Chapter 2

Background

This chapter will review the theory used in this dissertation. Much of the work that follows will use information the information in this chapter or be motivated by properties of the techniques outlined below.

The presentation begins with a consideration of optimisation algorithms in Section 2.1 and then moves on to consider the special case of genetic algorithms in Section 2.2. A number of important impedance matching algorithms are reviewed in Section 2.3 and some important discontinuity models are covered in Section 2.4.

2.1 Optimisation Algorithms

The ultimate objective of Computer Aided Design (CAD) is a computer system that can undertake design problems with no input from humans apart from the required specifications. While this objective may still be some way off, the current state of the art is found in optimisation algorithms. Optimisation algorithms are computer programs that modify the parameters of a system to improve its performance. This section will give a very brief overview of some of the most important classes of optimisation algorithm available

today. This presentation draws heavily on the review papers by Bandler [12] and Charalambous [13], the very practical presentation by Press *et al.* [14], and the detailed review of the field by Nocedal and Wright [15]. Some of the more advanced algorithms will not be considered here because the extra complexity required is not justified for this dissertation.

Section 2.1.1 will define the most important terms and concepts used in optimisation. Local optimisation in one dimension (line searches) will be covered in Section 2.1.2. Section 2.1.3 considers the most important classes of local optimisation techniques. Section 2.1.4 will present some of the main global optimisation techniques. Section 2.1.5 will give a brief overview of combinatorial optimisation. Some hybrid methods that combine more than one type of optimisation algorithm are considered in Section 2.1.6. Lastly, some methods for dealing with constraints are presented in Section 2.1.7.

2.1.1 Important Concepts

Optimisation is the process of adjusting the values of the parameters of a system so as to minimise or maximise some measure of the system's performance. The function used as a measure of a system's performance is known as the objective function. This section will only consider minimisation problems explicitly. This does not result in a loss of generality because maximisation problems can be formulated as minimisation problems by multiplying the objective function value by -1 [15].

Section 2.1.1.1 presents the basic formulation of an optimisation problem with minimax optimisation being considered in Section 2.1.1.2. Lastly, multi-objective optimisation is covered in Section 2.1.1.3.

2.1.1.1 Problem Formulation

This section will present the basic formulation of an optimisation problem, introduce important terminology, and consider some important special cases. Most of the work in this section is derived from Press *et al.* [14].

The mathematical form of the optimisation problem is

$$f(\mathbf{x}) \rightarrow \min \tag{2.1}$$

where $\mathbf{x}$ is some structure (usually a vector or a matrix) whose elements correspond to the system parameters, and $f(\mathbf{x})$ is the objective function which maps the system parameters to some measure of system performance that must be minimised. This case is known as unconstrained optimisation because there are no constraints on how the parameters' values can be assigned.

The simplest differentiable function that can have a well-defined minimum is a quadratic function [13]. The Taylor series expansion of a function is

$$f(\mathbf{x}+\Delta\mathbf{x}) = f(\mathbf{x}) + \sum_i \frac{\partial f}{\partial x_i}\Delta x_i + \frac{1}{2}\sum_{i,j}\frac{\partial^2 f}{\partial x_i \partial x_j}\Delta x_i \Delta x_j + \ldots \tag{2.2}$$

where $\Delta\mathbf{x}$ is the change in $\mathbf{x}$, x_i is the value of component i of $\mathbf{x}$, and Δx_i is the change in x_i. When the higher order terms are ignored and matrix representation is used, (2.2) can be rewritten as

$$f(\mathbf{x}+\Delta\mathbf{x}) \approx c + \mathbf{b}^T \cdot \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T \cdot \mathbf{A} \cdot \Delta\mathbf{x} \tag{2.3}$$

where

$$c \equiv f(\mathbf{x}) \qquad \mathbf{b} \equiv \nabla f|_{\mathbf{x}} \qquad [\mathbf{A}]_{ij} \equiv \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\mathbf{x}} \tag{2.4}$$

and all matrices except $\mathbf{A}$ are column matrices. The matrix $\mathbf{A}$, which contains the second partial derivatives of the objective function at $\mathbf{x}$, is known as the Hessian matrix of the function and is symmetrical. If the first and second derivatives of the objective function exist, the gradient vector is zero, and the Hessian matrix is positive definite then there is a minimum at point $\mathbf{x}$ [12]. The quadratic approximation will be used later in connection with conjugate directions, and the Newton and quasi-Newton methods.

Constrained optimisation is the case where parameters or functions of the parameters are constrained to be within some range or equal to a specified value. This can be written as

$$\begin{aligned} g_i(\mathbf{x}) &\geq 0 \\ h_j(\mathbf{x}) &= 0 \end{aligned} \tag{2.5}$$

where $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$ are both functions of the system parameters, and the subscripts indicate that there can be more than one of each type of constraint. Note that no generality is lost by only considering cases with zero because constants can be included in the functions. The situation where a function is less than or equal to a constant can be converted to the case where the function is greater than or equal to a constant by multiplying by -1. The subset of the problem space where all the constraints are satisfied is known as the feasible region.

Various special cases of objective function and its value can now be defined. A local minimum occurs when the value of the objective function is lower than all other objective function values in a subset of the feasible region. The point that has the lowest objective function value in the feasible region is known as the global minimum. Most optimisation algorithms consider the case where the parameters are continuous variables, but optimisation problems that require the best combination or permutation of parameters are also possible. These problems are known as combinatorial optimisation problems. A good example of a combinatorial problem is the classic travelling salesman problem where a salesman has to visit every city in a given area once and only once while minimising the total distance travelled. Other special cases are minimax and multi-objective optimisation which will be considered in the next two sections.

2.1.1.2 Minimax Optimisation

Minimax optimisation considers the problem of minimising the maximum value of an objective function. This is an extremely important problem which is very common in engineering where specifications usually have to be satisfied in a minimax way. The basic formulation

of a minimax problem along with an example and some difficulties with minimax objective functions are considered below.

A minimax problem is one with an objective function of the form

$$M(\mathbf{x}) = \max_{i \in I} [f_i(\mathbf{x})] \tag{2.6}$$

where $M(\mathbf{x})$ is the objective function, $f_i(\mathbf{x})$ are sub-functions, and I is the set of all sub-functions. The sub-functions in minimax problems are normally the value of the objective function at a number of values of some independent variable like frequency or temperature.

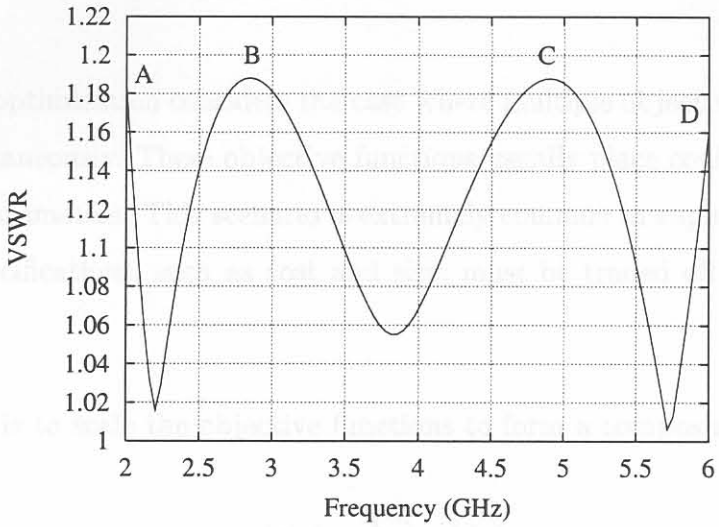
An example of a sixth-order minimax matching problem taken from Abrie [2] is given in Figure 2.1. Figure 2.1(a) shows the how the VSWR of a matching network varies with frequency and Figure 2.1(b) shows the effect of varying one of the component values around its optimum value. One of the four local maxima (A, B, C, and D in Figure 2.1(a)) will determine the minimax VSWR value of the system. This is shown in Figure 2.1(b) where distinct kinks are observed at A and B as different local maxima start to dominate the objective function.

The major difficulty with minimax problems is that the gradient is discontinuous meaning that many useful approximations such as the Taylor expansion are not valid. Gradient discontinuities are clearly seen at points A and B in Figure 2.1(b). It is also clear that a parabolic approximation to the curve in Figure 2.1(b) would be extremely poor, especially if all the points used to form the approximation were before A, between A and B, or after B.

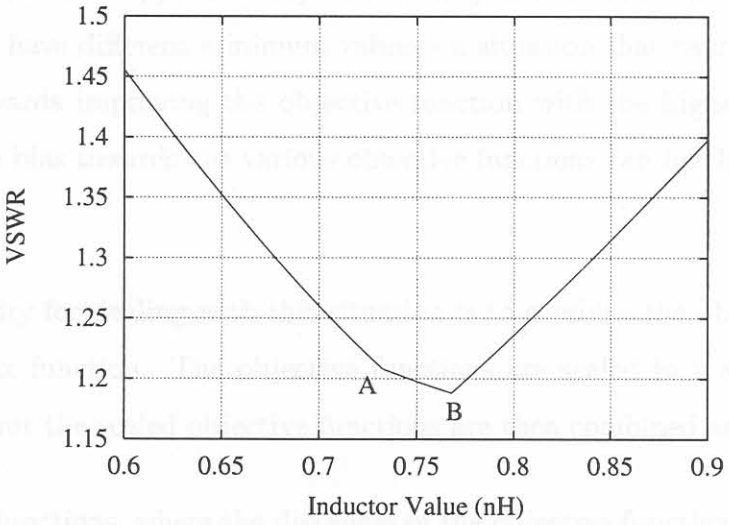
Charalambous [13] reviews a method for converting a minimax problem to a form where the objective function is smooth and has continuous gradients. This conversion is based on

$$M(\mathbf{x}) = \lim_{p \to \infty} \left([f_i(\mathbf{x})]^p \right)^{1/p} \tag{2.7}$$

where $M(\mathbf{x})$ is the minimax error. The main difficulty with the formulation given in (2.7) is that p must be very large for the approximation to be accurate, leading to ill-conditioned problems due to the finite precision of a computer. The techniques reviewed by Charalam-



(a) Frequency response.



(b) Variation with one variable.

Figure 2.1: Minimax Function.

bous [13] are significantly more powerful than (2.7) and relax the requirement for very large values of p.

2.1.1.3 Multi-Objective Optimisation

Multi-objective optimisation considers the case where multiple objective functions must be optimised simultaneously. These objective functions usually place conflicting requirements on the system parameters. This scenario is extremely common in engineering design where a number of specifications, such as cost and size, must be traded off to arrive at a good solution.

The first option is to scale the objective functions to form a composite objective function of the form

$$P(\mathbf{x}) = \sum_{i \in I} a_i f_i(\mathbf{x}) \tag{2.8}$$

where a_i are weights used to scale the objective functions $f_i(\mathbf{x})$, and $P(\mathbf{x})$ is the composite objective function. This approach helps limit the problems that can arise when the objective functions have different minimum values - a situation that causes the optimisation to be biased towards improving the objective function with the highest minimum value. Additionally, the bias towards the various objective functions can be changed by adjusting the weights.

Another possibility for dealing with this situation is to combine the objective functions to create a minimax function. The objective functions are scaled in a similar way to that shown in (2.8), but the scaled objective functions are then combined according to (2.6).

Fuzzy objective functions, where the distances of the objective functions' values from their minima are combined using fuzzy logic, can also be used [16]. This approach requires a knowledge of the optimum value of each objective function, but this is comparatively simple to calculate by solving the problem once for each objective function while ignoring the others.

When specifications for the worst acceptable value of each objective function exist, one of the objective functions can be optimised while the others are considered as constraints. This is a particularly useful approach where the primary requirement is to minimise one of

the objective functions, subject to the other objective functions meeting their specifications.

The last approach to multi-objective optimisation that will be considered here uses the concept of Pareto optimality. Pareto optimality [17] provides a way of ranking a number of solutions to an optimisation problem. This means that Pareto optimality is difficult to apply to algorithms that only process one solution, and is much better suited to algorithms where a number of solutions exist such as genetic algorithms. Pareto optimality relies on the concepts of dominated and non-dominated solutions to a problem. A solution to a problem dominates a second solution if the first solution is at least as good as the second solution for every objective function, and the first solution is better than the second for at least one objective function. In mathematical terms this means that solution $\mathbf{x}_1$ dominates solution $\mathbf{x}_2$ if

$$\forall i \in \{1, 2, \ldots, n\}, f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2) \wedge \exists i \in \{1, 2, \ldots, n\}, f_i(\mathbf{x}_1) < f_i(\mathbf{x}_2) \ . \tag{2.9}$$

Pareto optimality is implemented by finding all non-dominated solutions in the current set of solutions and giving them an objective function value. Those individuals are then removed from consideration and the next layer of non-dominated solutions are found. These solutions are given a worse objective function value than the previous layer and the process is then repeated until all solutions have been considered. The major advantage of an algorithm that uses Pareto optimality is that a number of results presenting various compromises between the objective functions are available.

Borghi *et al.* [16] consider multi-objective optimisation applied to Loney's solenoid problem which has two objective functions related to a solenoid's fields. The weighted objective functions, fuzzy, constraint, and Pareto approaches were tested and compared. The best results were obtained in the weighted and Pareto cases with the Pareto case requiring more function evaluations. These extra function evaluations are compensated for by the fact that the Pareto case gives the designer a large number of options to choose from.

2.1.2 Line Searches

Local optimisation is the process of taking a good starting point and finding the best result near that point. Line search algorithms are a special case of local optimisation where the optimisation takes place in only one dimension. This is a very important case that forms an integral part of many of the multi-dimensional algorithms described in Section 2.1.3. Many of the problems encountered in line searches are also applicable to the higher dimension cases considered later, so the study of line searches is extremely useful. This section will review the most important topics dealing with line searches.

2.1.2.1 Bracketing an Optimum

All the line search methods considered in subsequent sections assume that an interval with one and only one optimum is available. This is not always the case and a method is required to initially bracket a unique minimum before a line search can be used. This bracketing problem has not been extensively considered [14], but some algorithms are available.

One bracketing technique is to fit a parabolic function through three points and then to choose a point beyond the parabolic function's optimum. This procedure can be repeated until an optimum is bracketed. Care must be taken to ensure that a maximum is not bracketed when a minimum is required (or vice versa, as the case may be).

Another possibility is to use some heuristic technique to search along a line until an optimum is bracketed. The most common way of doing this is to move in the direction of the line search with steps of increasing size until an optimum is bracketed. An optimum is bracketed when the middle point has a lower objective function value than the first and last points, or two points have gradient values with different signs.

The main problem with these approaches is that it is possible to bracket more than one minimum and the line search methods considered below require an interval with a unique minimum.

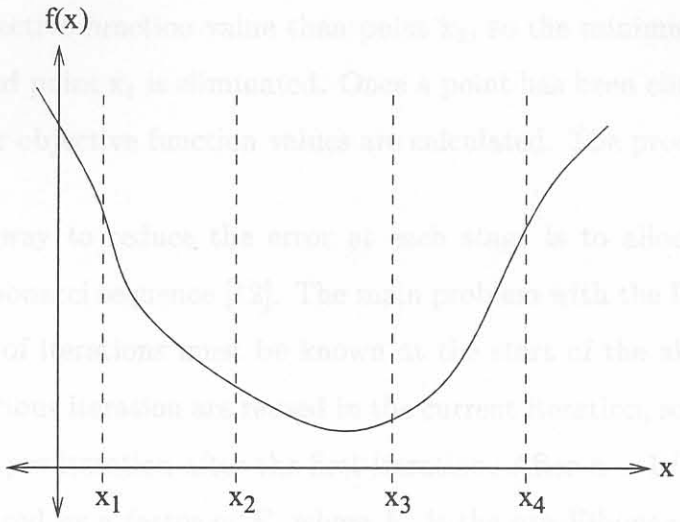


Figure 2.2: Fibonacci line search.

2.1.2.2 Direct Elimination Methods

Direct elimination methods operate by reducing the range around a minimum at each iteration of the algorithm. This means that the algorithm will monotonically converge to an optimum. Direct elimination methods only use the values of the objective function and do not require gradient or any other information about a function. This means that direct elimination methods have the advantage that they can be used on any objective function. The disadvantage of this approach is that useful information such as the gradient of a function is ignored even when it is available.

It is very important that a minimum is bracketed before using a direct elimination method because otherwise the algorithm can converge to an incorrect result. It is also very important that the bracketed range includes only one minima for the same reason.

Direct elimination methods work with four points because it is impossible to determine whether the upper or lower part of a range must be eliminated with only three points. During each iteration the two interior points' values are used to determine whether the first or last point of the current range must be eliminated. For example in Figure 2.2, point

x_2 has a higher objective function value than point x_3, so the minimum must be between points x_2 and x_4, and point x_1 is eliminated. Once a point has been eliminated, new points are chosen and their objective function values are calculated. The procedure then repeats.

The most efficient way to reduce the error at each stage is to allocate the four points according to the Fibonacci sequence [12]. The main problem with the Fibonacci line search is that the number of iterations must be known at the start of the algorithm. The three points from the previous iteration are reused in the current iteration, so only one new point has to be evaluated per iteration after the first iteration. After $n-1$ iterations, the initial search range is reduced by a factor of F_n where F_n is the nth Fibonacci number.

The golden ratio search is very similar to the Fibonacci algorithm except that the points are chosen according to the golden ratio. As with the Fibonacci line search, only one new point has to be evaluated at each iteration after the first iteration. After $n-1$ iterations, the initial search range is reduced by a factor of τ^{n-1} where τ is the golden ratio. The advantage of this formulation is that the number of steps does not have to be known in advance because the use of the golden ratio ensures that iterations can continue forever if rounding errors can be ignored. The penalty is that the golden ratio search is not quite as efficient as the Fibonacci algorithm, and will have an error approximately 17% worse than the Fibonacci algorithm after a large number of iterations [12].

The golden ratio line search is generally favoured over the Fibonacci line search because the number of iterations does not have to be known before the line search commences. This is a major advantage where the line search is done to a specified precision and the initial interval size is not fixed.

2.1.2.3 Interpolation Methods

Interpolation methods approximate the function by a low order polynomial and calculate the minimum of that polynomial to find the next point in the line search. The advantage of this approach is that convergence can be greatly accelerated.

The convergence can become quadratic when a parabolic interpolation function is used; that is the precision of the solution doubles at each iteration [14]. The main difficulty with interpolation methods is that they only work well for smooth functions and can actually be slower than direct elimination methods in some cases. Care must also be taken to ensure that the algorithm converges to a minimum and not a maximum (or vice versa, as the case may be). More information on interpolation methods can be found in Press *et al.* [14] and Jang *et al.* [18].

2.1.2.4 Line Searches with Gradients

The direct elimination and interpolation line search algorithms ignore gradient information. While this approach is justified when gradient information is not available, it does not make sense to ignore useful information about a function when it is available.

Press *et al.* [14] advocate a conservative approach based on the bisection method [19]. Three points are used, with the gradient at the centre point being used to determine whether the upper or lower portion of the range must be eliminated. This technique will halve the range at each step, representing a tremendous advantage over the direct elimination methods considered above. The initial range is reduced by a factor of 2^n after n iterations [19].

Jang *et al.* [18] suggest using Newton's method or the secant method [19] to find the point where the gradient is zero. Newton's method uses the first and second derivatives of the objective function to rapidly converge to the minimum. This approach has the disadvantages that the second derivatives must be calculated, and it can fail to converge. The secant method is similar to Newton's method except that it uses the last two points tested to generate an approximation to the second derivative of the objective function. The secant method's convergence rate can be faster or slower than the bisection method depending on the objective function, and the secant method can fail to converge for some problems [14].

The bisection and secant methods are recommended over Newton's method because they do

not require second derivatives. The convergence rate of the secant method will generally, but not always, be better than the bisection method. The bisection method has the advantages that it is simple to implement, and the number of iterations to achieve a specified precision can be calculated.

2.1.3 Local Optimisation

Local optimisation is the process of taking a good starting point and finding the best result near that point. The performance of a local optimisation algorithm thus depends heavily on the starting point chosen. This section will review the most important classes of local optimisation algorithms.

Simplex, direction set, and gradient methods will be considered in Sections 2.1.3.1 to 2.1.3.3. The gradient methods considered include steepest descent, Newton's method and quasi-Newton methods, conjugate gradient methods, and trust region methods.

2.1.3.1 Simplex Methods

Simplex methods are heuristic optimisation algorithms that do not require gradient evaluations or line searches. However this does not mean that they require less function evaluations than other algorithms, in fact the opposite is often true [14], but simplex methods can be used to quickly arrive at a useful solution.

For a problem with n variables, a simplex method requires $n+1$ points chosen so as to give a non-zero volume. The algorithm proceeds by modifying the position of the points in an attempt to improve the objective function value. The most popular simplex method is the Nelder-Mead algorithm [20], but other options do exist [21].

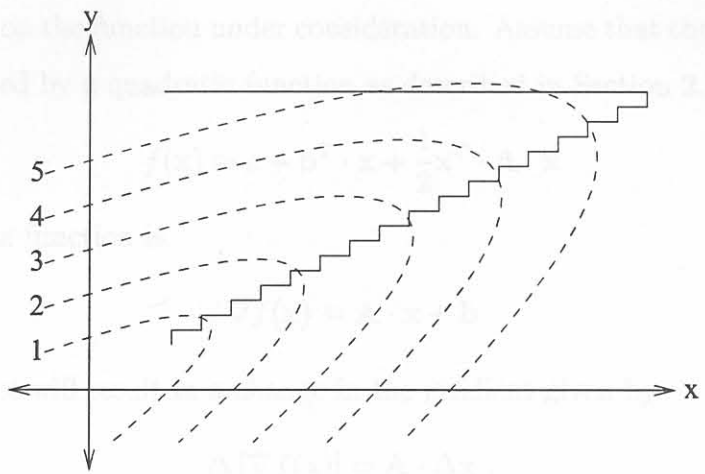


Figure 2.3: Optimisation by line searches along coordinate directions.

2.1.3.2 Direction Set Methods

This section will consider optimisation techniques that use line searches, but do not require gradient information. The most important concept in this section is the notion of conjugate directions. This concept also forms the basis of the conjugate gradient techniques covered later. The information in this section is adapted from Press *et al.* [14].

Line searches are a very powerful method for optimising in one dimension. Unfortunately the principles applied in line searches are not directly applicable to higher order problems. This leads to the possibility of using line searches as part of an optimisation algorithm that is applicable to high order problems. The main difficulty is choosing the line search directions. One possibility is to optimise each variable in turn, but this approach is very inefficient in cases like the one shown in Figure 2.3. The main problem with this approach is that each line search spoils the results of the previous line searches so the algorithm zig-zags down the valley, converging towards the optimum very slowly. This problem can be overcome by using previous results to move down the valley by searching in directions that do not interfere.

Directions that do not interfere are known as conjugate directions. Which directions are

conjugate depends on the function under consideration. Assume that the objective function can be approximated by a quadratic function as described in Section 2.1.1.1.

$$f(\mathbf{x}) \approx c + \mathbf{b}^T \cdot \mathbf{x} + \frac{1}{2}\mathbf{x}^T \cdot \mathbf{A} \cdot \mathbf{x} \tag{2.10}$$

The gradient of this function is

$$\nabla f(\mathbf{x}) = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \tag{2.11}$$

and any change in $\mathbf{x}$ will result in a change in the gradient given by

$$\Delta\left[\nabla f(\mathbf{x})\right] = \mathbf{A} \cdot \Delta\mathbf{x} \ . \tag{2.12}$$

Assume that a line search has found a optimum in some direction $\mathbf{u}$ and a line search in some direction $\mathbf{v}$ must now take place. The directions $\mathbf{u}$ and $\mathbf{v}$ are conjugate if the gradient change due to the second line search is orthogonal to the first line search direction. Two vectors are orthogonal when their scalar product is zero giving

$$0 = \mathbf{u} \cdot \Delta\left[\nabla f(\mathbf{x})\right] \ . \tag{2.13}$$

Now the gradient change can be replaced by the result in (2.12) giving

$$0 = \mathbf{u} \cdot \mathbf{A} \cdot \mathbf{v} \ . \tag{2.14}$$

Thus two directions $\mathbf{u}$ and $\mathbf{v}$ are said to be conjugate when (2.14) holds. Obviously this result is only strictly valid when the objective function is quadratic, but in practice it is a good approximation to most smooth functions.

The main task of a direction set method is thus to generate a set of conjugate directions. Once a set of conjugate directions are found, it can be shown that convergence will be quadratic. A more complete discussion of conjugate direction set methods is given by Powell [21].

2.1.3.3 Gradient Methods

This section will consider optimisation algorithms that use gradient information to improve convergence rate. Obviously these algorithms can only be used where gradient information

is available or can be inferred from other calculations. The information in this section is based on Press *et al.* [14].

The presentation will start by considering some methods to obtain the gradient of a function. The optimisation algorithms considered are the steepest descent algorithm, Newton's method and quasi-Newton methods, conjugate gradient algorithms, trust region methods, and algorithms that use gradient information, but do not require line searches.

Calculating the Gradient

Some methods for calculating the gradient will be considered in this section. Obviously the simplest approach is to perform symbolic differentiation of the objective function and supply this gradient function to the algorithm. When this is not possible, finite differences or automatic differentiation can be used. The information in this section is obtained from Nocedal and Wright [15].

Finite difference methods are based on the definition of a derivative given in (2.15).

$$\frac{\partial f}{\partial x_i} = \lim_{h_i \to 0} \frac{f(x_i + h_i) - f(x)}{h_i} \quad \text{for} \quad i = 1, 2, \ldots, n \tag{2.15}$$

where h_i is some deviation in direction x_i, and n is the number of variables. By removing the limit in (2.15), the gradient can be approximated by

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + \Delta d_i) - f(x)}{\Delta d_i} \quad \text{for} \quad i = 1, 2, \ldots, n \tag{2.16}$$

where Δd_i is some small deviation in direction x_i. This approach in known as the finite difference method because finite differences are used instead of the infinitesimal differences required in (2.15). The main implementation difficulty is to determine the size of Δd_i so that it is small enough to give a good approximation, but not so small that the finite precision of the computer introduces errors. The main problem with finite-difference methods is that a function with n variables requires n function evaluations to calculate the gradient (assuming that the function value at the current point is already known). A more detailed discussion of finite difference methods can be found in Mathews [19].

Automatic differentiation algorithms rely on the fact that any function consists of a number of simple sub-functions. The chain rule can then be used in conjunction with the derivatives of these simple sub-functions to calculate the derivative of the complete function. The result of this approach is a gradient function which is then used to compute gradients.

Steepest Descent Algorithm

The steepest descent algorithm is the simplest gradient-based optimisation algorithm. The principle is to perform a line search in the direction of steepest descent from the current point. It would seem that this algorithm should perform very well, but this is not the case.

The direction of steepest descent is the opposite direction to the gradient vector. When a line search has been completed, the optimum in the direction of the line search has been found, so the component of the gradient in that direction must be zero. This means that the gradient at the end of a line search will be perpendicular to the direction of the line search, and each line search performed by the steepest descent algorithm will thus be perpendicular to the previous line search. The objective function shown in Figure 2.4 has a narrow valley and the steepest descent algorithm performs very poorly because it zig-zags across the valley rather than moving along it. This problem is seen to be very similar to that shown in Figure 2.3. For this reason steepest descent is not a popular optimisation algorithm.

The major advantages of the steepest descent algorithm are that no data has to be stored between steps and second derivatives are not required.

Newton's Method

Newton's method works by finding the minimum of the quadratic approximation to a function. The formulation allows rapid quadratic convergence, but requires a large number of gradient calculations.

Section 2.1.1.1 shows that any function can be approximated as a quadratic function by

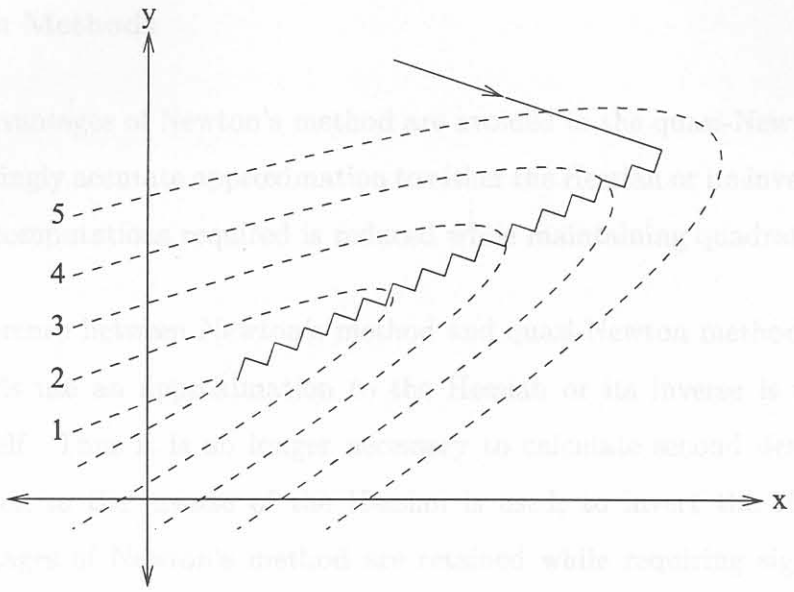


Figure 2.4: Steepest descent algorithm in a narrow valley.

using the Taylor expansion as shown in (2.3). The gradient of this function is

$$\nabla f(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \mathbf{A} \cdot \Delta\mathbf{x} \; . \tag{2.17}$$

At an optimum the gradient will be equal to zero, so the step to reach the optimum ($\Delta\mathbf{x}$) is given by

$$\Delta\mathbf{x} = -\mathbf{A}^{-1} \cdot \mathbf{b} \; . \tag{2.18}$$

The next optimisation step is thus calculated from the inverse of the Hessian and the gradient.

There are a number of problems with Newton's method. The first problem is that the Hessian and its inverse must be calculated at each step, requiring a large amount of processing power and storage. Newton's method can fail to converge or can converge to an incorrect result when the Hessian is not positive definite. Modifications to the basic version of Newton's method that overcome these difficulties do exist [15], but the quasi-Newton methods are significantly more popular.

Quasi-Newton Methods

The main disadvantages of Newton's method are avoided in the quasi-Newton methods by using an increasingly accurate approximation to either the Hessian or its inverse. In this way the number of computations required is reduced while maintaining quadratic convergence.

The major difference between Newton's method and quasi-Newton methods is that quasi-Newton methods use an approximation to the Hessian or its inverse is used instead of the Hessian itself. Thus it is no longer necessary to calculate second derivatives, and if an approximation to the inverse of the Hessian is used, to invert the Hessian. In this way the advantages of Newton's method are retained while requiring significantly fewer computations.

Once an approximation to the Hessian or its inverse has been formed, quasi-Newton methods use the Newton step given in (2.18) as a line search direction. Unlike other algorithms, the line search does not have to be highly accurate [12, 14].

The major disadvantages of quasi-Newton methods is that a large matrix must be manipulated and this can cause difficulties with problems with surprisingly few variables. The main advantages of quasi-Newton methods are rapid quadratic convergence (faster than the conjugate gradient methods [15]) and the fact that accurate line searches are not required. This means that the number of function evaluations required by the line search algorithm can be reduced. More information on quasi-Newton methods is given by Charalambous [13], Press *et al.* [14], and Nocedal and Wright [15].

Conjugate Gradient Algorithms

The conjugate gradient algorithms extend the notion of conjugate directions discussed in Section 2.1.3.2 to the case where gradient information is available. This results in a dramatic acceleration of the convergence because the extra information provided by the gradient is effectively used.

As with the direction set methods considered in Section 2.1.3.2, the main problem is to establish a set of mutually conjugate directions. The basic principle is to use the previous search direction and the gradient at the current point to determine the next search direction.

The major advantages of conjugate gradient techniques are that only the last search direction needs to be stored, and they converge quadratically (although slower than the quasi-Newton methods [15]). More information on conjugate gradient techniques is given by Press *et al.* [14], Nocedal and Wright [15], and Jang *et al.* [18].

Trust-Region Methods

Trust-region methods use similar principles to Newton's method and quasi-Newton methods. The difference arises because trust-region methods do not use line searches, but instead search within some volume round the current point.

Trust region methods are based on the assumption that any approximation to a problem will be most accurate near the current point. The region where a model is assumed to be accurate, a trust-region, is thus established round the current point. The most common way to do this is to have a radius which is adjusted after each step depending on how accurate the approximation proved to be. The radius is increased when the model proves to be accurate and decreased when the model is inaccurate.

A number of steps are performed for each iteration of a trust-region method. The first step is to calculate the minimum value of the approximation within the trust-region. The value at this point is then compared to the objective function value and the point is accepted if the agreement is satisfactory. If the agreement is unsatisfactory, the new point is discarded, the trust region volume decreased, and the process restarted. The Hessian can be computed directly as in Newton's method, an approximation can be constructed in the same way as the quasi-Newton method, or some other method can be used. Methods for finding the minimum of the approximation within the trust-region are considered by Nocedal and Wright [15].

The most important advantage of trust-region methods is that the Hessian is not always required to be positive-definite. This means that problems that cannot be solved with other methods can be solved by trust-region methods.

Methods That do not Require Line Searches

Apart from the trust region methods, all the methods considered so far require line searches to determine the size of movement in the search direction. Some algorithms that do not require line searches are briefly considered in this section.

Jang *et al.* [18] present a number of methods based on steepest descent that do not require line searches. These methods work by taking a single step in the search direction. The size of the step can be constant or can be adjusted by a heuristic to obtain faster convergence.

The Leap-Frog algorithm developed by Snyman [22–24] is another example of a local optimiser that does not require line searches. The leap-frog optimisation algorithm is based on the movement of a particle in a conservative force field. Such a particle will attempt to minimise its energy by moving to a point with lower energy. Friction is ignored by the algorithm, so a heuristic braking technique is used to ensure that the particle eventually settles at a local optimum. The major advantage of the Leap-Frog algorithm is that it is very robust – it is able to solve problems that other algorithms cannot. Some of the reasons for this robustness are the fact that it is not based on a quadratic approximation to the problem, and the absence of line searches allows it to cope with noisy objective functions.

2.1.4 Global Optimisation

The majority of practical optimisation problems have a number of local optima with the best local optimum being known as the global optimum. Local optimisation techniques can only give a good result if the starting point is near a good solution. For the purposes of this dissertation, global optimisation is defined as the process of finding a starting point

for a local optimisation algorithm that is near the global optimum or an optimum with a similar objective function value to the global optimum.

The main problem with global optimisation is that there is no simple approximation like the quadratic approximation for local optimisation. This means that global converge criteria cannot be mathematically derived, and global optimisation algorithms are thus heuristic. The quality of the final result will improve as the number of points sampled increases until the global optimum is found.

A reasonably intuitive approach is to use an exhaustive search. A grid is applied to the problem and each point on the grid is sampled and compared to other points. The best point can the be used either as the global optimum or as the starting point for a local optimiser. Other options are to apply a local optimiser to every grid point or a subset of the grid points. The choice of grid size is critical because a very fine grid will take a long time to sample, but a very coarse grid could easily miss a good solution. The major problem with an exhaustive search is the extremely large number of points that must be sampled. This has led to the development of the heuristic techniques below.

Clustering methods [25] reduce the number of points that have to be passed through a local optimiser by identifying points that are near the same local optimum. The reduces the unnecessary extra computations that result whenever a local optimum is found more than once.

Simulated annealing attempts to emulate the annealing process in materials [14]. A material is heated to a high temperature and then slowly cooled during annealing. Large changes in crystal structure can occur when the temperature is high, but much smaller changes occur at low temperatures. The crystal structure of the material is in a much lower energy state (fewer dislocations and grains) after annealing than before annealing [26] so this approach forms a good basis for an optimisation algorithm. This is done by allowing large changes near the beginning of a run when the current solution is probably far from a minimum, and allowing only small changes near the end of a run when the current solution is assumed to be

close to a minimum. The search space is thus efficiently sampled near the start of a run and good solutions are well optimised near the end of a run. This variation in allowable changes is accomplished by slowly reducing the value of a variable analogous to temperature in annealing. The main difference between simulated annealing and the algorithms described above is that changes that produce a deterioration in objective value are sometimes allowed. This means that simulated annealing can escape from local optima and find good results in the global sense. The main differences between various simulated annealing algorithms lie in cooling schemes and the probabilities with which changes that worsen the objective value are accepted [27].

Genetic algorithms imitate evolution and natural selection by maintaining a population of solutions and applying genetic operators such as crossover (breeding). The motivation for this approach is the remarkable way in which natural systems have developed to survive in even the harshest environments. The fact that a population of solutions is maintained means that genetic algorithms sample the entire problem space in a structured manner. Genetic algorithms will be discussed in considerably more detail in Section 2.2.

The use of a population to efficiently explore the problem space has led to the development of a number of other global optimisation methods based on a population of individuals. Particle swarm optimisation [28] attempts to imitate the motion of groups of animals such as a swarm of insects, a flock of birds, or a school of fish. The basic principle is that, while each individual is locally searching for a good solution, the interaction between individuals leads to a global search effect. The major advantage of this approach is that each individual can be very simple without compromising the optimisation performance of the swarm as a whole.

2.1.5 Combinatorial Optimisation

Combinatorial optimisation consists of selecting the best combination or permutation of elements. A good example of a combinatorial optimisation problem is the classic travelling

salesman problem where the total distance travelled to visit every node in a given area once and only once must be minimised. This section will consider some of the main combinatorial optimisation algorithms.

Simulated annealing has already been considered in connection with global optimisation. Simulated annealing can be applied to combinatorial optimisation problems because the principle of reducing the size of allowable changes during a run is also applicable to combinatorial optimisation.

Genetic algorithms have also been mentioned above in terms of their global optimisation properties. Genetic algorithms can also be applied to combinatorial optimisation problems by modifying the representation and genetic operators used.

Tabu search is a combinatorial optimisation technique that works by disallowing some modifications to the current solution (making those modifications taboo) [29]. The main motivation for this approach is to avoid revisiting areas of the problem space that have already been tested. A simple example is to ensure that the last modification is not undone, thereby returning to the previous solution. The recent history of the algorithm is used in conjunction with a number of heuristic techniques to establish which modifications will be disallowed.

2.1.6 Hybrid Optimisation Algorithms

The optimisation methods described above apply a number of different techniques to the problem of finding the best solution to a problem. Each of these algorithms has advantages and disadvantages. Recently, hybrid algorithms that combine more than one of the algorithms considered above have been developed in an attempt to accentuate the advantages and reduce the disadvantages of each algorithm. The basis for these developments is the "No Free Lunch" Theorem.

Recently a result known as the "No Free Lunch" Theorem was published by Wolpert and

Macready [30]. The basic premise is that any algorithm that performs very well on a particular type problem pays for this with poor performance on another type of problem. A simple example of this principle is that global optimisation algorithms perform very well on global optimisation problems, but very poorly on local optimisation problems. This suggests that hybrid algorithms should perform better than single algorithms when a wide variety of problem types is considered.

Groenwold and Hindley [31] used this principle in the development of a global optimisation algorithm. Their approach applies a number of optimisation algorithms to a problem, and then uses a Bayesian criterion to combine those results and determine when to terminate the search. The results were superior to those obtained when the sub-algorithms were used independently. The major advantage of this approach is that the algorithms can be run independently in parallel, allowing maximum advantage to be taken of clusters of computers.

Another approach to the development of hybrid optimisation algorithms is to integrate the algorithms so that they no longer function independently. Duch and Korczak [32] list a number of possibilities like the integration of simulated annealing and a genetic algorithm. The benefit of this approach is seen by the fact that Michalewicz [33] found that a genetic mutation operator based on simulated annealing (non-uniform mutation) significantly improved the performance of a genetic algorithm. Duch and Korczak [32] suggest that the integration of local and global optimisation algorithms should produce good results.

Genetic algorithms have been combined with local optimisers by Renders and Flasse [34], and Salomon [35]. Renders and Flasse [34] imitate learning in natural systems by creating a new generation using a genetic algorithm and then running a local optimiser to simulate learning. The results of the hybrid algorithm are significantly more reliable than the local optimiser and more accurate than the genetic algorithm, while requiring fewer function evaluations than the genetic algorithm. Salomon [35] compares genetic algorithms and gradient based local optimisers and gives some similarities and differences. This discussion

forms the basis for the development of a new algorithm which is a combination of the two approaches. The performance of the hybrid algorithm is better than more established algorithms on some problems.

2.1.7 Constraints

All the algorithms considered so far do not consider constraints and are known as unconstrained optimisation algorithms. Most practical problems have constraints imposed by physical laws such as the speed of light, and other limitations such as size and cost. The process of finding the values of inputs to a system that satisfy all constraints and give the optimum objective function value is known as constrained optimisation. This section will consider some methods of accounting for constraints. The majority of the information contained in this section was obtained from Nocedal and Wright [15].

A simple method of dealing with constraints is to add some penalty to the objective function value whenever a constraint is violated and then to use an unconstrained optimisation algorithm. This principle can be represented as

$$f_c(\mathbf{x}) = f(\mathbf{x}) + \frac{1}{\mu}\sum_i g_i^m(\mathbf{x}) + \frac{1}{\mu}\sum_j h_j^n(\mathbf{x}) \tag{2.19}$$

where $f_c(\mathbf{x})$ is the new objective function, $f(\mathbf{x})$ is the objective function, μ is the penalty parameter which is greater than zero, $g_i(\mathbf{x})$ are the inequality constraints, $h_j(\mathbf{x})$ are the equality constraints, and m and n are values greater than or equal to one, and i and j are the indices of constraints that are violated. The constraint functions are less than zero when a constraint is satisfied and are greater than zero when a constraint is violated, so they are only included in (2.19) when a constraint is violated. As n is increased or μ is decreased the penalty for violating a constraint increases. The main difficulty with this approach is that the Hessian may become ill-conditioned meaning that most optimisation methods will perform poorly. A further problem is that the penalty function can shift the positions of optima and this can cause solutions that violate the constraints by a small amount to be produced. Penalty functions have however been successfully applied by

Snyman [24] for an algorithm that does not use the Hessian. Nocedal and Wright [15] review the augmented Lagrangian method which reduces the problems with ill-conditioning, but the theory required for this method is beyond the scope of this work.

Another method of dealing with constraints is to transform the problem so that the new independent variables can have any value. A possibility reviewed by Bandler [12] is

$$x_i = x_{li} + \frac{1}{\pi}(x_{ui} - x_{li}) \operatorname{arccot}(x_i') \tag{2.20}$$

where x_i is an independent variable of the original problem, x_{ui} and x_{li} are respectively the upper and lower bounds of the independent variable, and x_i' is the new independent variable. An unconstrained optimisation algorithm can now be applied to the new problem with $\mathbf{x}'$ as the independent variable. Nocedal [15] considers similar algorithms known as barrier function methods. The main problem with barrier function methods is that ill-conditioning of the Hessian can occur.

Other approaches using approximations of both the problem and the constraints in the neighbourhood of the current point have been developed. Nocedal and Wright [15] review the quadratic programming and sequential quadratic programming approaches. Snyman [36, 37] has recently proposed a new algorithm that uses a very simple approximation to the problems yet achieves excellent results. The main difficulty with these approaches is that a complicated sub-problem must be solved at each iteration, limiting the value of these algorithms where the evaluation of the objective function is fast. The main advantage of these approaches is that they typically require fewer objective function evaluations than the algorithms considered above making them very well suited to the case where the evaluation of the objective function is costly.

2.2 Genetic Algorithms

Genetic algorithms are numerical techniques that attempt to imitate evolution and natural selection. The motivation for this approach is the remarkable way in which natural systems