matrix notation where convenient in the text and write (8), (9) and (10) as:

$$\max_{\mathbf{x}} \text{or} \min_{\mathbf{x}} \left[\mathbf{c}^T \mathbf{x} \right]$$

subject to:

$$\left. \begin{matrix} \geq \\ = \\ \leq \end{matrix} \right\} \mathbf{b}$$

$$\mathbf{Ax}$$

$$\mathbf{x} \geq \mathbf{0}$$

2.2 A Second Example: A Feed Mix Problem.

In this section we will formulate a mixing problem illustrating a more real-life example of Linear Programming. An animal feed supplier uses two different types of grain in differing proportions to provide feeds with different levels of the 3 main nutrients A, B and C. Having mixed suitable amounts of the two grains so as to provide the required nutrients the weight is made up with a filler containing none of the nutrients A, B or C. Grain #1 costs £5 per unit and Grain #2 costs £4 per unit (# denotes *number*). The Table below shows the nutrient composition of the grains. If the feed supplier has an order for feed containing at least 60 units of nutrient A, 84 units of nutrient B and 36 units of nutrient C per ton, how many units of grains 1 and 2 should he put into each ton of feed?

Nutrient / Grain	A	B	C
#1	6	6	2
#2	3	6	6

Nutrient content per unit of grains 1 and 2.

The Formulation. There are many mixtures of the grains which will contain at least as much of the nutrients as the purchaser has specified. Thus a mixture of 10 units of #1 and 10 units of #2 gives 90 units of A, 120 of B and 80 of C - a surfeit of all nutrients. Alternatively 20 units of grain #2 only provides 60 units of A, 120 of B and 120 of C. This is exactly enough of A, but an excess of B and C. Amongst all the possible mixtures that satisfy the nutrient constraints we want to find the cheapest. The objective function will therefore be the cost of the mixture and we wish to minimise this cost.
If we define x_1 as the number of units of grain #1 and x_2 as the number of units of grain #2 then the objective is

$$\min [5x_1 + 4x_2]$$

since the costs of the grains are £5 and £4 per unit.

We now write down the constraints. There must be at least 60 units of nutrient A and x_1 units of grain #1 contain $6x_1$ units of A, while x_2 units of grain # 2 contain $3x_2$ units of A so we can write:

Nutrient A constraint: $6x_1 + 3x_2 \geq 60$

Similarly

Nutrient B constraint: $6x_1 + 6x_2 \geq 84$
Nutrient C constraint: $2x_1 + 6x_2 \geq 36$

As the quantity of grain in a mixture can never be negative we also have the non-negativity constraint:

$$x_1, x_2 \geq 0$$

The problem can be displayed in a matrix form (see last section) with $\mathbf{c} = [5, 4]$; $\mathbf{b} = [60, 84, 36]$ and

$$\mathbf{A} = \begin{bmatrix} 6 & 3 \\ 6 & 6 \\ 2 & 6 \end{bmatrix}$$

2.3 A Third Example: the Clifton Metal Artwork company

The Clifton Metal Artwork Company makes public statues and sculptures out of bronze (a copper/tin alloy). They are commissioned to create a large bronze sculpture for the entrance lobby of a building. The commission requirement is that the sculpture is to be made of bronze with a general upper limit that *there must not be more than 5 tonnes Copper and 5 tonnes Tin used in the sculpture.* The company has two alloys in stock:

7 tonnes of alloy 1 which is 2:1 Copper:Tin
6 tonnes of alloy 2 which is 1:3 Copper:Tin

Let x_i be the weight of alloy i used in making the sculpture. The constraints on the amounts of alloy available give:

$$x_1 \leq 7 \qquad x_2 \leq 6 \qquad \text{or} \qquad \mathbf{x} \leq [7, 6]$$

The 5 tonne copper and 5 tonne tin constraints are thus:

$$\text{For} \quad \text{Copper} \quad : \quad \frac{2}{3}x_1 + \frac{1}{4}x_2 \leq 5$$
$$\text{For} \quad \text{Tin} \quad : \quad \frac{1}{3}x_1 + \frac{3}{4}x_2 \leq 5$$

The amounts of alloy used cannot be negative so we have the non-negativity constraint:

$$\mathbf{x} \geq \mathbf{0}$$

The problem can be written in standard matrix form as:

$$\max \left[\mathbf{c}^T \mathbf{x}\right]$$

subject to:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \frac{2}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{3}{4} \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} 7 \\ 6 \\ 5 \\ 5 \end{bmatrix}$$

$$\mathbf{x} \geq \mathbf{0}$$

Inequalities are algebraically inconvenient to manipulate but we can get rid of them by introducing extra variables. The inequality:

$$x_1 \leq 7$$

is exactly equivalent to the equation:

$$x_1 + s_1 = 7$$

provided the variable s_1, called a *slack variable*, is non-negative. The name slack variable is descriptive since, for instance, if $x_1 = 4$ then $s_1 = 3$ takes up the slack in the equality $x_1 \leq 7$. By introducing slack variables s_1, s_2, s_3, s_4 and multiplying the copper and tin inequalities by 12, we can rewrite the above as:

$$\max \left[\mathbf{c}^T \mathbf{x}\right]$$

subject to

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 8 & 3 & 0 & 0 & 1 & 0 \\ 4 & 9 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 7 \\ 6 \\ 60 \\ 60 \end{bmatrix}$$

if the objective is to maximise the savings to the company, say:

$$\max \left[c_1 x_1 + c_2 x_2 = \mathbf{c}^T \mathbf{x}\right]$$

where c_1 is the saving made from using alloy 1 and c_2 the saving from using alloy 2.

2.4 A Graphical Solution.

The Clifton Metal Artwork example has only two variables and so all possible solutions can be represented by points on a graph. The constraints similarly can be represented as areas of the graph in which the solution may not lie. Taking first the non-negativity constraint $\mathbf{x} \geq \mathbf{0}$, this constrains solution points $\mathbf{x} = [x_1, x_2]$ to lie in the positive quadrant see Figure 1. In Figure 2 the constraint $x_2 \leq 6$ has been added while in Figure 3 the constraint $x_1 \leq 7$ has been added further constraining the feasible region of possible solutions $\mathbf{x}$. We now come to the constraint:

$$\frac{2}{3}x_1 + \frac{1}{4}x_2 \leq 5$$

or multiplying through by 12:

$$8x_1 + 3x_2 \leq 60$$

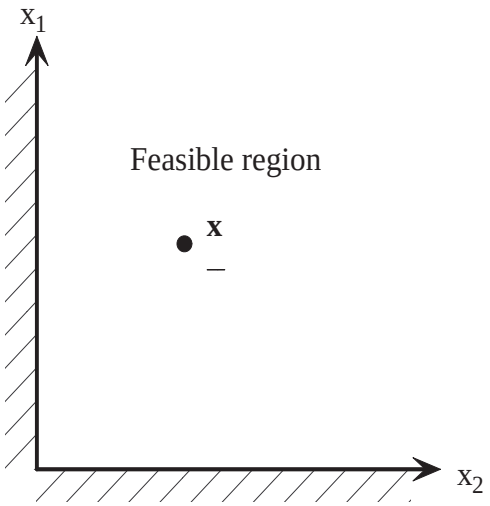

Figure 1: Positivity of the variables x_1 and x_2 means the solution must lie in the positive quadrant illustrated here.

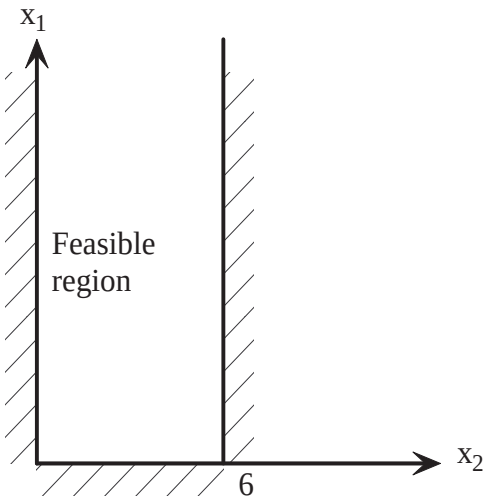

Figure 2: Addition of the constraint: $x_2 \leq 6$

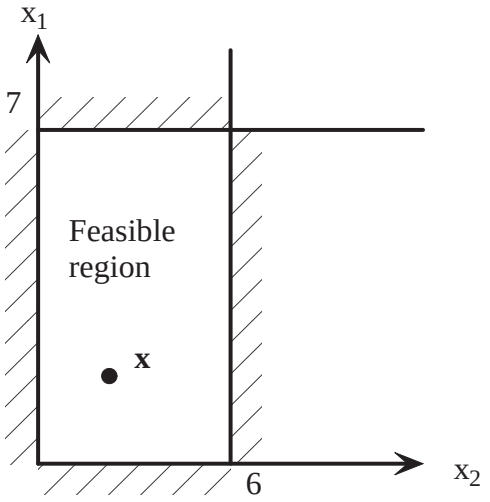

Figure 3: Addition of the constraint: $x_1 \leq 7$

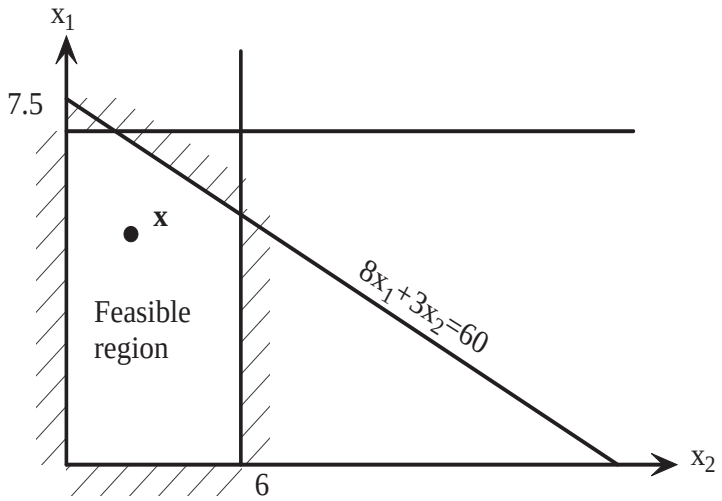

Figure 4: Addition of the constraint: $8x_1 + 3x_2 \leq 60$

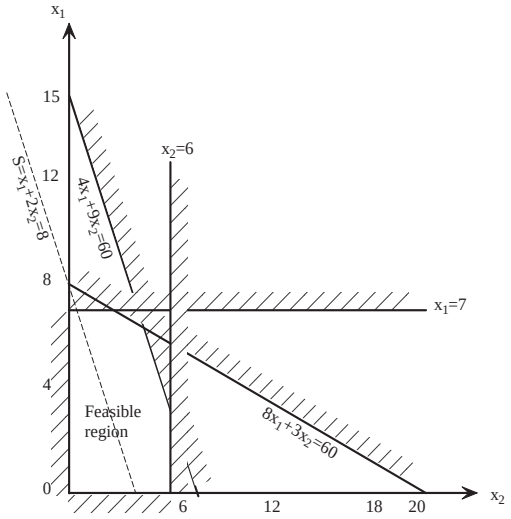

Figure 5: Addition of the constraint: $4x_1 + 9x_2 \leq 60$

In Figure 4 the line $8x_1 + 3x_2 = 60$ has been added. The easiest way to check if the feasible region has been identified correctly is to sample points and see if they satisfy the inequalities. Figure 5 shows the feasible region after the final constraint

$$\begin{aligned} \frac{1}{3}x_1 + \frac{3}{4}x_2 &\leq 5 \\ \text{or} \quad 4x_1 + 9x_2 &\leq 60 \end{aligned}$$

has been added. The feasible region is a *convex polygon* and all Linear Programming problems have feasible regions that are either *convex polygons*, if there are two variables, or *convex polyhedra* when there are more variables. We wish to find the point or set of points inside the feasible region which maximises the savings to the company, $S = \mathbf{c}^T\mathbf{x} = c_1x_1 + c_2x_2$.
For any chosen constant value of S, the equation $S = c_1x_1 + c_2x_2$ is a straight line in (x_1, x_2) space and since c_1 and c_2 are positive constants, maximising the saving S will correspond to choosing a line as far from the origin into the positive quadrant as possible. The gradient of the line is $dx_1/dx_2 = -c_2/c_1$. To obtain a feasible solution a line is moved in towards the origin until at least one point on the line is in the feasible region. This point will then be the optimal solution and the corresponding value of S will be the maximum saving. Consider the case $c_1 = 1,\ c_2 = 2$ the line has a gradient of -2 and in Figure 5 the case $S = x_1 + 2x_2 = 8$ has been drawn. This line crosses the feasible region but by moving the line away from the origin we can increase the savings S, so we move the line, maintaining the gradient constant, until it just touches the edge of the feasible region. It then passes through the point where the lines

$$\begin{aligned} 4x_1 + 9x_2 &= 60 \\ 8x_1 + 3x_2 &= 60 \end{aligned}$$

meet i.e. $x_1 = 6$, $x_2 = 4$. The maximum saving is thus

$$S = x_1 + 2x_2 = 14$$

Whatever the gradient of the line $s_1x_1 + s_2x_2 = S$, it will touch the feasible region at one of the corners or *extreme points* of the feasible region. This result is true for all Linear Programming problems and may be stated as a Theorem:

Theorem 1 *The optimal solution of a Linear Programming problem is always at an extreme point of the feasible region.*

Since there are a finite number of extreme points in the feasible region it is possible to evaluate the objective function S, at each point and thus find the optimal solution.

2.5 For what types of problem is a graphical solution feasible?

In the Clifton Metal Artwork example, which we solved in the last section, there were only two variables and so it was easy to draw a graph of the feasible region. Any problem that can be reduced to two variables can be solved this way. For example

$$\min\left[2x_1 + x_2 - x_3\right]$$

subject to:

$$\begin{aligned} 2x_2 + x_3 &\geq 3 \\ -x_1 + 2x_2 &\leq 3 \\ x_1 - x_2 - x_3 &= 0 \\ x_3 &\leq 1 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

The equation $x_1 - x_2 - x_3 = 0$ can be used to remove one variable from the problem. If we substitute for x_3 the problem becomes

$$\min [z = x_1 + 2x_2]$$

subject to

$$\begin{aligned} x_1 + x_2 &\geq 3 \\ -x_1 + 2x_2 &\leq 3 \\ x_1 - x_2 &\leq 1 \\ x_1, x_2 &\geq 0 \\ x_1 &\geq x_2 \end{aligned}$$

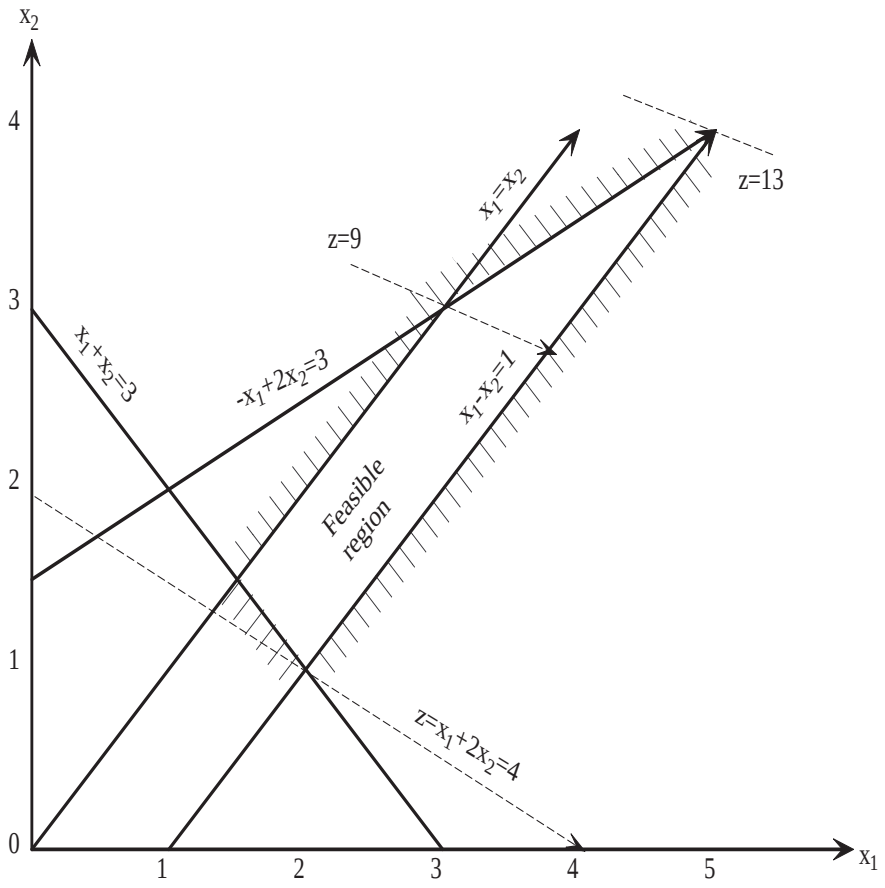

Figure 6: Feasible region described in Section 2.5.

The constraints are drawn in on Figure 6 and since this problem requires a minimisation we move a line of gradient $dx_2/dx_1 = 1/2$ as far in towards the origin as possible while retaining a point in the feasible region.

The line obtained in this way is $x_1 + 2x_2 = 4$ which passes through the point $\mathbf{x} = [2, 1]$. The solution to the original problem can be obtained by back substitution. The objective function is:

$$z = 2x_1 + x_2 - x_3 = 4$$

and $\mathbf{x} = [x_1, x_2, x_3] = [2, 1, 1]$. The graphical method has the advantage that problems with nonlinear constraints but linear objective functions can be solved - however, since the method will not solve problems with more than two or three variables, we have to look to numerical methods for most applications.

2.6 Redundancy and Inconsistency of Equations.

The equation $\mathbf{a}^T\mathbf{x} = b$ can be read as meaning the vector $\mathbf{x}$ lies in a line, plane or hyperplane which is normal to the vector $\mathbf{a}$ and at a distance b from the origin. Suppose the 2-component $\mathbf{x}$ is subject to the three constraints:

$$\begin{aligned} \mathbf{a}_1^T\mathbf{x} &= b_1 \\ \mathbf{a}_2^T\mathbf{x} &= b_2 \\ \mathbf{a}_3^T\mathbf{x} &= b_3 \end{aligned}$$

In two-dimensions they can be viewed as three lines (in more dimensions, *planes* or *hyperplanes*). If the three lines meet at a point (see Figure 7) then this is equivalent to one equation providing no extra information: it is said to be *redundant*. This last equation is said to be *consistent* with the other two equations. On the other hand if the three equations do not meet at one point (see Figure 8) then the set of equations is said to be *inconsistent* and there is no-vector $\mathbf{x}$ which will satisfy all three equations.

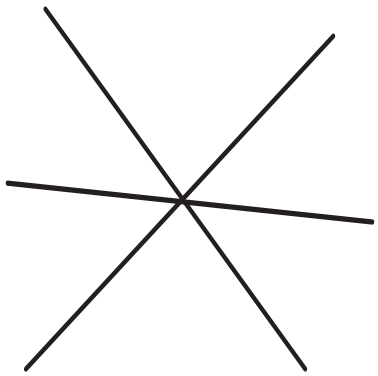

Figure 7: In 2D, if three lines meet at one point, there is a unique solution but one line is redundant for determining this point.

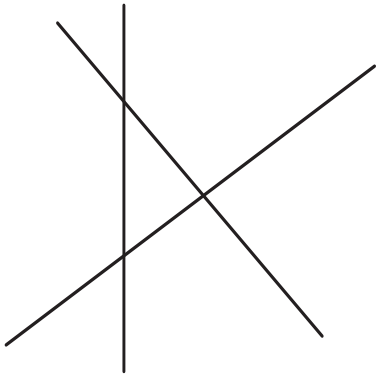

Figure 8: In 2D, if the three lines do not meet at a point, the corresponding set of equations is not consistent.

Example 1

$$3x_1 + 2x_2 = 9 \tag{11}$$

$$2x_1 - x_2 = 1 \tag{12}$$

$$5x_1 + x_2 = 10 \tag{13}$$

Equation (13) is just the sum of equations (11) and (12): any two equations will define $\mathbf{x} = [x_1, x_2]$ so the set of equations is *redundant*.

Example 2

$$\mathbf{Ax} = \begin{bmatrix} 3 & 2 \\ 2 & -1 \\ 5 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 9 \\ 1 \\ 7 \end{bmatrix}$$

Here the row vectors on the LHS are dependent as before i.e. $\mathbf{a}_1 + \mathbf{a}_2 = \mathbf{a}_3$, but the right hand sides are not similarly related so the equations are *inconsistent*. More generally for a system of m equations in n unknowns,

$$\mathbf{Ax} = \mathbf{b}$$

suppose we can find a set of scalars λ_i such that

$$\sum_{i=1}^{m} \lambda_i \mathbf{a}_{i\bullet} = 0$$

(when this condition holds the vectors $\mathbf{a}_i$ are said to be *linearly dependent*, where $\mathbf{a}_{i\bullet}$ are rows i of $\mathbf{A}$). Then the system of equations is *consistent* if:

$$\sum_{i=1}^{m} \lambda_i b_i = 0$$

but, otherwise, it is inconsistent.

A simple way to check for consistency is to row reduce the set of equations as in the Gauss Elimination method of solving equations.

Example:

$$\begin{bmatrix} 2 & 4 & 2 & 0 \\ 1 & 4 & 3 & 2 \\ 1 & 3 & 2 & 1 \\ 1 & 4 & 3 & 2 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 2 \\ 5 \\ 3 \\ 8 \end{bmatrix}$$

For ease of calculation we can write the equations as an augmented matrix $(\mathbf{A}, \mathbf{b})$ thus:

$$\begin{bmatrix} 2 & 4 & 2 & 0 & 2 \\ 1 & 4 & 3 & 2 & 5 \\ 1 & 3 & 2 & 1 & 3 \\ 1 & 4 & 3 & 2 & 8 \end{bmatrix} \qquad \begin{matrix} \text{R1} \\ \text{R2} \\ \text{R3} \\ \text{R4} \end{matrix}$$

which on dividing the first equation by two and subtracting as specified on the left becomes:

$$\begin{matrix} \text{R1/2} \\ \text{R2-R1/2} \\ \text{R3-R1/2} \\ \text{R4-R1/2} \end{matrix} \begin{bmatrix} 1 & 2 & 1 & 0 & 1 \\ 0 & 2 & 2 & 2 & 4 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 2 & 2 & 2 & 7 \end{bmatrix} \qquad \begin{matrix} \text{R5} \\ \text{R6} \\ \text{R7} \\ \text{R8} \end{matrix}$$

and then:

$$\begin{matrix} \text{R5} \\ \text{R6/2} \\ \text{R7-R6/2} \\ \text{R8-R6} \end{matrix} \begin{bmatrix} 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

From which we can see that equation R3 is linearly dependent on R1 and R2 and so is redundant while equation R4 is inconsistent with the other equations.

2.7 Basic Solutions

If we have a set of equations:

$$\mathbf{A}_{m\times n}\mathbf{x} = \mathbf{b}$$

i.e. m linear equations in n unknowns and the set of equations are neither redundant nor inconsistent, then the number of equations m must be less than or equal to the number of variables. In the case of equality, $n = m$, there is a unique solution but when there are fewer equations than unknowns there is an infinite number of feasible solutions. Adding $(n - m)$ extra equations will again specify a unique solution provided the extra equations are neither redundant nor inconsistent. When $(n - m)$ extra equations of the form $x_i = 0$ are added the resulting solutions are called *Basic Solutions*. There are ${}^nC_{n-m}$ ways of choosing the set of variables x_i which are defined to be zero, so there are up to ${}^nC_{n-m}$ Basic Solutions where:

$${}^nC_{n-m} = \frac{n!}{m!(n-m)!}$$

Example 1: Let us consider the following two equations with five unknowns:

$$\begin{aligned} x_1 + x_2 + 4x_3 + 2x_4 + 3x_5 &= 8 \\ 4x_1 + 2x_2 + 2x_3 + x_4 + 6x_5 &= 4 \end{aligned}$$

The maximum number of possible basic solutions is $\frac{5!}{3!2!} = 10$. A basic solution can only include two non-zero variables, which means the associated number of zero nonbasic variables must be three:

Case 1. A *basic feasible solution*: the zeroed (nonbasic) variables are x_2, x_4 and x_5. Thus:

$$\begin{aligned} x_1 + 4x_3 &= 8 \\ 4x_1 + 2x_3 &= 4 \end{aligned}$$

The solution is *unique* with $x_1 = 0$ and $x_3 = 2$: this is a *feasible* solution since both variables are non-negative.

Case 2. A *basic infeasible solution*: the zeroed (nonbasic) variables are x_3, x_4 and x_5. Thus:

$$\begin{aligned} x_1 + x_2 &= 8 \\ 4x_1 + 2x_2 &= 4 \end{aligned}$$

The solution is *unique* with $x_1 = -6$ and $x_3 = 14$: this is an *infeasible* solution since $x_1 < 0$.

Case 3. An *infinity of solutions*: the zeroed (nonbasic) variables are x_1, x_2 and x_5. Thus:

$$\begin{aligned} 4x_3 + 2x_4 &= 8 \\ 2x_3 + x_4 &= 4 \end{aligned}$$

In this case there is no unique solution because the equations are dependent (the lower is twice the above). Fixing one variable determines the other and thus there are an *infinity of solutions*.

Case 4. A *nonexisting solution*: the zeroed (nonbasic) variables are: x_1, x_3 and x_4. Thus:

$$\begin{aligned} x_2 + 3x_5 &= 8 \\ 2x_2 + 6x_5 &= 4 \end{aligned}$$

In this case there is no solution because the equations are *inconsistent*.

Example 2. Let us consider the set of equations:

$$\begin{aligned} x_1 + x_2 + x_3 + 2x_4 &= 1 \\ 2x_1 - x_2 + 3x_3 + 6x_4 &= -6 \\ x_1 + 2x_2 + 2x_3 + 4x_4 &= 1 \end{aligned}$$

Now we will use a matrix formulation and Gaussian elimination:

$$\begin{bmatrix} 1 & 1 & 1 & 2 \\ 2 & -1 & 3 & 6 \\ 1 & 2 & 2 & 4 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ -6 \\ 1 \end{bmatrix}$$

This has three equations in four unknowns and so one variable can be chosen arbitrarily. If we choose one variable to be zero the corresponding solution is a Basic Solution. There are ${}^4C_1$ ways of choosing a variable to equate to zero so there are up to 4 Basic Solutions. We now solve the above set of equations by using row operations to change the augmented matrix $(\mathbf{A}, \mathbf{b})$ into the form $(\mathbf{I}, \mathbf{C}, \mathbf{d})$ where $\mathbf{I}$ is a unit matrix, $\mathbf{C}$ is the transformed part of $\mathbf{A}$ which has not been changed into a unit matrix and $\mathbf{d}$ is the transformed RHS (right hand side). Thus:

$$\begin{bmatrix} 1 & 1 & 1 & 2 & 1 \\ 2 & -1 & 3 & 6 & -6 \\ 1 & 2 & 2 & 4 & 1 \end{bmatrix} \quad \begin{matrix} \text{R1} \\ \text{R2} \\ \text{R3} \end{matrix}$$

$$\begin{matrix} (1) \\ (2)\text{-}2(1) \\ (3)\text{-}(1) \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 2 & 1 \\ 0 & -3 & 1 & 2 & -8 \\ 0 & 1 & 1 & 2 & 0 \end{bmatrix} \quad \begin{matrix} \text{R4} \\ \text{R5} \\ \text{R6} \end{matrix}$$

$$\begin{matrix} \text{R4} \\ \text{R6} \\ \text{R5+3R6} \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 2 & 1 \\ 0 & 1 & 1 & 2 & 0 \\ 0 & 0 & 4 & 8 & -8 \end{bmatrix} \quad \begin{matrix} \text{R7} \\ \text{R8} \\ \text{R9} \end{matrix}$$

$$\begin{matrix} \text{R7-R8} \\ \text{R8-R9/4} \\ \text{R9/4} \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 2 & -2 \end{bmatrix}$$

From this we can read off the Basic Solution corresponding to $x_4 = 0$, i.e. $\mathbf{x} = [1, 2, -2, 0]$. The solution is read off by looking for a 1 in a *unit vector* column (e.g. the first column) then locating the matching number in the final column.

It is convenient to give the name *Basic Variables* to *those variables that we have not defined to be zero* and to call the set of basic variables the *Basis* i.e. x_1, x_2, x_3 are Basic Variables and form the *Basis*.

To find the Basic Solution corresponding to $x_3 = 0$ (x_3 can be called the Non-Basic Variable) we manipulate the augmented matrix into the form

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & \frac{1}{2} & 1 & -1 \end{bmatrix}$$

Where the RH column gives the value of the vector of Basic Variables $\mathbf{x} = [1, 2, 0, -1]$. The constraints $x_1 = 0$ and $x_2 = 0$ are both inconsistent with the initial set of constraints so although there were potentially ${}^4C_1$ Basic Solutions, only two of them are feasible.

2.8 Degenerate Basic Solutions

We have said that for a consistent non-redundant set of equations

$$\mathbf{A}_{m\times n}\mathbf{x} = \mathbf{b}$$

there are up to ${}^nC_{n-m}$ Basic Solutions. Each Basic Solution corresponds to a different set of $(n-m)$ variables being set equal to zero. If several of these Basic Solutions are identical then the set of identical Basic Solutions are known as a degenerate set and any one such solution is said to be *degenerate*.

Example. Consider the system of equations:

$$\begin{bmatrix} 1 & 2 & 1 & 3 & 4 & 1 & 0 \\ 2 & 1 & 2 & 1 & 2 & 1 & 0 \\ 1 & 2 & 4 & 4 & 3 & -1 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Here we have 3 equations and 7 unknowns so there will be a Basic Solution corresponding to each way of setting $7 - 3 = 4$ variables to zero - a total of ${}^7C_4 = 35$ possible Basic Solutions. All these Basic Solutions may not be different and if more than 4 variables are zero in any Basic Solution then there will be as many identical Basic Solutions as there are ways of choosing 4 variables out of the set of variables that are zero.
We can find the Basic Solution corresponding to $x_1 = x_2 = x_4 = x_5 = 0$ by solving the residual equations:

$$\begin{bmatrix} 1 & 1 & 0 \\ 2 & 1 & 0 \\ 4 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_3 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

or $x_3 = 1$; $x_6 = 0$; $x_7 = -1$ or $\mathbf{x} = [0, 0, 1, 0, 0, 0, -1]$ a Basic Solution. If we set $x_2 = x_4 = x_5 = x_6 = 0$ then

$$\begin{bmatrix} 1 & 1 & 0 \\ 2 & 2 & 0 \\ 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

for which the solution is $x_1 = 0, x_3 = 1, x_7 = -1$ giving the identical solution $\mathbf{x} = [0, 0, 1, 0, 0, 0, -1]$.
These Basic Solutions are said to be *degenerate* and there are a set of 4 additional degenerate Basic Solutions identical to the above.

2.9 Various Theorems for the Simplex Method.

Linear Programming problems have, by definition:

(1) a linear objective function

$$\max_{\mathbf{x}} \left[z = \mathbf{c}^T \mathbf{x} \right] \tag{14}$$

(2) a set of linear constraints

$$a_i^T x \left\{ \begin{array}{l} \geq \\ = \\ \leq \end{array} \right. b_i \quad ; \quad i = 1, 2, \ldots, m \tag{15}$$

(3) a restriction that the solution vector $\mathbf{x}$ lies in the positive quadrant

$$\mathbf{x} \geq \mathbf{0} \tag{16}$$

For this particular class of problems we can identify several general properties which are useful in solving these problems. We shall state these properties as Theorems.

Theorem 2 *If a Linear Programming problem has a set of feasible solutions then this set is a convex set of points* $\mathbf{x}$.

A *convex* set of points is a set of points surrounded by *convex boundaries*. Figure 9 is an example. In Figure 10, however, the region is not convex. Although in Figure 11 the feasible region is unbounded, it is a convex region. This is made clear by the definition that follows:

A *convex set* of points is such that for any two points x_1, x_2 in the set, all the points on the line segment joining them are also in the set. This is not true for Figure 10.
An arbitrary point x_3 on the line between x_1 and x_2 is given by

$$\mathbf{x}_3 = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \quad ; \quad 0 \leq \lambda \leq 1 \tag{17}$$

This is called a *convex linear combination* of points. If $\lambda = 0$, $\mathbf{x}_3 = \mathbf{x}_2$; if $\lambda = 1$, $\mathbf{x}_3 = \mathbf{x}_1$ and for intermediate values of $\lambda, \mathbf{x}_3$ takes other values on the line between $\mathbf{x}_1$ and $\mathbf{x}_2$.

Before we discuss the next few theorems, it is useful to define some commonly used terms:

- A *feasible solution* to an LP problem is a solution $\mathbf{x}$ which satisfies all the constraints.
- An *optimal solution* is a solution $\mathbf{x}$ which is not only feasible but also maximises (minimises) the objective function.

Consider now a LP problem whose constraints define a feasible region as shown in Figure 12. The objective function $z = c_1 x_1 + c_2 x_2$ has a gradient vector $\mathbf{c} = dz/d\mathbf{x} = [c_1, c_2]$ which is a constant vector, so that at any point in (x_1, x_2) space the direction $\mathbf{c}$ in which the objective function z increases most rapidly is always the same. If A and B are two points on the boundaries of the feasible region, then because the feasible region is a convex set we know that any point on the line segment AB is also in the feasible region. As P moves towards B, the objective function z will increase (as PB has a positive resolved part along the gradient vector $\mathbf{c}$) and we can infer that no interior point of the feasible region will ever be optimal. In addition, unless $\mathbf{c}$ is orthogonal

(at right angles) to DC, either D or C will always give a larger objective function value than an intermediate point B. Thus optimal solutions will always lie at an extreme point or corner point of the feasible region.

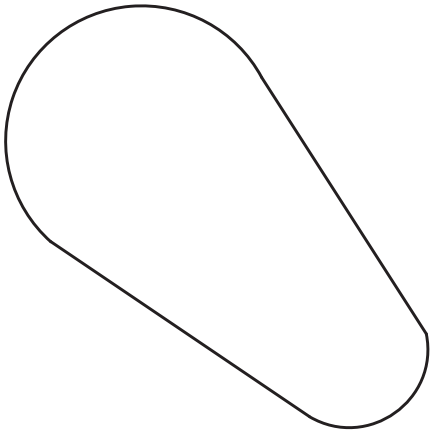

Figure 9: A *convex* region: if we draw a straight line between any two points in the region then all points on the line necessarily lie within the region or on an edge.

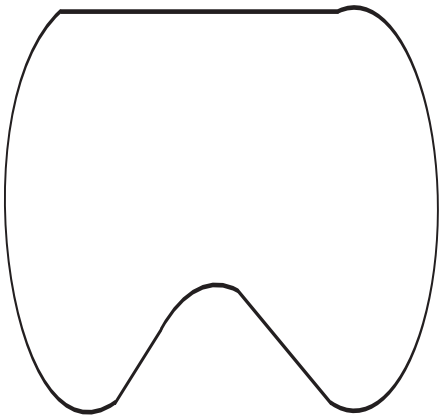

Figure 10: A *non-convex* region: it is possible to draw straight lines between two points in the region such that there are some points on the line which are outside the region.

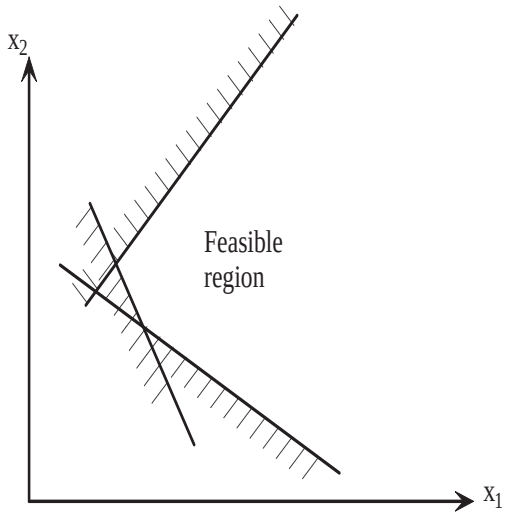

Figure 11: An open feasible region which is convex.

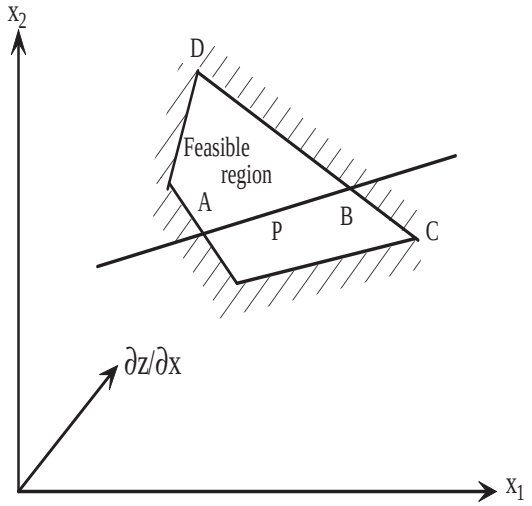

Figure 12: A closed feasible region which is convex.

In the special case when $\mathbf{c}$ is orthogonal to the constraint line DC, all points on the line DC are equally good so that any $\mathbf{x}_b$ is optimal given that

$$x_b = \lambda x_d + (1-\lambda)x_c \qquad \lambda \in [0,1] \tag{18}$$

This brings us to

Theorem 3 *The optimal solution of an LP problem, if one exists, is always at a corner point of the feasible region or, if multiple optimal solutions exist, at points defined by convex linear combination of optimal corner points.*

Equation (18) is an example of a convex linear combination: it is a way of defining points that lie in a linear space (line, plane, hyperplane) which is convex and has as its corner points, the vectors which are linearly combined. Thus a linear convex space having corner points $x_1, x_2, \ldots, x_n$ would have a set of members x such that

$$\begin{aligned} \mathbf{x} &= \lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2 + \ldots + \lambda_n \mathbf{x}_n \\ \text{and} \quad \sum_{i=1}^{n} \lambda_i &= 1 \end{aligned}$$

Theorem 3 is an expanded version of Theorem 1.

2.10 The Clifton Metal Artwork problem solved by enumeration.

In section 2.3 we reformulated the CliftonMetal Artworkk problem by introducing slack variables as:

$$\max \left[\mathbf{c}^T \mathbf{x}\right] \qquad\qquad \mathbf{c} = [1, 2]$$

subject to:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 8 & 3 & 0 & 0 & 1 & 0 \\ 4 & 9 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 7 \\ 6 \\ 60 \\ 60 \end{bmatrix} \tag{19}$$

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6]$$

In Figure 13 we have redrawn the feasible region in (x_1, x_2) space, using the constraint equations. Theorem 3 states that the optimal solution will always lie at a corner point or on the line (or linear space) between optimal corner points. This means that we can guarantee to find the optimal solution by evaluating the objective function at each of the corner points of the feasible region. The next pertinent question is to ask if there is a simple algebraic way of identifying corner points or *extreme points* as they are often called in the literature. The extreme points have been labelled (1) to (6) in Figure 13 and, in Table 1, we show that each of the extreme points corresponds to a solution to the equations (19) where two variables have been set to zero. There are 4 equations and 6 variables in (19) so that solutions obtained by setting two variables to zero are Basic Solutions. There are more Basic Solutions than the 6 extreme points but all the other Basic Solutions are infeasible either because the set of equations is inconsistent as when $x_3 = x_4 = 0$, or because some of the variables break the non-negativity constraints. Thus the extreme points of the feasible region are identical with basic feasible solutions. Theorem 3 can be restated in a more useful form:

Theorem 4 *The optimal solution of an LP problem, if one exists, is a basic feasible solution of the set of constraint equations or, if multiple optima exist, the optimal solutions are convex linear combinations of those basic feasible solutions which are themselves optimal.*

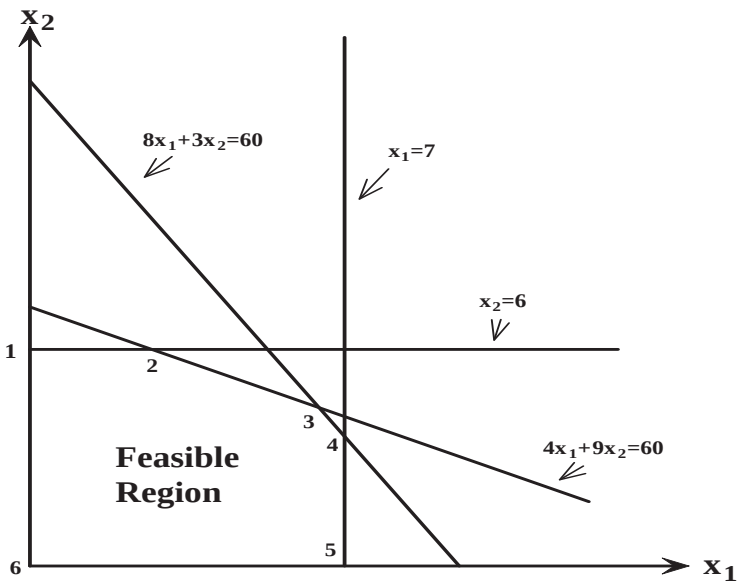

Figure 13: The constraints implied by the Clifton Metal Artwork problem.

Theorem 4 provides a basis for a simple numerical technique for solving Linear programming problems. The steps of the technique are:

(1) Convert the constraints of the problem into equations.
(2) Solve the constraint equations for all basic feasible solutions.
(3) Evaluate the objective function corresponding to each basic feasible solution and thus find the optimal solution. The optimal solution will either be one optimal basic feasible solution (BFS) or if more than one BFS has the same optimal value of the objective function, then any convex linear combination of the optimal BFSs will be optimal.

In Table 1 we have followed this technique and can identify extreme point 3 as the unique optimal solution. A better technique (the Simplex Method) is to start at one basic feasible solution, for example extreme point 1 in Table 1, and then move to adjacent basic feasible solutions in such a manner as to always improve or maintain the value of the objective function. The sequence of extreme points 1,2,3 are all adjacent, each to the next, in the sense that one new variable enters the basis (set of basic variables) and one variable leaves and becomes a non-basic variable at each stage in the sequence. For this scheme to be a successful optimisation algorithm we need to be able to:
(a) test the effect on the objective function of introducing a particular variable into the basis
(b) be sure that the algorithm will find the global optimum.

Consider the requirement (a). If we have an objective function

$$z = c_1x_1 + c_2x_2 + \ldots + c_nx_n = \mathbf{c}^T\mathbf{x}$$

then the partial derivative $\partial z/\partial x_i|_x$ shows us whether the objective function increases as x_i increases at the point $\mathbf{x}$.

Extreme point	Variables set to zero	x_1	x_2	z
	(Non-Basic Variables)			
1	x_1, x_4	0	6	12
2	x_4, x_6	1.5	6	13.5
3	x_6, x_5	6	4	14
4	x_5, x_3	7	4/3	$9\frac{2}{3}$
5	x_3, x_2	7	0	7
6	x_2, x_1	0	0	0

Table 1: Extreme Points of the Feasible Space in Figure 13

When a variable x_i is introduced into the basis it changes from being defined as zero to some value $x_i \geq 0$. Thus if $\partial z/\partial x_i|_{\mathbf{x}} \geq 0$ and $\mathbf{x}$ is a basic solution, then introducing x_i into the basis will increase (or maintain) the value of z. When $\partial z/\partial x_i|_{\mathbf{x}^*} \leq 0$ for all x_i then x^* is a local optimum - and, as we shall prove, a global optimum.
Suppose A is a possible local maximum of the objective function and B is a supposed global maximum. Linear Programming problems have convex feasible regions so any point T on the line joining A and B is also a feasible solution. If $\mathbf{x}_T = t\mathbf{x}_A + (1-t)\mathbf{x}_B$, $t \in: [0,1]$ and the value of the objective function z is z_A at A and z_B at B then

$$z_T = tz_A + (1-t)z_B$$

as z is a linear function of $\mathbf{x}$. Now we postulated that B was the global maximum, so:

$$z_T = tz_A + (1-t)z_B > z_A$$

and we can take T arbitrarily close to A which implies that A is not a local maximum. This contradiction allows us to state:

Theorem 5 *In a Linear Programming problem a Basic Feasible Solution that is a local optimum is the global optimum over the whole feasible region.*

We will now describe the Simplex Method which relies upon these results to find optimal solutions to an LP problem.

2.11 The Simplex Method.

The Simplex Method for solving Linear Programming problems was devised by Dantzig and first described in 1951. Because of the economic importance of the method considerable effort has gone into producing modifications to the original algorithm which improve its efficiency for large problems and to make other extensions. Here we will only be concerned with the basic algorithm. Solving a problem using the Simplex Method involves the following stages:

1. Formulate the problem as a Linear Programming problem.
2. Convert the formulation into the standard (canonical) form

$$\max_{\mathbf{x}} \left[\mathbf{c}^T\mathbf{x}\right] \qquad \text{or} \qquad \min_{\mathbf{x}} \left[\mathbf{c}^T\mathbf{x}\right]$$

subject to

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

3. Find an initial Basic feasible Solution.
4. Find a variable which when introduced into the basis will improve the objective function.
5. Evaluate the new Basic Feasible Solution.
6. Repeat **4** and **5** until no improvement can be found.

In this section we will take the Clifton Metal Artwork problem for which stages **1** and **2** have been completed and we will apply the Simplex Method to obtain the solution.

The standard formulation for the Clifton Metal Artwork problem is (choosing some appropriate coefficients in the objective function):

$$\max[z = x_1 + 2x_2]$$

subject to

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 8 & 3 & 0 & 0 & 1 & 0 \\ 4 & 9 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 7 \\ 6 \\ 60 \\ 60 \end{bmatrix}$$

$$\mathbf{x} > \mathbf{0}$$

It is convenient to do the calculations required for the Simplex Method in a Tableau and we shall display the calculations by writing the constraint equations $\mathbf{Ax} = \mathbf{b}$ with an equation for the objective function $-c^T x + z = 0$ added below, so:

x, z	
$(A, 0)$	b
$-c^T, 1$	0

When the Clifton Metal Artwork problem is displayed in this way we obtain the Tableau as illustrated in Table 2. Each row in the Tableau corresponds to an equation with the bottom row, the objective function row, being an equation for the value of the objective function z, in terms of the variables. When a Tableau has a unit vector corresponding to each row, as this one does, it is possible to read off a Basic feasible solution from the solution column. Thus in this Tableau we can read off the basic solution corresponding to the variables x_1 and x_2 set to zero; the basis is $[x_3, x_4, x_5, x_6, z] = [7, 6, 60, 60, 0]$.
This completes stage 3 and now we choose a variable to bring into the basis which will improve the objective function as much as possible. The equation in the objective function row may be rewritten:

$$z = x_1 + 2x_2$$

so that

x_1	x_2	x_3	x_4	x_5	x_6	z	Solution Column
1	0	1	0	0	0	0	7
0	1	0	1	0	0	0	6
8	3	0	0	1	0	0	60
4	9	0	0	0	1	0	60
-1	-2	0	0	0	0	1	0

Table 2: Initial Tableau for the Clifton Metal Artwork problem

$$\frac{\partial z}{\partial x_1} = 1 \qquad\qquad \frac{\partial z}{\partial x_2} = 2$$

We are maximising z and locally the rate of increase of z with x_2 is largest so we choose to introduce x_2 into the basis.
In terms of the Simplex Tableau we can generalise this to a rule:

Rule 1

When maximising introduce the variable whose coefficient in the objective function row is the largest negative number. When all coefficients are non-negative then an optimum solution has been found. In the Tableau we ring the column corresponding to x_2 to show that this variable is to be introduced into the basis. Now (stage **5**) we need to evaluate the new Basic Feasible Solution. As x_2 is being introduced to the basis one other variable must leave. This corresponds to moving from one extreme point of the feasible region to an adjacent extreme point by increasing x_2 from its present zero value. Since we wish to move to an extreme point, we increase x_2 as much as possible within the constraints. From the Tableau we can rewrite the constraints as equations for x_2 thus:

$$\begin{aligned} x_2 &= 6 - x_4 \\ 3x_2 &= 60 - x_5 - 8x_1 \\ 9x_2 &= 60 - x_6 - 4x_1 \end{aligned}$$

All the variables are non-negative so we can rewrite these equations as inequalities:

$$\begin{aligned} x_2 &\leq 6 \\ x_2 &\leq \frac{60}{3} \\ x_2 &\leq \frac{60}{9} \end{aligned}$$

The harshest constraint is $x_2 \leq 6$ so we increase x_2 to 6. The new basic feasible solution in which x_1 is still non-basic and $x_2 = 6$ may be obtained by substituting into the Tableau thus:

$x_3 = 7$; $x_4 = 0$ i.e. x_4 has left the basis.
$x_5 = 42$; $x_6 = 6$; $z = 12$ (an improvement).

All the operations described above are performed in a tableau format as follows:

Simplex algorithm for maximising a function.

(1) The column containing the largest negative element of the objective function row is circled.

(2) Each element of the solution column is divided by the corresponding element in the column circled in **(1)** and the result placed in a column headed *Ratio*. The row in which the smallest non-negative ratio occurs is then circled.

(3) The third stage involves creating a new Tableau which looks similar to the previous Tableau but has the new basis variables with unit vectors under them, so that we can read off the values of the basis variables directly from the solution column.
In order to get a *unit vector* (inclusive of the objective function row) under the new basis variable (in the ringed column) we:

- divide the ringed row by the element in both the ringed row and the ringed column (called the *pivot*).
- add or subtract the required multiple of this new row from each of the other rows (equations), so that all elements in the ringed column, except the pivot, are zero (inclusive of the objective function row). For example to remove the 3 in the ringed column we subtract three times the ringed row from the row containing the 3.

(4) If there are no negative elements in the objective function row then **stop**, else go to step **(1)**.

Example 1: Consider maximisation of the objective function:

$$z = 60x_1 + 90x_2 + 300x_3$$

subject to the constraints:

$$\begin{aligned} x_1 + x_2 + x_3 &\leq 600 \\ x_1 + 3x_2 &\leq 600 \\ 2x_1 + x_3 &\leq 900 \end{aligned}$$

1. Thus our starting Tableau is:

x_1	x_2	x_3	s_1	s_2	s_3	z	Solution Column
1	1	1	1	0	0	0	600
1	3	0	0	1	0	0	600
2	0	1	0	0	1	0	900
-60	-90	-300	0	0	0	1	0

The top row identifies the variables. s_1, s_2 and s_3 are slack variables. The bottom row comes from setting the equation: $z = 60x_1 + 90x_2 + 300x_3$ to 0, thus: $-60x_1 - 90x_2 - 300x_3 + z = 0$.

2. Choose the pivot column (given here in boldface):

x_1	x_2	$\mathbf{x}_3$	s_1	s_2	s_3	z	Solution Column
1	1	**1**	1	0	0	0	600
1	3	**0**	0	1	0	0	600
2	0	**1**	0	0	1	0	900
-60	-90	$\mathbf{-300}$	0	0	0	1	0

The pivot column is the column with the most negative entry in the objective function row.

3. Choose the pivot row:

x_1	x_2	$\mathbf{x}_3$	s_1	s_2	s_3	z	Solution Column	Ratio
1	**1**	$\langle\mathbf{1}\rangle$	**1**	**0**	**0**	**0**	**600**	$\frac{600}{1}=\mathbf{600}$
1	3	**0**	0	1	0	0	600	$\frac{600}{0}=\infty$
2	0	**1**	0	0	1	0	900	$\frac{900}{1}=900$
-60	-90	$\mathbf{-300}$	0	0	0	1	0	$-$

We divide the solution column by the entries in the pivot column. *The pivot row is the row with the least non-negative ratio (this can include zero).* Negative values or undefined values are ignored. The pivot row is in bold type and the *pivot* is indicated by $< \ldots >$ here.

4. We now perform the following row manipulations to create a unit vector in the pivot column:

x_1	x_2	$\mathbf{x}_3$	s_1	s_2	s_3	z	Solution Column
1	1	**1**	1	0	0	0	600
1	3	**0**	0	1	0	0	600
1	-1	**0**	-1	0	1	0	300
-60	-90	$\mathbf{-300}$	0	0	0	1	0

We have obtained a 0 in the third row by mutiplying row1 by -1 and adding it to row3. Similarly for the bottom (objective function row):

x_1	x_2	$\mathbf{x}_3$	s_1	s_2	s_3	z	Solution Column
1	1	**1**	1	0	0	0	600
1	3	**0**	0	1	0	0	600
1	-1	**0**	-1	0	1	0	300
240	210	**0**	300	0	0	1	180,000

We have multiplied row1 by 300 and added it to row4 (the objective function row). Pivoting is now complete. We could have done the above two steps in one go but it is usually advisable to break down the calculations, step-by-step, to avoid numerical errors.

5. There are *no more negative elements* in the bottom row so we have finished with pivoting and we can read of the solution:

$$\begin{aligned} x_1 &= 0 \\ x_2 &= 0 \\ x_3 &= 600 \\ s_1 &= 0 \\ s_2 &= 600 \\ s_3 &= 300 \\ z &= 180,000 \end{aligned}$$

To do so we proceed through the variables in the top row $(x_1, x_2, \ldots, s_3)$ and when we find the column is not a unit vector we set the corresponding variable to zero. Thus:

$$\begin{aligned} x_1 &= 0 \\ x_2 &= 0 \end{aligned}$$

If the column is a unit vector then we look down the column and locate the 1, we then read across the row and set this variable equal to the corresponding element in the solution column (whose numbers have changed during the row manipulations process). Thus for x_3 we get:

$$x_3 = 600$$

We proceed through the slack variables to determine these too in the same fashion.

Being rigorous engineers we should then check our solution satisfies the constraints and that the objective function value is correct (in the lower right-hand side of the Final Tableau), thus:

$$\begin{aligned} 0 + 0 + 600 &\leq 600 \\ 0 + 3(0) &\leq 600 \\ 2(0) + 600 &\leq 900 \end{aligned}$$

and $180,000 = 60(0) + 90(0) + 300(600)$.

Example 2. For the Clifton Metal Artwork problem mentioned earlier three pivot operations are required:

1. The Initial Tableau is (the pivot is marked by $< \ldots >$):

x_1	x_2	s_1	s_2	s_3	s_4	Solution Column	Ratio
1	0	1	0	0	0	7	$\frac{7}{0} = \infty$
0	$\langle 1 \rangle$	0	1	0	0	6	6
8	3	0	0	1	0	60	20
4	9	0	0	0	1	60	$\frac{60}{9} = 6.67$
-1	-2	0	0	0	0	0	$-$

2. After the first pivot we get:

x_1	x_2	s_1	s_2	s_3	s_4	Solution Column	Ratio
1	0	1	0	0	0	7	7
0	1	0	1	0	0	6	$\frac{6}{0} = \infty$
8	0	0	-3	1	0	42	$\frac{42}{8} = 5.25$
$\langle 4 \rangle$	0	0	-9	0	1	6	$\frac{6}{4} = 1.5$
-1	0	0	2	0	0	12	$-$

3. After the second pivot we get:

x_1	x_2	s_1	s_2	s_3	s_4	Solution Column	Ratio
0	0	1	$\frac{9}{4}$	0	$-\frac{1}{4}$	5.5	2.44
0	1	0	1	0	0	6	6
0	0	0	$\langle 15 \rangle$	1	-2	30	2
1	0	0	$-\frac{9}{4}$	0	$\frac{1}{4}$	1.5	-0.66
0	0	0	$-\frac{1}{4}$	0	$\frac{1}{4}$	13.5	$-$

4. After the third pivot we get:

x_1	x_2	s_1	s_2	s_3	s_4	Solution Column
0	0	1	0	$-\frac{3}{20}$	$\frac{1}{20}$	1
0	1	0	0	$-\frac{1}{15}$	$\frac{2}{15}$	4
0	0	0	1	$\frac{1}{15}$	$-\frac{2}{15}$	2
1	0	0	0	$\frac{3}{20}$	$-\frac{1}{20}$	6
0	0	0	0	$\frac{1}{60}$	$\frac{13}{60}$	14

so the solution is:

$$\begin{aligned} x_1 &= 6 \\ x_2 &= 4 \\ z &= 14 \end{aligned}$$

2.12 Comments on the Simplex Method

2.12.1 Standard Form.

We have described the Simplex Method as follows:

$$\max_{\mathbf{x}} \left[\mathbf{c}^T \mathbf{x} \right]$$

subject to

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

This is called *standard form*.

2.12.2 Pivoting on a zero ratio

It is possible to pivot of a ratio of zero but the zero must be 'positive'. Suppose we divide 0 by 4 then this is 'positive' which we could write as 0+. If we divided 0 by -4 we would call this a 'negative zero', which we could write 0−. So pivoting on 0+ is fine and would be the best choice but we do not pivot on 0−. If two ratios are the same then we could pivot on either: we do not have the information about which choice is best.

2.12.3 Minimising a function.

For the Simplex Method for Maximisation we note that we place the coefficients of the objective function (written as $z - c_1x_1 - c_2x_2 - \ldots = 0$) in the bottom row. We then pivot on the column with the most negative element. This is because, according to the gradients derived from $\partial z/\partial x_i$:

- a negative entry in the objective function row will take the solution towards the maximum. Choosing the most negative element gives the most rapid approach to the maximum.
- a zero entry in the objective function row means the gradient is zero: if under a non-basic variable variable it would mean we can introduce that variable into the basis without changing the value of z. This is the situation we encounter when multiple optima are present.
- a positive entry in the objective function row means that incorporating this variable into the basis would take the solution towards the minimum.

Hence to minimise a function we can use the Simplex Method as before with one change. We set up the initial tableau as before *but* we select *the most positive element* in the objective function row and pivot on this column. We stop the pivoting procedure when all entries in the objective function row *are zero or negative.*

2.12.4 Surplus variables

We have already seen how inequalities of the form:

$$\mathbf{a}^T\mathbf{x} \leq b_1$$

can be converted into equations by adding a *slack variable* s_1

$$\mathbf{a}^T\mathbf{x} + s_1 = b_1$$

The slack variable must be constrained to be non-negative for these equations to hold. In a similar way we can convert a $\geq$ inequality into an equation by subtracting a *surplus variable* i.e.

$$\mathbf{a}^T\mathbf{x} \geq b_2$$

is equivalent to:

$$\mathbf{a}^T\mathbf{x} - s_2 = b_2 \qquad\qquad s_2 \geq 0$$

The surplus variable s_2 is well named because if $\mathbf{a}^T\mathbf{x}$ is greater than b_2 then s_2 gives the size of the surplus.

2.12.5 Unbounded Variables

One of the requirements of the Simplex Method is that *all variables must be non-negative.* In most problems the variables are naturally non-negative but this is not always so. Sometimes we may know the lower bound of a variable - for instance if t is a temperature measured in ^oC then $t > -274$ so we can change the variable to $T = t + 274$ and be confidant that T is non-negative.
In the event that no lower bound is known we can represent a variable which is unconstrained in sign by the difference of two non-negative variables. Thus the problem:

$$\max [x_1 + 3x_2]$$

subject to

$$\begin{array}{rcl} x_1 + 2x_2 & \leq & 5 \\ x_1 - 3x_2 & \geq & 20 \\ x_1 & \geq & 0 \\ & & x_2 \quad \text{unconstrained} \end{array}$$

can be converted by a change of variable $x_2 = y - z$, to

$$\max [x_1 + 3y - 3z]$$

subject to:

$$\begin{array}{rcl} x_1 + 2y - 2z & \leq & 5 \\ x_1 - 3y - 3z & \geq & 20 \\ x_1, y, z & \geq & 0 \end{array}$$

As the Simplex Method always finds Basic Solutions either y or z will always be zero with the other variable taking the value of $|x_2|$. We can convert to standard form by introducing slack and surplus variables so the problem becomes:

$$\max [x_1 + 3y - 3z]$$

subject to

$$\begin{array}{rcl} x_1 + 2y - 2z + s_1 & = & 5 \\ x_1 - 3y - 3z - s_2 & = & 20 \\ x_1, y, z, s_1, s_2 & \geq & 0 \end{array}$$

or, in matrix notation:

$$\max \left[c^T q\right]$$

$$c = [1, 3, -3, 0, 0]$$

subject to

$$\begin{bmatrix} 1 & 2 & -3 & 1 & 0 \\ 1 & -3 & -3 & 0 & -1 \end{bmatrix} \mathbf{q} = \begin{bmatrix} 5 \\ 20 \end{bmatrix}$$

$$\mathbf{q} \geq \mathbf{0}$$

If a problem has several variables which are unconstrained in sign it is not necessary to introduce a new variable for each one but we may define $x_j = y_i - z$ and use the same z for each x_j. z will then take the absolute value of the most negative variable and y_i will be the amount by which x_j exceeds this most negative variable.

2.13 Finding an initial basic feasible solution.

The Simplex Method is an algorithm which generates new (usually better) basic feasible solutions to a problem given a basic feasible solution from which to start. The addition of slack variables to the constraint equations created an initial basic feasible solution. In the example that follows we show how other types of problem can be modified so as to obtain an initial basic feasible solution.

For a Tableau to correspond to a basic feasible solution:

(1) *the numbers in the solution column except the one in the objective function row must be positive or zero.* This corresponds to the non-negativity constraint.

(2) *there must be enough unit vectors in columns of the Tableau to form an identity matrix.*

Example:

$$\min [2x_1 + x_2]$$

subject to:

$$\begin{aligned} 3x_1 + x_2 &= 3 \\ 4x_1 + 3x_2 &\geq 6 \\ -x_1 - 2x_2 &\geq -3 \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

As a first step we turn all the inequalities into equations by inserting surplus variables s_1 and s_2 obtaining:

$$\begin{aligned} 3x_1 + x_2 &= 3 \\ 4x_1 + 3x_2 - s_1 &= 6 \\ -x_1 - 2x_2 - s_2 &= -3 \end{aligned}$$

The initial tableau is:

x_1	x_2	s_1	s_2	Solution Column
3	1	0	0	3
4	3	-1	0	6
-1	-2	0	-1	-3
-2	-1	0	0	0

We cannot read off a feasible basic solution. We can multiply the third row to get:

x_1	x_2	s_1	s_2	Solution Column
3	1	0	0	3
4	3	-1	0	6
1	2	0	1	3
-2	-1	0	0	0

but we still get $s_1 = -6$. Thus we introduce *artificial variables* to create a feasible basic solution:

$$\begin{aligned} 3x_1 + x_2 + a_1 &= 3 \\ 4x_1 + 3x_2 - s_1 + a_2 &= 6 \\ x_1 + 2x_2 + s_2 &= 3 \end{aligned}$$

Since the initial tableau becomes:

x_1	x_2	s_1	s_2	a_1	a_2	Solution Column
$\langle 3 \rangle$	1	0	0	1	0	3
4	3	-1	0	0	1	6
1	2	0	1	0	0	3
-2	-1	0	0	0	0	0

with a basic solution $x_1 = 0$, $x_2 = 0$, $s_1 = 0$, $s_2 = 3$, $a_1 = 3$ and $a_2 = 6$. We introduce artificial variables to create an identity matrix in the part of the tableau corresponding to the contraint equation rows (excluding the solution column). We can introduce any number of artificial variables, an excess doesn't matter since they will be eliminated from the eventual solution, all that matters is that an Basic Feasible Solution is created.
Introducing artificial variables into the problem will not change the solution of the problem provided we can be certain that in the optimal solution all the artificial variables are zero i.e. non-basic variables. The objective function is $S = 2x_1 + x_2$ and we are minimising. If we introduce costs for the artificial variables into the objective function e.g.

$$S = 2x_1 + x_2 + 10a_1 + 10a_2$$

then the optimal solution will tend to a solution in which both the artificial variables are zero. The costs of the artificial variables have been chosen to be 10 to make hand-done arithmetic easier. A cost of 1 would have been easier but this is not large compared with the cost for x_1 of 2. In the Table below the modified objective function has been inserted and then in order to get back to a basic solution ten times both of rows (a) and (b) have been added to the objective function row and the result (e) shown below the objective function row: we had to create this new objective function row because, for minimisation, we had to have positive entries in the objective function row to start the calculation rolling. However, adding and substracting constraint equations to each other or to the objective function equation is always OK:

x_1	x_2	s_1	s_2	a_1	a_2	Soln col.	
3	1	0	0	1	0	3	(a)
4	3	-1	0	0	1	6	(b)
1	2	0	1	0	0	3	(c)
-2	-1	0	0	-10	-10	0	(d)
68	39	-10	0	0	0	90	(e)

This Table, excluding row (d), now represents an initial Tableau for the Simplex method and the initial basic feasible solution can be read off as

$$\mathbf{x} = [x_1, x_2, x_3, x_4, a_1, a_2, S] = [0, 0, 0, 3, 3, 6, 90].$$

We introduce x_1 into the basis obtaining the Tableau in the Table below:

x_1	x_2	x_3	x_4	a_1	a_2	Soln col.	
1	$\frac{1}{3}$	0	0	$\frac{1}{3}$	0	1	(a)
0	$1\frac{2}{3}$	-1	0	$-\frac{4}{3}$	1	2	(b)
0	$1\frac{2}{3}$	0	1	$-\frac{1}{3}$	0	2	(c)
0	$16\frac{1}{3}$	-10	0	$-\frac{68}{3}$	0	22	

The artificial variable a_1 has left the basis and can be left out of future Tableau. The value of the objective function has been reduced to 22. In the next iteration we introduce x_2 into the basis. Both constraints (b) and (c) require the same maximum value of x_2 and either could be used. (b) has the advantage that a_2 will leave the basis on the next iteration as shown in the Table below:

x_1	x_2	x_3	x_4	a_2	Soln. col.
1	0	$\frac{1}{5}$	0	$-\frac{1}{5}$	$\frac{3}{5}$
0	1	$-\frac{3}{5}$	0	$\frac{3}{5}$	$\frac{6}{5}$
0	0	1	1	-1	0
0	0	$-\frac{1}{5}$	0	$-\frac{49}{5}$	$\frac{12}{5}$

This is the final and optimal Tableau as all coefficients in the objective function row are now *negative* (for a minimisation problem). The optimal solution to the original problem is thus $x_1 = \frac{3}{5}$, $x_2 = \frac{6}{5}$ and the value of the objective function is $\frac{12}{5}$.

The artificial variable approach described above creates a new problem, by introducing extra variables, whose solution set contains the solution to the original problem. If there is no solution to the original problem then it will prove impossible to get rid of the artificial variables. In a computer implementation of the method described above the objective function is written as

$$\text{New objective function} = \text{old objective function} + \text{M}\times(\text{sum of artificial variables})$$

where M is taken as a large number e.g. 999,999,999 (we use $-M(sum)$ for maximisation and $+M(sum)$ for minimisation) in computer implementations (in calculations we may use a free-floating M). This method is called the *Big M Method* in most textbooks.
The large size of M makes sure that the artificial variables leave the basis if at all possible. However, as noted above, we may have to do some manipulations with constraints to set up a reasonable starting point for the new objective function row. This is standard practice: *we use the constraint equations with artificial variables to eliminate all artificial variables from the objective function and use this new objective function for our calcultions.* Thus, in our case:

$$z = 2x_1 + x_2 + M\,(a_1 + a_2)$$

We can write:

$$\begin{aligned} a_1 &= 3 - 3x_1 - x_2 \\ a_2 &= 6 - 4x_1 - 3x_2 + s_1 \end{aligned}$$

so:

$$z = x_1\,(2 - 7M) + x_2\,(1 - 4M) + 9M + Ms_1$$

and the initial tableau is:

x_1	x_2	s_1	s_2	a_1	a_2	Solution Column
$\langle 3 \rangle$	1	0	0	1	0	3
4	3	-1	0	0	1	6
1	2	0	1	0	0	3
$(7M - 2)$	$(4M - 1)$	$-M$	0	0	0	$9M$

We pivot on the first column because it is the most positive for large M, thus:

x_1	x_2	s_1	s_2	a_1	a_2	Solution Column
1	$\frac{1}{3}$	0	0	$\frac{1}{3}$	0	1
0	$\left\langle \frac{5}{3} \right\rangle$	-1	0	$-\frac{4}{3}$	1	2
0	$\frac{5}{3}$	0	1	$-\frac{1}{3}$	0	2
0	$\frac{5}{3}M - \frac{1}{3}$	$-M$	0	$\frac{1}{3}\,(2 - 7M)$	0	$2 + 2M$

We then pivot on the second column, to get:

x_1	x_2	s_1	s_2	a_1	a_2	Solution Column
1	0	$\frac{1}{5}$	0	$\frac{3}{5}$	$-\frac{1}{5}$	$\frac{3}{5}$
0	1	$-\frac{3}{5}$	0	$-\frac{4}{5}$	$\frac{3}{5}$	$\frac{6}{5}$
0	0	1	1	1	-1	0
0	0	$-\frac{1}{5}$	0	$\left(\frac{2}{5} - M\right)$	$\left(\frac{1}{3} - \frac{5}{3}M\right)$	$\left(\frac{12}{5}\right)$

which only has negative entries on the objective function row (hence we terminate for minimisation). Thus the solution is:

$$\begin{aligned} x_1 &= \frac{3}{5} \\ x_2 &= \frac{6}{5} \\ s_1 &= 0 \\ s_2 &= 0 \\ a_1 &= 0 \\ a_2 &= 0 \end{aligned}$$

and the value of the objective function is 12/5.

2.14 The Two-Phase Method (non-examinable)

With the use of artificial variables, the two-phase method is designed to eliminate the big-M in the Big M method from the problem. There are two phases: Phase 1 attempts to find a starting basic feasible solution. If such a solution can be found, Phase 2 uses it to solve the original problem.

Phase 1. Put LP problem in standard form, add artifical variables to get a starting basic solution. Next, find a basic solution that minimizes the sum of the artificial variables. If the minimum value of the sum is positive, the LP problem has no feasible solution, otherwise go to Phase 2.

Phase 2. Use the feasible solution obtained in Phase 1 as a starting basic solution for the original problem.

Example: suppose we have the LP problem:

$$\begin{aligned} \min z &= 4x_1 + x_2 \\ 3x_1 + x_2 &= 3 \\ 4x_1 + 3x_2 &\geq 6 \\ x_1 + 2x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Phase 1. Minimization of the sum of the artificial variables gives the following problem:

$$\min r = a_1 + a_2$$

subject to:

$$\begin{aligned} 3x_1 + x_2 + a_1 &= 3 \\ 4x_1 + 3x_2 - s_1 + a_2 &= 6 \\ x_1 + 2x_2 + s_2 &= 4 \\ x_1, x_{2,} s_1, s_2, a_1, a_2 &\geq 0 \end{aligned}$$

The associated tableau is then:

Basic	x_1	x_2	s_1	a_1	a_2	s_2	Solution
a_1	3	1	0	1	0	0	3
a_2	4	3	-1	0	1	0	6
s_2	1	2	0	0	0	1	4
r	0	0	0	-1	-1	0	0

As in the Big M method, a_1 and a_2 are substituted out in the r-row by using the following computations:

Old r-row	(	0	0	0	-1	-1	0	—	0)
$+1 \times a_1$-row	(	3	1	0	1	0	0	—	3)
$+1 \times a_2$-row	(	4	3	-1	0	1	0	—	6)
=New r-row	(	7	4	-1	0	0	0	—	9)

The new r-row is:

$$r + 7x_1 + 4x_2 - x_3 + 0a1 + 0a_2 + 0s_2 = 9$$

which is used to solve Phase 1 of the problem, yielding the following optimal tableau:

Basic	x_1	x_2	s_1	a_1	a_2	s_2	Solution
x_1	1	0	$\frac{1}{5}$	$\frac{3}{5}$	$-\frac{1}{5}$	0	$\frac{3}{5}$
x_2	0	1	$-\frac{3}{5}$	$-\frac{4}{5}$	$\frac{3}{5}$	0	$\frac{6}{5}$
s_2	0	0	1	1	-1	1	1
r	0	0	0	-1	-1	0	0

Because $r = 0$, Phase 1 produces the basic feasible solution $x_1 = \frac{3}{5}$, $x_2 = \frac{6}{5}$ and $s_2 = 1$. The artificial variables have now completed their use, we eliminate their columns from the tableau and move to Phase 2. Basically the point of Phase 1 is to provide a basic feasible solution for the problem.

Phase 2. After deleting the artificial variable columns, the original problem is written as:

$$\begin{aligned} \min z &= 4x_1 + x_2 \\ x_1 + \frac{1}{5}s_1 &= \frac{3}{5} \\ x_2 - \frac{3}{5}s_1 &= \frac{6}{5} \\ s_1 + s_2 &= 1 \\ x_1, x_2, s_1, s_2 &\geq 0 \end{aligned}$$

and the corresponding tableau is:

Basic	x_1	x_2	s_1	s_2	Solution
x_1	1	0	$\frac{1}{5}$	0	$\frac{3}{5}$
x_2	0	1	$-\frac{3}{5}$	0	$\frac{6}{5}$
s_2	0	0	1	1	1
z	-4	-1	0	0	0

Because the basic variables x_1 and x_2 have nonzero coefficients in the z-row, they must be substituted out. The computation for modifying the z-row is:

$$\begin{array}{lccccccc} \text{Old } z\text{-row} & (& -4 & -1 & 0 & 0 & — & 0 &) \\ +4 \times x_1\text{-row} & (& 4 & 0 & \frac{4}{5} & 0 & — & \frac{12}{5} &) \\ +1 \times x_2\text{-row} & (& 0 & 1 & -\frac{3}{5} & 0 & — & \frac{6}{5} &) \\ =\text{New } z\text{-row} & (& 0 & 0 & \frac{1}{5} & 0 & — & \frac{18}{5} &) \end{array}$$

The initial tableau for Phase 2 is thus:

Basic	x_1	x_2	s_1	s_2	Solution
x_1	1	0	$\frac{1}{5}$	0	$\frac{3}{5}$
x_2	0	1	$-\frac{3}{5}$	0	$\frac{6}{5}$
s_2	0	0	1	1	1
z	0	0	$\frac{1}{5}$	0	$\frac{18}{5}$

Since we are minimizing, s_1 must enter the basis. Further application of the simplex method then gives the answer.

2.15 Multiple Optima in the Simplex Method

Earlier we saw that an optimal solution of a Linear Programming problem is always at an extreme point of the feasible region and noted that, if the gradient vector of the objective function is orthogonal to a bounding plane or hyperplane, then any point in that bounding surface will have the same value of the objective function. If this bounding hyperplane has several optimal basic feasible solutions as corner points, say x_1, $x_2, \ldots, x_n$, then any

$$x = \lambda_1 x_1 + \lambda_2 x_2 + \ldots \lambda_n x_n \tag{20}$$

such that $\lambda_1 + \lambda_2 + \ldots \lambda_n = 1$, is also an optimal solution. While this idea is comprehensible from the geometry of the situation, LP problems are solved by the Simplex Method and so we need to find such multiple solutions via the Simplex algorithm. The following simple example illustrates the method.

Example:

$$\max [x_1 + 2x_2]$$

subject to

$$\begin{aligned} x_1 + 2x_2 + x_3 &\geq 2 \\ x_1 + 4x_2 + 2x_3 &\leq 5 \\ x_2 + x_3 &= 1 \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

Inserting slack and surplus variables turns the constraints into:

$$\begin{aligned} x_1 + 2x_2 + x_3 - s_1 &= 2 \qquad (2) \\ x_1 + 4x_2 + 2x_3 + s_2 &= 5 \qquad (3) \\ x_2 + x_3 &= 1 \qquad (4) \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

both the first and last equation need an artificial variable to obtain an initial Basic feasible solution as shown in Tableau 1. We are using the big M method with the choice $M = 10$ to show actual numbers (the explains the bottom row in Tableau 1). The next Tableau 2 is obtained by introducing x_2 into the basis. The method proceeds until we eventually get Tableau 4 (all entries are positive or zero).

Tableau 1

x_1	x_2	x_3	s_1	s_2	a_1	a_2	Soln. col.	
1	2	1	−1	0	1	0	2	(a)
1	4	2	0	1	0	0	5	(b)
0	1	1	0	0	0	1	1	(c)
−1	−2	0	0	0	10	10	0	(d)
−11	−32	−20	10	0	0	0	−30	(d)-10(a)-10(c)

Tableau 2

x_1	x_2	x_3	s_1	s_2	a_1	a_2	Soln. col.
1	0	-1	-1	0	1	-2	0
1	0	-2	0	1	0	-4	1
0	1	1	0	0	0	1	1
-11	0	12	10	0	0	32	2

Tableau 3

x_1	x_2	x_3	s_1	s_2	a_1	Soln. col.
1	0	-1	-1	0	1	0
0	0	-1	1	1	-1	1
0	1	1	0	0	0	1
0	0	1	-1	0	11	2

Tableau 4

x_1	x_2	x_3	s_1	s_2	Soln. col.
1	0	-2	0	1	1
0	0	-1	1	1	1
0	1	1	0	0	1
0	0	0	0	1	3

Tableau 5

x_1	x_2	x_3	s_1	s_2	Soln. col.
1	2	0	0	1	3
0	1	0	1	1	2
0	1	1	0	0	1
0	0	0	0	1	3

Tableau 4 is known to be optimal as none of the coefficients in the objective function row are negative. However the coefficient under the non-basic variable x_3 is zero. This indicates that x_3 can be introduced into the Basis with no change in the objective function: the resulting tableau is Tableau 5.

This gives us a second optimal basic solution and any convex linear combination of optimal basic solutions is also optimal. Thus the general solution to this problem is:

$$x = [x_1, x_2, x_3] = \lambda[1, 1, 0] + (1 - \lambda)[3, 0, 1] \qquad 0 \leq \lambda \leq 1$$

in terms of the original problem variables and the optimal value of the objective function is 3.
The example we have just completed illustrates the way in which multiple optima appear in the Simplex Method. When an optimal basic solution has been found: *if a non-basic variable has a zero coefficient in the objective function that variable may be introduced into the basis to form another optimal basic solution.* There may be more than two optimal basic solutions but the method is the same irrespective of the number of optimal basic solutions.

2.16 Duality

In this section we introduce the concept of *duality*. Duality is an interesting concept which applies to geometry, nonlinear optimisation and other areas, in addition to linear programming. In short, for every stated linear programming problem there is a dual problem which, if solved, gives the solution to the original problem (called the *primal problem*). As a practical application from my own research I'm interested in constructing an optimal classifier (machine learning) for classifying cancer patients into 'predict relapse' and 'predict no-relapse'. The data for the classifier comes from about 30 patients with about 30,000 gene measurements. The 'primal' problem

would involve 30,000 variables for all the gene measurements and, because of the size of this number, the task of constructing the classifier is impractical. Fortunately, I can formulate the dual problem and this problem only involves 30 variables for the 30 patients: this is a practical proposition and I can build the classifier without a problem. In Geometry the following are dual statements:

- Two points determine a line
- Two lines determine point

Both these statements have the remarkable property of being true in a two dimensional space and yet one can be obtained from the other by just exchanging the words line and point. This line-point duality allows us to define dual pairs. For example the dual of a figure consisting of four points, no three of which lie on the same line, is a figure consisting of four lines no three of which pass through the same point.

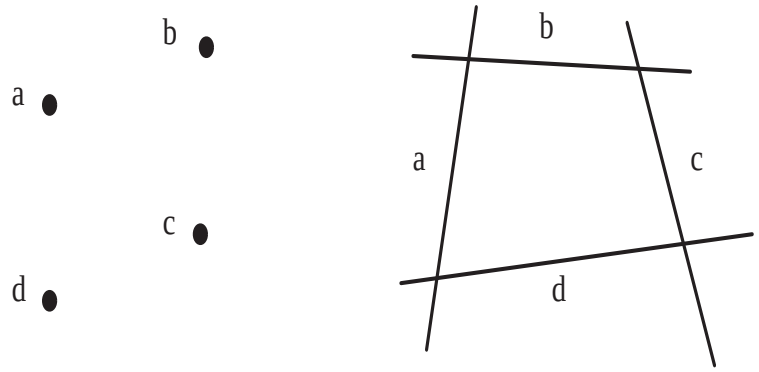

Figure 14:

We shall call the following pair of problems the "classic" Primal-Dual pair of Linear Programming Problems

Primal Problem	**Dual Problem**
$\min_{\mathbf{x}} \left[\mathbf{c}^T\mathbf{x} = S_x\right]$	$\max_y \left[\mathbf{b}^T\mathbf{y} = S_y\right]$
subject to $\mathbf{Ax} \geq \mathbf{b}$	subject to $\mathbf{A}^T\mathbf{y} \leq \mathbf{c}$
$\mathbf{x} \geq \mathbf{0}$	$\mathbf{y} \geq \mathbf{0}$

The two problems are closely related *and have the remarkable property that the optimal values of both objective functions*, $\mathbf{c}^T\mathbf{x}$ *and* $\mathbf{b}^T\mathbf{y}$, *are equal.*
Explicitly, for a linear programming problem the *primal form* is:

$$\text{Minimise}\,(c_1x_1 + c_2x_2 + \ldots + c_nx_n)$$

subject to:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &\geq b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n &\geq b_2 \\ &\ldots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n &\geq b_n \end{aligned}$$

where:

$$x_1, x_2, \ldots, x_n \geq 0$$

the associated *dual linear program* is given by:

$$\text{Maximise}\,(b_1y_1 + b_2y_2 + \ldots + b_ny_n)$$

subject to:

$$\begin{aligned} a_{11}y_1 + a_{21}y_2 + a_{31}y_3 + \ldots + a_{n1}y_n &\leq c_1 \\ a_{12}y_1 + a_{22}y_2 + a_{32}y_3 + \ldots + a_{n2}y_n &\leq c_2 \\ &\ldots \\ a_{1n}y_1 + a_{2n}y_2 + a_{3n}y_3 + \ldots + a_{nn}y_n &\leq c_n \end{aligned}$$

where:

$$y_1, y_2, \ldots, y_n \geq 0$$

Thus for all primal-dual pairs:

(a) the functional MIN and MAX swop from Primal to Dual.
(b) the constraint matrix $\mathbf{A}$ is transposed.
(c) the cost vector $\mathbf{c}$ becomes the right hand side of the constraint and vice versa.
(d) The dual of a dual problem is the original primal problem.

2.17 Duality Theorems

In the last section we saw how to find the Dual Problem given a Primal problem and, since the Primal is the Dual of the Dual, we have seen how to do the inverse operation. The Dual pair of problems involve exactly the same coefficients $\mathbf{A}, \mathbf{b}, \mathbf{c}$ and yet appear to be very different problems. However, as we see from the Theorems that follow, the solution of one problem contains the solution to the other.
Consider the classic dual pair

$$\begin{array}{r|r} \min\left[S_x = \mathbf{c}^T\mathbf{x}\right] & \max\left[S_y = \mathbf{b}^T\mathbf{y}\right] \\ \text{subject to } \mathbf{Ax} \geq \mathbf{b} & \text{subject to } \mathbf{A}^T\mathbf{y} \leq \mathbf{c} \\ \mathbf{x} \geq \mathbf{0} & \mathbf{y} \geq \mathbf{0} \end{array}$$

For any pair of feasible solutions $\mathbf{x}, \mathbf{y}$ we know that

$$\mathbf{Ax} \geq \mathbf{b} \qquad \text{and} \qquad \mathbf{y} \geq \mathbf{0}$$

so we can multiply both sides of the constraint by $\mathbf{y}$ obtaining

$$\mathbf{y}^T\mathbf{Ax} \geq \mathbf{y}^T\mathbf{b} = \mathbf{b}^T\mathbf{y} = S_y$$

Similarly $\mathbf{x}^T\mathbf{A}^T\mathbf{y} \leq \mathbf{x}^T\mathbf{c} = \mathbf{c}^T\mathbf{x} = S_x$. Combining these two results

$$S_y = \mathbf{b}^T\mathbf{y} \leq \mathbf{y}^T\mathbf{Ax} \leq \mathbf{c}^T\mathbf{x} = S_x$$

The values S_x and S_y are not optimal values but any feasible values. This relationship must however hold for optimal values. Hence:

Theorem 6 *If there are feasible x and y such that*

$$S_y = \mathbf{b}^T\mathbf{y} = \mathbf{y}^T\mathbf{A}\mathbf{x} = \mathbf{c}^T\mathbf{x} = S_x$$

then these $\mathbf{x}$ *and* $\mathbf{y}$ *must be optimal values.*

The Duality Theorem makes the matter clear:

Theorem 7 *For any Dual pair of Linear Programming problems such that a finite optimal solution exists for one problem, the objective functions of both problems have an identical optimal value i.e.*

$$S_x^* = S_y^*$$

where the askerisk indicates optimality.

This Duality Theorem tells us that provided there is a finite optimal solution to one of a dual pair of problems this relationship holds. This equation may be rewritten as

$$\begin{aligned} \mathbf{y}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) &= 0 \\ \mathbf{x}^T(\mathbf{A}^T\mathbf{y} - \mathbf{c}) &= 0 \end{aligned}$$

Hence:

Theorem 8 *(The Theorem of Complementary Slackness).*
If $\mathbf{x}$ *and* $\mathbf{y}$ *are corresponding feasible solutions to a Primal-Dual pair of Linear Programming problems then both are optimal if and only if*

$$\begin{aligned} \mathbf{y}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) &= 0 \\ \mathbf{x}^T(\mathbf{A}^T\mathbf{y} - \mathbf{c}) &= 0 \end{aligned}$$

These equations have the following interpretation: either an inequality constraint in the Primal (Dual) problem is satisfied by the equality relationship (i.e. $\mathbf{A}\mathbf{x} = \mathbf{b}$ or $\mathbf{A}^T\mathbf{y} = \mathbf{c}$) or the corresponding Dual (Primal) variable is zero.
An alternative way of writing these relationships would be

$$\begin{aligned} y_i p_i &= 0 \\ x_i d_i &= 0 \end{aligned}$$

where p_i and d_i are slack or surplus variables in the i^{th} primal and dual constraint respectively.

2.18 An Example

In this section a Primal-Dual pair of problems are both solved to illustrate the relationships between such pairs. Consider the problem:

$$\begin{array}{rl|rl} \min\left[2x_1 + x_3\right] & & \min\left[\mathbf{c}^T\mathbf{x}\right] \\ \text{subject to } x_1 - x_2 + x_3 \geq 5 & & \mathbf{Ax} \geq \mathbf{b} \\ x_1 - 2x_2 - 4x_3 \geq 8 & & \\ \mathbf{x} \geq \mathbf{0} & & \mathbf{x} \geq \mathbf{0} \\ \text{whose dual is} & & \\ \max\left[5y_1 + 8y_2\right] & & \max\left[\mathbf{b}^T\mathbf{y}\right] \\ \text{subject to} & & \\ y_1 + y_2 \leq 2 & & \mathbf{A}^T\mathbf{y} \leq \mathbf{c} \\ -y_1 - 2y_2 \leq 0 & & \\ y_1 - 4y_2 \leq 1 & & \\ \mathbf{y} \geq \mathbf{0} & & \mathbf{y} \geq \mathbf{0} \end{array}$$

- We first solve the Dual problem adding slack variables d_i to obtain an initial Tableau (Tableau 1, below)
- and thus obtain an optimal solution (Tableau 2 below)
- We now solve the Primal problem by introducing surplus variables p_i (Tableau 3 below).
- There is no basic solution so we start to create one by making a unit vector under x_1 (Tableau 4 below),
- and then introduce a unit vector under p_1 giving the optimal Tableau (Tableau 5 below)

Tableau 1

y_1	y_2	d_1	d_2	d_3	S_y	Solution column
1	1	1	0	0	0	2
−1	−2	0	1	0	0	0
1	−4	0	0	1	0	1
−5	−8	0	0	0	0	0

Tableau 2

y_1	y_2	d_1	d_2	d_3	S_y	Solution column
1	1	1	0	0	0	2
1	0	2	1	0	0	4
5	0	4	0	1	0	9
3	0	8	0	0	1	16

Tableau 3

x_1	x_2	x_3	p_1	p_2	S_x	Solution column
−1	−1	1	−1	0	0	5
1	−2	−4	0	−1	0	8
−2	0	−1	0	0	1	0

Tableau 4

x_1	x_2	x_3	p_1	p_2	S_x	Solution column
1	-1	1	-1	0	0	5
0	-1	-5	1	-1	0	3
0	-2	1	-2	0	1	10

Tableau 5

x_1	x_2	x_3	p_1	p_2	S_x	Solution column
1	-2	-4	0	-1	0	8
0	-1	-5	1	-1	0	3
0	-4	-9	0	-2	1	16

Comparing the two optimal solutions we see

$$S_x = S_y = 16$$

The solution to the Primal problem is $[x_1, x_2, x_3] = [8, 0, 0]$. The coefficients in the objective function row of the dual problem under the slack variables d_1, d_2, d_3 are $[8, 0, 0]$. The solution to the Dual Problem is $[y_1, y_2] = [0, 2]$. The coefficients in the objective function row of the Primal problem corresponding to the surplus variables p_1, p_2 are $[0, -2]$. We thus see that the solution to a dual problem may be read from the optimal simplex Tableau of the primal problem:

Theorem 9 *If the optimal solution to the primal problem has been found by the Simplex Method, then the optimal value of the i^{th} dual variable may be found in the objective function row under the slack variable corresponding to the i^{th} constraint. Surplus variables give the negative of the dual variable.*

The example in this section also illustrates the Theorem of Complementary Slackness: $y_1 = 0$ and $y_2 = 2$ so x_1 need not be zero and it is equal to 8 but x_2 must be 0.

2.19 The Dual Simplex Method

This method is motivated by the relationship between the primal and dual problems and it is designed to solve LP problems efficiently. In the dual simplex algorithm the algorithm starts better than or equal to the optimum and infeasible. Successive iterations are designed to move toward feasibility without violating optimality. When feasibility is restored, the algorithm terminates. This method is in contrast with the regular (primal) simplex method which starts with a feasible but non-optimal solution and continues until feasibility and optimality is achieved.
In the dual simplex method the starting tableau must have an optimum objective function row with at least one infeasible (i.e. with a value less than 0) basic variable. To maintain optimality and simultaneously move toward feasibility at each iteration, the following two conditions are employed:

1. Dual feasibility condition. The leaving variable x_r is the basic variable having the most negative value (any tied values are broken arbitrarily). If all the basic variables are non-negative , the algorithm ends.

2. Dual optimality condition. The entering variable is determined from among the nonbasic variables as the one corresponding to:

$$\min_{Nonbasic\ x_j} \left\{ \left| \frac{\beta_j}{\alpha_{rj}} \right|, \alpha_{rj} < 0 \right\}$$

where α_{rj} is the constraint coefficient of the tableau associated with the row of the leaving variable x_r and the column of the entering variable x_j and β_j is the objective function coefficient of x_j in the simplex tableau.

This whole procedure is best illustrated with an Example.

Example:

$$\min z = 3x_1 + 2x_2$$

subject to:

$$\begin{aligned} 3x_1 + x_2 &\geq 3 \\ 4x_1 + 3x_2 &\geq 6 \\ x_1 + x_2 &\leq 3 \\ x_1, x_2 &\geq 0 \end{aligned}$$

The starting tableau would be:

x_1	x_2	s_1	s_2	s_3	Solution Column
-3	-1	1	0	0	-3
-4	-3	0	1	0	-6
1	1	0	0	1	3
-3	-2	0	0	0	0

In this method we will use a slightly different tableau format with the objective function row at the top (we will introduce a Ratio row at the bottom). We will also introduce an initial column which indicates the basis. Thus the initial tableau is:

Basic	x_1	x_2	s_1	s_2	s_3	Solution Column
z	-3	-2	0	0	0	0
s_1	-3	-1	1	0	0	-3
s_2	-4	-3	0	1	0	-6
s_3	1	1	0	0	1	3

The variables s_1 and s_2 are surplus and s_3 is a slack variable. We have multiplied each of the equations associated with the surplus basic variables s_1 and s_2 by -1 so that the right-hand side will directly show which basic variables are infeasible ($s_1 = -3$, $s_2 = -6$ and $s_3 = 3$). The current solution is $x_1 = 0$, and $x_2 = 0$ which with the requirement $x_1, x_2 \geq 0$ is clearly optimal for a minimisation problem. This the current solution is optimal and infeasible.

The *dual feasibility condition* indicates that s_2 ($s_2 = -6$) is the leaving variable. Next, we apply the *dual optimality condition* to determine the entering variable using the following summary table.

Variable	x_1	x_2	s_1	s_2	s_3
β_j (z-row)	-3	-2	0	0	0
s_2-row, $\alpha_{s_2 j}$	-4	-3	0	1	0
Ratio $\frac{\beta_j}{\alpha_{s_2 j}}$	$\frac{3}{4}$	$\frac{2}{3}$	–	–	–

The ratios show that x_2 (the smallest ratio) is the entering variable. *Note* that a variableis a candidate for entering the basic solution only if its α_{4j} is strictly negative. Thus , the basic variables s_1, s_2 and s_3 should not be considered. The next tabeau is obtained using the usual row manipulations:

Basic	x_1	x_2	s_1	s_2	s_3	Solution Column
z	$-\frac{1}{3}$	0	0	$-\frac{2}{3}$	0	4
s_1	$-\frac{5}{3}$	0	1	$-\frac{1}{3}$	0	-1
x_2	$\frac{4}{3}$	1	0	$-\frac{1}{3}$	0	2
s_3	$-\frac{1}{3}$	0	0	$\frac{1}{3}$	1	1
Ratio	$\frac{1}{5}$	–	–	2	–	

We get the ratio $1/5$ from $(-1/3)/(-5/3) = 1/5$. Thus s_1 leaves and x_1 enters, thus yielding the following tableau:

Basic	x_1	x_2	s_1	s_2	s_3	Solution Column
z	0	0	$-\frac{1}{5}$	$-\frac{3}{5}$	0	$\frac{21}{5}$
x_1	1	0	$-\frac{3}{5}$	$\frac{1}{5}$	0	$\frac{3}{5}$
x_2	0	1	$\frac{4}{5}$	$-\frac{3}{5}$	0	$\frac{6}{5}$
s_3	0	0	$-\frac{1}{5}$	$\frac{2}{5}$	1	$\frac{6}{5}$

The last tableau is feasible and optimal, thus the algorithm has terminated. The solution is therefore:

$$x_1 = \frac{3}{5} \qquad x_2 = \frac{6}{5} \qquad z = \frac{21}{5}$$

The trajectory taken by the dual simplex method is shown in the plot below:

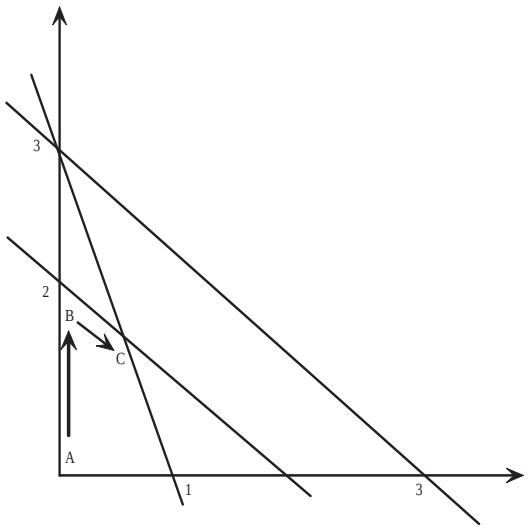

Figure 15: Trajectory taken by algorithm.

The algorithm starts at A (infeasible and better than the optimum). It moves to B (still infeasible and better than the optimum). It terminates at C (feasible and optimal).

2.20 Sensitivity Analysis

In many real problem situations the data provided can change with time, so one needs to know how sensitive the solution is to small changes. The constraints are also sometimes a bit elastic. For instance, if more resources could be made available, we may need to be able to say how much extra output could be obtained at the price of a marginal increase in resources. In this section we consider an example showing how it is possible to read from the final Tableau a lot of information about the sensitivity of the solution to small changes in the model.

Example: The Great Press Company makes ashtrays, wheel covers and scatter dishes. It has a press shop, plating shop, polishing shop and trim shop but finds that it is the press shop and trim shop that limit the amount of work it can get done. Ash trays require 2 man-hours per thousand in the press shop and 3 hours per thousand in the trim shop. Wheel covers take 1 hour per thousand in both the press and trim shops. Scatter dishes require 5 hours per thousand in the press shop and 3 hours per thousand in the trim shop. The sales team feel that not more than 5,000 ashtrays can be sold. The net profit from selling ashtrays is £15 each, while for wheel covers and scatter dishes the net profits are £6 and £9 respectively. How many of each article should Great Press make if they have available 10 hours of press shop time and 12 hours of trim shop time. If we define:

x_1 = the number of thousands of ash trays made

x_2 = the number of thousands of wheel covers made

x_3 = the number of thousands of scatter dishes made

the problem can be stated as

$$\max [z = 15x_1 + 6x_2 + 9x_3]$$

subject to:

Press shop: $2x_1 + x_2 + 5x_3 \leq 10$

Trim shop: $3x_1 + x_2 + 3x_3 \leq 12$

Sales constraint: $x_1 \leq 5$

Non negativity Constraint: $\mathbf{x} \geq 0$

Adding slack variables s_1, s_2 and s_3 gives an initial and final Simplex Tableau as follows:

Initial Tableau for the Great Press Co. Problem.

x_1	x_2	x_3	s_1	s_2	s_3	z	Solution Column
2	1	5	1	0	0	0	10 $(+\delta)$
3	1	3	0	1	0	0	12
1	0	0	0	0	1	0	5 $(+\mu)$
−15	−6	−9	0	0	0	1	0

Final Tableau for Great Press Co. Problem.

x_1	x_2	x_3	s_1	s_2	s_3	z	Solution Column
0	1	9	3	−2	0	0	6
1	0	−2	−1	1	0	0	2
0	0	2	1	−1	1	0	3
0	0	15	3	3	0	1	66

From which we could advise Great Press Co. to make 2000 ash trays, 6000 wheel covers and no scatter dishes. The corresponding expected profit is £66, 000.

2.20.1 Sensitivity to Changes in Constraint Limits.

In the problem as posed, only 10 hours were said to be available in the Press shop, but there might be more time if overtime were worked. It will only be worth using overtime if the extra labour costs incurred are more than paid for by the increased profit. This leads us to ask how much the profit would increase if an extra hour of Press shop time were available. This extra profit is called the *marginal value* of an hour of Press shop time. It is the rate at which profit increases with Press shop time when one is making a small change in its availability away from the normal value.
In Table 1 below a small change δ has been added to the amount of Press shop time available. The δ occurs only in the first row and similarly the slack variable s_1 occurs only in the first row. The Final Tableau is obtained by adding or subtracting multiples of the rows to other rows and every time row 1 has been added to another row we can see the multiple of s_1 occuring. δ would be treated in the same way as s_1 so that Table 1 shows how a change in the availability of Press shop hours would alter the situation.

Table 1

x_1	x_2	x_3	s_1	s_2	s_3	z	Solution Column
0	1	9	3	-2	0	0	$6+3\delta$
1	0	-2	-1	1	0	0	$2-\delta$
0	0	2	1	-1	1	0	$3+\delta$
0	0	15	3	3	0	1	$66+3\delta$

From this we can see that the Marginal value of an hour of Press shop time is £3000 and, if the Press shop time were increased by an hour the new optimal solution would be to make 1000 ashtrays, 9000 wheel covers and no scatter dishes.
The marginal value applies not only to increases in Press shop time but also to decreases and it is important to know the range over which the marginal value is valid. The range is calculated by modifying δ until the solution given by the Tableau becomes infeasible i.e. one of the values in the solution column becomes negative. The range in this example is:

$$-2 \leq \delta \leq 2$$

For values of δ outside this range it is necessary to solve for another Basic feasible solution.

2.20.2 Changes in the Coefficients of the Objective Function

The net profit on each type of article is an estimate - it may prove necessary to reduce prices to clear stock or it may be possible to increase prices. Because of the uncertainty in the true level of prices it is important to know how sensitive the solution is to price levels. This is usually expressed by calculating the range within which objective function coefficients can change without changing the basic solution.
In the Simplex Method the objective function row is never added or substracted from any other row in the Tableau, and it is never multiplied by any constant, so if the price of a scatter dish changes from £9 to $9+\epsilon$, the number in the objective function row under x_3 changes to $15-\epsilon$. This means that the optimal solution will only change if $15-\epsilon \leq 0$, i.e. $\epsilon \geq 15$.
In the example above we considered changes in the objective function coefficient of the non-basic variable x_3. If we change the coefficient of the basic variable x_1 from 15 to $15+\epsilon$ this produces a $-\epsilon$ in the objective function row of the final tableau:

x_1	x_2	x_3	s_1	s_2	s_3	Solution Column
0	1	9	3	-2	0	6
1	0	-2	-1	1	0	2
0	0	2	1	-1	1	3
$-\epsilon$	0	15	3	3	0	66

if we perform the usual Simplex algorithm, ignoring the presense of the ϵ. We can remove the $-\epsilon$ term in the objective function row by adding ϵ times the second row to the objective function row. This gives:

x_1	x_2	x_3	s_1	s_2	s_3	Solution Column
0	1	9	3	-2	0	6
1	0	-2	-1	1	0	2
0	0	2	1	-1	1	3
0	0	$(15-2\epsilon)$	$(3-\epsilon)$	$(3+\epsilon)$	0	$(66+2\epsilon)$

Hence $x_1 = 2$ and $x_2 = 6$ provided:

$$-3 \leq \epsilon \leq 3$$

The optimal solution thus does not change for changes in the price of ashtrays between 12 and 18 pounds.

2.21 Problems

1. Solve the following Linear Programming problem by drawing a graph of the feasible region:

$$\max[x_1 + 2x_2]$$

subject to:

$$\begin{aligned} x_1 &\leq 2 \\ x_1 + x_2 &\leq 3 \\ -x_1 + x_2 &\leq 1 \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

2. A manufacturer produces an article in two finishes A, B. He employs 8 skilled men and 4 labourers each of whom works a maximum of 40 hours a week. Each article with finish A requires 4 hours work by a skilled man and 6 hours by a labourer: with finish B, it needs 8 hours work by a skilled man and 2 hours by a labourer. On each article with finish A the profit is £4, with finish B it is £6. How many articles of each type should be produced each week so as to maximise his profit. (The employees are paid for a 40 hour week -even if they do not work that number of hours due to lack of work).

3. Solve the Linear Programming problem

$$x_0 = \max\left[x_1 + x_2\right]$$

subject to:

$$\begin{aligned} x_1 + 2x_2 &\leq 8 \\ 3x_1 + 2x_2 &\leq 12 \\ 2x_1 + 2x_2 &\leq 9 \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

4. Solve the problem formulated in section 2.2 (Feed Mix problem) by a graphical method.

(5,a) Maximise $Z = 3x_1 + x_2 + 3x_3$ subject to:

$$\begin{aligned} 2x_1 + x_2 + x_3 &\leq 2 \\ x_1 + 2x_2 + 3x_3 &\leq 5 \\ 2x_1 + 2x_2 + x_3 &\leq 6 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

(5,b) Maximise $Z = 8x_1 + 4x_2$ subject to:

$$\begin{aligned} x_1 + x_2 &\leq 10 \\ 5x_1 + x_2 &\leq 15 \\ x_1, x_2 &\geq 0 \end{aligned}$$

(5,c) Maximise $Z = 3x_1 + 2x_2 + 5x_3$ subject to:

$$\begin{aligned} x_1 + 2x_2 + x_3 &\leq 430 \\ 3x_1 + 2x_3 &\leq 460 \\ x_1 + 4x_2 &\leq 420 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

(5,d) Minimise $Z = 11 - x_3 - x_4 - x_5$ subject to:

$$\begin{aligned} x_1 + x_3 - x_4 + 2x_5 &= 2 \\ x_2 - x_3 + 2x_4 - x_5 &= 1 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

(5,e) Maximise $Z = 3x_1 + 9x_2$ subject to:

$$\begin{aligned} x_1 + 4x_2 &\leq 8 \\ x_1 + 2x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \end{aligned}$$

6. A patient who regularly visits his doctor complaining of nervousness has been recommended to take at least 24 units of vitamin C and at least 25 units of vitamin B_2 every day. The 2 local Health Shop sells two sorts of vitamin pills; 'vitalife' pills which cost lp each and contain 1 unit of vitamin C and 5 units of vitamin B_2 and 'pick-you-up' pills at 3p each which contain 4 units of vitamin C and 1 unit of B_2. How many pills of each type should the patient consume each day in order to obtain the required vitamins at minimum cost?

Find out, using a graphical method, how the time of each type of machine should be divided so as to maximise the rate of production of completed items. What is the maximum output rate.

7. The Clifton Metal Artwork Company has excess capacity which can be devoted to one or two new products (product 1 and 2). The available capacity on the production machines and the number of machine hours required for each unit is summarised in the table below:

Machine	Available time (machine hours/week)	Product 1	Product 2
Milling	400	5	4
Lathe	300	3	3.75
Grinder	70	0	1

The unit profit rate would be £20 and £30 respectively on products 1 and 2 and the objective is to maximise profitability.

(a) Formulate the problem as a linear programming task.

(b) Find the optimal feasible solution graphically.

(c) Using the Simplex Method solve the problem formulated in (a) and show that it agrees with the solution proposed in (b).

(d) For product 2 determine the range of the unit profit rate for which the current solution remains optimal given the unit profit rate on product 1 remains fixed at £20.

(e) The Company investigates the market for two new products labelled 3 and 4 and discovers there is apparently unlimited potential for the sale of these products at home and overseas. The sales department formulates the sales task as:

Maximise $z = 200x_3 + 100x_4$

subject to:

$$\begin{aligned} -x_3 + x_4 &\leq 1 \\ x_3 - 2x_4 &\leq 2 \\ x_3, x_4 &\geq 0 \end{aligned}$$

where x_3 and x_4 are the proposed sales volume for these two new products. What is wrong with this formulation and what are the implications for the potential sales volume of products 3 and 4?

2.22 Answers

1. 1,2,5
2. 16,32,256
3. Any point on the line $x_1 + x_2 = 4.5$ provided $x_1 \in [1, 3]$. The objective function always has the value 4.5.
4. 6,8,62

(5,a)

x_1	x_2	x_3	s_1	s_2	s_3	Solution Column	Ratio
$\langle 2 \rangle$	1	1	1	0	0	2	1
1	2	3	0	1	0	5	5
2	2	1	0	0	1	6	3
-3	-1	-3	0	0	0	0	–

x_1	x_2	x_3	s_1	s_2	s_3	Solution Column	Ratio
1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	1	2
0	$\frac{3}{2}$	$\langle \frac{5}{2} \rangle$	$-\frac{1}{2}$	1	0	4	$\frac{8}{5}$
0	-1	0	-1	0	1	4	∞
0	$\frac{1}{2}$	$-\frac{3}{2}$	$\frac{3}{2}$	0	0	0	3

x_1	x_2	x_3	s_1	s_2	s_3	Solution Column
1	$-\frac{1}{5}$	0	$\frac{3}{5}$	$-\frac{1}{3}$	0	$\frac{1}{5}$
0	$\frac{3}{5}$	1	$-\frac{1}{5}$	$\frac{2}{3}$	0	$\frac{8}{5}$
0	-1	0	-1	0	1	4
0	$\frac{7}{5}$	0	$\frac{6}{5}$	$\frac{2}{3}$	0	$\frac{27}{5}$

so $x_1 = 1/5$, $x_2 = 0$ and $x_3 = 8/3$ with $z = 27/5$.

(5,b)

x_1	x_2	s_1	s_2	Solution Column	Ratio
1	1	1	0	10	10
5	1	0	1	15	3
-8	-4	0	0	0	–

x_1	x_2	s_1	s_2	Solution Column	Ratio
0	$\frac{4}{5}$	1	$-\frac{1}{5}$	7	$\frac{35}{4}$
1	$\frac{1}{5}$	0	$\frac{1}{5}$	3	15
0	$-\frac{12}{5}$	0	$\frac{8}{5}$	24	–

x_1	x_2	s_1	s_2	Solution Column
0	1	$\frac{5}{4}$	$-\frac{1}{4}$	$\frac{35}{4}$
1	0	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{5}{4}$
0	0	3	1	45

$x_1 = 5/4$, $x_2 = 35/4$ and $z = 45$

(5,c)

x_1	x_2	x_3	s_1	s_2	s_3	Solution Column	Ratio
1	2	1	1	0	0	430	430
3	0	$\langle 2 \rangle$	0	1	0	460	230
1	4	0	0	0	1	420	∞
-3	-2	-5	0	0	0	0	0

x_1	x_2	x_3	s_1	s_2	s_3	Solution Column	Ratio
$-\frac{1}{2}$	2	0	1	$-\frac{1}{2}$	0	200	100
$\frac{3}{2}$	0	1	0	$\frac{1}{2}$	0	230	∞
1	4	0	0	0	1	420	105
$\frac{9}{2}$	-2	0	0	0	$\frac{5}{2}$	0	1150

x_1	x_2	x_3	s_1	s_2	s_3	Solution Column
$-\frac{1}{4}$	1	0	$\frac{1}{2}$	$-\frac{1}{4}$	0	100
$\frac{3}{2}$	0	1	0	$\frac{1}{2}$	0	230
0	0	0	-2	1	1	20
4	0	0	1	2	0	1350

(5,d) The constant 11 makes no difference to the minimisation task and so we discard it. Furthermore minimising $-x_3 - x_4 - x_5$ can be viewed as maximising $z = x_3 + x_4 + x_5$ so our initial tableau is:

x_1	x_2	x_3	x_4	x_5	Solution Column	Ratio
1	0	$\langle 1 \rangle$	-1	2	2	2
0	1	-1	2	-1	1	-1
0	0	-1	-1	-1	0	0

we could choose any of the three columns with a -1 in the objective function row.

x_1	x_2	x_3	x_4	x_5	Solution Column	Ratio
1	0	1	-1	2	2	-2
1	1	0	$\langle 1 \rangle$	1	3	3
1	0	0	-2	1	2	–

x_1	x_2	x_3	x_4	x_5	Solution Column
2	1	1	0	3	5
1	1	0	1	1	3
3	2	0	0	3	8

so $x_1 = 0$, $x_2 = 0$, $x_3 = 5$, $x_4 = 3$ and $x_5 = 0$ and $z = 11 - 5 - 3 - 0 = 3$.

(5,e)

x_1	x_2	s_1	s_2	Solution column	Ratio
1	$\langle 4 \rangle$	1	0	8	2
1	2	0	1	4	2
-3	-9	0	0	0	–

x_1	x_2	s_1	s_2	Solution column	Ratio
$\frac{1}{4}$	1	$\frac{1}{4}$	0	2	8
$\langle \frac{1}{2} \rangle$	0	$-\frac{1}{2}$	1	0	0
$-\frac{3}{4}$	0	$\frac{9}{4}$	0	18	–

x_1	x_2	s_1	s_2	Solution column
0	1	$\frac{1}{2}$	$-\frac{1}{2}$	2
1	0	-1	2	0
0	0	$\frac{3}{2}$	$\frac{3}{2}$	18

so $x_1 = 0$, $x_2 = 2$ and $z = 18$.

6. 4,5,19

7 (a) Let x_i be weekly production of product i where $i = 1, 2$, then the task is:

$$\max \; z = 20x_1 + 30x_2$$

where:

$$\begin{aligned} 5x_1 + 4x_2 &\leq 400 \\ 3x_1 + 3.75x_2 &\leq 300 \\ x_2 &\leq 70 \end{aligned}$$

and $x_i \geq 0$.

(b)

The optimal point is at the intersection of constraints (2) and (3) and it is given by $(12.5, 70)$ with $z = 2350$.

(c) The LP problem is:

$$\max \; z = 20x_1 + 30x_2$$

where:

$$\begin{aligned} 5x_1 + 4x_2 + s_1 &= 400 \\ 3x_1 + 3.75x_2 + s_2 &= 300 \\ x_2 + s_3 &= 70 \end{aligned}$$

so:

x_1	x_2	s_1	s_2	s_3	Sol.	Ratio
5	4	1	0	0	400	100
3	3.75	0	1	0	300	80
0	$\langle 1 \rangle$	0	0	1	70	70
-20	-30	0	0	0	0	

where $\langle\rangle$ indicates the pivot element:

x_1	x_2	s_1	s_2	s_3	Sol.	Ratio
5	0	1	0	-4	120	24
$\langle 3 \rangle$	0	0	1	-3.75	37.5	12.5
0	1	0	0	1	70	∞
-20	0	0	0	30	2100	

x_1	x_2	s_1	s_2	s_3	Sol.
0	0	1	$-\frac{5}{3}$	2.25	57.5
1	0	0	$\frac{1}{3}$	-1.25	12.5
0	1	0	0	1	70
0	0	0	$\frac{20}{3}$	5	2350

so the solution is $(x_1, x_2, s_1, s_2, s_3) = (12.5, 70, 57.5, 0, 0)$, or in terms of the physical variables in the problem:

$$x_1 = 12.5 \qquad\qquad x_2 = 70$$

(d) This problem can be approved several ways but the best approach would be to return to the sketch in (b), look at the sketched objective function line and look to see how the solution may change as c_2 is varied (alternatively we could use a tableau approach). With the graphical approach
$z = 20x_1 + c_2x_2$ can be written:

$$x_2 = \frac{z}{c_2} - \left(\frac{20}{c_2}\right) x_1$$

In the (x_2, x_1) plane this is a line with gradient:

$$gradient = -\frac{20}{c_2}$$

If:

$$-\frac{20}{c_2} \leq -\frac{4}{5}$$

then $(12.5, 70)$ is not optimal but $\left(44\frac{4}{9}, 44\frac{4}{9}\right)$ is the optimal solution. This last inequality can be written:

$$25 \leq c_2$$

(e) The problem is unbounded! This is why you should always check whether the initial formulation of the problem looks correct with a quick sketch.

3 Integer Programming

3.1. Introduction
3.2. Examples of Modeling with Integer Variables

3.2.1 Capital Budgeting
3.2.2 Multiperiod Capital Budgeting
3.2.3 Rucksack problem
3.2.4 Set Covering
3.2.5 Traveling Salesman Problem

3. Solving Integer Programs
4. Relationship to Linear Programming
5. Branch and Bound

3.1 Introduction

Consider the manufacture of television sets. A linear programming model might give a production plan of 205.7 sets per week. In such a model, most people would have no trouble stating that production should be 205 sets per week (or even "roughly 200 sets per week"). On the other hand, suppose we were buying warehouses to store finished goods, where a warehouse comes in a set size. Then a model that suggests we purchase 0.7 warehouse at some location and 0.6 somewhere else would be of little value. Warehouses come in integer quantities, and we would like our model to reflect that fact.
This *integrality restriction* may seem rather innocuous, but in reality it has far reaching effects. On the one hand, modeling with integer variables has turned out to be useful far beyond restrictions to integral production quantities. With integer variables we can model logical requirements (*true* or *false*), fixed costs (the costs may be discrete e.g. in units of £50K), sequencing and scheduling requirements and many other problems.
The problem with this approach, however, is that problems with as few as 40 variables can be beyond the capabilities of even the most sophisticated computers. While these small problems are somewhat artificial, most real problems with more than 100 or so variables are not possible to solve unless they show specific exploitable structure. Despite the possibility (or even likelihood) of enormous computing times, there are methods that can be applied to solving integer programs. The LINDO solver in LINGO (a widely available software package) is based on a method called *branch and bound* but there are other approaches. The purpose of this chapter is to illustrate some interesting integer programming applications and to describe some of these solution techniques as well as possible pitfalls.
First we introduce some terminology. An integer programming problem in which all variables are required to be integer is called a *pure integer programming problem.* If some variables are restricted to be integer and some are not then the problem is a *mixed integer programming problem.* The instance where the integer variables are restricted to be 0 or 1 appears surprisingly often. Such problems are called pure (mixed) *0-1 programming problems* or pure (mixed) *binary integer programming* problems.

3.2 Modeling with Integer Variables

In this section we will illustrate a number of classic problems in which integer programming features. They are Capital Budgeting, Multiperiod Capital Budgeting, Rucksack, Set Covering and the classic Traveling Salesman Problem. They are outlined here to illustrate the many important applications of integer programming: many of the integer programming problems encountered in real life can be mapped across to one of these problems.

3.2.1 The Capital Budgeting Problem

Suppose we wish to invest £14,000. We have identified four investment opportunities. Project 1 requires an investment of £5,000 and has a return of £8,000 (at the end of the term assets are therefore the investment and this additional return); project 2 requires £7,000 and has a return of £11,000; project 3 requires £4,000 and has a return of £6,000; and project 4 requires £3,000 and has a return of £4,000. Into which investment projects should we place our money so as to maximise our total return?
As in linear programming, our first step is to decide on our variables. This can be much more difficult in integer programming because there are many ways to use integrality restrictions. In this case, we will use a 0-1 variable for each investment. If is 1 then we will make investment j. If it is 0, we will not make the investment. This leads to the 0-1 programming problem:

$$\begin{array}{rl} \text{maximise} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x_j \in \{0, 1\}\ j = 1, \ldots 4. \end{array}$$

It would appear that project 1 offers the most efficient investment. In fact, ignoring integrality constraints, the optimal linear programming solution is $x_1 = 1, x_2 = 1, x_3 = 0.5, x_4 = 0$ with a return of £22,000. Unfortunately, this solution is not integral. Rounding down x_3 to 0 gives a feasible solution with a value of £19,000. There is a better integer solution, however, of $x_1 = 0, x_2 = x_3 = x_4 = 1$ with a value of £21,000. This example shows that rounding does not necessarily give an optimal value. There are a number of additional constraints we might want to add. For instance, consider the following constraints:
1. We can only complete two projects.

2. If project 2 is completed, project 4 must also be completed.

3. If project 1 is made, project 3 cannot be completed.

All of these, and many more logical restrictions, can be enforced using 0-1 variables. In these cases, the constraints are:

1. $x_1 + x_2 + x_3 + x_4 \leq 2$

2. $x_2 - x_4 \leq 0$

3. $x_1 + x_3 \leq 1$

3.2.2 The Multiperiod Capital Budgeting Problem

In the preceding example, we considered making a single investment in a project for a duration of time and receiving the return on the investment at the end of this duration. In practice, we may face a choice among projects that require investments of different amounts in each of several periods (with possibly different budgets available in each period), with the return being realized over the life of the project. In this case, we can still model the problem with variables:

$$x_j = \begin{cases} 1 & \text{if we invest in project } j \\ 0 & \text{otherwise,} \end{cases}$$

the objective is still to maximise the sum of the returns on the projects selected, and there is now a budget constraint for each period. For example, suppose we wish to invest £14,000, £12,000, and £15,000 in each month of the next quarter. We have identified four investment opportunities. Investment 1 requires an investment of £5,000, £8,000, and £2,000 in month 1, 2, and 3, respectivey, and has a return of £8,000; investment 2 requires

£7,000 in month 1 and £10,000 in period 3, and has a return of £11,000; investment 3 requires £4,000 in period 2 and £6,000 in period 3, and has a return of £6,000; and investment 4 requires £3,000, £4,000, and £5,000 and has a return of £4,000. The corresponding integer program is

$$\begin{array}{rl} \text{maximise} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 \qquad\quad + 3x_4 \leq 14 \\ & 8x_1 \qquad + 4x_3 + 4x_4 \leq 12 \\ & 2x_1 + 10x_2 + 6x_3 + 5x_4 \leq 15 \\ & x_j \in \{0, 1\}. \end{array}$$

3.2.3 The Rucksack (Knapsack) Problem

The rucksack problem is a particularly simple integer program: it has only one constraint. Furthermore, the coefficients of this constraint and the objective are all non-negative. For example, the following is a rucksack problem:

$$\begin{array}{rl} \text{maximise} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x_j \in \{0, 1\}. \end{array}$$

The traditional story is that there is a rucksack (here of capacity 14). There are a number of items (here there are four items, labelled $x_1, \ldots, x_4$), each with a size and a value (here item 2 has size 7 and value 11). The objective is to maximise the total value of the items in the rucksack. rucksack problems are interesting because they are (usually) easy to solve.

To solve *the associated linear program*, i.e. variables not restricted to be integer, it is simply a matter of determining which variable has the most "value" or potential return. If you take c_j/a_j (the objective coefficient/constraint coefficient) for each variable, the one with the highest ratio is the best item to place in the rucksack. Then the item with the second highest ratio is put in and so on until we reach an item that cannot fit. At this point, a fractional amount of that item is placed in the rucksack to completely fill it. In our example, the variables here have already been ordered by this ratio. We would first place item 1 in, then item 2, but item 3 doesn't fit: there are only 2 units left, but item 3 has size 4. Therefore, we take half of item 3. The solution $x_1 = 1, x_2 = 1, x_3 = 0.5, x_4 = 0$ is the optimal solution to the linear program.

As already observed, this solution is quite different from the optimal solution to the rucksack problem with integer variables. Nevertheless, it will play an important role in the solution of the problem by branch and bound as we will see shortly.

3.2.4 The Set Covering Problem

To illustrate this model, consider the following location problem: a city is reviewing the location of its fire stations. The city is made up of a number of neighbourhoods, as illustrated in Figure 16. A fire station can be placed in any neighbourhood. It is able to handle the fires for both its neighbourhood and any adjacent neighbourhood (any neighbourhood with a non-zero border with its home neighbourhood). The objective is to minimise the number of fire stations used.

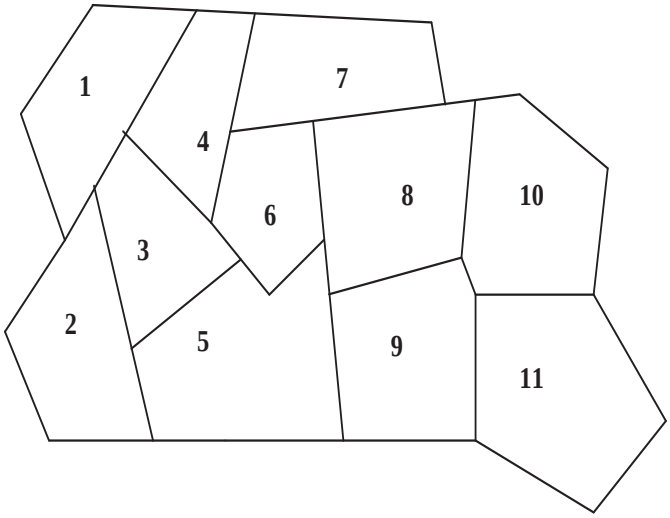

Figure 16: A map of the city: a fire station can handle fires in its neighbourhood and those in the adjacent neighbourhoods

We can create one variable for each neighbourhood j. This variable will be 1 if we place a station in the neighbourhood, and will be 0 otherwise. This leads to the following formulation:

$$
\begin{array}{rl}
\text{minimise} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \\
\text{subject to} & x_1 + x_2 + x_3 + x_4 \geq 1 \\
& x_1 + x_2 + x_3 + x_5 \geq 1 \\
& x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 1 \\
& x_1 + x_3 + x_4 + x_6 + x_7 \geq 1 \\
& x_2 + x_3 + x_5 + x_6 + x_8 + x_9 \geq 1 \\
& x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \geq 1 \\
& x_4 + x_6 + x_7 + x_8 \geq 1 \\
& x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \geq 1 \\
& x_5 + x_8 + x_9 + x_{10} + x_{11} \geq 1 \\
& x_8 + x_9 + x_{10} + x_{11} \geq 1 \\
& x_9 + x_{10} + x_{11} \geq 1 \\
& x_j \in \{0,1\} \quad j = 1, \ldots, 11
\end{array}
$$

The first constraint states that there must be a station either in neighbourhood 1 or in some adjacent neighbourhood. The next constraint is for neighbourhood 2 and so on. Notice that the constraint coefficient is 1 if neighbourhood i is adjacent to neighbourhood j or if $i = j$, and 0 otherwise. The jth column of the constraint matrix represents the set of neighbourhoods that can be served by a fire station in neighbourhood j. We are asked to find a set of such subsets j that covers the set of all neighbourhoods in the sense that every neighbourhood appears in the service subset associated with at least one fire station. One optimal solution to this is $x_3 = x_8 = x_9 = 1$ and the rest equal to 0.
This is an example of the *set covering problem*. The set covering problem is characterized by having binary variables, $\geq$ constraints each with a right hand side of 1, and having simply sums of variables as constraints. In general, the objective function can have any coefficients, though here it is of a particularly simple form.

3.2.5 The Traveling Salesman Problem

Consider a traveling salesman who must visit each of 20 cities before returning home. He knows the distance between each of the cities and wishes to minimise the total distance traveled while visiting all of the cities. In what order should he visit the cities?
Let there be n cities, numbered from 1 up to n. For each pair of cities (i, j) let c_{ij} be the cost of going from city i to city j, or from city j to city i. Let x_{ij} be 1 if the person travels between cities i and j (either from city i to city j or from j to i). This problem is known as the *symmetric TSP*. In the *asymmetric TSP*, the cost to travel in one direction may differ from the cost to travel in the opposite direction, and the decision variables must distinguish between the two directions. Clearly the asymmetric problem is the more general.

The objective is to minimise $\sum_{i=1}^{n} \sum_{j=1}^{i-1} c_{ij} x_{ij}$. The constraints are harder to find. Consider the following set:

$$\sum_{j \neq i} x_{ij} = 2 \quad \forall i$$

These constraints say that every city must be visited. These constraints are not enough, however, since it is possible to have multiple cycles (subtours), rather than one big cycle (tour) through all the points. To handle this condition, we can use the following set of subtour elimination constraints:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \quad \forall S \subset N$$

This set states that, for any subset of cities S, the tour must enter and exit that set. These, together with $x_{ij} \in \{0, 1\}$ is sufficient to formulate the traveling salesman problem (TSP) as an integer program.

3.3 Solving Integer Programs

We have gone through a number of examples of integer programs. A natural question is: how can we get solutions to these models? There are two common approaches. Historically, the first method developed was based on *cutting planes* (adding constraints to force integrality). In the last twenty years or so, however, the most effective technique has been based on dividing the problem into a number of smaller problems in a method called *branch and bound*. Both these approaches involve solving a series of linear programs. So that is where we will begin.

3.4 Relationship to Linear Programming

Given an integer program (**IP**):

$$\begin{array}{rrl} \text{minimise} \quad (\text{or} \quad \text{maximise}) & cx & \\ \text{subject} \quad \text{to} & Ax = b & \\ & x \geq 0 \quad \text{and} \quad \textit{integer} & \end{array} \tag{21}$$

there is an associated linear program called the linear relaxation (**LR**) formed by dropping the integrality restrictions:

$$\begin{array}{rrl} \text{minimise} \quad (\text{or} \quad \text{maximise}) & cx & \\ \text{subject} \quad \text{to} & Ax = b & \\ & x \geq 0. & \end{array} \tag{22}$$

Since (LR) is less constrained than (IP), the following are immediate:

- If (IP) is a minimisation, the optimal objective value for (LR) is less than or equal to the optimal objective for (IP).
- If (IP) is a maximisation, the optimal objective value for (LR) is greater than or equal to that of (IP).
- If (LR) is infeasible, then so is (IP).
- If (LR) is optimized by integer variables, then that solution is feasible and optimal for (IP).
- If the objective function coefficients are integer, then for minimisation, the optimal objective for (IP) is greater than or equal to the "round up" of the optimal objective for (LR). For maximisation, the optimal objective for (IP) is less than or equal to the "round down" of the optimal objective for (LR).

So solving (LR) does give some information: it gives a bound on the optimal value, and, if we are lucky, may give the optimal solution to IP. We saw, however, that rounding the solution of LR will not in general give the optimal solution of (IP). In fact, for some problems it is difficult to round and even get a feasible solution.

3.5 Branch and Bound

We will explain branch and bound by using the capital budgeting example from the previous section. In that problem, the model is:

$$\begin{array}{rl} \text{maximise} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x_j \in \{0,1\}; j = 1, \ldots, 4. \end{array}$$

The linear relaxation solution is $x_1 = 1, x_2 = 1, x_3 = 0.5, x_4 = 0$ with a return of 22. We know that no integer solution will have value more than 22. Unfortunately, since x_3 is not integer, we do not have an integer solution yet.
We want to force x_3 to be integer. To do so, we *branch* on x_3, creating two new problems. In one, we will add the constraint $x_3 = 0$. In the other, we add the constraint $x_3 = 1$. This is illustrated in Figure 17.

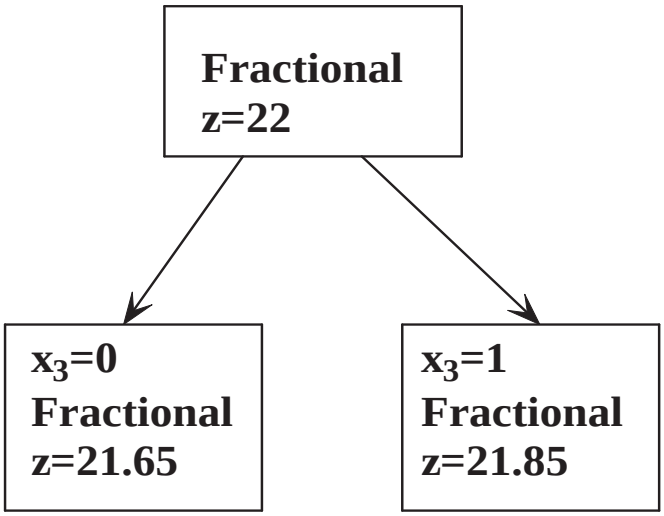

Figure 17: First Branching

Note that any optimal solution to the overall problem must be feasible to one of the subproblems. If we solve the linear relaxations of the subproblems, we get the following solutions:

$x_3 = 0$: objective 21.65, $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0.667$;
$x_3 = 1$: objective 21.85, $x_1 = 1, x_2 = 0.714, x_3 = 1, x_4 = 0$.
At this point we know that the optimal integer solution is no more than 21.85 (we actually know it is less than or equal to 21 - why?), but we still do not have any feasible integer solution. So, we will take a subproblem and branch on one of its variables. In general, we will choose the subproblem as follows:

- We will choose an *active subproblem*, which so far only means one we have not chosen before, and
- We will choose the subproblem with the *highest* solution value for *maximisation* (*lowest* for *minimisation*).

In this case, we will choose the subproblem with $x_3 = 1$, and branch on x_2. After solving the resulting subproblems, we have the branch and bound tree in Figure 18.

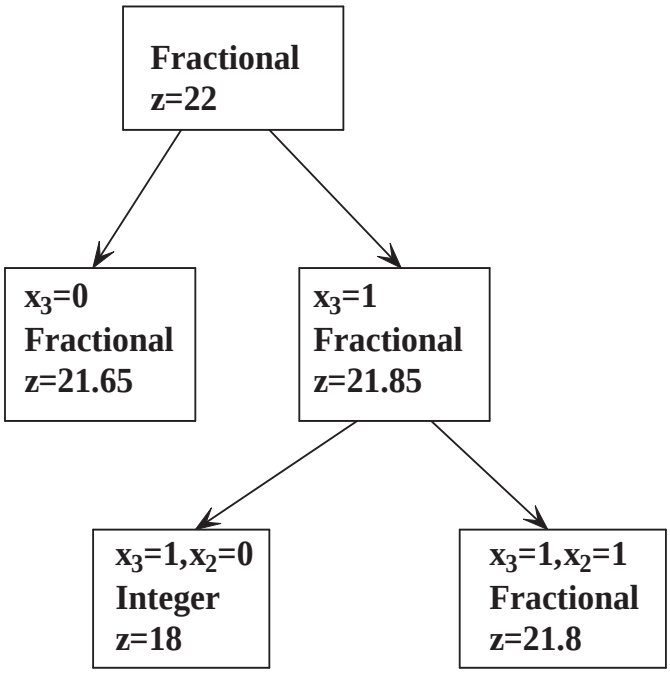

Figure 18: Second Branching

The solutions are:

$x_3 = 1, x_2 = 0$: objective 18, $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$;
$x_3 = 1, x_2 = 1$: objective 21.8, $x_1 = 0.6, x_2 = 1, x_3 = 1, x_4 = 0$
We now have a feasible integer solution with value 18. Furthermore, since the problem gave an integer solution, no further branching on that problem is necessary. *It is not active due to integrality of solution.* There are still active subproblems that might give values more than 18. Using our rules, we will branch on problem $x_3 = 1, x_2 = 1$ by branching on x_1 to get Figure 19.

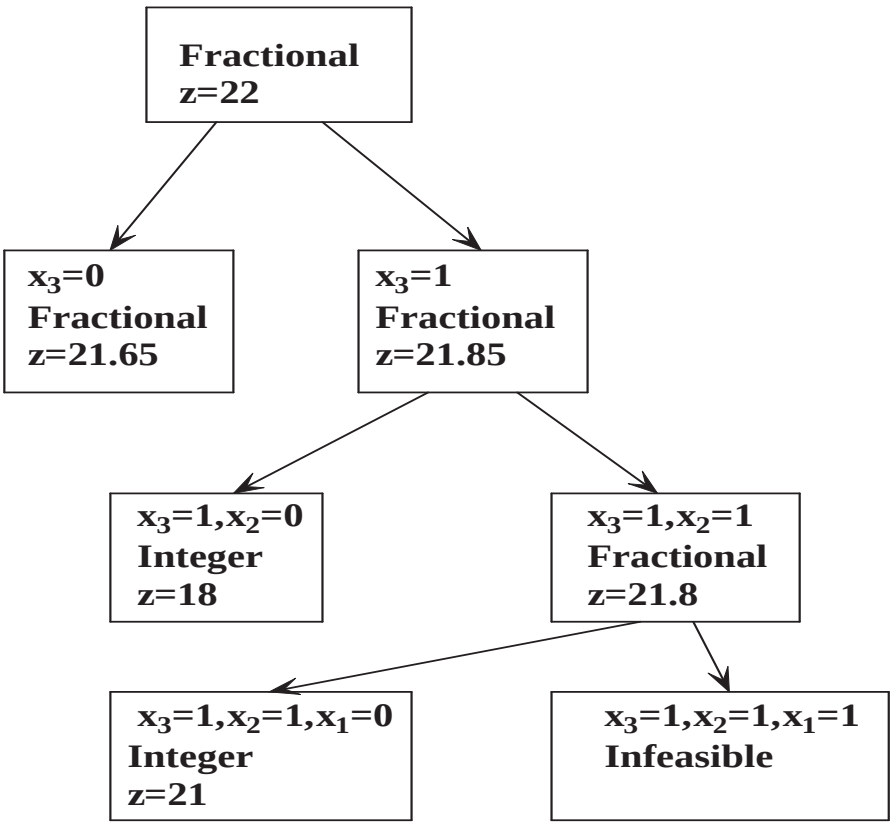

Figure 19: Third Branching

The solutions are:

$x_3 = 1, x_2 = 1, x_1 = 0$: objective 21, $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1$;
$x_3 = 1, x_2 = 1, x_1 = 1$: infeasible.
Our best integer solution now has value 21. The subproblem that generates this solution is not active due to integrality of solution. The other subproblem generated is not active due to infeasibility. There is still a subproblem that is active. It is the subproblem with solution value 21.65. By our "round-down" result, there is no better solution for this subproblem than 21. But we already have a solution with value 21. It is therefore not useful to search for another such solution. We can *fathom* this subproblem based on the above bounding argument and mark it not active. There are no longer any active subproblems, so the optimal solution value is 21.

We have seen all parts of the branch and bound algorithm. The essence of the algorithm is therefore as follows:

Branch and Bound Algorithm.

1. Solve the linear relaxation of the problem. If the solution is integer, then we are done. Otherwise create two new subproblems by branching on a fractional variable.

2. A subproblem is *not active* when any of the following occurs:
2.1. You used the subproblem to branch on,
2.2. All variables in the solution are integer,
2.3. The subproblem is infeasible,
2.4. You can fathom the subproblem by a bounding argument.

3. Choose an active subproblem and branch on a fractional variable. *Repeat until there are no active subproblems.*

That's all there is to branch and bound! Depending on the type of problem, the branching rule may change somewhat. For instance, if x is restricted to be integer (but not necessarily 0 or 1), then if $x = 4.27$ you would branch with the constraints $x \leq 4$ and $x \geq 5$ (not on $x = 4$ and $x = 5$). In the worst case, the number of subproblems can get huge. For many problems, however, the number of subproblems is quite reasonable.

4 Nonlinear Optimisation

Summary

4.1. Introduction.

4.2. Definitions

4.3. A classification scheme for optimisation problems

4.4. Optimality conditions

4.5. Examples of nonlinear optimisation tasks

4.6. Convexity and concavity

4.7. Methods for unconstrained optimisation.

4.8. Nonderivative methods

4.11. Gradient methods: steepest descent

4.12. The conjugate gradient method

4.13. The Newton method and the Levenberg-Marquardt algorithm

4.14. The Davidon-Fletcher-Powell Method

4.15. One-dimensional line search methods

4.16. Constrained optimisation

4.17. Penalty function methods

4.18. The SUMT method

4.19. Further types of penalty function

4.1 Introduction

The purpose of this chapter is intoduce techniques for solving *general nonlinear optimisation problems.* At the beginning we introduce terminology and the ways in which problems and their solutions are formulated and classified. Subsequent sections consider the most appropriate methods for several classes of optimisation problem, with emphasis placed on powerful, versatile algorithms well suited to optimizing functions of many variables on high performance computational platforms.

4.2 Definitions

The goal of an optimisation problem can be formulated as follows: find the combination of parameters (independent variables) which optimise a given quantity, possibly subject to some restrictions on the allowed parameter ranges. We have seen that the quantity to be optimised (maximised or minimised) is termed the *objective function.* The parameters which may be changed in the quest for the optimum are often called *control or decision variables.* The restrictions on allowed parameter values are known as *constraints.* A maximum of a function f is a minimum of $-f$. Thus, the general optimisation problem may be stated as:

$$\begin{array}{lll} \text{minimise} & f(\mathbf{x}) & \mathbf{x} = (x_1, x_2, \ldots, x_n)^T \\ \text{subject to} & c_i(\mathbf{x}) = 0 & i = 1, 2, \ldots, m' \\ & c_i(\mathbf{x}) \geq 0 & i = m' + 1, \ldots, m \end{array}$$

where $f(\mathbf{x})$ is the objective function, $\mathbf{x}$ is the column vector of the n independent variables, and $c_i(\mathbf{x})$ is the set of constraint functions. Constraint equations of the form $c_i(x) = 0$ are termed *equality constraints*, and those of the form $c_i(x) \geq 0$ are *inequality constraints*. Taken together, $f(\mathbf{x})$ and $c_i(\mathbf{x})$ are known as the *problem functions*.

4.3 A Classification Scheme for Optimisation Problems

In the previous chapters we introduced linear programming and integer programming. Probably the most important characteristic is the nature of the objective function. This function is *linear* if the relationship between $f(\mathbf{x})$ and the control variables is of the form

$$f(\mathbf{x}) = \mathbf{b}^T\mathbf{x} + \mathbf{c}$$

where b is a constant valued vector and c is a constant. A function is *quadratic* if

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + \mathbf{c}$$

where A is a constant valued matrix. Special methods can locate the optimal solution very efficiently for linear and quadratic functions, for example. In many cases the objective function may be a nonlinear function of variables other than quadratic and this is the focus of the current chapter. General nonlinear optimisation problems may be *unconstrained* or *constrained*. They may be *univariate* (one variable) or *multivariate* (many variables). In the Table below we have tried to illustrate the types of problems we may encounter in practice:

Characteristic	Property	Classification
Number of control variables	One	Univariate
	More than one	Multivariate
Type of control variables	Continuous real numbers	Continuous
	Integers	Integer or Discrete
	Both continuous real numbers and integers	Mixed Integer
	Integers in permutations	Combinatorial
Problem functions	Linear functions of the control variables	Linear
	Quadratic functions of the control variables	Quadratic
	Other nonlinear functions of the control variables	Nonlinear
Problem formulation	Subject to constraints	Constrained
	Not subject to constraints	Unconstrained

Table 3: Classification of Optimisation Problems

Indeed there are other types of optimisation problem beyond this classification scheme and which are beyond this course. For example, there is a special class of problems, examples of which are particularly common in the fields of operations research and engineering design, in which the task is to find the optimum permutation

of some control variables. These are known as *combinatorial optimisation problems.* The *traveling salesman problem (TSP)* in which the shortest path visiting N cities is sought can be viewed as such. The solutions to such problems are usually represented as ordered lists of integers (indicating the cities in the order they are to be visited in the TSP, for example).

4.4 Optimality Conditions

Before describing methods for nonlinear optimisation we need to describe the conditions which hold at an optimum. The strict definition of the *global optimum* $\mathbf{x}^\star$ of $f(\mathbf{x})$ is that

$$f(\mathbf{x}^\star) < f(\mathbf{y}) \qquad \forall \mathbf{y} \in V(\mathbf{x}), \mathbf{y} \neq \mathbf{x}^\star \tag{23}$$

where $V(\mathbf{x})$ is the set of feasible values of the control variables $\mathbf{x}$. Obviously, for an unconstrained problem $V(\mathbf{x})$ is infinitely large. A point $\mathbf{y}^\star$ is a *strong local minimum* of $f(\mathbf{x})$ if

$$f(\mathbf{y}^\star) < f(\mathbf{y}) \qquad \forall \mathbf{y} \in N(\mathbf{y}^\star, \eta), \mathbf{y} \neq y^\star \tag{24}$$

where $N(\mathbf{y}^\star, \eta)$ is defined as the set of feasible points contained in the neighbourhood of $\mathbf{y}^\star$, i.e., within some arbitrarily small distance η of $\mathbf{y}^\star$. For $\mathbf{y}^\star$ to be a *weak local minimum* only an inequality need be satisfied

$$f(\mathbf{y}^\star) \leq f(\mathbf{y}) \qquad \forall \mathbf{y} \in N(\mathbf{y}^\star, \eta), \mathbf{y} \neq \mathbf{y}^\star \tag{25}$$

More useful definitions, i.e., more easily identified optimality conditions, can be provided if $f(\mathbf{x})$ is a smooth function with continuous first and second derivatives for all feasible $\mathbf{x}$. Then a point $\mathbf{x}^\star$ is a stationary point of $f(\mathbf{x})$ if

$$g(\mathbf{x}^\star) = 0, \tag{26}$$

where $g(\mathbf{x})$ is the *gradient* of $f(\mathbf{x})$. This first derivative vector has components given by

$$g_i(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x_i} \tag{27}$$

The point $\mathbf{x}^\star$ is also a *strong local minimum* of $f(\mathbf{x})$ if the *Hessian matrix* $\mathbf{H}(\mathbf{x})$ i.e. the symmetric matrix of second derivatives with components

$$H_{ij}(\mathbf{x}) = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \tag{28}$$

is *positive definite* at $\mathbf{x}^\star$ if:

$$\mathbf{u}^T \mathbf{H}(\mathbf{x}^\star)\mathbf{u} > 0 \qquad \forall \mathbf{u} \neq 0. \tag{29}$$

where u is a scalar or vector (as appropriate) with real-valued components.
The Figure on the next page illustrates the different types of stationary points for unconstrained univariate functions. For constrained optimisation problems, the presence of a constraint boundary acts as a simple bound on the permitted values of the control variable so the global minimum may be an extreme value or *extremum* (an endpoint) rather than a true stationary point. Some methods of treating constraints transform the optimisation problem into an equivalent unconstrained one, with a different objective function: these will be discussed later.

4.5 Examples of Nonlinear Optimisation Tasks

Example 1: Consider minimisation of:

$$f(x, y) = (x - 0.5)^4 + (y - 0.25)^4 \tag{30}$$

with respect to x and y. This is a *unconstrained multivariate optimisation task* (with a solution $x = 0.5$ and $y = 0.25$).

Example 2: For the subject of pattern recognition a *classifier* maps inputs x_j to targets (labels) y_i and one way of finding the variables to perform this classification task is to maximise the following expression:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right) \tag{31}$$

with respect to the α_i subject to the constraints:

$$\alpha_i \geq 0 \qquad \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{32}$$

with the data objects (x_j and y_i) held constant. This is an example of a *constrained quadratic programming* task

Example 3: *Rosenbrock's function* is often used as a minimisation test problem for *nonlinear continuous problems*, since its minimum lies at the base of a "banana-shaped valley" and can be difficult to locate. This function is defined for even integers n as the following sum:

$$f(x) = \sum_{j=1,3,5,\ldots,(n-1)} \left[(1 - x_j)^2 + 100(x_{j+1} - x_j^2)^2\right]. \tag{33}$$

The *contour plot* of Rosenbrock's function for $n = 2$ is shown in Figure 20. In general, contour maps show surfaces in the $(n + 1)-$dimensional space defined by $f(x) = \gamma$ where γ is a constant. For $n = 2$, plane curves correspond to various values of γ. We see from the figure that the minimum point (dark circle) is at $(1, 1)^T$, where $f(x) = 0$. The gradient components of this function are given by

$$\left.\begin{array}{c} g_{j+1} = 200(x_{j+1} - x_j^2) \\ g_j = -2\left[x_j g_{j+1} + (1 - x_j)\right] \end{array}\right\}, j = 1, 3, 5, \ldots, (n - 1), \tag{34}$$

and the Hessian is the 2×2 block diagonal matrix with entries

$$\left.\begin{array}{l} H_{(j+1),(j+1)} = 200 \\ H_{(j+1),j} = -400 x_j \\ H_{j,j} = -2(x_j H_{(j+1),j} + g_{(j+1)} - 1) \end{array}\right\}, j = 1, 3, 5, \ldots, (n - 1) \tag{35}$$

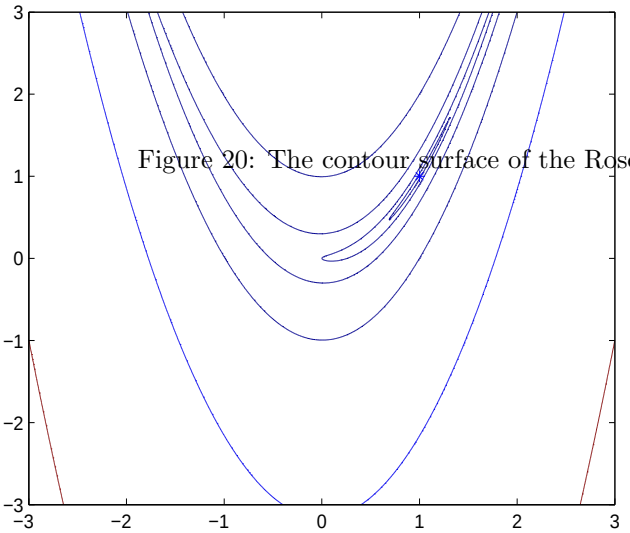

Figure 20: The contour surface of the Rosenbrock function

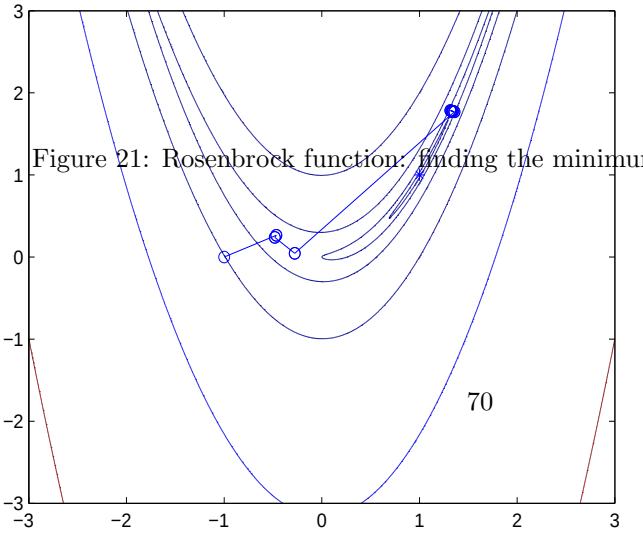

Figure 21: Rosenbrock function: finding the minimum using steepest descent

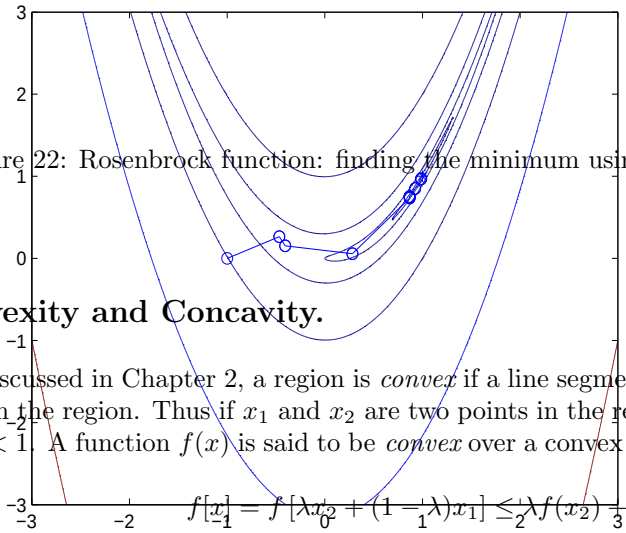

Figure 22: Rosenbrock function: finding the minimum using the conjugate gradient method

4.6 Convexity and Concavity.

As we have discussed in Chapter 2, a region is *convex* if a line segment joining any two points in that region lies entirely within the region. Thus if x_1 and x_2 are two points in the region then so is every point $\lambda x_2 + (1-\lambda)x_1$ where $0 < \lambda < 1$. A function $f(x)$ is said to be *convex* over a convex domain X if for any two points $x_1, x_2 \in X$:

$$f[x] = f\left[\lambda x_2 + (1-\lambda)x_1\right] \leq \lambda f(x_2) + (1-\lambda)f(x_1) \tag{36}$$

for $0 < \lambda < 1$. For a function of one variable this means that $f[x]$ lies below the chord joining any two points on its graph.

For a *concave* function defined on a convex domain we reverse the inequality to obtain:

$$f[x] = f\left[\lambda x_2 + (1-\lambda)x_1\right] \geq \lambda f(x_2) + (1-\lambda)f(x_1) \tag{37}$$

In this case the function lies above the chord joining two points on its graph. If the inequalities in the above ($\leq$ or $\geq$) are replaced by strict inequalities ($<$ or $>$) then $f(x)$ is said to be *strictly convex* or *strictly concave*. We can deduce the following property for convex (concave) functions from the above:

Theorem: If the Hessian matrix:

$$H = \frac{\partial^2 f(x)}{\partial x_i \partial x_j} \tag{38}$$

is *positive definite* (*negative definite*) the function $f(x)$ is *convex* (*concave*).

Sketch of the proof for positive definite/convex function:

For a convex function:

$$f[x] = f\left[\lambda x_2 + (1-\lambda)x_1\right] \leq \lambda f(x_2) + (1-\lambda)f(x_1) \tag{39}$$

so with double derivatives on both sides:

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \leq \lambda \frac{\partial^2 f(x_2)}{\partial x_i \partial x_j} + (1-\lambda)\frac{\partial^2 f(x_1)}{\partial x_i \partial x_j} \tag{40}$$

Pre and post-multiplying by u^T and u respectively we get:

$$u^T \frac{\partial^2 f(x)}{\partial x_i \partial x_j} u \leq \lambda u^T \frac{\partial^2 f(x_2)}{\partial x_i \partial x_j} u + (1-\lambda) u^T \frac{\partial^2 f(x_1)}{\partial x_i \partial x_j} u \tag{41}$$

so if $f(x)$ is a convex function then so is $u^T H u$ and vice versa, where H is the Hessian matrix.
Let x_2 and x_1 be any two values for x and let $u = \lambda x_2 + (1-\lambda)x_1$ where $0 < \lambda < 1$. Then:

$$\begin{aligned} & u^T H u - \lambda x_2^T H x_2 - (1-\lambda) x_1^T H x_1 \\ = \;& \lambda^2 x_2^T H x_2 + 2\lambda(1-\lambda) x_2^T H x_1 + (1-\lambda)^2 x_1^T H x_1 - \lambda x_2^T H x_2 - (1-\lambda) x_1^T H x_1 \\ = \;& -\lambda(1-\lambda)(x_2 - x_1)^T H (x_2 - x_1) \end{aligned} \tag{42}$$

since $0 < \lambda < 1$, $(1-\lambda) > 0$ and so if H is positive definite the final expression is less than or equal to zero. Thus:

$$u^T H u - \lambda x_2^T H x_2 - (1-\lambda) x_1^T H x_1 \leq 0$$

so:

$$u^T H u = [\lambda x_2 + (1-\lambda)x_1]^T H [\lambda x_2 + (1-\lambda)x_1] \leq \lambda x_2^T H x_2 + (1-\lambda) x_1^T H x_1 \tag{43}$$

thus, referring back to the definition of convexity, we see that $u^T H u$ is convex, which from, earlier comments means $f(x)$ is convex. For convex functions of one variable this says that the second derivative is non-negative i.e. the first derivative is an increasing function which can only be zero at one point, thus such a function will have one minimum point. *If the Hessian matrix is negative definite the function is concave by a similar argument.*

Example 1: Is the function $3x^3 + 4x^2 + 5x + 6$ convex or concave?

Answer: In this case the Hessian $\partial^2 f(x)/\partial x_i \partial x_j$ is simply $d^2 f(x)/dx^2$. In this case:

$$\frac{df}{dx} = 9x^2 + 8x + 5$$

so:

$$\frac{d^2 f}{dx^2} = 18x + 8$$

This can be both positive and negative on the range $(-\infty, \infty)$ for x. Hence this function is neither concave or convex.

Example 2: Is the function $f(x) = 4x^2 - 8$ convex or concave?

Answer:

$$\frac{df}{dx} = 8x$$

$$\frac{d^2f}{dx^2} = 8$$

so the function is strictly convex.

Example 3: Is $f(x_1, x_2) = x_1^2 + x_1x_2 + 2x_2^2$ a convex or concave function?

Answer: The Hessian of this function is:

$$H = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$$

We proceed to evaluate the eigenvalues for this Hessian matrix, since if the eigenvalues are all positive then $u^THu = u^T\lambda u = \lambda u^Tu > 0$ and if the eigenvalues are all negative then $u^THu = u^T\lambda u = \lambda u^Tu > 0$. In this case:

$$\det(H - \lambda I) = 0$$

$$\det \begin{vmatrix} 2-\lambda & 1 \\ 1 & 4-\lambda \end{vmatrix} = \lambda^2 - 6\lambda + 7 = 0$$

The roots of this equation are $\lambda = 3 \pm \sqrt{2}$. Since both values of λ are positive, H is positive definite, so $f(x_1, x_2)$ is *convex*.

4.7 Methods for Unconstrained Continuous Multivariate Problems

In this section we outline some basic techniques involving algorithms for finding *local minima* of multivariate functions whose arguments are continuous and on which no restrictions are imposed. It should be emphasized that finding the *global minimum* is an entirely different, and more challenging, problem which will not be addressed here. Some algorithmic approaches, not covered in this course, are called *stochastic* since the approach uses probabilistic techniques. An example is the simulated annealing algorithm which is slow but very good for finding a global minimum. The algorithms covered in this course are called *deterministic* by contrast since probabilistic concepts are not used. For a minimisation task the simplest strategy is to descend, following the local gradient. The first approach which we outline (the downhill simplex method) uses only function calls to estimate the descent direction. In general, however, it is best to use gradient and second order derivative information to improve convergence to a minimum. The simplest idea is *steepest descent* but, in order to remove oscillations, a better scheme is to choose the descent direction using the approach of the *conjugate gradient* algorithm. The conjugate gradient algorithm implicitly uses second derivative information.

4.8 Unconstrained minimisation using function values only (not examinable)

Generally the use of gradient information gives a much faster approach to an optimum of a function than using function calls only (in the worse case we can estimate a derivative using function calls only). In this section we will introduce one method based purely on function calls - the *downhill simplex method* due to Nelder and Mead. This method isn't related to the simplex method of linear programming but has the same name because geometric objects called simplexes are used in the method. On the average it will be slower than the gradient descent methods we will mention shortly. Nonetheless it is still of interest for the following reasons:

- it is robust and simple and will sometimes succeed when other methods fail
- there is no need to evaluate the derivative - only function calls are needed.
- in contrast to other approaches the method is geometric and simple to use.

In this section we will only give the essential outline of the method (a more detailed description can be found in textbooks e.g. Press et al. Numerical Recipes, Cambridge).

A *simplex* is a geometric object with $(n+1)$ points (vertices) in n dimensions together with lines, planar faces, etc which connect these vertices. In two dimensions a simplex is a triangle, in three dimensions a simplex is a tetrahedron (which is not necessarily regular - i.e. of equal side). These simplexes are nondegenerate, that is, they enclose a finite n-dimensional volume.

In the *downhill simplex method* we start with some arbitrary set of points defining a simplex and (hopefully but not necessarily) enclosing the minimum. The values of the function are evaluated at all the vertices of the simplex and there will typically be a highest value and a lowest value for the function at these vertices. A series of reflections, reflections and expansions or contractions of the simplex are pursued which consistently move the simplex away from the highest-valued regions towards the lowest-valued regions. The simplex may be contracted onto the minimum in the vicinity of the minimum. These geometric operations are illustrated with the following Figures. In the last Figure we show a 2-dimensional example (a 3-vertex triangle) in which the simplex is moving towards the lowest-valued region of the space. Thissequence of maneouvers will always converge on a minimum of the function.

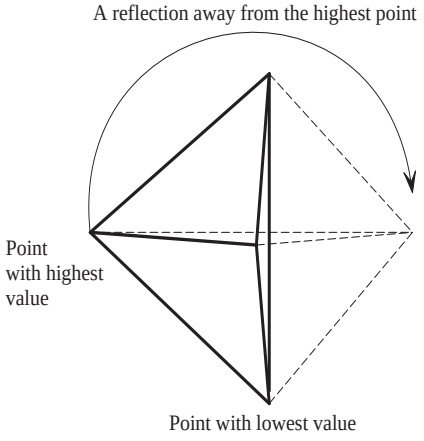

Figure 23: Downhill Simplex method: a reflection

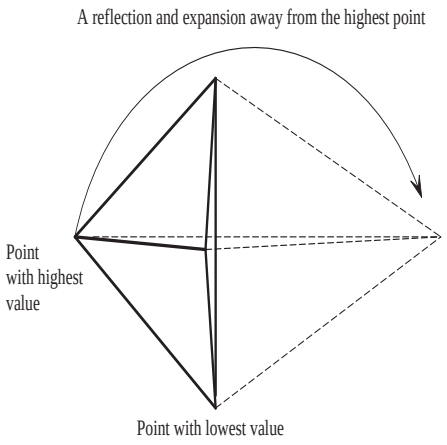

Figure 24: A reflection and expansion

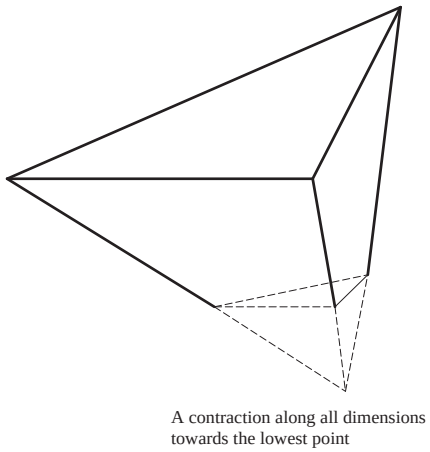

Figure 25: A reflection with contraction

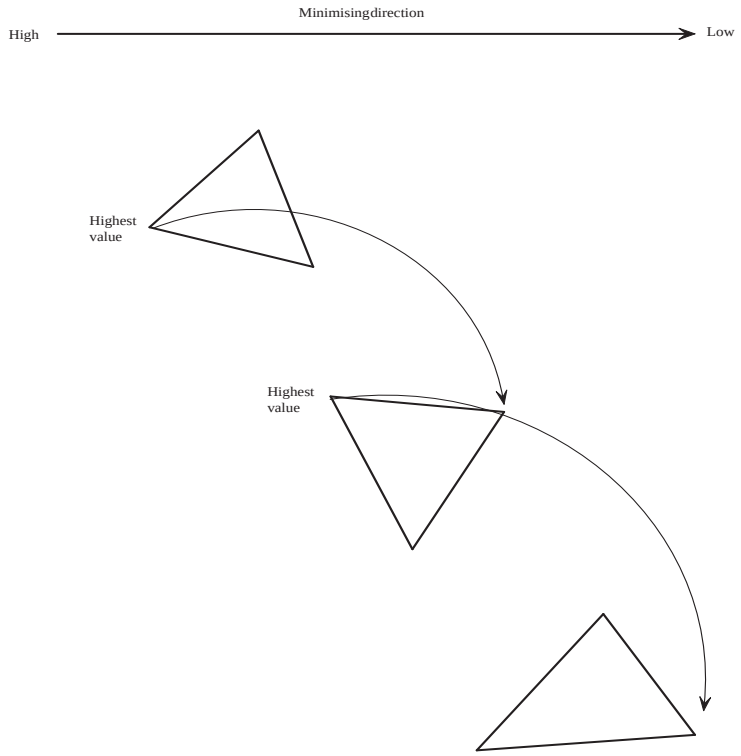

Figure 26: The Downhill Simplex method: a two-dimensional example. The simplex, in a n-dimensional space, is a $(n+1)$-dimensional polytope, here a 3-sided triangle in 2-dimensions. It moves through the space in an amoeboid fashion heading toward a maximum or minimum through a sequence of reflections, expansions and contractions. This method is robust and does not use derivative information, only function calls.

4.9 Gradient descent techniques

The fundamental structure of local iterative techniques for solving unconstrained minimisation problems is simple. A starting point is chosen, a direction of movement is given according to some algorithm, and a *line search* is performed to determine an appropriate next step. The process is repeated at the new point and the algorithm continues until a local minimum is found. Schematically, a model local minimiser method can be sketched as follows:

A Basic Local Minimiser Algorithm

$\mathbf{x}^{(t)}$ is the position at iteration t:

Supply an initial guess for the position at iteration $t = 0$, i.e. $\mathbf{x}^0$

For $t = 0, 1, 2, \ldots$ until convergence:

1. Test $\mathbf{x}^t$ for convergence, if converged on solution then **stop**, else:
2. Calculate a search direction $\mathbf{d}^{(t)}$
3. Determine an appropriate step length λ in this direction (we can call this a *line search* since λ is a scalar and differing values of λ take us along a line)

4. Calculate a new position $\mathbf{x}^{(t+1)}$ from $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \lambda \mathbf{d}^{(t)}$ and go to step **1.**

Descent Directions: For minimisation, it is reasonable to choose a search vector $\mathbf{d}^{(t)}$ which is a descent direction i.e. a direction leading to function reduction. If we use a Taylor series approximation:

$$f(\mathbf{x} + \lambda \mathbf{d}) - f(\mathbf{x}) \approx \lambda \mathbf{g}(\mathbf{x})^T \mathbf{d}$$

we see that negativity of the left-hand side, for positive λ, implies that a suitable descent direction $\mathbf{d}$ is such that:

$$\mathbf{g}(\mathbf{x})^T \mathbf{d} < 0$$

Thus, for minimisation, one obvious way of achieving this is to use a direction which is simply the negative gradient i.e.

$$\mathbf{d} = -\mathbf{g}$$

since then $\mathbf{g}^T \mathbf{d} = -\mathbf{g}^T \mathbf{g} < 0$. We would use the positive gradient ($\mathbf{d} = \mathbf{g}$) if we wish to maximise the function. This method is therefore called steepest descent or steepest ascent for this reason.

Different methods are distinguished by their choice of search directions. Algorithms can be classified into *nonderivative* (function calls only), *gradient* and *second derivative* methods depending on the technique used to determine $\mathbf{d}$ in the Basic Local Minimiser Algorithm given above. Generally there are computational benefits (in terms of speed of convergence) if we use derivative information rather than function calls only with the most efficient algorithms using both first and second order derivative information. Two common gradient methods discussed here are Steepest Descent (SD) and Conjugate Gradient (CG). Both are fundamental techniques that are often incorporated into various iterative algorithms in many areas of scientific computing.

4.10 Steepest Descent (SD): using the function values and first derivatives of the function

SD is one of the oldest and simplest methods. It is actually more important as a theoretical, rather than a practical, reference by which to test other methods. However, steepest descent steps are often incorporated into other methods (e.g. Conjugate Gradient, Newton methods). At each iteration t of SD, the search direction is taken as $-\mathbf{g}^{(t)}$. Steepest ascent is the same idea but with $\mathbf{d}^{(t)} = \mathbf{g}^{(t)}$.
SD is simple to implement and requires modest storage requirements. However, progress toward a minimum may be very slow, especially near a solution. For example, the convergence rate of SD when applied to a convex quadratic function is only linear. Thus, the SD search vectors may in some cases exhibit very inefficient paths toward a solution, especially in the proximity of that solution.
The minimisation performance of Rosenbrock's is shown in Figures 20, 21, 22 for SD and the Conjugate Gradient Method which will be discussed later. We show the progress of the method by superimposing the resulting iterates from each minimisation application on the contour plot. Recall that the minimum point for Rosenbrock's function lies at $\mathbf{x} = (1, 1)^T$, where $f(\mathbf{x}) = 0$. We see the characteristic behavior of the SD method: relatively good progress at first, but very slow convergence in the vicinity of the solution.

Example: As a practical example let us consider the following problem. Maximise:

$$f(x_1, x_2) = -x_1^2 + 4x_1 + 2x_1 x_2 - 2x_2^2$$

starting at the point:

$$x_1 = 0 \qquad x_2 = 0$$

using a gradient method with line search.

Iteration 1.

$$\nabla f\left(x_1, x_2\right) = \left(-2x_1 + 2x_2 + 4, 2x_1 - 4x_2\right)$$

Thus:

$$\nabla f\left(x_1^0, x_2^0\right) = (4, 0)$$

for $x^0 = (0, 0)$. So:

$$\begin{aligned} f\left[x^0 + \lambda \nabla f(x^0)\right] &= \max f\left[(0,0) + \lambda(4,0)\right] \\ &= \max f\left[4\lambda, 0\right] \\ &= \max\left[-16\lambda^2 + 16\lambda\right] \end{aligned}$$

where $\lambda > 0$. In this case it is possible to use differentiation to determine the optimal value of λ:

$$\frac{d}{d\lambda}\left(-16\lambda^2 + 16\lambda\right) = -32\lambda + 16 = 0$$

so the optimal value of λ is:

$$\lambda^\star = \frac{1}{2}$$

so:

$$\mathbf{x}_1 = \mathbf{x}_0 + \lambda \nabla f(\mathbf{x}_0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 4 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

Iteration 2.

$$\nabla f(x_1 x_2) = (-4 + 4, 4) = (0, 4)$$

and so:

$$\begin{aligned} f\left[x_1^0 + \lambda \nabla f\left(x_1^0\right)\right] &= \max f\left[(2,0) + \lambda(0,4)\right] \\ &= \max f\left[2, 4\lambda\right] \\ &= \max\left[4 + 16\lambda - 32\lambda^2\right] \end{aligned}$$

and the optimal value of λ is found from:

$$\frac{d}{d\lambda}\left(4 + 16\lambda - 32\lambda^2\right) = 16 - 64\lambda = 0$$

thus:

$$\lambda^\star = \frac{1}{4}$$

and:

$$\mathbf{x}_2 = \mathbf{x}_1 + \lambda \nabla f(x_1) = \begin{pmatrix} 2 \\ 0 \end{pmatrix} + \frac{1}{4} \begin{pmatrix} 0 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

we keep repeating this process. You can verify that the subsequent points in this sequence are:

$$\begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ \frac{3}{2} \end{pmatrix}, \begin{pmatrix} \frac{7}{2} \\ \frac{3}{2} \end{pmatrix}, \begin{pmatrix} \frac{7}{2} \\ \frac{7}{4} \end{pmatrix} \cdots$$

Eventually this process will terminate at $(4, 2)$.

4.11 The Conjugate Gradient Method

The conjugate gradient (CG) method was originally designed to minimise convex quadratic objective functions but, through several variations, has been extended to general functions. The first iteration in CG is the same as in SD, but successive directions are constructed so that they form a set of mutually *conjugate vectors* with respect to the (positive definite) Hessian H of a general convex quadratic function $q(x)$: we will explain this shortly.

During the line search step we try and minimise the function (in λ) along a suitable search direction. Suppose we have already minimised λ along the proposed search direction. It turns out that using the gradient vector at the new position does not represent the best choice for the next search direction. To see this we note that the minimum of the line search for $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \lambda \mathbf{d}^{(t)}$ occurs when:

$$\frac{\partial}{\partial \lambda} f\left(\mathbf{x}^{(t)} + \lambda \mathbf{d}^{(t)}\right) = \frac{\partial f\left(\mathbf{x}^{(t)} + \lambda \mathbf{d}^{(t)}\right)}{\partial\left(\mathbf{x}^{(t)} + \lambda \mathbf{d}^{(t)}\right)} \frac{\partial\left(\mathbf{x}^{(t)} + \lambda \mathbf{d}^{(t)}\right)}{\partial \lambda} = 0 \tag{44}$$

which gives:

$$g(\mathbf{x}^{(t+1)})^T \mathbf{d}^{(t)} = 0 \tag{45}$$

where $\mathbf{g}(\mathbf{x})$ is the gradient (derivative of $f(\mathbf{x})$) and where $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \lambda \mathbf{d}^{(t)}$. Thus the gradient at the new point $\mathbf{x}^{(t+1)}$ is orthogonal to the previous search direction. *Hence using the local gradient at each point can lead to a trajectory involving oscillatory behaviour as it approaches the minimum*: the algorithm may take many steps to converge.
The solution to this problem is as follows. *We choose successive search directions* $\mathbf{d}^{(t)}$ *so that, at each step of the algorithm, we retain a component parallel to the previous search direction.* In other words, rather than the:

$$\mathbf{d}^{(t+1)} = -\mathbf{g}^{(t+1)} \tag{46}$$

of the gradient descent method we use:

$$\mathbf{d}^{(t+1)} = -\mathbf{g}^{(t+1)} + \beta \mathbf{d}^{(t)} \tag{47}$$

instead (think of this as smoothing out the trajectory). Rather than the notations:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \lambda \mathbf{d}^{(t)} \tag{48}$$

and:

$$\mathbf{d}^{(t+1)} = -\mathbf{g}^{(t+1)} + \beta \mathbf{d}^{(t)} \tag{49}$$

we will, instead, use the conventional notation used in this context, i.e.

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{d}_j \tag{50}$$

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j \tag{51}$$

Thus j is the iteration index, $\mathbf{x}_j$ is the position at iteration j, $\mathbf{d}_j$ is the corresponding direction vector and α_j and β_j are parameters involved in the updates of the position and direction vector respectively (so we use the conventional notation β_j in place of our previous usage $\lambda^{(t)}$).
For reasons which are apparent later on, this method will work if successive search directions are *conjugate*, that is, if $\mathbf{d}_j$ is the current search direction and $\mathbf{d}_{j+1}$ the search direction after the next update, then:

$$[\mathbf{d}_{j+1}]^T \mathbf{H} \mathbf{d}_j = 0 \tag{52}$$

where $\mathbf{H}$ is a shorthand for *the Hessian matrix*. Successive search directions which satisfy (52) are said to be *conjugate*.

To introduce the conjugate gradient algorithm we will first consider the case of a quadratic objective function:

$$f(\mathbf{x}) = f_0 + \mathbf{b}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} \tag{53}$$

in which the parameters $\mathbf{b}$ and $\mathbf{H}$ are constant (for a quadratic function the second derivatives of the function are zero or constant so the Hessian only has zero or constant components). The gradient of this function is then:

$$\mathbf{g}(\mathbf{x}) = \mathbf{b} + \mathbf{H} \mathbf{x} \tag{54}$$

and a solution, $\mathbf{x}^\star$, therefore satisfies:

$$\mathbf{b} + \mathbf{H} \mathbf{x}^\star = \mathbf{0} \tag{55}$$

since, at the optimum, the gradient in all directions will be zero.
Suppose the dimensionality of the space is n (i.e. number of components in the vector $\mathbf{x}$). The space can be *spanned* by n independent vectors i.e. this set of vectors forms a *basis* for this space (any vector in this space can be expanded as a linear combination of these vectors). Thus, in $n = 3$ dimensions, any arbitrary vector can be written as a combination of the 3 vectors $\mathbf{i} = (1, 0, 0)$, $\mathbf{j} = (0, 1, 0)$ and $\mathbf{k} = (0, 0, 1)$. As noted, we further assume that these n linearly independent vectors $\mathbf{d}_i$ are *mutually conjugate* with respect to $\mathbf{H}$ i.e.

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_i = 0 \qquad j \neq i \tag{56}$$

Suppose we start from a position $\mathbf{x}_1$ and wish to get to the minimum $\mathbf{x}^\star$ of the objective function. The difference between these two vectors is also a vector lying within this space and hence it can be expressed as a linear combination of the conjugate direction vectors. Thus, iterating:

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{d}_j \tag{57}$$

we get:

$$\mathbf{x}_j = \mathbf{x}_1 + \sum_{i=1}^{j-1} \alpha_i \mathbf{d}_i \tag{58}$$

and eventually:

$$\mathbf{x}^\star - \mathbf{x}_1 = \sum_{i=1}^{n} \alpha_i \mathbf{d}_i \tag{59}$$

where the α_j are step lengths multiplying the direction vectors.

In order to find an expression for the α we multiply both sides of (59) by $\mathbf{d}_j^T\mathbf{H}$ and make use of (55) to get:

$$-\mathbf{d}_j^T(\mathbf{b} + \mathbf{H}\mathbf{x}_1) = \sum_{i=1}^{n} \alpha_i \mathbf{d}_j^T\mathbf{H}\mathbf{d}_i \tag{60}$$

From (56) we therefore see that the terms on the right hand side of this equation are all zero except when $j = i$ so that:

$$-\mathbf{d}_j^T(\mathbf{b} + \mathbf{H}\mathbf{x}_1) = \alpha_j \mathbf{d}_j^T\mathbf{H}\mathbf{d}_j \tag{61}$$

Thus we can get an explicit expression for α_j:

$$\alpha_j = -\frac{\mathbf{d}_j^T(\mathbf{b} + \mathbf{x}_1)}{\mathbf{d}_j^T\mathbf{H}\mathbf{d}_j} \tag{62}$$

The ability to obtain such an explicit expression for the step length α_i originated from the use of conjugate direction vectors: without conjugacy we would not be able to decouple the step lengths α_i. We can write (62) in a more direct form as follows. From (58) we have:

$$\mathbf{d}_j^T\mathbf{H}\mathbf{x}_j = \mathbf{d}_j^T\mathbf{H}\mathbf{x}_1 \tag{63}$$

where we again use the conjugacy condition (56). We add $\mathbf{d}_j^T\mathbf{b}$ to both sides of (63) and use (54) to get:

$$\mathbf{d}_j^T\left(\mathbf{b} + \mathbf{H}\mathbf{x}_1\right) = \mathbf{d}_j^T\left(\mathbf{b} + \mathbf{H}\mathbf{x}_j\right) = \mathbf{d}_j^T\mathbf{g}_j \tag{64}$$

where $\mathbf{g}_j = \mathbf{g}(\mathbf{x}_j)$. Thus we can write:

$$\alpha_j = -\frac{\mathbf{d}_j^T\mathbf{g}_j}{\mathbf{d}_j^T\mathbf{H}\mathbf{d}_j} \tag{65}$$

The question then arises: *how do we find a set of mutually conjugate direction vectors?* The first direction vector is chosen as the negative gradient (if minimising, say):

$$\mathbf{d}_1 = -\mathbf{g}_1 \tag{66}$$

Each successive direction is a linear combination of the current steepest descent direction and the previous direction vector:

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j \tag{67}$$

To satisfy the conjugacy condition $\mathbf{d}_j^T\mathbf{H}\mathbf{d}_i = 0$ for $j \neq i$. Thus since $j + 1 \neq j$ we get (post-multiplying by $\mathbf{H}\mathbf{d}_j$):

$$\mathbf{d}_{j+1}\mathbf{H}\mathbf{d}_j = 0 = -\mathbf{g}_{j+1}\mathbf{H}\mathbf{d}_j + \beta_j \mathbf{d}_j\mathbf{H}\mathbf{d}_j \tag{68}$$

or:

$$\beta_j = \frac{\mathbf{g}_{j+1}^T\mathbf{H}\mathbf{d}_j}{\mathbf{d}_j^T\mathbf{H}\mathbf{d}_j} \tag{69}$$

Hence from (65) and (69) we see that we have a means of successively updating the direction vectors and the step lengths taken in each direction while maintaining conjugacy. From position $\mathbf{x}_j$ we use the current $\mathbf{g}_j$, $\mathbf{d}_j$ and $\mathbf{H}$ to find α_j from (65). We then determine $\mathbf{x}_{j+1}$ (from (50)) and hence $\mathbf{g}_{j+1}$. Hence, knowing $\mathbf{g}_{j+1}$, $\mathbf{d}_j$ and $\mathbf{H}$ we can find β_j from (69). Hence we can find $\mathbf{d}_{j+1}$ from (67). We are now at a new position $\mathbf{x}_{j+1}$ with a corresponding $\mathbf{d}_{j+1}$.

4.12 Comments and Extensions

So far our discussion of the conjugate gradient technique has been restricted to a quadratic error function. *How might we extrapolate this approach to a general nonlinear optimisation task?* In the neighbourhood of a point $\mathbf{x}$ the error function of an arbitrary error function is approximately quadratic from the Taylor series exansion:

$$f(\mathbf{x}+\delta\mathbf{x}) = f(\mathbf{x}) + (\nabla f)^T\delta\mathbf{x} + \frac{1}{2}(\delta\mathbf{x})^T\mathbf{H}\delta\mathbf{x} + O(\delta\mathbf{x}^3) \tag{70}$$

so we could expect the conjugate gradient method carries over to general nonlinear functions. In fact this method can be applied to arbitrary nonlinear functions but the full formal proof is beyond this course.

However, for a general nonlinear objective function there is an issue to be considered: for a quadratic function the components in the Hessian are constant or zero but for a general nonlinear function the components will depend on $\mathbf{x}$. Both α_j and β_j depend on the Hessian from (65) and (69) respectively. It would be computationally expensive to evaluate the Hessian at each step of the algorithm. How do we circumvent this problem? In fact, it turns out that α_j and β_j can be found without an explicit knowledge of the Hessian using several methods (the proof of these is beyond the course):

1. Determination of the coefficient β_j. Using further manipulations we can get:

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1}-\mathbf{g}_j)}{\mathbf{d}_j^T(\mathbf{g}_{j+1}-\mathbf{g}_j)} \tag{71}$$

This is known as the *Hestenes-Stiefel form.*

or:

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1}-\mathbf{g}_j)}{\mathbf{g}_j^T\mathbf{g}_j} \tag{72}$$

which is known as the *Polak-Ribiere form.*

We can simplify the latter further giving:

$$\beta_j = \frac{\mathbf{g}_{j+1}^T\mathbf{g}_{j+1}}{\mathbf{g}_j^T\mathbf{g}_j} \tag{73}$$

which is known as the *Fletcher-Reeves form.*

If the objective function is quadratic then these three forms are equivalent, but for a general nonlinear function they will give different behaviours. The Polak-Ribiere form appears to give the best results.

2. Determination of the coefficient α_j. For a quadratic objective function the components of the Hessian H are constants and we could use:

$$\alpha_j = -\frac{d_j^T g_j}{d_j^T H d_j} \tag{74}$$

as before. However, in generality, the Hessian could change as we proceed (it would be $\mathbf{x}$ dependent) and we would have to repeatedly re-calculate this matrix. *Thus for a general nonlinear objective function we tend to use a one-dimensional line minimisation (in α) in the direction* $\mathbf{d}_j$. That is, rather than use this explicit formula we use a one-dimensional minimiser on the scalar value α in $f(\mathbf{x}^{(t)} + \alpha\mathbf{d}^{(t)})$. In summary the algorithm works as follows:

Summary: The Conjugate Gradient Algorithm.

1. Choose an initial point $\mathbf{x}_1$.
2. Evaluate the gradient vector $\mathbf{g}_1$ and set the initial search direction $\mathbf{d}_1 = -\mathbf{g}_1$.
3. At each step j minimise $f(\mathbf{x}_j+\alpha\mathbf{d}_j)$ with respect to the one-dimensional variable α to give $\mathbf{x}_{j+1} = \mathbf{x}_j+\alpha_{\min}\mathbf{d}_j$.
4. Test to see if the stopping criterion is satisfied.
5. Evaluate the new gradient $\mathbf{g}_{j+1}$.
6. Evaluate the new search direction using $\mathbf{d}_{j+1} = -g_{j+1} + \beta_j\mathbf{d}_j$ where β_j is found from the Hestenes-Stiefel formula (71), the Polak-Ribiere formula (72) or the Fletcher-Reeves formula (73).
7. Set $j = j + 1$ and return to 3.

4.13 Newton's Method and the Levenberg-Marquardt Method

The method of Newton is an algorithmic procedure in which the steepest descent (or ascent) direction is obtained by premultiplication by the inverse of the Hessian matrix $\mathbf{H}(\mathbf{x})$. The motivation for the method comes from a truncated approximation of the Taylor series expansion. Specifically we retain only the first three terms up to the quadratic expression in $\mathbf{x}$. This approximation is sound since, as the update of the solution $\mathbf{x}$ approaches the minimum the higher order terms are negligible and the quadratic term is most dominant. Thus suppose $q(\mathbf{x}_j)$ is this approximation to the objective function $f(\mathbf{x}_j)$, at a new point $\mathbf{x}_j$ at iteration j, then:

$$q(\mathbf{x}) = f(\mathbf{x}_j) + \nabla f(\mathbf{x}_j)^T(\mathbf{x} - \mathbf{x}_j) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_j)^T\mathbf{H}(\mathbf{x} - \mathbf{x}_j)$$

where $\mathbf{H} \equiv \mathbf{H}(\mathbf{x}_j)$ is the Hessian matrix for f at $\mathbf{x}_j$. A necessary condition for a minimum of this quadratic approximation q, is that $\nabla q(\mathbf{x}) = \mathbf{0}$. ∇ is taken with respect to $\mathbf{x}$ so we could also write $\nabla_{\mathbf{x}} q(\mathbf{x}) = 0$. Thus:

$$\nabla q(\mathbf{x}) = \nabla f(\mathbf{x}_j) + \mathbf{H}(\mathbf{x} - \mathbf{x}_j) = 0 \tag{75}$$

so, premultiplying by the inverse of the Hessian matrix $\mathbf{H}(\mathbf{x}_j)$, and letting $\mathbf{x} = \mathbf{x}_{j+1}$ be our new estimated position of this minimum, we get:

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \mathbf{H}(\mathbf{x}_j)^{-1}\nabla f(\mathbf{x}_j)$$

This defines the recursive form of the method for the multidimensional case Newton method. This method has some similarities to the Newton-Raphson method which is based on a *linear* (two term) approximation for a Taylor series expansion and and update of the form $x_{j+1} = x_j - f(x_j)/f'(x_j)$, where $f(x)$ is the function to be minimised. If you have experience with the Newton-Raphson method you will know that it can generally settled in a sub-optimal solution if the intial starting point, x_0 is poorly chosen, and this is also true for the Newton method.
Thus for the multidimensional Newton method we can therefore find the following:

- The method will fail if the Hessian matrix cannot be inverted
- If the method is initialised poorly then the algorithm may get caught in a sub-optimal solution, also
- The search direction may not necessarily be a descent direction.

However, it is certainly appealing as a method since it will move rapidly through a multidimensional space towards a solution. In fact, it is possible to show that if Newton's method is initialised close enough to a minimum, with a positive definite Hessian $\mathbf{H}$, then the method will converge with *quadratic efficiency* toward this solution, that is, the difference between the current position and the final solution follows a quadratic dependency against the iteration index

We now discuss a modification of Newton's method that overcomes the above limitations and guarantees convergence, irrespective of the start point. Given $\mathbf{x}$, let us consider the direction $\mathbf{d}_j = -\mathbf{B}\nabla f(\mathbf{x})$, where $\mathbf{B} = (\epsilon\mathbf{I} + \mathbf{H})^{-1}$ will be a symmetric positive definite matrix. The successor position in the space is then given by:

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \lambda \mathbf{d}_j$$

where we might find λ by a one-dimensional line search (see next Section). The scalar $\epsilon_j \geq 0$ is found as follows. Let us select a small but positive variable δ such that $\delta > 0$. Then the scalar ϵ is the smallest value that would make all the eigenvalues of matrix $(\epsilon\mathbf{I}+\mathbf{H})$ greater than or equal to δ. Since the eigenvalues of $\mathbf{B}$ would therefore all be positive it follows from matrix theory that this matrix is positive definite and it is also invertable. A matrix which has all its eigenvalues positive is a positive definite matrix. As a consequence, since $\mathbf{B}$ is positive definite, at any intermediate step $\mathbf{d}_j = -\mathbf{B}\nabla f(\mathbf{x})$ is a descent direction. This statement follows from taking $\mathbf{d}_j = -\mathbf{B}\nabla f(\mathbf{x})$, pre-multiplying both sides by $\nabla f(\mathbf{x})^T$, hence $\nabla f(\mathbf{x})^T \mathbf{d}_j = -\nabla f(\mathbf{x})^T \mathbf{B} \nabla f(\mathbf{x}) < \mathbf{0}$ from the positive definiteness condition. As the $\mathbf{x}_j$ approach the solution, ϵ approaches 0 and therefore it approaches the Newton method in the vicinity of the solution. Thus it generalises the Newton method and also enjoys its advantage of having an order-two or quadratic rate of convergence.

This algorithmic scheme of determining the new iterate $\mathbf{x}_{j+1}$, from the current iterate $\mathbf{x}_j$, is based on finding the solution to the system of equations:

$$[\epsilon_j \mathbf{I} + \mathbf{H}(\mathbf{x}_j)]\,(\mathbf{x}_{j+1} - \mathbf{x}_j) = -\nabla f(\mathbf{x}_j) \tag{76}$$

in lieu of (75). This approach is commonly known as the **Levenberg-Marquardt method**. The full operational description of the method is as follows. Given the current iterative solution $\mathbf{x}_j$ and a parameter $\epsilon_j \geq 0$ at step j, we must first ascertain the positive definiteness of the matrix $M = \epsilon_j \mathbf{I} + \mathbf{H}(\mathbf{x}_j)$. This is actually achieved by what is known as a *Cholesky factorisation*. The details of this step are a little beyond the course but the Cholesky factorisation amounts to writing $\mathbf{M} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is a triangular matrix with positive diagonal elements and non-zero components in the lower triangle ($\mathbf{L}^T$ is the transpose matrix). If this step is unsuccessful, then we multiply by a factor of 4, and repeat until the factorisation is possible. We then solve the equations in via (76), which is readily achieved by exploiting the triangularity of $\mathbf{L}$. Incrementing j by 1 we iterate this procedure until convergence, when zero gradient has been obtained.

4.14 The Davidon-Fletcher-Powell Method (non-examinable)

There are further algorithmic variants on the Newton method. The method we now briefly describe is an example of a *quasi-Newton method*. The basic idea of this method is that rather than using the inverse-Hessian matrix, we use a running approximation to the inverse-Hessian which we denote by the matrix $\mathbf{D}$. This approximation to the inverse-Hessian becomes exact as we approach the solution. Though we give no details the essential steps of this procedure are a search direction:

$$\mathbf{d}_j = -\mathbf{D}_j \nabla f(\mathbf{x}_j)$$

for an update step:

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \lambda \mathbf{d}_j$$

where:

$$\mathbf{D}_{j+1} = \mathbf{D}_j + \frac{1}{(\mathbf{p}_j^T \mathbf{q}_j)} \mathbf{p}_j \mathbf{p}_j^T + \frac{1}{(\mathbf{q}_j^T \mathbf{D}_j \mathbf{q}_j)} \mathbf{D}_j \mathbf{q}_j \mathbf{q}_j^T \mathbf{D}_j$$

and:

$$\begin{aligned} \mathbf{p}_j &= \lambda_j \mathbf{d}_j = \mathbf{x}_{j+1} - \mathbf{x}_j \\ \mathbf{q}_j &= \nabla f(\mathbf{x}_{j+1}) - \nabla f(\mathbf{x}_j) \end{aligned}$$

4.15 One-dimensional line search (non-examinable)

A one-dimensional line search uses or finds a descent direction $\mathbf{d}$ along which an objective function f will be reduced. It computes a scalar step size parameter λ which determines how far the position vector $\mathbf{x}$ should be moved along that direction i.e. $\mathbf{x} \rightarrow \mathbf{x} + \lambda \mathbf{d}$. Line search routines are a detail which is a little beyond the course but the following are applicable:

- Secant method
- Newton-Raphson method
- Nelder-Mead Method
- Polynomial interpolation methods
- Golden Search

We mentioned the Newton-Raphson method above and the Nelder-Mead method earlier. Of these techniques, the Golden Search method is very efficient. The function $f(\lambda)$ is evaluated at three points in λ, say λ_1, λ_2 and λ_3. If $\lambda_1 < \lambda_2 < \lambda_3$ and $f(\lambda_2)$ is smaller than $f(\lambda_1)$ and $f(\lambda_3)$ then the minimum lies between λ_1 and λ_3. We now introduce a new value λ_4. This is placed in the larger of the two intervals $\{\lambda_1, \lambda_2\}$ or $\{\lambda_2, \lambda_3\}$, say it is the interval $\{\lambda_2, \lambda_3\}$. If $f(\lambda_4)$ is smaller than $f(\lambda_2)$ and $f(\lambda_3)$ then the minimum is on this interval and the new search interval is $\{\lambda_2, \lambda_4, \lambda_3\}$, if not, we follow an alternative procedure to narrow the search. Basically, Golden Search follows an optimised process to continuously narrow the search interval.

4.16 Constrained Optimisation

4.16.1 General Theory: Equality Constraints

For purposes of illustration let us consider a two-variable minimisation problem with an objective function:

$$z = f(x, y)$$

and with the variables x and y constrained by the relationship:

$$g(x, y) = 0$$

In some instance it might be possible to use the constraint $g(x, y) = 0$ to solve for y as a function x, thus $y = h(x)$. However, it is frequently the case that we cannot obtain such an explicit form for y. But let us suppose that $y = h(x)$ exists in some implicit sense, then:

$$z = f(x, y) = f\left(x, h(x)\right)$$

which is a function of the one independent variable x. For a minimum of z the necessary condition is:

$$\frac{dz}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} = 0$$

However, from implicit differentiation we can write:

$$\frac{dy}{dx} = \frac{d}{dx}h(x) = -\left[\frac{\partial g}{\partial y}\right]^{-1}\frac{\partial g}{\partial x}$$

so:

$$\frac{\partial f}{\partial x} - \frac{\partial f}{\partial y}\left[\frac{\partial g}{\partial y}\right]^{-1}\frac{\partial g}{\partial x} = 0$$

Let us suppose we are at the minimum so $x = x^\star$ and $y = y^\star$ and we define:

$$\lambda = -\left[\frac{\partial g}{\partial y}(x,y)\right]^{-1}\left[\frac{\partial f}{\partial y}(x,y)\right]$$

then at the minimum we have:

$$\begin{aligned} g(x,y) &= 0 \\ \frac{\partial f}{\partial x}(x,y) + \lambda\frac{\partial g}{\partial x}(x,y) &= 0 \\ \frac{\partial f}{\partial y}(x,y) + \lambda\frac{\partial g}{\partial y}(x,y) &= 0 \end{aligned}$$

These three conditions can be generated if we consider *the Lagrange function*:

$$F(x,y,\lambda) = f(x,y) + \lambda g(x,y)$$

i.e. the sum of the objective function and the product of λ with the constraint. The necessary conditions for a constrained minimum of $f(x,y)$ can then be written:

$$\begin{aligned} \frac{\partial F}{\partial x}(x,y,\lambda) &= \frac{\partial f}{\partial x}(x,y) + \lambda\frac{\partial g}{\partial x}(x,y) = 0 \\ \frac{\partial F}{\partial y}(x,y,\lambda) &= \frac{\partial f}{\partial y}(x,y) + \lambda\frac{\partial g}{\partial y}(x,y) = 0 \\ \frac{\partial F}{\partial \lambda}(x,y,\lambda) &= g(x,y) = 0 \end{aligned}$$

Example 1: Find the minimum of x^2+y^2 subject to $x+y=4$. Here $f(x,y) = x^2+y^2$ and $g(x,y) = 4-x-y = 0$.

The *Lagrange function* is:

$$F(x,y) = x^2 + y^2 + \lambda\,(4 - x - y)$$

so:

$$\begin{aligned}\frac{\partial F}{\partial x} &= 2x - \lambda = 0 \\ \frac{\partial F}{\partial y} &= 2y - \lambda = 0 \\ \frac{\partial F}{\partial \lambda} &= 4 - x - y = 0\end{aligned}$$

Thus the solution is: $x = y = 2$, $\lambda = 4$.

This approach can be generalised to m variables. Thus if we define the Lagrange function:

$$F(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x})$$

then the necessary conditions for a constrained minimum of $f(\mathbf{x})$ are:

$$\begin{aligned}\frac{\partial F}{\partial x_j} &= \frac{\partial f}{\partial x_j} + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i}{\partial x_j} = 0 \qquad j = 1, 2, \ldots, n \\ \frac{\partial F}{\partial \lambda_i} &= g_i(\mathbf{x}) = 0 \qquad i = 1, \ldots, m\end{aligned} \tag{77}$$

At a constrained minimum $x^\star$:

$$f(\mathbf{x}^\star + \mathbf{h}) - f(\mathbf{x}^\star) \geq 0$$

where $\mathbf{h}$ satisfies $g_i(\mathbf{x}^\star + \mathbf{h}) = 0$ for all i.

Thus we shall have:

$$F(x^\star + h) - F(x^\star) = \sum_{j=1}^{m} \frac{\partial F}{\partial x_j} h_j + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} h_i \frac{\partial^2 F}{\partial x_i \partial x_j} h_j + \ldots \geq 0$$

where the derivatives are evaluated at $x^\star$, $\lambda^\star$.

Thus using (78) above:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} h_i \frac{\partial^2 F}{\partial x_i \partial x_j} h_j \geq 0$$

for all h which satisfy the constraints.

Thus a sufficient condition for a constrained minimum is that the Hessian of $F(\mathbf{x})$ is a positive definite matrix.

Example 2: Show that $(4, 1)$ is a constrained minimum of $x_1^2 + 4x_2^2$ subject to the constraint $x_1 + x_2 = 5$. With:

$$F(x, \lambda) = x_1^2 + 4x_2^2 + \lambda (5 - x_1 - x_2)$$

we have see that:

$$\frac{\partial F}{\partial x_1} = 0, \qquad \frac{\partial F}{\partial x_2} = 0$$

when $x_1 = 4$, $x_2 = 1$ and $\lambda = 8$. Thus it is a minimum and the Hessian matrix is:

$$\begin{bmatrix} 2 & 0 \\ 0 & 8 \end{bmatrix}$$

which is positive definite, hence establishing the result.

4.16.2 Inequality Constraints

We will extend the idea of Lagrange multipliers to the case of inequality constraints. Thus we consider the general problem:

$$\text{Minimise} \qquad f(\mathbf{x})$$

subject to the m constraints:

$$g_i(x) \leq b_i$$

where $i = 1, 2, \ldots, m$. This formulation is entirely general since a constraint $h(x) \geq c$ can be written $-h(x) \leq -c$.

The inequality constraints can be transformed into equality constraints by the addition of a non-negative *slack variable* u_i^2 (note that u_i^2 will always be positive). This we obtain:

$$g_i(x) + u_i^2 = b_i$$

i.e.:

$$g_i(x) + u_i^2 - b_i = 0$$

Thus the problem is to minimise $f(x)$ subject to the m equality constraints $g_i(x) + u_i^2 - b_i = 0$. In line with the previous section we form the Lagrange function:

$$F(\mathbf{x}, \lambda, \mathbf{u}) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \left[g_i(\mathbf{x}) + u_i^2 - b_i \right]$$

The necessary conditions to be satisfied at a stationary point are:

$$\frac{\partial F}{\partial x_j} = 0 = \frac{\partial f}{\partial x_j} + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i}{\partial x_j}; \qquad j = 1, 2, \ldots, n \tag{78}$$

$$\frac{\partial F}{\partial \lambda_i} = 0 = g_i(x) + u_i^2 - b_i; \qquad i = 1, 2, \ldots, m \tag{79}$$

$$\frac{\partial F}{\partial u_i} = 0 = 2\lambda_i u_i \qquad i = 1, 2, \ldots, m \tag{80}$$

The last condition when multiplied by $u_i/2$ gives:

$$\lambda_i u_i^2 = 0$$

i.e.:

$$\lambda_i \left[b_i - g_i(\mathbf{x}) \right] = 0, \qquad i = 1, 2, \ldots, m \tag{81}$$

The first two equations of (78,79,80) and equation (81) are necessary conditions for a constrained minimum $\mathbf{x}^\star$. Equation (79) is just a restatement of the constraints $g_i(x) \leq 0$. Condition (81) states that one of λ_i or $(b_i - g_i(\mathbf{x}^\star))$ is zero. If λ_i is not zero then $g_i(x^\star) = b_i$, and the constraint is said to be *active* and satisfied as an equation. On the other hand if the constraint is satisfied as a strict inequality so $g_i(x^\star) < b$ then the corresponding Lagrange multiplier λ_i is zero. As far as the constrained minimum is concerned, if $g_i(x^\star) < b_i$, this constraint is inactive and could be ignored, and the coresponding λ_i is zero. Of course, initially we do not know which constraints can be ignored.

At the constrained minimum the constraint $\lambda_i \geq 0$ must also be satisfied.

Suppose equations (78), (79) and (81) are satisfied at the point $(x^\star, \lambda^\star, u^\star)$. If the actual constrained function minimum is $z = f(x^\star)$ then we can regard z as a function of the b_i, in that changing the b_i will modify the constraints and so will change z.

We will show that:

$$\frac{\partial z}{\partial b_i} = -\lambda_i^\star$$

$$\frac{\partial z}{\partial b_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} . \frac{\partial x_j}{\partial b_i}$$

where the partial derivatives are evaluated at $x^\star$.

Since $g_k(x) + u_k^2 = b_k$:

$$\frac{\partial g_k}{\partial b_i} = \sum_{j=1}^{n} \frac{\partial g_k}{\partial x_j} . \frac{\partial x_j}{\partial b_i} = \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases}$$

Thus:

$$\frac{\partial z}{\partial b_i} + \sum_{k=1}^{m} \lambda_k^\star \frac{\partial g_k}{\partial b_i} = \frac{\partial z}{\partial b_i} + \lambda_i^\star = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_j} + \sum_{k=1}^{m} \lambda_k^\star \frac{\partial g_k}{\partial x_j} \right) \frac{\partial x_j}{\partial b_i}$$

Thus:

$$\frac{\partial z}{\partial b_i} = -\lambda_i^\star$$

As b_i is increased the constraint region is enlarged which cannot result in a higher value for z, the minimum of $f(\mathbf{x})$ within this region, although it could reduce z. So we shall have:

$$\frac{\partial z}{\partial b_i} \leq 0$$

i.e. $\lambda_i^\star \geq 0$. The necessary conditions to be satisfied at the minimum of $f(x)$ where x must satisfy $g_i(x) \leq b_i$ $(i = 1, 2, \ldots, m)$ are that we can find x and λ which satisfy:

$$\begin{aligned} \frac{\partial f}{\partial x_j} + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i}{\partial x_j} &= 0 \qquad j = 1, 2, \ldots, n. \\ g_i(x) &\leq b_i \qquad i = 1, 2, \ldots, m \\ \lambda_i \left[g_i(x) - b_i \right] &= 0 \qquad i = 1, 2, \ldots, m \\ \lambda_i &\geq 0 \qquad i = 1, 2, \ldots, m \end{aligned} \tag{82}$$

The sign of λ_i is reversed if we are dealing with a maximum. These conditions (in (83)) are known as *the Kuhn-Tucker conditions.*

Example 3: Find the Kuhn-Tucker conditions for the minimum of:

$$3x_1^2 + 4x_1x_2 + 5x_2^2$$

subject to $x_1 \geq 0$, $x_2 \geq 0$ and $x_1 + x_2 \geq 4$.

This problem can be restated as minimise:

$$3x_1^2 + 4x_1x_2 + 5x_2^2$$

suibject to:

$$\begin{aligned} -x_1 &\leq 0 \\ -x_2 &\leq 0 \\ -x_1 - x_2 &\leq -4 \end{aligned}$$

The Lagrange function $F(x, u, \lambda)$ is:

$$F = 3x_1^2 + 4x_1x_2 + 5x_2^2 + \lambda_1(u_1^2 - x_1) + \lambda_2(u_2^2 - x_2) + \lambda_3(u_3^2 - x_1 - x_2 + 4)$$

The necessary conditions are therefore:

$$\begin{aligned} 6x_1 + 4x_2 - \lambda_1 - \lambda_3 &= 0 \\ 4x_1 + 10x_2 - \lambda_2 - \lambda_3 &= 0 \\ -x_1 &\leq 0 \\ -x_2 &\leq 0 \\ -x_1 - x_2 &\leq -4 \\ \lambda_1 x_1 &= 0 \\ \lambda_2 x_2 &= 0 \\ \lambda_3(4 - x_1 - x_2) &= 0 \\ \lambda_1, \lambda_2, \lambda_3 &\geq 0 \end{aligned}$$

It is easy to verify that these conditions are satisfied by $x_1 = 3$, $x_2 = 1$, $\lambda_1 = 0$, $\lambda_2 = 0$, $\lambda_3 = 22$. The minimum of the function is 44 at $x_1 = 3$, $x_2 = 1$. The contours of $f(x)$ are the ellipses:

$$3x_1^2 + 4x_1x_2 + 5x_2^2 = c$$

The unconstrained minimum of $f(x)$ is zero at the origin.

4.17 Penalty Function Methods

The idea behind penalty function methods is to transform the problem of minimising:

$$z = f(\mathbf{x})$$

subject to certain constraints on x into the problem of finding the unconstrained minimum of:

$$Z = f(\mathbf{x}) + P(\mathbf{x})$$

$P(\mathbf{x})$ is called *the penalty function*. It should have the property that if the constraints are violoated then a high value will be assigned to Z so that the minimum of Z will not arise outside the constrained region. Formally let us suppose the constrained problem in the form:

$$\textit{Minimise} \qquad z = f(\mathbf{x})$$

subject to:

$$c_j(\mathbf{x}) \geq 0 \qquad j = 1, 2, \ldots, m$$

A useful form for $P(\mathbf{x})$ is then:

$$P(\mathbf{x}) = r \sum_{j=1}^{m} \left(\frac{1}{c_j(\mathbf{x})} \right)$$

where r is positive. The function $Z = \phi(\mathbf{x}, r)$ then takes the form:

$$Z = \phi(\mathbf{x}, r) = f(\mathbf{x}) + r \sum_{j=1}^{m} \left(\frac{1}{c_j(\mathbf{x})} \right)$$

The effect of $P(\mathbf{x})$ is to create a steep sided ridge along each of the constraint boundaries. Thus if we start with a feasible point and try to find an unconstrained minimum of $\phi(\mathbf{x}, r)$ it will certainly lie within the feasible region of our constrained problem. By giving r a suitably small value $P(\mathbf{x})$ is small at the minimum point, we may be able to make this unconstrained minimum point for $\phi(\mathbf{x}, r)$ coincide with the constrained minimum of $f(\mathbf{x})$.

Example 4: Minimise:

$$f(x) = x$$

subject to:

$$x \geq 2$$

or, equivalently: $x - 2 \geq 0$. The minimum is clearly 2, when $x = 2$. To approach this problem using the penalty function method consider the function:

$$\phi(x, r) = x + \frac{r}{(x-2)}$$

The effect of the penalty function is diminished by decreasing r. In fact we see that:

$$\frac{d\phi}{dx} = 1 - \frac{r}{(x-2)^2}$$

Thus when:

$$\frac{d\phi}{dx} = 0 \qquad\qquad (x-r)^2 = r$$

so that:

$$\begin{aligned} x &= 2 \pm \sqrt{r} \\ \frac{d^2\phi}{dx^2} &= \frac{2r}{(x-2)^3} \end{aligned}$$

and the minimum arises when $x = 2 + \sqrt{r}$, within the constrained region. Clearly as $r \to 0$ the unconstrained minimum of $\phi(x, r)$ is at $x = 2$.

In general it is not possible to locate the position of the minimum of $\phi(x, r)$ analytically as a simple function of r, and it is necessary to resort to numerical techniques instead.

Example 5: As a second example (again with an analytical solution), consider the following problem, minimise:

$$f(x_1, x_2) = \frac{1}{3}(x_1 + 1)^3 + x_2$$

where $x_1 - 1 \geq 0$ and $x_2 \geq 0$. Thus we use:

$$\phi(\mathbf{x}, r) = \frac{1}{3}(x_1 + 1)^3 + x_2 + r\left(\frac{1}{x_1 - 1} + \frac{1}{x_2}\right)$$

The necessary conditions for the minimum of ϕ give:

$$\begin{aligned} (x_1 + 1)^2 - \frac{r}{(x_1 - 1)^2} &= 0 \\ 1 - \frac{r}{x_2^2} &= 0 \end{aligned}$$

with solutions:

$$x_1(r) = (1 + \sqrt{r})^{1/2} \qquad\qquad x_2(r) = \sqrt{r}$$

The minimum value of $\phi(x, r)$ is then:

$$\begin{aligned} \phi^\star(r) &= \left\{\frac{1}{3}\left[(1+\sqrt{r})^{1/2} + 1\right]^3 + \sqrt{r} + r\left[\frac{1}{\sqrt{r}} + \frac{1}{(1+\sqrt{r})^{1/2} - 1}\right]\right\} \\ &= \left\{\frac{1}{3}\left[(1+\sqrt{r})^{1/2} + 1\right]^3 + \sqrt{r} + r\left[\frac{1}{\sqrt{r}} + \frac{(1+\sqrt{r})^{1/2} + 1}{\sqrt{r}}\right]\right\} \\ &= \left\{\frac{1}{3}\left[(1+\sqrt{r})^{1/2} + 1\right]^3 + \sqrt{r} + r\left[1 + 1 + (1+\sqrt{r})^{1/2}\right]\right\} \end{aligned}$$

Thus we see that as $r \to 0$

$$x_1(r) \to 1, \qquad x_2(r) \to 0$$

and:

$$\phi^\star(r) \to f(1, 0) = \frac{8}{3}$$

4.18 The SUMT Method

The results of the previous section show that we can solve a constrained minimisation problem, minimise $f(\mathbf{x})$ subject to $c_j(\mathbf{x}) \geq 0$, by solving the sequence of unconstrained problems:

$$\textit{Minimise} \;\; \phi(\mathbf{x}, r) = f(\mathbf{x}) + r \sum_{j=1}^{m} \frac{1}{c_j(\mathbf{x})}$$

with a sequence of r values which tend to zero.

The *sequential unconstrained minimisation technique (SUMT)* was originally suggested by Carroll in 1961 and implements this idea. Given a function $f(\mathbf{x})$ and constraint functions $c_j(\mathbf{x}) \geq 0$ we choose an initial value for r namely r_0 to form the function $\phi(\mathbf{x}, r_0)$. This function is minimised with the minimisation treated as an unconstrained minimisation problem. The conjugate gradient method mentioned earlier would be an example of an approach that could be used to do this.

Having found the minimum of $\phi(\mathbf{x}, r_0)$ we must now reduce the value of r. A simple and effective way to do this is simply to choose $r_1 = r_0/c$ where c is a constant greater than 1. A common choice for c is 10. We then minimise $\phi(\mathbf{x}, r_1)$ again using the conjugate gradient technique. This process is iterated. Thus we minimise $\phi(\mathbf{x}, r_{k+1})$ where $r_{k+1} = r_k/c$ and the r_k form a decreasing sequence tending to zero. The scheme can be depicted as follows:

To implement the SUMT method is practice we need to note some points:

1. Feasiblility of each point. It is assumed that we have a feasible point at the beginning. It is also important that subsequent points always remain within the feasible region. As x approaches a constraint from within the feasible region $\phi(\mathbf{x}, r) \to \infty$ and as $\mathbf{x}$ approaches a constraint from outside the feasible region $\phi(\mathbf{x}, r) \to -\infty$. This point must be handled in the program written to implement the SUMT algorithm.

2. The initial value for r. The initial value of r can be important in reducing the number of iterations required to minimise $\phi(x, r)$. If r is chosen too small the function $\phi(x, r)$ will change rapidly in the vicinity of its minimum poses a problem for a gradient based approach. If r is too large the penalty function $P(\mathbf{x})$ becomes too dominant. For many problems a value of $r_0 = 1$ is reasonable. A more analytical approach is to note that at the minimum of:

$$\phi(\mathbf{x}, r) = f(\mathbf{x}) + rP(\mathbf{x})$$

the gradient is:

$$\nabla\phi(\mathbf{x}, r) = \nabla f(\mathbf{x}) + r\nabla P(\mathbf{x})$$

and this should be small in magnitude (so effectively $\nabla\phi(\mathbf{x}, r) \simeq 0$). The squared magnitude of this vector is:

$$\nabla f(\mathbf{x})^T \nabla f(\mathbf{x}) + 2r\nabla f(\mathbf{x})^T \nabla P(\mathbf{x}) + r^2 \nabla P(\mathbf{x})^T \nabla P(\mathbf{x})$$

and this is a minimum when:

$$r = -\frac{\nabla f(\mathbf{x})^T \nabla P(\mathbf{x})}{\nabla P(\mathbf{x})^T \nabla P(\mathbf{x})}$$

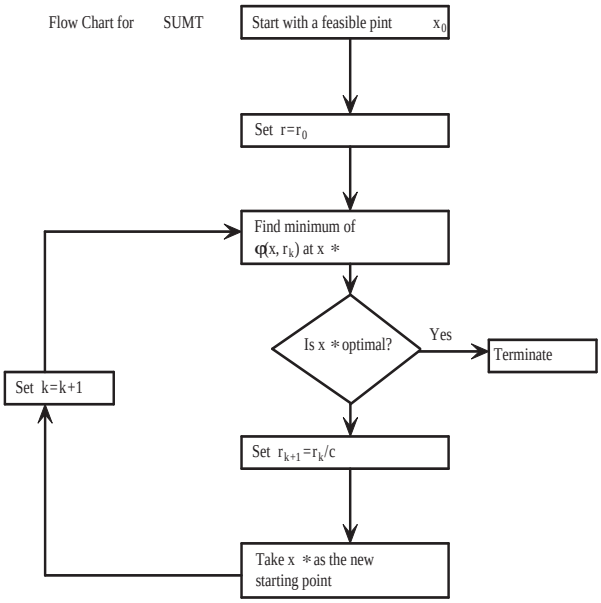

Figure 27: Schematic layout of the SUMT method.

4.19 Further Types of Penalty Function

For the minimisation task:

$$\text{Minimise} \qquad f(\mathbf{x}) \qquad \text{subject to} \qquad c_j(\mathbf{x}) \geq 0$$

we saw that a suitable minimisation approach is to minimise:

$$\phi(\mathbf{x}, r) = f(\mathbf{x}) + r \sum_{j=1}^{m} \frac{1}{c_j(\mathbf{x})}$$

with a sequence of values for r which tend to zero.

Many other penalty functions are used. For example:

1. Quadratic Penalty Functions.

Consider the task:

$$\text{Minimise} \qquad f(\mathbf{x}) \qquad \text{subject to} \qquad c_j(\mathbf{x}) = 0$$

The following type of penalty function is often used:

$$\phi(x, r) = f(\mathbf{x}) + \frac{1}{2\mu} \sum_{j=1}^{m} c_j^2(\mathbf{x})$$

where $\mu > 0$ in the penalty parameter. By driving μ to zero, we penalize the constraint violations with increasing severity. For an algorithm such as SUMT we consider a sequence of values $\{\mu_k\}$ with μ_k decreasing as k increaes.

This type of penalty function is well-behaved (no discontinuities) and we can use techniques from unconstrained optimisation to search for values for x_k.

Example 1. Consider the problem:

$$\text{Minimise} \qquad x_1 + x_2 \qquad \text{subject to} \qquad x_1^2 + x_2^2 - 2 = 0$$

An appropriate penalty function would be:

$$\phi(\mathbf{x}, r) = x_1 + x_2 + \frac{1}{2\mu}\left(x_1^2 + x_2^2 - 2\right)^2$$

Starting with $\mu = 1$ we eventually obtain the solution $(-1, -1)^T$.

2. Logarithmic Barrier Functions.

Consider the task:

$$\text{Minimise} \qquad f(\mathbf{x}) \qquad \text{subject to} \qquad c_j(\mathbf{x}) \geq 0$$

A logarithmic penalty function is of the form $-\ln(c_j(\mathbf{x}))$, thus:

$$\phi(\mathbf{x}, r) = f(\mathbf{x}) - \mu \sum_{j=1}^{m} \ln\left[c_j(\mathbf{x})\right]$$

Example 2. Consider the following problem with a single variable x:

$$\text{Minimise} \qquad x \qquad \text{subject to} \qquad x \geq 0, \qquad 1 - x \geq 0$$

then an appropriate penalty function would be:

$$\phi(x, r) = x - \mu \ln\left[x\right] - \mu \ln\left[1 - x\right]$$

For small values of μ the function $\phi(x, r)$ is close to the objective function value $f(\mathbf{x})$. It is only large in narrow "boundary layers".

Example 3. Consider the following problem with a two variables :

$$\text{Minimise} \qquad (x_1 + 0.5)^2 + (x_2 - 0.5)^2$$

subject to:

$$x_1 \in [0, 1]$$

and:

$$x_2 \in [0, 1]$$

then the log-barrier function is:

$$\phi(\mathbf{x}, r) = (x_1 + 0.5)^2 + (x_2 - 0.5)^2 - \mu \left[\ln(x_1) + \ln(1 - x_1) + \ln(x_2) + \ln(1 - x_2)\right]$$

3. Penalty functions with max or min. For a task such as:

$$\text{Minimise} \qquad f(\mathbf{x}) \qquad \text{subject to} \qquad c_j(\mathbf{x}) \leq 0$$

a penalty function such as:

$$\phi(\mathbf{x}, r) = f(\mathbf{x}) + \frac{1}{2\mu} \sum_{j=1}^{m} c_j^2(\mathbf{x})$$

the penalty term would still contribute if μ is large but not infinite. A better choice would be a penalty which is only active if the point x is not feasible i.e. $c_j(\mathbf{x}) < 0$. A suitable unconstrained optimisation task is thus:

$$\phi(\mathbf{x}, r) = f(\mathbf{x}) + \mu \sum_{j=1}^{m} \max\left[0, c_j(\mathbf{x})\right]$$

If $c_j(\mathbf{x}) \leq 0$ then maximum$\{0, c_j(\mathbf{x})\} = 0$ and no penalty is incurred. On the other hand if $c_j(\mathbf{x}) > 0$ then maximum$\{0, c_j(\mathbf{x})\} > 0$ and the penalty term $\mu c_j(\mathbf{x})$ is realised. However, at x such that $c_j(\mathbf{x}) = 0$ the function $\phi(x, r)$ may not be differentiable. If differentiability is required then a suitable penalty function would be of the type $\mu \left[\max\{0, c_j(\mathbf{x})\right]^2$.

Example 4. Consider the following function:

$$\text{Minimise} \qquad x \qquad \text{subject to} \qquad -x + 2 \leq 0$$

with $p(\mathbf{x}) = \left[\max\{0, c_j(\mathbf{x})\}\right]^2$ we would then use:

$$P(\mathbf{x}) = \begin{cases} 0 & \text{if} \quad x \geq 2 \\ (-x+2)^2 & \text{if} \quad x < 2 \end{cases}$$

In general, a suitable penalty function must incur a positive penalty for infeasible points and no penalty for feasible points. If the constraints are of the form $c_i(\mathbf{x}) \leq 0$ for $i = 1, \ldots, m$ and $h_i(\mathbf{x}) = 0$ for $i = 1, \ldots, l$ then a suitable penalty function is of the form:

$$P(\mathbf{x}) = \sum_{i=1}^{m} \phi\left[c_i(\mathbf{x})\right] + \sum_{i=1}^{l} \psi\left[h_i(\mathbf{x})\right]$$

where ϕ and ψ are continuous functions satisfying the following:

$$\begin{array}{llllll} \phi(y) = 0 & \text{if } y \leq 0 & \text{and} & \phi(y) > 0 & \text{if } y > 0 \\ \psi(y) = 0 & \text{if } y = 0 & \text{and} & \psi(y) > 0 & \text{if } y \neq 0 \end{array}$$

Typically, ϕ and ψ are of the form:

$$\begin{aligned} \phi(y) &= \left[\max\{0, y\}\right]^p \\ \psi(y) &= |y|^p \end{aligned}$$

where p is a positive integer. Thus, the penalty function is usually of the form:

$$P(\mathbf{x}) = \sum_{i=1}^{m} \left[\max\{0, c_i(\mathbf{x})\}\right]^p + \sum_{i=1}^{l} |h_i(\mathbf{x})|^p$$

4.20 Questions on Nonlinear Optimisation.

Question 1. Use the steepest ascent method to maximise:

$$f(\mathbf{x}) = 2x_1x_2 + 2x_2 - x_1^2 - 2x_2^2$$

starting from $\mathbf{x} = (0, 0)$. *Only evaluate the first iteration of the steepest ascent method.*

Question 2. Use the steepest ascent method to maximise:

$$f(\mathbf{x}) = 2x_1 + 5x_2 + x_1x_2 - x_1^2 - x_2^2 + 10$$

starting from $\mathbf{x} = (0, 0)$. *Only evaluate the first iteration of the steepest ascent method.*

Question 3. Use the steepest descent method to *minimise*:

$$f(\mathbf{x}) = (x_1 - 3x_2)^2 + (x_2 - 1)^2$$

starting from $\mathbf{x} = (2, 2)$. *Only evaluate the first iteration of the steepest descent method.*

Question 4. Is the following function concave or convex or neither:

$$f(x) = 3x^3 + 4x^2 + 6x + 5$$

Question 5. Is the following function concave or convex or neither:

$$f(x) = x^2 - 9x + 10$$

Question 6. Is the following function concave or convex or neither:

$$f(x) = 5x_1 - 2x_1^2 + 3x_1x_2 - 2x_2^2$$

Question 7. Starting from $(100, 100)$ evaluate the first iteration of the Newton-Raphson method based on:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{H}_i^{-1}\mathbf{g}_i$$

with $\mathbf{H}$ the Hessian matrix to find the maximum of:

$$z = (x + y + 10)\left(\frac{2272000}{xy} + 2y + 5\right)$$

Question 8. Starting from $(1, 1)$ evaluate the first iteration of the Newton-Raphson method to find the *minimum* of:

$$z = x_1^2 + x_1^2x_2^2 + 3x_2^4$$

Can you spot the expected minimum?

Answers

1.

$$\begin{aligned}\frac{\partial f}{\partial x_1} &= 2x_2 - 2x_1 \\ \frac{\partial f}{\partial x_2} &= 2x_1 + 2 - 4x_2\end{aligned}$$

so $\mathbf{g} = \nabla f = (0, 2)$. Thus:

$$f\left(\mathbf{x}' + \lambda \nabla f(\mathbf{x}')\right) = f\left(0, 2\lambda\right) = 4\lambda - 8\lambda^2$$

where $\mathbf{x}'$ is the new point and λ the step length. Thus to maximise $f(\mathbf{x})$ we wish to maximise $(4\lambda - 8\lambda^2)$. Thus:

$$\frac{d}{d\lambda}\left(4\lambda - 8\lambda^2\right) = 4 - 16\lambda = 0$$

so $\lambda = 1/4$ and:

$$\mathbf{x}' = (0, 0) + \frac{1}{4}(0, 2) = \left(0, \frac{1}{2}\right)$$

For the next iteration we find $\lambda = 1/2$ and $\mathbf{x}'' = \left(\frac{1}{2}, \frac{1}{2}\right)$. Continuing this way we eventually the solution $(1, 1)$.

2. The gradient for this function is:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 2 + x_2 - 2x_1 \\ 5 + x_1 - 2x_2 \end{pmatrix}$$

and its value at the starting point is:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 5 \\ 5 \end{pmatrix}$$

so:

$$f\left(\mathbf{x}' + \lambda \nabla f(\mathbf{x}')\right) = f\left(2\lambda, 5\lambda\right) = 10 + 29\lambda - 19\lambda^2$$

where $\mathbf{x}'$ is the new point and λ the step length. Thus to maximise $f(\mathbf{x})$ we wish to maximise $(10 + 29\lambda - 19\lambda^2)$. Thus:

$$\frac{d}{d\lambda}\left(10 + 29\lambda - 19\lambda^2\right) = 29 - 38\lambda = 0$$

so $\lambda = 0.763$ and:

$$\mathbf{x}' = (0, 0) + 0.763(2, 5) = (1.526, 3.815)$$

After a number of iterations the method converges on $(3, 4)$ with the objective function equal to 22.71.

3. The gradient vector is given by:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 2x_1 - 6x_2 \\ -6x_1 + 20x_2 - 2 \end{pmatrix}$$

and its value at the starting point is:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} -8 \\ 26 \end{pmatrix}$$

Since this is *minimisation task* we should move *downwards* i.e. use a negative gradient so:

$$\mathbf{x}' = \mathbf{x} - \lambda \nabla f(x) = \begin{pmatrix} 2 \\ 2 \end{pmatrix} - \lambda \begin{pmatrix} -8 \\ 26 \end{pmatrix} = \begin{pmatrix} 2 + 8\lambda \\ 2 - 26\lambda \end{pmatrix}$$

so for a given value of λ the value of the objective is equal to:

$$\begin{aligned} h(\lambda) &= f\left(\mathbf{x} - \lambda \nabla f(x)\right) = f\left(2 + 8\lambda, 2 - 26\lambda\right) \\ &= 17 - 740\lambda + 8072\lambda^2 \end{aligned}$$

and we wish to minimise this expression. Hence we set the derivative of this expression to zero:

$$\frac{dh(\lambda)}{d\lambda} = -740 + 16144\lambda = 0$$

This yields a step length $\lambda = 0.0458$ so that:

$$x' = \begin{pmatrix} 2.3667 \\ 0.8082 \end{pmatrix}$$

and the objective function has decreased from $f(x) = 17$ to $f(x') = 0.0402$.

After a number of iterations we eventually converge on the solution $(3, 1)$.

4. $f(x) = 3x^3 + 4x^2 + 6x + 5$ so:

$$\begin{aligned} \frac{df}{dx} &= 9x^2 + 8x + 6 \\ \frac{d^2 f}{dx^2} &= 18x + 8 \end{aligned}$$

so $(d^2 f/dx^2)x^2$ is not greater than or equal to zero (or less than or equal to zero) for every value of x from $-\infty$ to ∞ (i.e. not positive or negative definite). *Thus this function is neither convex or concave.*

5. $f(x) = x^2 - 9x + 10$ so:

$$\begin{aligned} \frac{df}{dx} &= 2x - 9 \\ \frac{d^2 f}{dx^2} &= 2 \end{aligned}$$

so $(d^2f/dx^2)x^2$ is always greater than or equal to zero and the function is *strictly convex*.

6. The derivatives are:

$$\begin{aligned}
\frac{\partial f}{\partial x_1} &= 5 - 4x_1 + 3x_2 \\
\frac{\partial f}{\partial x_2} &= 3x_1 - 4x_2 \\
\frac{\partial^2 f}{\partial x_1^2} &= -4 \\
\frac{\partial^2 f}{\partial x_2^2} &= -4 \\
\frac{\partial^2 f}{\partial x_1 \partial x_2} &= 43
\end{aligned}$$

so the Hessian matrix is:

$$H = \begin{bmatrix} -4 & +3 \\ -3 & -4 \end{bmatrix}$$

so:

$$\begin{aligned}
|H - \lambda I| &= \begin{vmatrix} 4+\lambda & -3 \\ -3 & 4+\lambda \end{vmatrix} \\
&= (\lambda + 4)^2 - 9 \\
&= \lambda^2 + 8\lambda + 7 \\
&= 0
\end{aligned}$$

so $\lambda_1 = -7$ and $\lambda_2 = -1$. Since both eigenvalues of the Hessian are negative, the objective function is *concave*.

7.

$$\begin{aligned}
\frac{\partial z}{\partial x} &= \left(\frac{2272000}{xy} + 2y + 5\right) - (x + y + 10)\,\frac{2272000}{x^2 y} \\
\frac{\partial z}{\partial y} &= \left(\frac{2272000}{xy} + 2y + 5\right) + (x + y + 10)\left(-\frac{2272000}{xy^2} + 2\right) \\
\frac{\partial^2 z}{\partial x^2} &= 2(y + 10)\frac{2272000}{x^3 y} \\
\frac{\partial^2 z}{\partial x \partial y} &= 2 + \frac{10 \times 2272000}{x^2 y^2} \\
\frac{\partial^2 z}{\partial y^2} &= 4 + \frac{2 \times 2272000(x + 10)}{xy^3}
\end{aligned}$$

and for the first iteration:

$$\mathbf{x} = \begin{bmatrix} 100 \\ 100 \end{bmatrix}$$

$$\begin{aligned}\mathbf{g} &= \begin{bmatrix} -44.92 \\ 375.1 \end{bmatrix} \\ \mathbf{H} &= \begin{bmatrix} 4.998 & 2.227 \\ 2.227 & 8.998 \end{bmatrix}\end{aligned}$$

so:

$$x_{new} = \begin{bmatrix} 100 \\ 100 \end{bmatrix} - \begin{bmatrix} 0.2249 & -0.0557 \\ -0.0557 & 0.1249 \end{bmatrix} \begin{bmatrix} -44.92 \\ 375.1 \end{bmatrix} = \begin{bmatrix} 131 \\ 50.6 \end{bmatrix}$$

8. The derivatives are:

$$\begin{aligned}\frac{\partial z}{\partial x_1} &= 2x_1 + 2x_1x_2^2 \\ \frac{\partial z}{\partial x_2} &= 2x_1^2x_2 + 12x_2^3 \\ \frac{\partial^2 z}{\partial x_1^2} &= 2 + 2x_2^2 \\ \frac{\partial^2 z}{\partial x_1 \partial x_2} &= 4x_1x_2 \\ \frac{\partial^2 z}{\partial x_2^2} &= 2x_1^2 + 36x_2^2\end{aligned}$$

The Hessian is:

$$H = \begin{bmatrix} 4 & 4 \\ 4 & 38 \end{bmatrix}$$

The new point after the first iteration is $(0.294, 0.706)$. However, note that from the first derivatives, or by observation, it is apparent that the solution is $(0, 0)$.

5 Dynamic Programming: Multi-Stage Decision Making

5.1. Backward Recursion

5.2 A Capital Budgeting Problem

5.3 A Scheme for formulating Dynamic Programming Problems

5.4 A Cargo Loading Problem

The method of Dynamic Programming was developed by Richard Bellman in the 1950s. He showed that problems in which a sequence of decisions are taken at different stages of a process can be solved by backward recursion. The name *Dynamic Programming* (DP) refers to the time dependence of these multi-stage decision problems but DP can be applied to any problem which can be split into stages even if the stages are not separated in time.
An example of the sort of multi-stage problem for which DP is used is that of a production manager who has to set production targets each week and set aside time for maintenance. The decisions he takes in one week will affect stock-levels and the reliability of the plant in the future so that although each week's decisions constitute a separate problem all the decisions are interdependent. The Dynamic Programming solution to this problem puts all the information that inter-relates one week's decisions with the next week into a set of variables called the *state* and thus allows each week's decisions to be considered as a separate problem.
We shall now consider a very simple example of the method of *backward recursion*. Backward recursion is the numerical method often used for solving problems formulated using the ideas of Dynamic Programming and it will allow us to introduce a DP *functional equation* in a simple example.

5.1 Backward Recursion

Figure 28 shows a network of nodes and the problem is to find the cheapest route from node A to any node on the line BC. The cost of travelling along each section between nodes is shown against the line. The nodes are numbered in terms of a *stage* variable 0,1,2,3,4 and a *state* variable which is the number of the node counting up from the line AC. Thus node (0,1) is stage 0 and the first node denoted A. Node (4,5) is B. The cost of going from node (0,1) to (1,2) is 2.

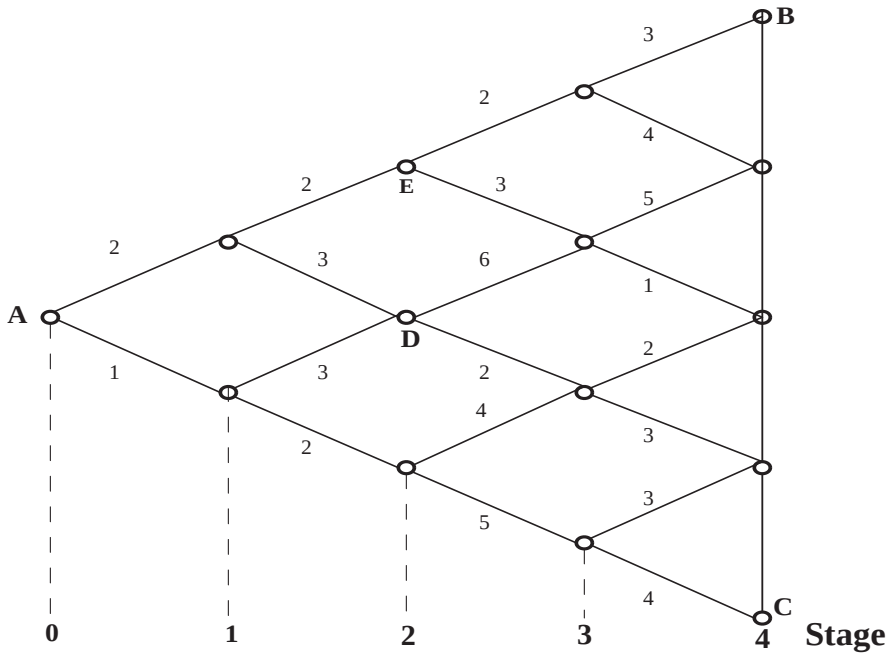

Figure 28: A network with stages labelled 0 to 4 and costs indicated next to each link. We wish to find the least cost route from node A to the line BC.

One way of solving this problem is to enumerate all the possible routes, find out the cost of each route, and then choose the cheapest route. This is a feasible method for this small problem, however we will want to solve much larger problems and so we need to be able to reduce the amount of computation. The dynamic programming solution to this problem involves defining an *optimal value function*, $S_n(x)$ which is the cost of the cheapest route from stage n, state x, i.e. from node *(n,x)* to the line BC. At any point on the line BC the optimal value function will be zero so we can move one stage backwards in time to stage 3 and for each node we write down $S_3(x)$. Thus for node (3,3) we can either go up (U) for 5 or down (D) for 1, so we insert in Table 4 the optimal decision D and the optimal value function $S_3(3) = 1$.

x	$d_0(x)$	$S_0(x)$	$d_1(x)$	$S_1(x)$	$d_2(x)$	$S_2(x)$	$d_3(x)$	$S_3(x)$	$S_4(x)$
5									0
4							U	3	0
3					D	4	D	1	0
2			U	6	D	4	U	2	0
1	U or D	8	U	7	U	6	U	3	0
STAGE		0		1		2		3	4

Table 4: the optimal decisions and associated values for a given stage and state.

Having filled in all the values for stage 3 we move back to stage 2. Consider node (2,2) or D, the choice is again

U or D but the cost of getting to the line BC is as follows:

For the decision:

U : 6 + the cost of getting from (3,3) to BC
i.e. $6 + S_3(3) = 6 + 1 = 7$.

D : 2 + the cost of getting from (3,2) to BC
i.e. $2 + S_3(2) = 2 + 2 = 4$.

We wish to find the minimum cost route so the calculations for node D can be succinctly summed up as:

$$S_2(2) = \min_{\text{UorD}} \left[\begin{array}{l} U : 6 + S_3(3) \\ D : 2 + S_3(2) \end{array} \right] = 4 \tag{83}$$

There is nothing peculiar about node D, the same decision process is true at any node and we can write down a more general *functional equation*, as equation (83) is called, thus:

$$S_n(x) = \min_{\text{UorD}} \left[\begin{array}{l} U : C_n(x, x+1) + S_{n+1}(x+1) \\ D : C_n(x, x) + S_{n+1}(x) \end{array} \right] \tag{84}$$

where $C_n(x, y)$ is the cost of going from node (n, x) to $(n + 1, y)$.

We will now use this to evaluate $S_2(3)$ i.e. substitute $n = 2$, $x = 3$ or node E. This tells us that:

$$\begin{aligned} S_2(3) &= \min_{\text{UorD}} \left[\begin{array}{l} U : C_2(3,4) + S_3(4) \\ D : C_2(3,3) + S_3(3) \end{array} \right] \\ &= \min_{\text{UorD}} \left[\begin{array}{l} U : 2 + 3 \\ D : 3 + 1 \end{array} \right] = 4 \end{aligned}$$

where $C_2(3, 4)$ and $C_2(3, 3)$ are the costs of the routes to the next nearest nodes from E on Figure 28: $S_3(3)$, $S_3(4)$ have already been calculated and entered in Table 4. The rest of the entries in Table 4 have been calculated using equation (84), working backwards through the table from stage 4 to stage 0. Hence the name *backwards recursion*. For the final entry $S_0(1) = 8$ the cost is the same whether the decision is U or D.
Table 4 now provides a solution to our problem as the optimal decision is given in the table for every node. There are two optimal routes because the decision at node (0,1) is not unique:

U at (0,1) which brings us to (1,2) where we go
U taking us to (2,3) and then
D to (3,3) and then
D to (4,3) at a total cost of 8.

The alternative sequence is: D U D U again at a cost of 8. Should a mistake occur when following the route, a new optimal route from the node at which we found ourselves could easily be worked out from Table 4. This feedback feature of the DP solution is often very valuable and the amount of computation in evaluating Table 4 is less than that required to evaluate all feasible routes.
The function of this section has been twofold. First to introduce functional equations, such as equation (84), in an example which is simple enough for the notation to be understood. Secondly to explain the way in which such functional equations may be solved starting from the boundary conditions ($S_4(x) = 0$, in this case) and evaluating the functions $S_n(x)$ in backwards sequence: $S_4, S_3, S_2, \ldots$ In the next section we consider another simple example.

5.2 A Capital Budgeting Problem

A large corporation has 3 subsidiaries each of which has put in a set of development plans. The subsidiaries have submitted either 2 or 3 alternative plans each with a specified capital cost c_{ij} (subsidiary i, plan j) and a corresponding expected return r_{ij}. The expected returns have been calculated according to an agreed corporate formula which takes account of how the income from the investment is spread out over the years to come and also of the possibility of loss or failure. The corporation has a capital budget of £14m - and wishes to choose one of each subsidiary's development plans in such a way as to maximise its expected return. The data for r_{ij} and c_{ij} are shown in Table 5.

Subsidiary (stage) i	1		2		3	
plan no, j	c_{1j}	r_{1j}	c_{2j}	r_{2j}	c_{3j}	r_{3j}
1	2	3	1	2	3	5
2	4	6	3	5	5	7
3	7	10			8	13

Table 5: the costs and returns (in £m) of the subsidiary's development plans

We can write down a mathematical formulation of the problem as:

$$\max_{d} [r_{1d_1} + r_{2d_2} + r_{3d_3}] \tag{85}$$

subject to:

$$c_{1d_1} + c_{2d_2} + c_{3d_3} \leq 14 \tag{86}$$

where d_i is the number of the development plan chosen for subsidiary i.

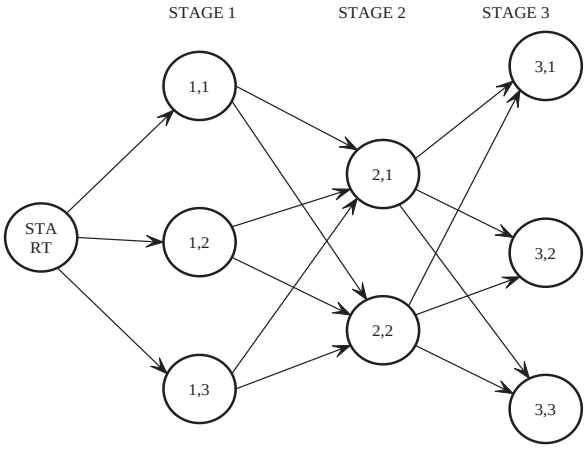

Figure 29: The investment decision tree for the capital budgeting problem.

In order to solve this problem using Dynamic Programming we decompose it into sub-problems or stages so that at each stage only one decision is taken. The decision tree in Figure 29 clearly suggests the stages and the sub-problems are:

Stage 1

Decide which plan should be implemented for subsidiary 1 when the amount of capital available for subsidiary 1 is £x_1 millions. We solve this simple sub-problem for all feasible values of x_1 i.e. $x_1 \in [2, 14]$.

Stage 2

Decide which plan should be implemented for subsidiary 2 when the capital available for both subsidiary 1 and 2 is £x_2 millions. We solve this for $x_2 \in [3, 14]$. (The minimum value of $x_2 = 3$ because we must implement a plan for both subsidiaries.)

Stage 3

We need only solve for the case $x_3 = 14$.

The decisions for the three subsidiaries are connected together because the objective function (85) and the constraint on capital spending (86) both contain contributions from all the decisions. DP allows us to separate the decisions by the device of carrying, from stage to stage, the necessary information concerning the contributions to the objective function and the spare capacity in the constraint. The information about the objective function contribution of earlier stages is held in the *optimal value function* $S_n(x)$ where n is the stage number and x tells us how much slack there is in the constraint i.e. how much capital is still available, is called the *state* variable. Thus $S_n(2)$ is the maximum contribution to the return made in the stage 1 problem when the capital available $x = 2$. From Table 5 we see that $S_1(2) = 3$; $S_1(3) = 3$; $S_1(4) = 6$ etc. These results are tabulated in Table 6. In stage 2 we decide the best plan for subsidiary 2 for the general case of a capital x_2 available for both subsidiary 1 and 2. Whatever capital c_{2j} we use for subsidiary 2, the capital left over for subsidiary 1 will be $x_2 - c_{2j}$ and this defines the optimal return from stage 1 as $S_1(x_2 - c_{2j})$ - a function we have already evaluated. So we can make a decision in stage 2 which maximises the return from both stage 1 and 2 thus:

$$S_2(x_2) = \max_{j \in \{1,2\}} [r_{2j} + S_1(x_2 - c_{2j})] \tag{87}$$

Using this equation we find the optimal return $S_2(x)$ available from subsidiaries 1 and 2 when the total capital available is x. For example $S_2(9)$ is the optimal return from stages 1 and 2 when £9m is available and:

$$S_2(9) = \max_{j \in \{1,2\}} \left[\begin{array}{l} j = 1 : 2 + S_1(9-1) = 2 + 10 \\ j = 2 : 5 + S_1(9-3) = 5 + 6 \end{array} \right] = 12 \tag{88}$$

The values of S_1 are looked up in Table 6 and the calculated values of $S_2(x)$ are filled in. Stage 3 is very similar. We wish to decide which plan to use for subsidiary 3 so that the total return is maximised. The total return is:

$$r_{3j} + S_2(x - c_{3j}) \tag{89}$$

where x is the capital available for all 3 subsidiaries so we can write:

$$S_3(x) = \max_{j \in \{1,2,3\}} [r_{3j} + S_2(x - c_{3j})] \tag{90}$$

Using (90) we evaluate $S_3(14)$ thus:

$$S_3(14) = \max_{j} \left[\begin{array}{l} j = 1 : 5 + S_2(14-3) = 20 \\ j = 2 : 7 + S_2(14-5) = 19 \\ j = 3 : 13 + S_2(14-8) = 21 \end{array} \right] = 21$$

The results from using (89) and (90) are shown in Table 6, both equations are special cases of the *general functional equation* for the system:

x	$d_1(2)$	$S_1(x)$	$d_2(x)$	$S_2(x)$	$d_3(x)$	$S_3(x)$
2	1	3				
3	1	3	1	5		
4	2	6	1	5		
5	2	6	1 or 2	8		
6	2	6	1 or 2	8		
7	3	10	1	11		
8	3	10	1	12		
9	3	10	1	12		
10	3	10	2	15		
11	3	10	2	15		
12	3	10	2	15		
13	3	10	2	15		
14	3	10	2	15	3	21

Table 6: The forward recursion calculations for the Capital Budgeting problem.

$$S_n(x) = \max_j [r_{nj} + S_{n-1}(x - c_{nj})] \tag{91}$$

We have solved this equation by forward recursion starting from the boundary conditions:

$$S_1(x) = \begin{cases} 3 & 2 \leq x < 4 \\ 6 & 4 \leq x < 7 \\ 10 & 7 \leq x \end{cases} \tag{92}$$

Once the forward recursion table (Table 6) has been generated complete with the decisions d_i, we can find the solution to the original problem by working backwards through the table. Thus the capital available for the 3 stages is £14m so we look up $S_3(14)$ and find the optimal decision is 3 i.e. implement plan 3 for subsidiary 3. This leaves $14 - c_{33} = 6$ as the state x for the 2 stage problem. We then look up $S_2(6)$ and find the decision 1 or 2, meaning that either plan 1 or plan 2 may be implemented. If we choose to implement plan 1 at a cost of £1m this leaves £5m for stage 1 and looking up $S_1(5)$ we see the decision is to implement plan 2. This gives a decision sequence $\mathbf{d} = [2, 1, 3]$. Alternatively if we choose to implement plan 2 for subsidiary 2 at a cost of £3m this would leave £3m for subsidiary 1 and looking up $S_1(3)$ we see that the decision is to implement plan 1 for subsidiary 1 giving a decision sequence $d = [1, 2, 3]$. If we add up the return from either of these alternative decision sequences we find that the return is $S_3(14) = 21$ as expected.
This problem has been solved using *forward recursion* while the network problem in the last section was solved by *backward recursion*. Either method could have been used for this problem and in general the only limitation on the choice comes from the boundary conditions on the functional equation. If these apply to the last stage of the problem one must work backwards from them, if they are available at the first stage then one can work forwards.

5.3 A Scheme for formulating Dynamic Programming Problems (not examinable).

Note: in this section I described how you might set up a functional equation. You are not expected to do this in an exam. In the next section I do this for the cargo loading problem. This last section is examinable, in that you could be expected to know and use the functional equation for the cargo loaduing problem or related (same functional equation) problems.

Dynamic Programming is not an algorithm like the Simplex method of Linear Programming but an approach to problem solving in which one generalises or embeds the given problem into a broader class of problems. Problems from the broad class are then solved in ascending order of complexity each utilising the solutions of the simpler problems. Formulating DP problems is often very difficult and so in this section we describe a step-by- step approach to the task.

Step 1: Formulate the problem without regard to DP.

Step 2: Generalise the formulation of Step 1 by changing the numerical limits of the constraints to variables and the upper limits of all summations to other variables. If several summations are over the same range then the upper limits should be replaced by the same variable.

Applying this to the capital budgeting problem we change the limit of the only constraint (86) to x instead of 14, giving:

$$c_{1d_1} + c_{2d_2} + c_{3d_3} \leq x$$

Now this inequality and the objective function both have summations in them so we need to rewrite them with a variable upper limit. They are both summed over the 3 subsidiaries so that the variable upper limit will be the same, say n. The reformulated problem is now:

$$\max_{d} \left[\sum_{i=1}^{n} r_{id_i} \right] \tag{93}$$

subject to:

$$\begin{aligned} \sum_{i=1}^{n} c_{id_i} &\leq x \\ d_1, d_3 &\in \{1,2,3\} \qquad d_2 \in \{1,2\} \end{aligned}$$

Step 3. Choose State and Stage Variables.

Dynamic Programming is concerned with multistage decision problems so, in order to formulate a problem, we must find a way of looking at the problem as a series of decisions. Sometimes the stage variable is obvious because the problem contains a sequence of decisions separated in time; in other problems one looks at the variables in the problem that are integers to see which of them could be used as stages. If, in the step 2 formulation, the objective function is a sum of similar terms e.g.

$$\sum_{i=1}^{N} f(x_i, d_i)$$

then N is usually the stage variable.
Because one solves the problem one stage at a time, it is essential to *embed* the original problem in which there are, say N stages and a starting state x_0, into the more general class of problems where the number of stages can take any value and the starting state can take any feasible value. The idea of the "state" of a system and hence the state variables is difficult to describe and hence to grasp. It is most easily understood in terms of time varying physical systems. Consider for example the state of a car - one you know. The function of a car is to go places so when we talk about its state we are usually thinking about it getting somewhere and the state of the car is the all the information you would need in order to be able to estimate when it would get

to its destination. You would need to know where it was, how fast it was travelling and whether there was enough petrol in the tank for it to get to its destination without stopping. These three variables would be state variables. Information such as the size of the engine is part of the system description and doesn't change as you drive along so we don't include that in the state. If however the radiator leaked this would add an extra dimension to the state as we would require to know if the engine was likely to seize up before arriving at the destination. The state then consists of those variables whose values you must know in order to predict the future (for a time varying problem) or to be able to solve the problem at all for static problems. Often the state variables can be identified by looking at the formulation done in stage 1 crossing out the stage variable which has already been chosen, crossing out all the decision or control variables (under the MAX or MIN operator and elsewhere) and then looking at the residual numbers or variables. These must be either a part of the system description (just technical data) or a part of the state. This classification is usually not difficult.
The state of a system is often closely related to the constraints of the problem. The more constraints there are the more variables one has in the state vector. It is sometimes useful to think of the state as the state of the constraints. If we consider again the Capital Budgeting example, the stage variable is clearly n as the objective function (85) is summed to n and this leaves:

d_1, d_2, d_3 which are decision variables
r_{id_i}, c_{id_i} which are part of the system description
and x which is the state variable.

Step 4 Define the Optimal Value Function

The optimal value function is a function whose value is that of the objective function for the generalised or imbedded problem. Thus the optimal value function has a parameter n which defines the sub-problem or stage, and a set of arguments which are the state. It is essential to have an unambiguous definition otherwise one's thinking easily becomes confused. In the Capital Budgeting problem the stage variable is n and the state is x so we can define the optimal value function $S_n(x)$ as the total return from the development plans of subsidiaries $n, n+l, ...$ when the capital available for all plans is x.

Step 5 Write down the functional equation and boundary conditions.

We have already discussed the generation of the functional equation (87) and its boundary condition (88) in the last section. In the next section we will formulate another simple problem explicitly following the above scheme for formulation.

5.4 A Cargo Loading Problem

A barge can carry 100 tons of cargo and there are four different types of cargo which can be loaded:

Type, i	Weight in tons/unit, w_i	Value/unit, v_i
1	40	40
2	30	25
3	15	10
4	10	6

Table 7: The weights and values of different types of cargo.

How many of each type of cargo should be loaded onto the barge so as to maximise its loaded value?

This is a small problem and could be solved by trying different combinations of numbers. It is however typical of an important wider class of problems which are usually too big to be efficiently solved by experimenting with

different combinations. We now formulate this problem using the scheme for formulating Dynamic Programming problems described in the last section.

Step 1. Formulate without regard to Dynamic Programming

If we define x_i as the number of packs of type i loaded then the objective function is

$$\max_x \left[\sum_{i=1}^{4} x_i v_i \right]$$

subject to the constraint on total weight

$$\begin{aligned} \sum_{i=1}^{4} x_i w_i &\leq 100 \\ x &\geq 0 \quad x \text{ integer} \end{aligned}$$

This particular problem is a form that could be solved by Linear Programming were it not for the integer constraint. It is a simple example of the class of problems called *allocation problems* as a limited resource (cargo carrying capacity in this example) is allocated between different activities or demands (the cargo types here).
Step 2. *We now generalise the problem i.e. imbed it in a more general recognisable class of problem.* Thus, instead of the constraint being a total weight of 100 tons, we say a total weight W tons, and instead of having 4 different types of cargo we have N. The formulation is now:

$$\max_x \sum_{i=1}^{N} x_i v_i$$

subject to the constraint

$$\begin{aligned} \sum_{i=1}^{N} x_i w_i &\leq W \\ x &\geq 0 \quad x \text{ integer} \end{aligned}$$

Step 3. Choose Stage and State Variables

The variables in this generalised problem are $x_i, i \in [I, N]$, N and W. All the variables are integers so this gives no guide as to how to choose the stage variable, however it is easier to think of choosing how many units of type 1 and then how many of type 2 etc, as stages than using any other variable as a stage variable. N is also the top limit of the sum in the objective function and this is usually the stage variable.
In choosing the stage variable we have already started the process of embedding the problem in a more general class of problems. The general problem is to load N types of cargo so the sub-problems involve loading a barge with type 1 cargo only, with types 1 and 2 only and so on. We now wish to identify the state. The x_i are the decision variables, $n \in [1, N]$ is the stage variable and so, looking at the step 2 formulation, we are left with v_i, w_i and W which is the carrying capacity of the barge. The v_i and w_i are problem-specific data and this leaves W the capacity of the barge as the state variable.

Step 4. Define the Optimal Value Function

The optimal value function is the optimal value of the objective function of the imbedded problem. It has a stage variables as a parameter and the state W as its argument. So we define $S_n(w)$ as the maximum value of

the cargo chosen from types 1, 2, which can fit into a barge of capacity w tons. In order not to confuse w (a variable weight value) with W (the total weight available for transport) we shall use the notation y_n below for the weight available at stage n.

Step 5. Write down the functional equation

In order to write down the functional equation we consider the n^{th} stage optimal value function $S_n(\ldots)$ and subdivide the contributions to the objective function into the contribution from the n^{th} stage, $x_n v_n$ and that from remaining $(n-1)$ stages, $S_{n-1}(\ldots)$. Let y_n be the weight available at stage n. Then the weight of cargo available for the remaining stages will be $(y_n - x_n w_n)$ x_n is the number of packs of type n loaded at stage n. So

$$S_n(y_n) = \max_{x_n} [x_n v_n + S_{n-1}(y_n - x_n w_n)] \tag{94}$$

where the first term on the right hand side is the value of the n^{th} cargo type and the second term on the right hand side is the value of all the other $n-1$ cargo types loaded. The choice of x_n is limited to integers in the range $[0, W/w_n]$.

A boundary value for this recurrence relation is calculated by considering the one stage problem of maximising the loaded value of type 1 cargo. This weighs 40 tons per unit and has a value of 40 per unit so

$$S_1(y) = \text{Int}\left(\frac{y}{40}\right) \times 40 \tag{95}$$

where the **Int** function takes the integer part of the argument y/40. This gives the left-hand column in the Table below.

We now solve the functional equation (94) using forward recursion i.e. we evaluate $S_1(y)$ for all y and then $S_2(y)$, $S_3(y)$ etc. $S_1(y)$ is given as a boundary condition in (95) but it could have been evaluated from the functional equation in (94) by using the further boundary condition:

$$S_0(y) = 0$$

As each value of $S_n(y_n)$ is evaluated using the functional equation (94), the optimising value of x_n is recorded as $x_n^\star$. The solution is laid out in Table 8. Only multiples of 5 are given for y. This is apparent as all y are multiples of 5.

y	$S_1(y)$	$x_1^\star$	$S_2(y)$	$x_2^\star$	$S_3(y)$	$x_3^\star$	$S_4(y)$	$x_4^\star$
100	80	2	90	2	90	1	92	2
95	80	2	80	0	90	1	90	0
90	80	2	80	0	80	0	86	1
85	80	2	80	0	80	0	80	0
80	80	2	80	0	80	0	80	0
75	40	1	65	1	65	0	65	0
70	40	1	65	1	65	0	65	0
65	40	1	55	2	55	1	56	1
60	40	1	50	2	50	1	52	2
55	40	1	40	0	50	1	50	0
50	40	1	40	0	40	0	46	1
45	40	1	40	0	40	0	40	0
40	40	1	40	0	40	0	40	0
35	0	0	25	1	25	0	25	0
30	0	0	25	1	25	0	25	0
25	0	0	0	0	10	1	16	1

Table 8: The solution of the Cargo Loading Problem by forward recursion.

The functional equation was solved by starting with the one-stage problem $S_1(y)$ and proceeding in turn to the 2 stage problem $S_2(y)$, the 3 stage problem $S_3(y)$ and then the 4 stage problem $S_4(y)$. In order to read off the optimal policy corresponding to the original problem of loading 4 types of cargo into a 100 ton barge *we go in reverse order* starting with the 4 stage problem and finding $x_4^\star$ corresponding to $W = 100$. This tells us to load 2 units of type 4 cargo. Having loaded these units we are confronted with the problem of how to load the residual weight $100 - 2 \times 10 = 80$ tons with 3 types of cargo. The optimal return from this sub-problem is $S_3(80) = 80$ with $x_3^* = 0$ i.e. no units of type 3 are loaded. We now consider the two stage problem and find the optimal value of $S_2(80) = 80$ with $x_2^\star = 0$ i.e. none of type 2 cargo is loaded so there is still a weight allowance of 80 tons for type 1 cargo. Against $S_1(80)$ we find $x_1^\star = 2$. The complete solution is thus to load 2 units of type 1 and 2 units of type 4 giving a total loaded weight of $2 \times 40 + 2 \times 10 = 100$ and an optimal loaded value $S_4(100) = 2 \times 40 + 2 \times 6 = 92$.
The Dynamic Programming solution of this problem provides much move information than just the answer to the original problem because Table 8 allows us to write down the solution to all possible sub-problems. Thus if the barge will only take 95 tons we load none of type 4 cargo, 1 of type 3 and 2 of type 1 or, if the Captain had a special reason for loading 3 of type 4 cargo, we can solve the residual 3 stage problem of allocating the remaining 70 tons amongst the other types of cargo $\{x_3^\star(70) = 0;\ x_2^\star(70) = 1;\ x_1^\star(70 - 30) = 1$, i.e. we load 1 of type 2 and 1 of type 1$\}$.
Of course, we could also have solved the problem by backward recursion. The functional equation is then:

$$S_n(y_n) = \max_{x_n = 0, \ldots, \frac{W}{w_n}} [x_n v_n + S_{n+1}(x_n - W_n x_n)] \tag{96}$$