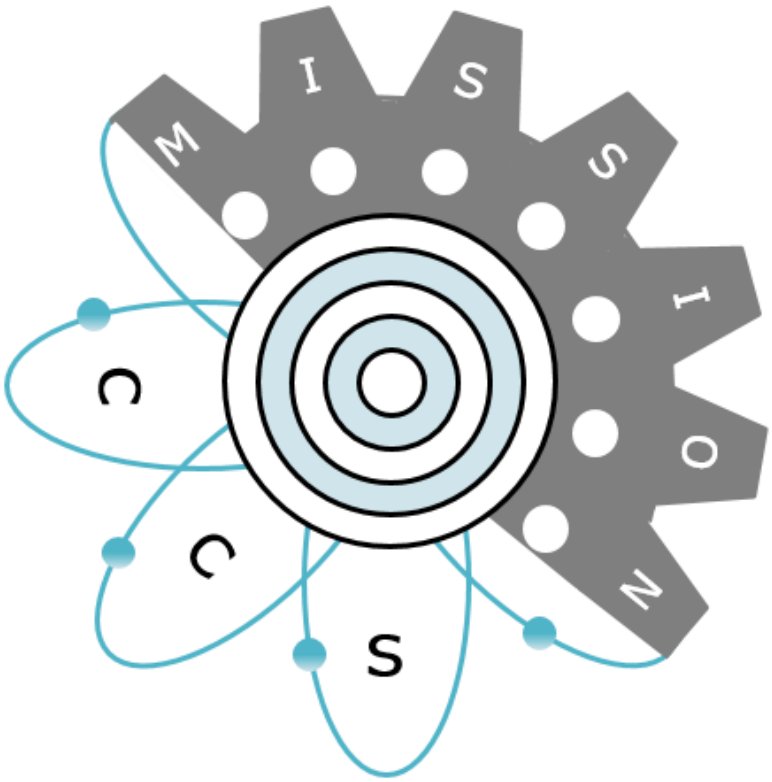


Introduction to Design Optimization

Harvey Thompson

Chapter 1

Introduction to Design Optimization

1.0 Introduction

The process of design and manufacture has developed over the centuries. Complex systems such as: buildings, bridges, cars, aircraft, space vehicles, are an excellent demonstration of the design process. However, the evolution of these systems has been slow.

The entire process is time-consuming and costly, requiring substantial human and material resources. Therefore, the procedure has been to design, manufacture and use a system, regardless of whether it is the best one. Improvements to these systems have been made only after a substantial investment has been recovered. The thing to appreciate is that several systems can usually accomplish the same task, and that some systems are better than others.

For example: The purpose of a bridge is to provide movement of people or vehicles from one side of a river or road to the other. Different types of bridges can serve this purpose. However, to analyse and design all possibilities can be time-consuming and costly. Usually one type is selected based on some preliminary analyses and then it is designed in detail.

Figures 1.1 to 1.5 show different solutions to the task of designing a foot bridge.

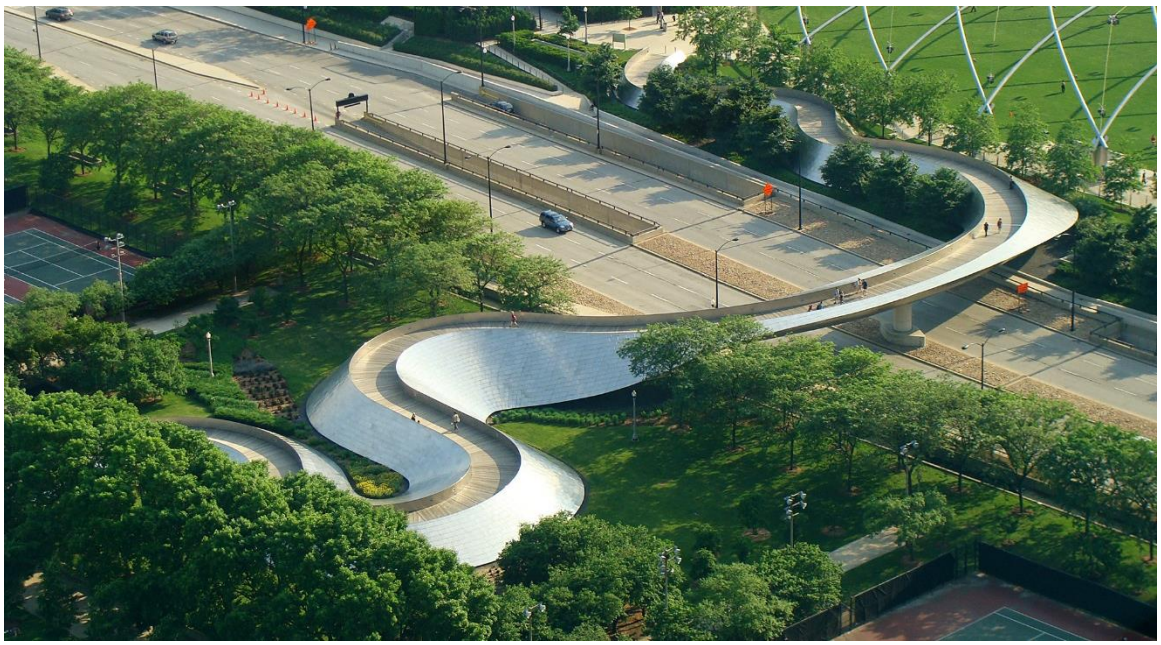


Figure 1.1: BP Pedestrian Bridge: concealed box girder footbridge Chicago, USA

Figure 1.2: Millennium bridge, on the river Thames, London.

Figure 1.3: Gateshead Millennium Bridge on Newcastle upon Tyne

Figure 1.4: Arched Pedestrian Bridge, Hachaturyana street, Moscow

Figure 1.5: Pedestrian footbridge based on Leonardo da Vinci's single span bridge near the town of Ås in Norway

In order to describe optimization concepts and methods, it is necessary to generate a mathematical statement for the optimum design problem. Such a mathematical model is defined as the minimization of a cost function while satisfying all equality and inequality constraints. This is the standard design optimization model used throughout this course.

1.1 Standard Design Optimization Model

The standard design optimization model, requires determining the values of a vector of ***n design variables = (x_1, x_2, . . ., x_n)*** in order to:

Minimize:

the cost function: $$f(x) = f(x_1, x_2, \cdots, x_n) \quad (1.1)$$

Subject to:

m inequality constraints: $$g_i(x) = g_i(x_1, x_2, \cdots, x_n) \le 0; \quad i = 1 \; to \; m \quad (1.2)$$

and ***p*** equality constraints: $$h_j(x) = h_j(x_1, x_2, \cdots, x_n) = 0; \quad j = 1 \; to \; p \quad (1.3)$$

Note that the limits on the design variables $x_i \ge 0$ or $x_{iL} \le x_i \le x_{iU}$ where x_{iL} and x_{iU} are the lower and higher limits for x_i, are included as inequality constraints. So, in its simplest form, a standard optimization problem is given by (1.4).

$$\begin{aligned} &\textbf{Minimize:} && f(x) \\ &\textbf{Subject to:} && g_i(x) \le 0; \quad i = 1 \; to \; m \\ & && h_j(x) = 0; \quad j = 1 \; to \; p \\ &\textbf{where:} && x_{iL} \le x_i \le x_{iU} \;\; or \;\; x_i \ge 0; \; i = 1 \; to \; n \end{aligned} \quad (1.4)$$

1.1.1 How to Treat Maximization Problems

The general design model treats only minimization problems. This is not a problem, since the maximization of a function ***F(x)*** is the same as minimization of the transformed function ***f(x)= –F(x)***. Considering the plot of Figure 1.6, of a function of 1 variable, with a maximum at ***x****, this is the same as the minimum of the negative of the function to maximise.

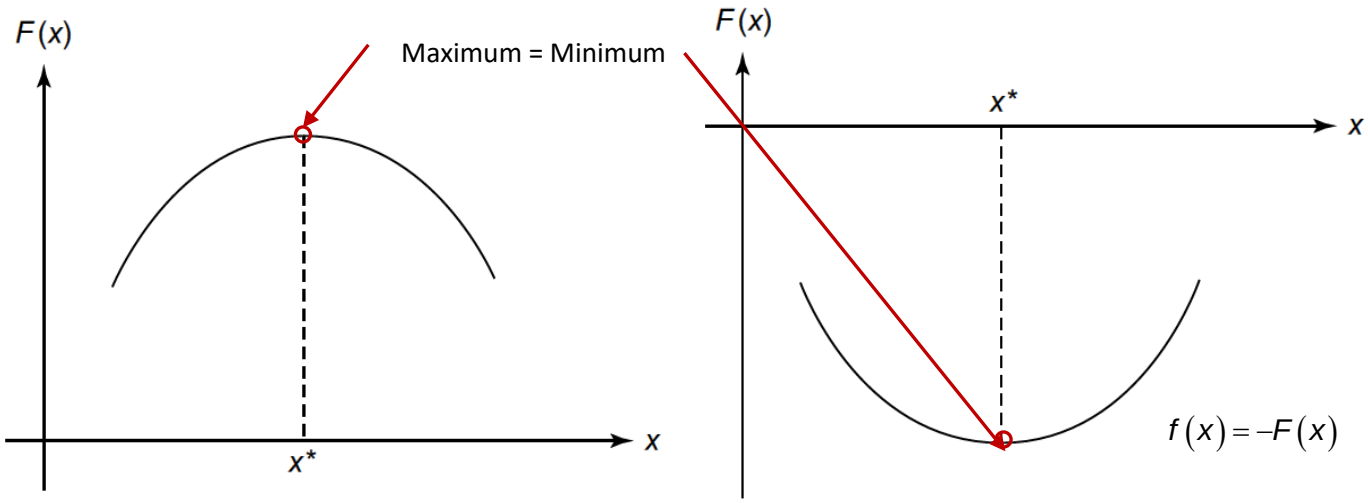


Figure 1.6: Plot of function of 1 variable: a) normal function showing the maximum at x*, b) the negative of the same function showing now the minimum at x*

1.1.2 Greater Than (≥) Constraints

The standard design optimization model only treats "Less Than (**≤)"** types of inequality constraints. But it is equally as likely for an optimization problem to have "Greater Than **≥** " type inequality constraints. It is relatively easy to convert from a "greater than" to "less than" type of inequality constraint.

So, if (1.5) is a greater than type of inequality constraint, all that is required, is to multiply (1.5) by –1, to convert it to the less than inequality constraint of (2.6).

$$G_j(x) \geq 0 \tag{1.5}$$

$$g_j(x) = -G_j(x) \leq 0 \tag{1.6}$$

1.1.3 Issues about the Standard Optimization Model

The following 6 issues need to be understood about the standard optimization model:

1. All functions f(x), h_j(x), and g_i(x) must depend on some or all of the design variables. Functions not depend on design variables can be ignored!
2. The number of equality constraints must be less than, or at the most equal to, the number of design variables $p \leq n$. If p > n, the system is overdetermined and some of the equality constraints are either redundant or inconsistent.
    a. If redundant: Constraints can be deleted until p < n, so that a solution is possible.
    b. If inconsistent: the design problem doesn't have a solution and the problem formulation needs to be re-examined. This means that two or more equations require different values for the same design variables, for example:

    $$x_1 = x_2, \ \textit{and} \ \ x_1 = \frac{1}{x_2}$$

    c. If p = n, no optimization of the system is necessary because the roots of the equality constraints are the only solution to the optimum design.
3. No restriction on number of inequality constraints. At the optimum, the total number of active constraints must be less than or at the most equal to the number of design variables.
4. Unconstrained problems: Some design problems may not have any constraints.
5. Linear programming problems: If all of the functions *f(x)*, $h_j(x)$, and $g_i(x)$ are linear with respect to the design variables x, then the problem is called a *linear programming problem* (LP). If any of these functions is nonlinear, the problem is called a *nonlinear programming problem* (NLP).
6. Scaling of problem functions: The cost function can be scaled by multiplying it with a positive constant. This has no effect on the optimum design. However, the optimum cost function value will change. Constants can also be added to the cost function without affecting the optimum design. Similarly, the inequality constraints can be scaled by any positive constant and the equalities by any constant.

1.1.4 Discrete and Integer Design Variables

Design variables x_i can have any numerical value within the feasible region. However, sometimes, these values may need to be discrete or integer, depending on the problem. So both need to be defined:

a) *Discrete Design Variables*: are those whose value must be selected from a given finite set of values. For example: A plate thickness must be one that is available commercially: 1/8”, 1/4", 3/8”, 1/2", 5/8”, 3/4", 1”, etc.
b) *Integer Design Variables:* Must have an integer value. For example: the number of bolts used, the number of coils in a spring, the number of items to be shipped, the number of pistons in an engine, etc. Problems with these design variables are called discrete and integer programming problems.

1.1.5 Types of Optimization Problems

The standard design optimization model can represent many different problem types. It can be used to represent unconstrained, constrained, linear programming, and nonlinear programming optimization problems.

It is also important to know other optimization problems encountered in practical applications. Many times, these problems can be transformed into the standard model and solved by the optimization methods presented.

There are a couple more types of optimization problems that need to be considered:

1. Continuous/Discrete-Variable Optimization Problems
2. Smooth/Non-smooth Optimization Problems.

1.1.5.1 Continuous/Discrete-Variable Optimization Problems

When the design variables can have any numerical value within their allowable range, the problem is called a continuous-variable optimization problem. When the problem has only discrete/integer variables, it is called a discrete/integer-variable optimization problem. When the problem has both continuous and discrete variables, it is called a mixed variable optimization problem.

1.1.5.2 Smooth/Non-smooth Optimization Problems

When the functions are continuous and differentiable, the problem is referred to as smooth (differentiable). There are also many practical applications where the problem functions are not differentiable or even discontinuous. Such problems are called nonsmooth (nondifferentiable). Numerical methods to solve these two classes of problems can be different. Theory and numerical methods for smooth problems are well developed. Therefore, it is most desirable to formulate the problem with continuous and differentiable functions as far as possible.

1.2 Example of Structural Optimization Design Problem

Consider the design of the cross-sectional dimensions of the rectangular beam of Figure 1.7 in order to minimize the area. At the same time it is desired to minimize the maximum shear stress in the beam corresponding to a unit shear force. Based on some physical constraints, the two variables, w and h, which are the width and height of the cross-section are limited to be in the range 0.5 < w, h < 30 mm.

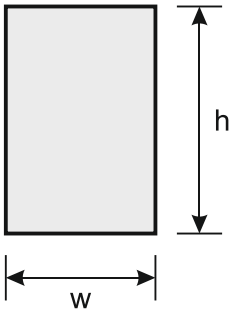


Figure 1.7: Beam cross-section to be minimised

The equation for the area and maximum shear stress are given by

$$f_1(w,h) = w \times h \tag{1.7}$$

$$f_2(w,h) = \frac{3V}{2w \times h} \tag{1.8}$$

Assume for this problem that we have an applied load which produces a shear force of V = 1000N. The contour lines for both objective functions are given by Figures 1.8 and 1.9.

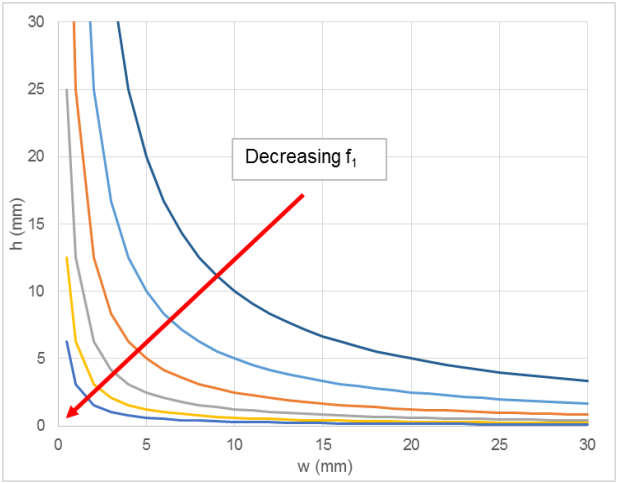


Figure 1.8: Design of beam cross-section for minimum area

The individual minima for the two functions are at the opposite corners of the design space, with the following values, $w_1^* = h_1^* = 0.5\,mm$ for minimum area and $w_2^* = h_2^* = 30\,mm$ for minimum shear stress with the associated function values of $f_1^*\left(w_1^*, h_1^*\right) = 0.25\,mm^2$ and $f_2^*\left(w_2^*, h_2^*\right) = 1.667\,MPa$ respectively.

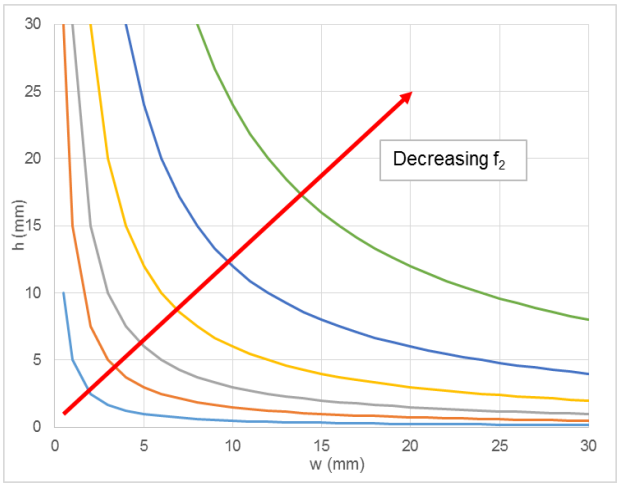


Figure 1.9: Design of beam cross-section for minimum shear stress

One way to solve this problem is to use the weighted objective function approach with equal weights for both objective functions, which results in the minimization of function (1.9).

$$F(w,h) = w \times h + \frac{3V}{2w \times h} \tag{1.9}$$

Since design variables *w* and *h* appear everywhere in the form of a product, we can treat this product as a single variable (x1), changing this equation into (1.10). The contour line for the new objective function is given in Figure 1.10.

$$F(x_1) = x_1 + \frac{3V}{2x_1} \tag{1.10}$$

Differentiating (1.10) and solving for the minimum gives:

$$F(x_1) = x_1 + \frac{3V}{2x_1}$$

$$\frac{d}{dx_1}F(x_1) = 1 - \frac{3V}{2x_1^2} = 0$$

$$x_1^2 = \frac{3V}{2}$$

$$\therefore x_1 = \sqrt{\frac{3V}{2}}$$

For V = 1000N,

$$F(x_1) = x_1 + \frac{3V}{2x_1}$$

$$x_1 = \sqrt{\frac{3V}{2}} = \sqrt{\frac{3 \times 1000}{2}} = \sqrt{1500} = 38.73\,mm^2$$

Which gives that $w^* = h^* = 6.22\,mm$, with objective function values of $f_1^* = 38.78\,mm^2$ and $f_2^* = 38.78\,MPa$

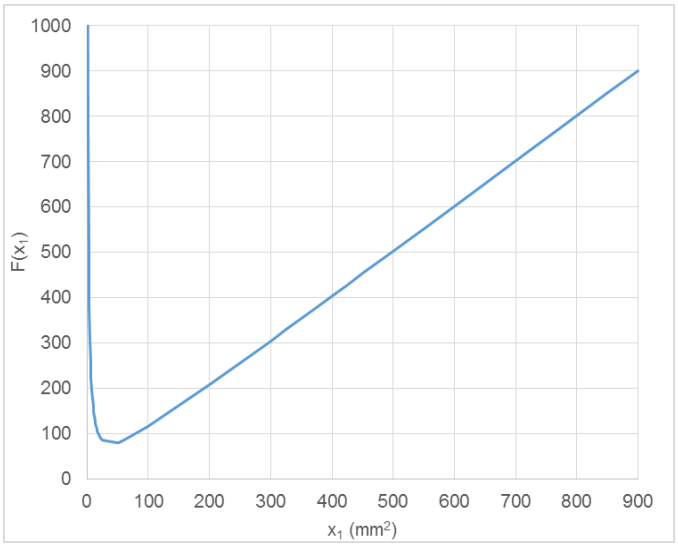


Figure 1.10: Design of beam cross-section for minimum equal weighted objective function

Alternatively, it may be more desirable to minimize the Euclidean norm between the individual minima and the final value. Which means minimising function (1.11).

$$F(w,h)=\left(\frac{wh-0.25}{0.25}\right)^2+\left(\frac{\frac{3V}{2wh}-1.667}{1.667}\right)^2 \tag{1.11}$$

The product *wh*, can again be treated as a single value to give (1.12). The contour line for the new objective function zoomed into the minimum region is given in Figure 1.11.

$$F(x_2)=\left(\frac{x_2-0.25}{0.25}\right)^2+\left(\frac{\frac{3V}{2x_2}-1.667}{1.667}\right)^2 \tag{1.12}$$

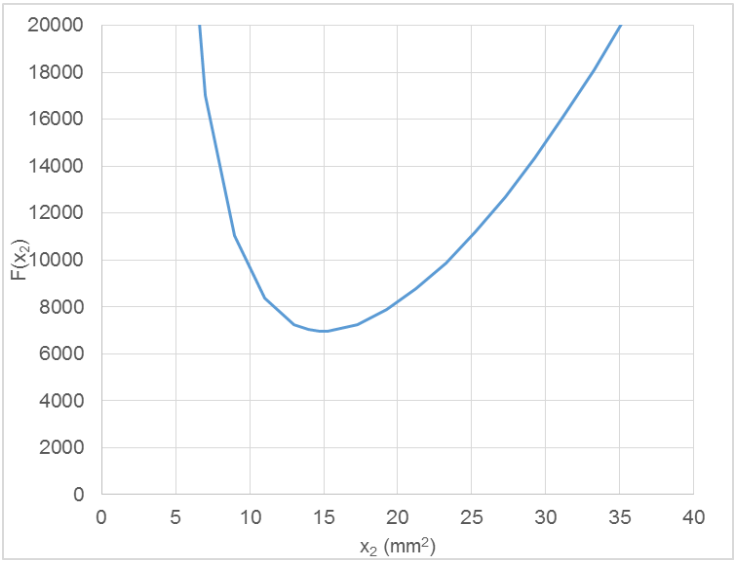


Figure 1.11: Design of beam cross-section for minimum Euclidean norm between the individual minima

Graphically, the optimum is found at $x_2^* = 15\,mm^2$ which gives that $w^* = h^* = 3.87\,mm$, with the objective function values of $f_1^* = 15\,mm^2$ and $f_2^* = 100\,MPa$.

Both of these designs are appropriate and optimal, but in different ways. Later on, when we look at Pareto Optimality, this will make more sense. Everything depends on how the optimization problem is formulated, which is what we are going to look at next.

1.3 Formulation of the Optimum Design Problem

To properly define and formulate an optimization problem, it takes approximately 50% of the total effort required to solve it. It is therefore necessary to follow a well-defined procedure for formulating the design optimization problems. Remember that the optimum solution will be only as good as the formulation. For example:

a. If a critical constraint is not included, then it will most probably be violated.
b. If there are too many constraints, or if they are inconsistent, then a solution may not be possible.

However, once the problem is properly formulated, good software is usually available to deal with it. For most design optimization problems, the formulation procedure requires the following 6 steps:

1. Project/problem description
2. Data and information collection
3. Definition of design variables
4. Optimization criterion
5. Formulation of constraints
6. Formulate the optimization problem

1.3.1 Project/Problem Description

The formulation process begins by developing a descriptive statement for the project/ problem, usually by the project's owner/sponsor. The statement describes the overall objectives of the project and the requirements to be met. This is also called the statement of work.

1.3.2 Data and Information Collection

To develop a mathematical formulation for the problem, it is necessary to obtain all available information on the: material properties, performance requirements, resource limits, cost of raw materials, etc. It is also necessary to determine how to analyse the designs. Therefore, the analysis procedures and tools must also be identified at this stage. For example: the finite-element analysis is commonly used for structural analysis, so the relevant software tool available needs to be identified. In many cases, the project statement is vague, and assumptions about modelling of the problem need to be made in order to formulate and solve it.

1.3.3 Definition of Design Variables

Identify a set of variables that describe the system, called the design variables. These are the optimization variables and are free so that any value can be assigned to them. The number of independent design variables gives the design degrees of freedom for the problem.

Design variables should be independent of each other as far as possible. If they are not, there must be some equality constraints between them. There must be a minimum number of design variables to properly formulate a design optimization problem. A numerical value should be given to each identified design variable to determine if a trial design of the system is specified.

1.3.4 Optimization Criterion

The optimization criterion is a scalar function which produces a numerical value once a design is specified; i.e. when the design variable vector x is substituted into it. This criterion is called the objective function for the optimum design problem, and it needs to be maximized or minimized depending on the problem. The selection of a proper objective function is an important decision in the design process. Some objective functions are: Cost (minimized); Profit (maximized), Weight (minimized), Energy expenditure (minimized), Vehicle ride quality (maximized).

1.3.5 Formulation of Constraints

All restrictions on the design are called constraints. It is necessary to identify all constraints and develop expressions for them. Most realistic systems must be designed and manufactured with the given resources and must meet performance requirements. For example:

a. Structural members should not fail under normal operating loads.
b. Structural vibration frequencies must be different from the operating frequency of the machine it supports; otherwise, resonance can occur and cause catastrophic failure.
c. Members must fit into the available space.
d. These constraints must depend on the design variables.
e. A meaningful constraint must be a function of at least one design variable.

1.3.6 Formulate the Optimization Problem

This is where everything from steps 2, 3, 4 and 5 are put together to formulate the optimization problem in the form of (1.4).

$$\begin{aligned} &\textbf{Minimize:} && f(x) \\ &\textbf{Subject to:} && g_i(x) \le 0; \quad i = 1 \; to \; m \\ & && h_j(x) = 0; \quad j = 1 \; to \; p \end{aligned} \tag{1.4}$$

Chapter 2

Graphical Method of Optimization

2.0 Introduction

Optimization problems having only two design variables can be solved by observing how they are graphically represented. All constraint functions are plotted, and a set of feasible designs (the feasible set) for the problem is identified. Objective function contours are then drawn, and the optimum design is determined by visual inspection. In this section, the graphical solution process will be introduced as well as several concepts related to optimum design problems. The method will be introduced using a simple example of profit maximisation followed by a further example.

2.1 Defining a Profit Maximization Example

Step 1: Project Description

A company manufactures two machines, A and B. Using available resources, either 28 A or 14 B can be manufactured daily. The sales department can sell up to 14 A machines or 24 B machines. The shipping facility can handle no more than 16 machines per day. The company makes a profit of £400 on each A machine and £600 on each B machine. How many A and B machines should the company manufacture every day to maximize its profit?

STEP 2: Data and information collection

Is all the information available to solve the problem? Data and information are defined in the project statement.

STEP 3: Definition of design variables

The following two design variables are identified in the problem statement:

x_1=Number of A machines made each day

x_2=Number of B machines made each day

STEP4: Optimization criterion:

The objective is to maximize daily profit, which can be expressed in terms of design variables as (2.1)

$$P = 400x_1 + 600x_2 \tag{2.1}$$

STEP 5*: *Formulation of constraints

Design constraints are placed on manufacturing capacity, on sales personnel and on shipping and handling facility. The constraint on the shipping and handling facility is quite straightforward and is given by (2.2).

Shipping and Handling Constraints

$$x_1 + x_2 \leq 16 \tag{2.2}$$

Constraints on manufacturing and sales facilities are a bit tricky. First, consider the manufacturing limitation. It is assumed that if the company is manufacturing x_1 A machines per day, then the remaining resources and equipment can be proportionately used to manufacture x_2 B machines, and vice versa. Therefore, noting that $x_1/28$ is the fraction of resources used to produce A and $x_2/14$ is the fraction used to produce B, the constraint is expressed as (2.3).

Manufacturing Constraint

$$\frac{x_1}{28} + \frac{x_2}{14} \leq 1 \tag{2.3}$$

Similarly, the constraint on sales department resources is given as (2.4).

Sales limitation

$$\frac{x_1}{14} + \frac{x_2}{24} \leq 1 \tag{2.4}$$

Finally, the design variables must be non-negative, given in (2.5).

$$x_1, x_2 \geq 0 \tag{2.5}$$

Note that for this problem, the formulation remains valid even when a design variable has zero value. The problem has two design variables and five inequality constraints. All functions of the problem are linear in variables x_1 and x_2. Therefore, it is a *linear programming* problem. Note also that for a meaningful solution, both design variables must have integer values at the optimum point.

2.2 Step by Step Graphical Solution Procedure

STEP 1: Coordinate system set-up

The first step in the solution process is to set up an origin for the x-y coordinate system and scales along the x- and y-axes. By looking at the constraint functions, a coordinate system for the profit maximization problem can be set up using a range of 0 to 25 along both the x and y axes, Figure 2.1 In some cases, the scale may need to be adjusted after the problem has been graphed because the original scale may provide too small or too large a graph for the problem.

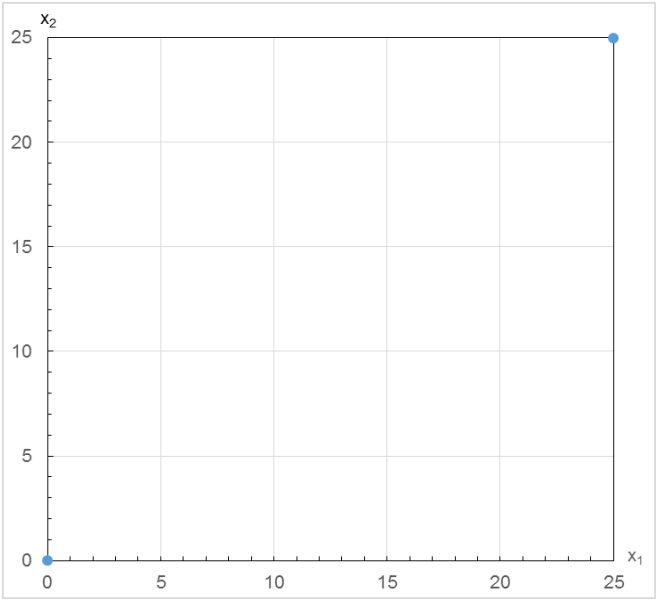


Figure 2.1: x-y coordinate system with the range of 0 to 25 along both the x and y axes.

STEP 2: Inequality constraint boundary plot

To illustrate the graphing of a constraint, let us consider the inequality $x_1 + x_2 \leq 16$ given in (2.2). To represent the constraint graphically, we first need to plot the constraint boundary; that is, the points that satisfy the constraint as an equality $x_1 + x_2 = 16$. This is a linear function of the variables x_1 and x_2. To plot such a function, we need two points that satisfy the equation $x_1 + x_2 = 16$. Let these points be calculated as (16, 0) and (0, 16). Locating them on the graph and joining them by a straight line produces the line F-J, as shown in Figure 2.2. Line F-J then represents the boundary of the feasible region for the inequality constraint $x_1 + x_2 \leq 16$. Points on one side of this line violate the constraint, while those on the other side satisfy it.

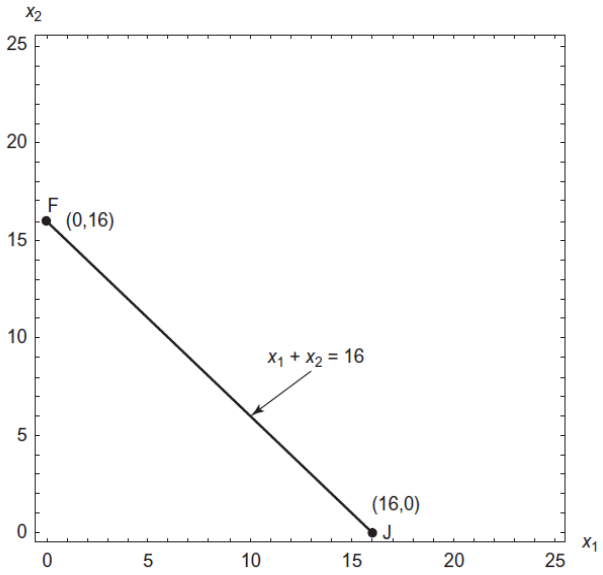


Figure 2.2: Constraint boundary for the inequality $x_1 + x_2 \leq 16$ in the profit maximization problem.

STEP 3: Identification of the feasible region for an inequality

The next task is to determine which side of constraint boundary F-J is feasible for the constraint $x_1 + x_2 \leq 16$. To accomplish this, we select a point on either side of F-J and evaluate the constraint function there. For example, at point (0,0), the left side of the constraint has a value of 0. Because the value is less than 16, the constraint is satisfied and the region below F-J is feasible. We can test the constraint at another point on the opposite side of F-J, say at point (10, 10). At this point the constraint is violated because the left side of the constraint function is 20, which is larger than 16. Therefore, the region above F-J is infeasible with respect to the constraint, as shown in Figure 2.3. The infeasible region is "*shaded-out,*" a convention that is used throughout this text. Note that if this were an equality constraint $x_1 + x_2 = 16$, the feasible region for it would only be the points on line F-J. Although there are infinite points on F-J, the feasible region for the equality constraint is much smaller than that for the same constraint written as an inequality. This shows the importance of properly formulating all the constraints of the problem.

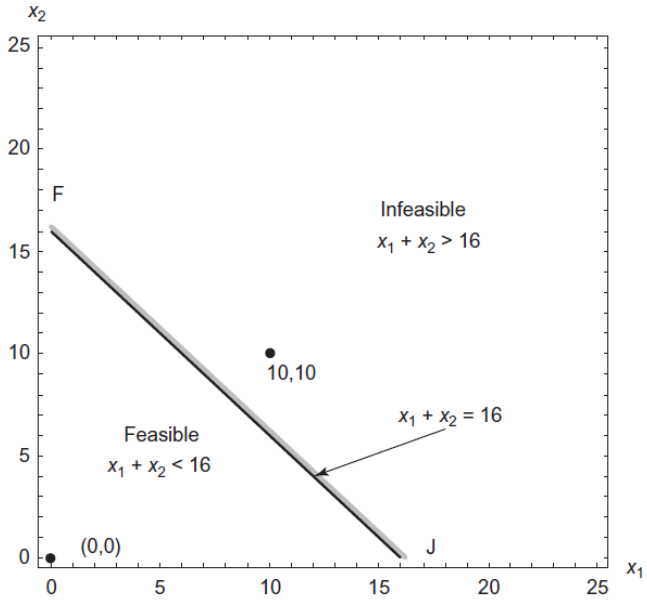


Figure 2.3: Feasible/infeasible side for the inequality $x_1 + x_2 \leq 16$ in the profit maximization problem.

STEP 4: Identification of the feasible region By following the procedure that is described in step 3, all inequalities are plotted on the graph and the feasible side of each one is identified (if equality constraints were present, they would also be plotted at this stage). Note that the constraints $x_1, x_2 \geq 0$ restrict the feasible region to the first quadrant of the coordinate system. The intersection of feasible regions for all constraints provides the feasible region for the profit maximization problem, indicated as ABCDE in Figure 2.4. Any point in this region or on its boundary provides a feasible solution to the problem.

STEP 5: Plotting of objective function contour

The next task is to plot the objective function on the graph and locate its optimum points. For the present problem, the objective is to maximize the profit $P = 400x_1 + 600x_2$, which involves three variables: P, x_1, and x_2. The function needs to be represented on the graph so

that the value of P can be compared for different feasible designs to locate the best design. However, because there are infinite feasible points, it is not possible to evaluate the objective function at every point. One way of overcoming this impasse is to plot the contours of the objective function.

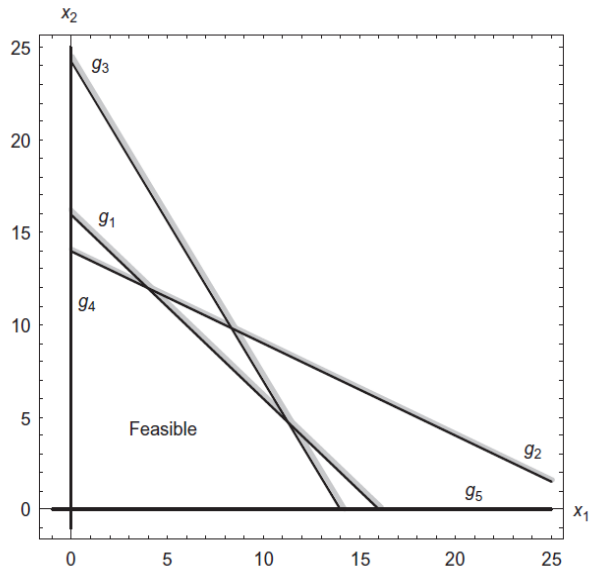


Figure 2.4: Feasible region for the profit maximization problem.

A contour is a curve on the graph that connects all points having the same objective function value. A collection of points on a contour is also called the level set. If the objective function is to be minimized, the contours are also called *isocost curves*. To plot a contour through the feasible region, we need to assign it a value. To obtain this value, consider a point in the feasible region and evaluate the profit function there. For example, at point (6,4), P is $P = 400 \times 6 + 600 \times 4 = 4800$. To plot the P=4800 contour, we plot the function $400x_1 + 600x_2 = 4800$ .This contour is a straight line, as shown in Figure 2.5.

STEP 6: Identification of the optimum solution

To locate an optimum point for the objective function, we need at least two contours that pass through the feasible region. We can then observe trends for the values of the objective function at different feasible points to locate the best solution point. Contours for P=2400, 4800, and 7200 are plotted in Figure 2.6. We now observe the following trend: As the contours move up toward point D, feasible designs can be found with larger values for P. It is clear from observation that point D has the largest value for P in the feasible region. We now simply read the coordinates of point D (4, 12) to obtain the optimum design, having a maximum value for the profit function as P=8800. Thus, the best strategy for the company is to manufacture 4 A and 12 B machines to maximize its daily profit. The inequality constraints in (2.2) and (2.3) are active at the optimum; that is, they are satisfied at equality. These represent limitations on shipping and handling facilities, and on manufacturing. The company can think about relaxing these constraints to improve its profit. All other inequalities are strictly satisfied and therefore inactive. Note that in this example the design variables must have integer values. Note also that for this example all functions are linear in design variables. Therefore, all curves in Figures 2.2 through 2.6 are straight lines. In general, the functions of a design problem may not be

linear, in which case curves must be plotted to identify the feasible region, and contours or isocost curves must be drawn to identify the optimum design. To plot a nonlinear function, a table of numerical values for x_1 and x_2 must be generated. These points must be then plotted on a graph and connected by a smooth curve.

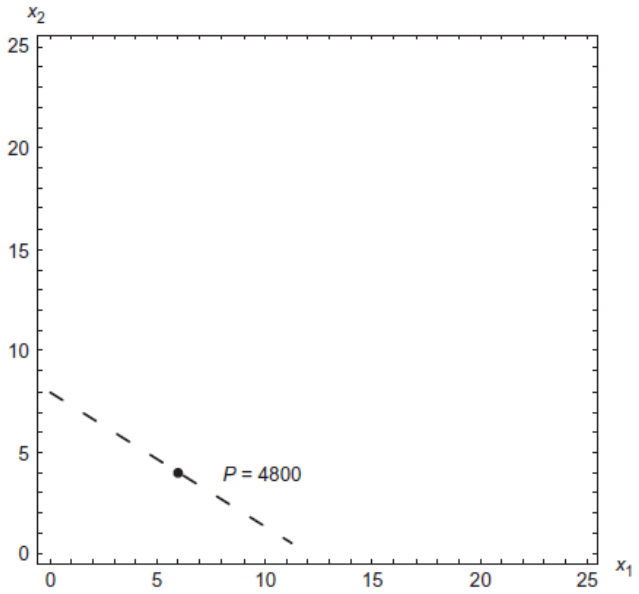


Figure 2.5: Plot of P=4800 objective function contour for the profit maximization problem.

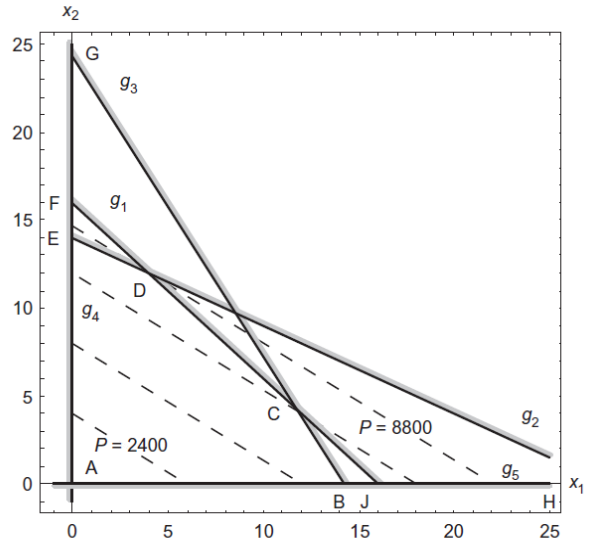


Figure 2.6: Graphical solution to the profit maximization problem: optimum point D = (4, 12); maximum profit, P = 8800.

2.3 Design Problem with Multiple Solutions

A situation can arise in which a constraint is parallel to the cost function. If the constraint is active at the optimum, there are multiple solutions to the problem. To illustrate this situation, consider the design problem of (2.6).

Minimize: $f(x) = -x_1 - 0.5x_2$ (2.6)

$$\begin{aligned} \textbf{Subject to:} \quad & 2x_1 + 3x_2 \leq 12 \\ & 2x_1 + x_2 \leq 8 \\ \textbf{where:} \quad & 0 \leq x_1; 0 \leq x_2 \end{aligned}$$

In this problem, the second constraint is parallel to the cost function. Therefore, there is a possibility of multiple optimum designs. Figure 2.7 provides a graphical solution to the problem. It is seen that any point on the line B-C gives an optimum design, giving the problem infinite optimum solutions.

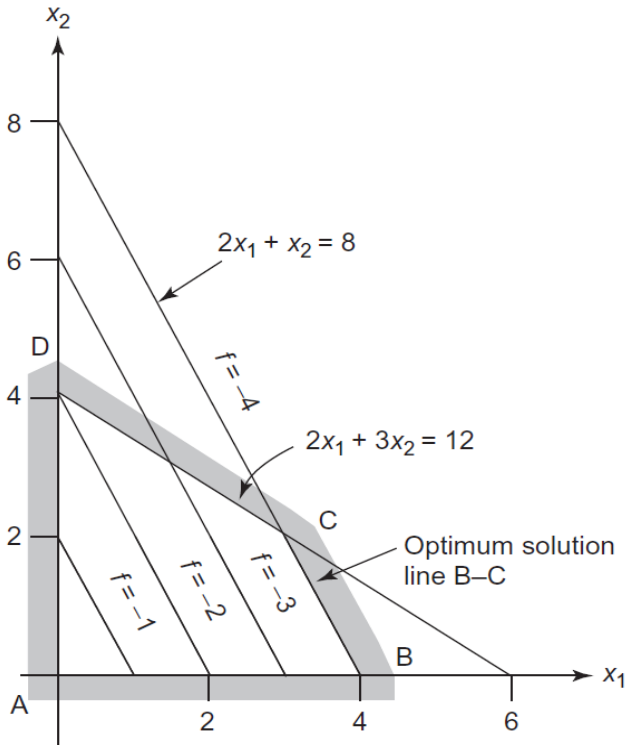


Figure 2.7: Example problem with multiple solutions

2.4 Design Problem with Unbounded Solutions

Some design problems may not have a bounded solution. This situation can arise if we forget a constraint or incorrectly formulate the problem. To illustrate such a situation, consider the design problem of (2.7).

$$\begin{aligned} \textbf{Minimize:} \quad & f(x) = -x_1 + 2x_2 \\ \textbf{Subject to:} \quad & -2x_1 + x_2 \leq 0 \\ & -2x_1 + 3x_2 \leq 6 \\ \textbf{where:} \quad & 0 \leq x_1; 0 \leq x_2 \end{aligned} \tag{2.7}$$

The feasible set for the problem is shown in Figure 2.8 with several cost function contours. It is seen that the feasible set is unbounded. Therefore, there is no finite optimum solution, and we must re-examine the way the problem was formulated to correct the situation. Figure 2.8 shows that the problem is underconstrained.

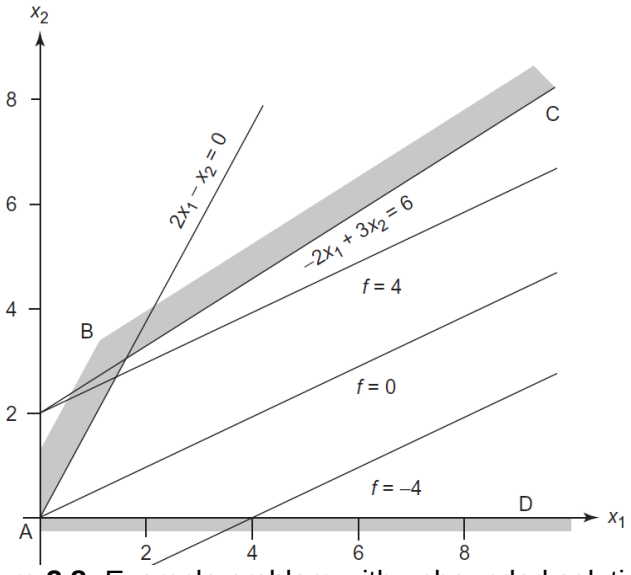


Figure 2.8: Example problem with unbounded solutions

2.5 Design Problem with Infeasible Solutions

If we are not careful in formulating it, a design problem may not have a solution, which happens when there are conflicting requirements or inconsistent constraint equations. There may also be no solution when we put too many constraints on the system; that is, the constraints are so restrictive that no feasible solution is possible. These are called infeasible problems. To illustrate them, consider the design problem of (2.8).

$$\begin{aligned} \textbf{Minimize:} \quad & f(x) = x_1 + 2x_2 \\ \textbf{Subject to:} \quad & 3x_1 + 2x_2 \le 6 \\ & 2x_1 + 3x_2 \ge 12 \\ \textbf{where:} \quad & x_1 \le 5;\ x_2 \le 5;\ x_1 \ge 0;\ x_2 \ge 0 \end{aligned} \tag{2.8}$$

Constraints for the problem are plotted in Figure 2.9 and their infeasible side is shaded out. It is evident that there is no region within the design space that satisfies all constraints; that is, there is no feasible region for the problem. Thus, the problem is infeasible. Basically, the first two constraints impose conflicting requirements. The first requires the feasible design to be below the line A-G, whereas the second requires it to be above the line C-F. Since the two lines do not intersect in the first quadrant, the problem has no feasible region.

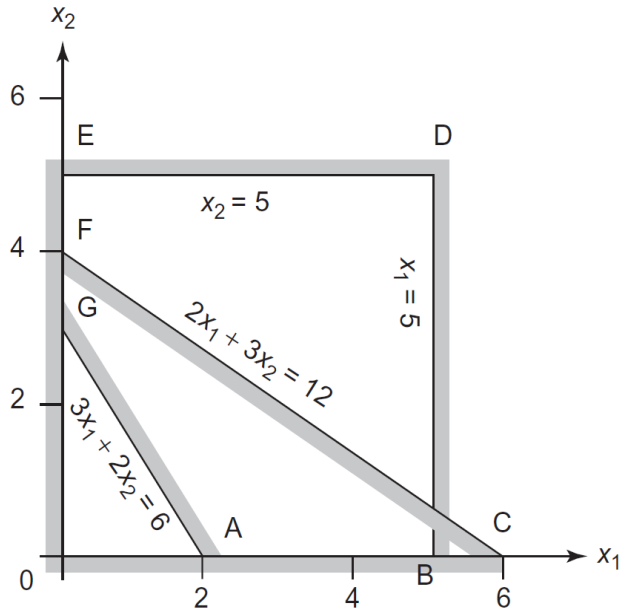


Figure 2.9: Example problem with infeasible solutions

2.6 Summary of Steps to Solve an Optimization Problem Graphically

The 6 steps outlined in section 2.2, related to solving the problem defined in section 2.1, which consisted of only inequality constraints. However, as optimization problems can have both equality and inequality constraints, the steps in solving a graphical optimization problem are best summarised as follow:

1) **Select a suitable coordinate system**, and choose an appropriate range for the optimization variables.
2) **Plot a contour for each equality and inequality constraint** function to define the boundaries of the design domain. If required, adjust the range of values to plot for the design variables.
3) **Identify the feasible region for each inequality constraint** function plotted.
4) **Identify the feasible design domain region.**
5) **Plot several contours of the objective function**, for different decreasing (if minimization) or increasing (if maximization) values of the objective function.
6) **Identity the optimal solution**. Read from the graph, or manipulate the constraint equations and the objective function to obtain the optimal solution.

2.7 Example

Solve the optimization problem of (2.9) graphically.

$$\begin{aligned} \textbf{Minimize:} \quad & f(x_1 x_2) = 4x_1^2 - 5x_1x_2 + x_2^2 \\ \textbf{Subject to:} \quad & g(x_1, x_2) = x_1^2 - x_2 + 2 \le 0 \\ & h(x_1, x_2) = x_1 + x_2 - 6 = 0 \end{aligned} \tag{2.9}$$

Following the 6 steps outlined in section 2.6, the solution to this problem is as follows:

1) Select a suitable coordinate system.

In looking at the objective function and constraints, all are a function of design variables x_1 and x_2. Consequently in solving this graphically, it is customary to place the design variable x_2 on the vertical axis, and the variable x_1 on the horizontal axis.

Note that: There are no limits placed on the design variables. Therefore, they could both be either positive or negative and have any value from -∞ to +∞. However, let's start by defining a plausible range for x_1, then substitute into the constraint as if they were all equality constraints to define a range for x_2.

Let's assume the range for x_1 to be: $-10 \le x_1 \le 10$. Now, rearrange the equality and inequality constraint equations, to calculate x_2 from the range of x_1.

$$g(x_1, x_2) = x_1^2 - x_2 + 2 \le 0 \Rightarrow x_1^2 - x_2 + 2 = 0$$

$$\therefore x_2 = x_1^2 + 2 \qquad \text{(a)}$$

$$h(x_1, x_2) = x_1 + x_2 - 6 = 0$$

$$\therefore x_2 = 6 - x_1 \qquad \text{(b)}$$

Now substitute the limits of x_1, into (a) and (b) to solve for the limits of x_2.

At $x_1 = -10$

$$x_2 = x_1^2 + 2;$$
$$x_2 = -10^2 + 2 = 102$$
$$x_2 = 6 - x_1;$$
$$x_2 = 6 - -10 = 16$$

At $x_1 = 10$

$$x_2 = x_1^2 + 2;$$
$$x_2 = 10^2 + 2 = 102$$
$$x_2 = 6 - x_1;$$
$$x_2 = 6 - 10 = -4$$

Which then means that the range of values for x_2 is: $-4 \le x_2 \le 102$

And the design domain area is then that given by Figure 2.10, where these limits have been rounded to the nearest 10, so from -10 up to 110.

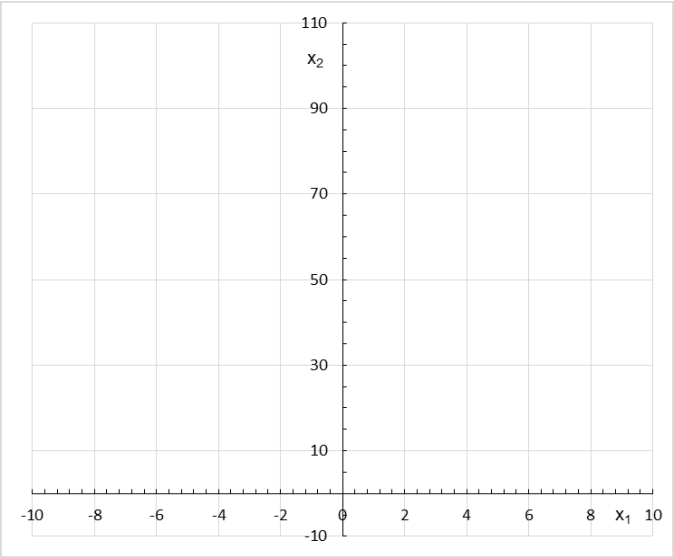


Figure 2.10: Coordinate system for problem (2.9).

2) Plot a contour for each equality and inequality constraint function

With the rearranged equations (a) and (b), now calculate values of x_2 in the range for x_1 of $-10 \le x_1 \le 10$.

Table 2.1: Calculated values of x_2 from range of x_1

	g	h
x_1	**x_2 (g)**	**x_2 (h)**
-10	102.0	16
-9	83.0	15
-8	66.0	14
-7	51.0	13
-6	38.0	12
-5	27.0	11
-4	18.0	10
-3	11.0	9
-2	6.0	8
-1	3.0	7
0	2.0	6
1	3.0	5
2	6.0	4
3	11.0	3
4	18.0	2
5	27.0	1
6	38.0	0
7	51.0	-1
8	66.0	-2
9	83.0	-3
10	102.0	-4

The values from table 2.1 are then plotted in Figure 2.11 to show the contours of the equality and inequality constraint. In looking at this figure, since the solution to the optimization

problem must lie on the equality constraint, it is easy to see, that the range of values for both design variables needs to be changed.

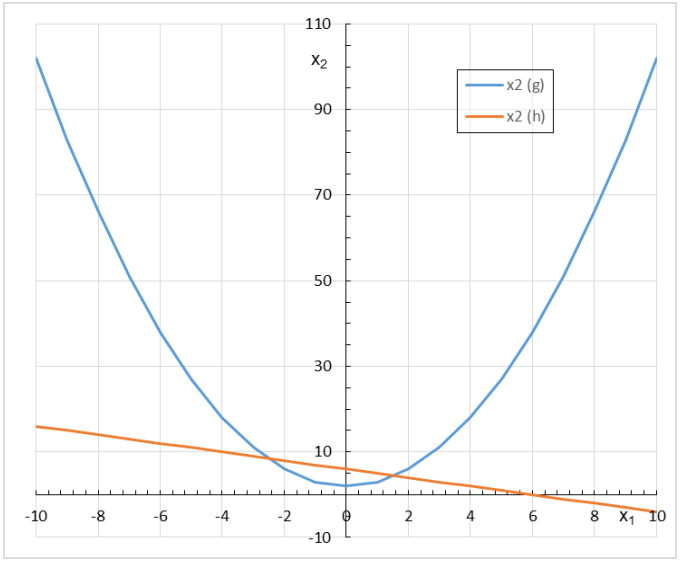


Figure 2.11: Coordinate system for problem (2.9).

The range to plot is therefore now changed to be $-3 \le x_1 \le 2$, and $0 \le x_2 \le 10$.

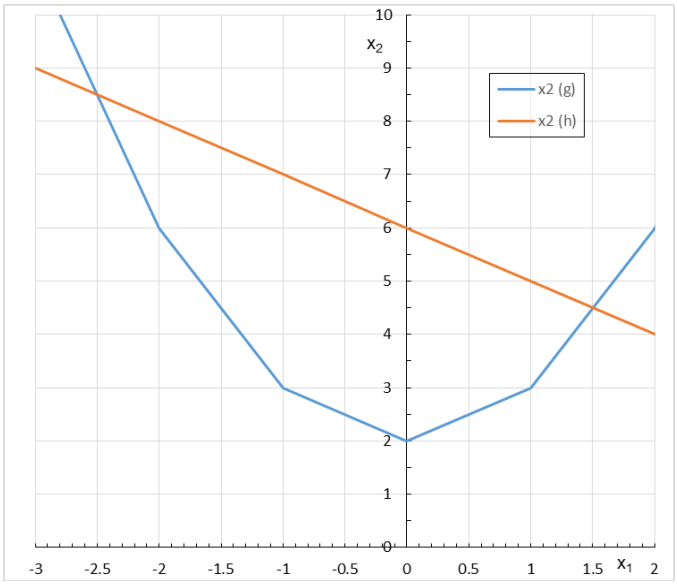


Figure 2.12: Coordinate system for problem (2.9) with new range for the design variables.

In looking at Figure 2.12, the plot of the constraint equations is not very smooth, so more points are required. Instead of the gap between each point set at 1, a new set of values was calculated with a gap between values of 0.1, to generate the smoother plot of Figure 2.13.

3) Identify the feasible region for each inequality constraint.

Now that the equality and inequality constraints have been plotted, we can identify the feasible region of the inequality constraint. To do this, we need to rearrange the inequality constraint to determine which values of x_2 are in the feasible domain.

$$g(x_1, x_2) = x_1^2 - x_2 + 2 \leq 0$$
$$\Rightarrow x_1^2 + 2 \leq x_2$$
$$\therefore x_2 \geq x_1^2 + 2 \qquad \text{(c)}$$

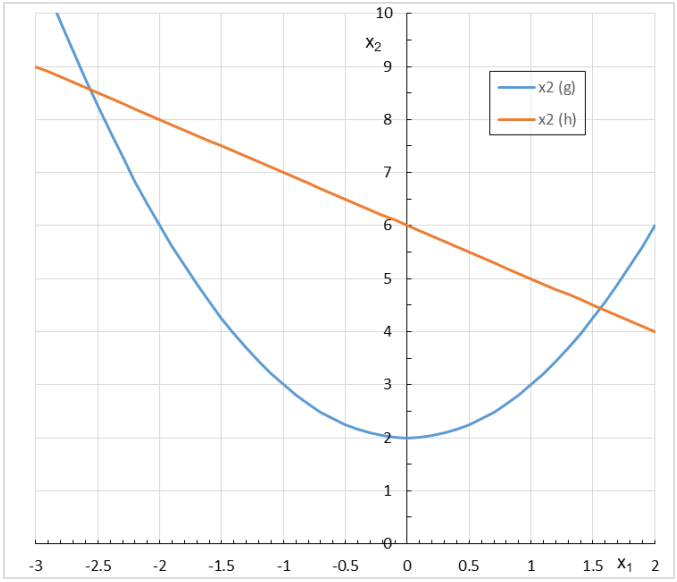


Figure 2.13: Coordinate system for problem (2.9) with smoother plot than Figure 2.12.

From equation (c) we can see that all values of x_2, greater than or equal to the boundary for the plot of x_2 in Figure 2.13, corresponds with the feasible domain for this inequality constraint. This is now shown in Figure 2.14 with a shaded area for the infeasible domain.

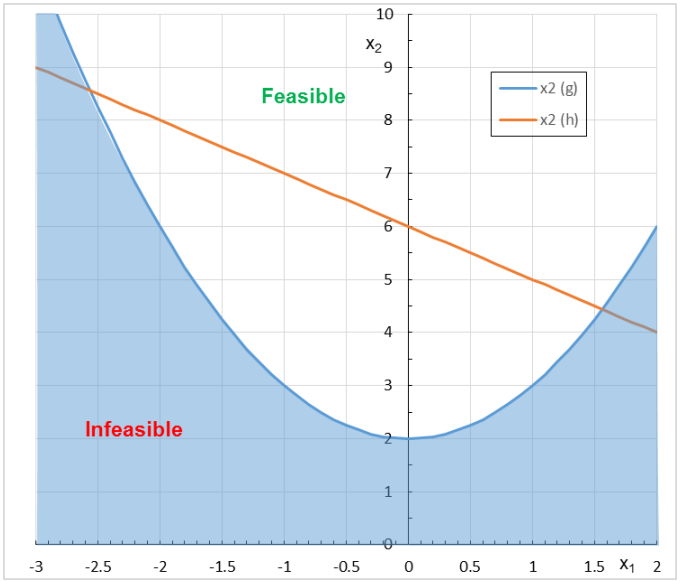


Figure 2.14: Plot of constraint equations showing the feasible region for the inequality constraint equation.

4) Identify the feasible design domain region.

Since the feasible region is above the plot of equation (a), and since the problem has an equality constraint, then the feasible design domain is that part of the plot of equation (b) within the feasible region. Figure 2.15, shows the feasible resign region.

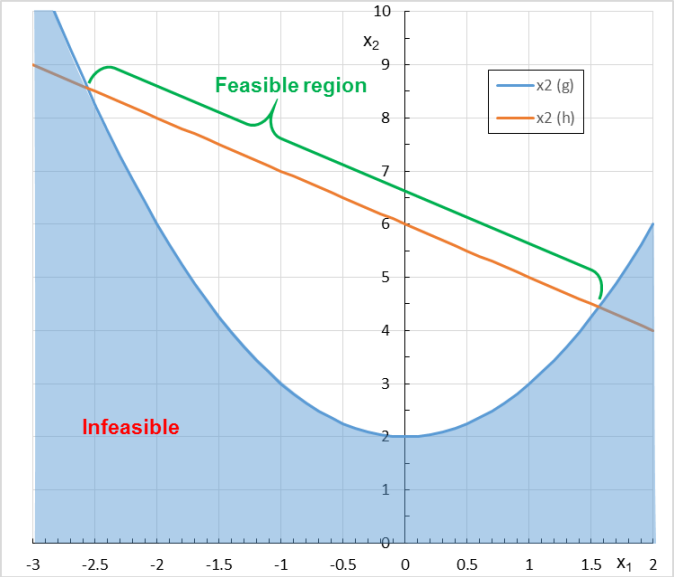


Figure 2.15: Plot of constraint equations showing the feasible region for the inequality constraint equation.

5) Plot several contours of the objective function

In order to plot contours of the objective function, it needs to be manipulated so that we can calculate values of x_2, as a function of values of x_1. So, rearranging the objective function gives:

$$f\left(x_1x_2\right) = 4x_1^2 - 5x_1x_2 + x_2^2$$
$$4x_1^2 - 5x_1x_2 + x_2^2 - f = 0$$
$$x_2^2 - 5x_1x_2 + 4x_1^2 - f = 0$$
$$\therefore x_2^2 - \left(5x_1\right)x_2 + \left(4x_1^2 - f\right) = 0 \qquad \text{(d)}$$

Equation (d) is in the form of a quadratic equation of the form of equation (e).

$$ax_2^2 + bx_2 + c = 0 \qquad \text{(e)}$$

where:

$$a = 1,\ b = -5x_1,\ c = 4x_1^2 - f \qquad \text{(f)}$$

and where the solution to equation (e) is found using equation (g)

$$x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad \text{(g)}$$

Now substituting (f) into (g) gives equation (h), which will allow for the calculation of x_2, at different values of x_1, for different objective function values of ***f***. Thus allowing for the generation of contour plots for different values of ***f***.

$$x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-(-5x_1) \pm \sqrt{(-5x_1)^2 - 4 \times 1 \times (4x_1^2 - f)}}{2 \times 1}$$

$$\therefore x_2 = \frac{5x_1 \pm \sqrt{25x_1^2 + 4f - 16x_1^2}}{2}$$

$$\therefore x_2 = \frac{5x_1 \pm \sqrt{9x_1^2 + 4f}}{2} \qquad \text{(h)}$$

When equation (h) is plotted, for different values of ***f***, the plot of Figure 2.17 is generated. To begin with, different values of ***f*** were tried until the plots of the contours started to appear within the feasible domain. Then only plots for values of f = 100, 50, 10, 0 and -5 were plotted.

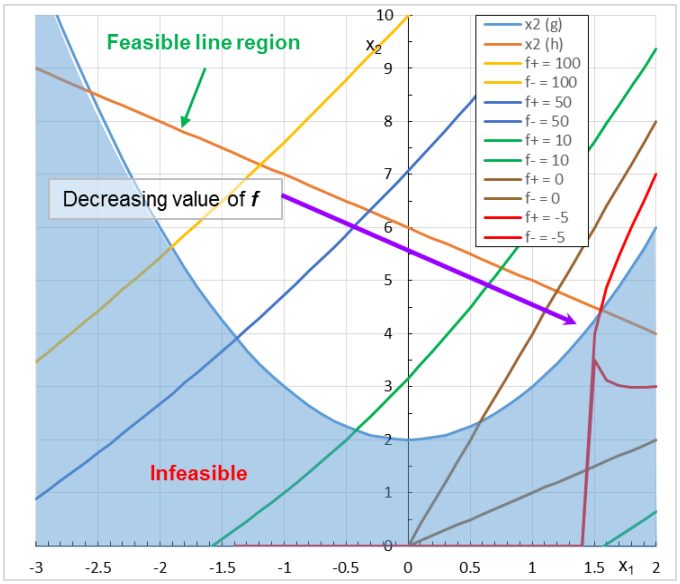


Figure 2.17: Contour plots of the objective function using equation (h).

As can clearly be seen from Figure 2.17, the contour of the objective function decrease in value as they get closer to the intersection of both constraints on the right hand quadrant of plot. Such that the contour for f = –5 is the lowest value which is within the feasible line region.

6) Identity the optimal solution

From Figure 2.17, we can see that the optimum value is at the intersection of both constraints on the right hand quadrant of plot. If both constraint equations (a) and (b) are now solved, we can find the values of x_1, x_2 and ***f***, which will be the optimum value for this problem.

As (a) and (b) are both equal to x_2, both equations are therefore equal to each other, which gives:

$$x_2 = x_1^2 + 2 = 6 - x_1$$
$$x_1^2 + x_1 + 2 - 6 = 0$$
$$\therefore x_1^2 + x_1 - 4 = 0 \qquad \text{(i)}$$

Solving the quadratic equation (i) gives:

$$x_1^2 + x_1 - 4 = 0$$
$$x_1 = \frac{-1 \pm \sqrt{1^2 - 4 \times 1 \times (-4)}}{2 \times 1} = \frac{-1 \pm \sqrt{1+16}}{2}$$
$$\therefore x_1 = \frac{-1 \pm \sqrt{17}}{2} = \frac{-1 \pm 4.1231}{2} = -2.5616 \text{ or } 1.5616 \qquad \text{(i)}$$

And from Figure 2.17, since x_1 has to be positive, then the value of x_1 is 1.5616, substituting into (b), and then into the objective function gives that the optimal values for the design variables and the optimum value for the objective function are:

$$x_1 = 1.5616$$
$$x_2 = 4.4384$$
$$f = -5.20125$$

These are then plotted onto Figure 2.18 to show the optimal solution to this problem

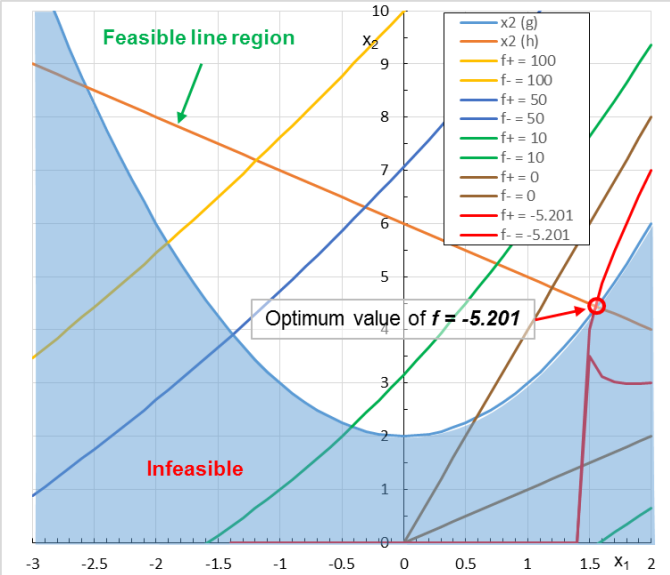


Figure 2.18: Contour plots of the objective function showing the optimal solution

Chapter 3

Unconstrained Optimization Methods

3.0 Introduction

This chapter looks at some methods for solving unconstrained nonlinear optimization problems.

3.1 The Bisector Method

In the interval halving method, exactly one-half of the current interval of uncertainty is deleted in every stage. It requires three experiments in the first stage and two experiments in each subsequent stage. The procedure can be described by the following four steps:

1. Divide the initial interval $[a,b]$ of uncertainty $L_0 = b - a$ into four equal parts $\Delta x = (b-a)/4$ and label the middle point $x_0 = a + 2\Delta x$ and the quarter-interval points $x_1 = a + \Delta x$ and $x_2 = a + 3\Delta x$.
2. Evaluate the function f (x) at the three interior points to obtain $f_1 = f(x_1)$, $f_0 = f(x_0)$, and $f_2 = f(x_2)$.
3. There are then 3 cases to consider corresponding with Figure 3.1, these are:
   a) If $f_2 > f_0 > f_1$ as shown in Figure 3.1(a), delete the interval (x_0, b), label x_1 and x_0 as the new x_0 and b, respectively, and go to step 4.
   b) If $f_2 < f_0 < f_1$ as shown in Figure 3.1(b), delete the interval (a, x_0), label x_2 and x_0 as the new x_0 and a, respectively, and go to step 4.
   c) If $f_1 > f_0$ and $f_2 > f_0$ as shown in Figure 3.1(c), delete both the intervals (a, x_1) and (x_2, b), label x_1 and x_2 as the new ***a*** and ***b***, respectively, and go to step 4.
4. Test whether the new interval of uncertainty, $L = b - a$, satisfies the convergence criterion L ≤ ε, where ε is a small quantity. If the convergence criterion is satisfied, stop the procedure. Otherwise, set the new $L_0 = L$ and go to step 1.

Remarks:

1. In this method, the function value at the middle point of the interval of uncertainty, f_0, will be available in all the stages except the first stage.
2. The interval of uncertainty remaining at the end of n experiments (n ≥ 3 and odd) is given by (3.1).

$$L_n = \left(\frac{1}{2}\right)^{\frac{(n-1)}{2}} \times L_0 \qquad (3.1)$$

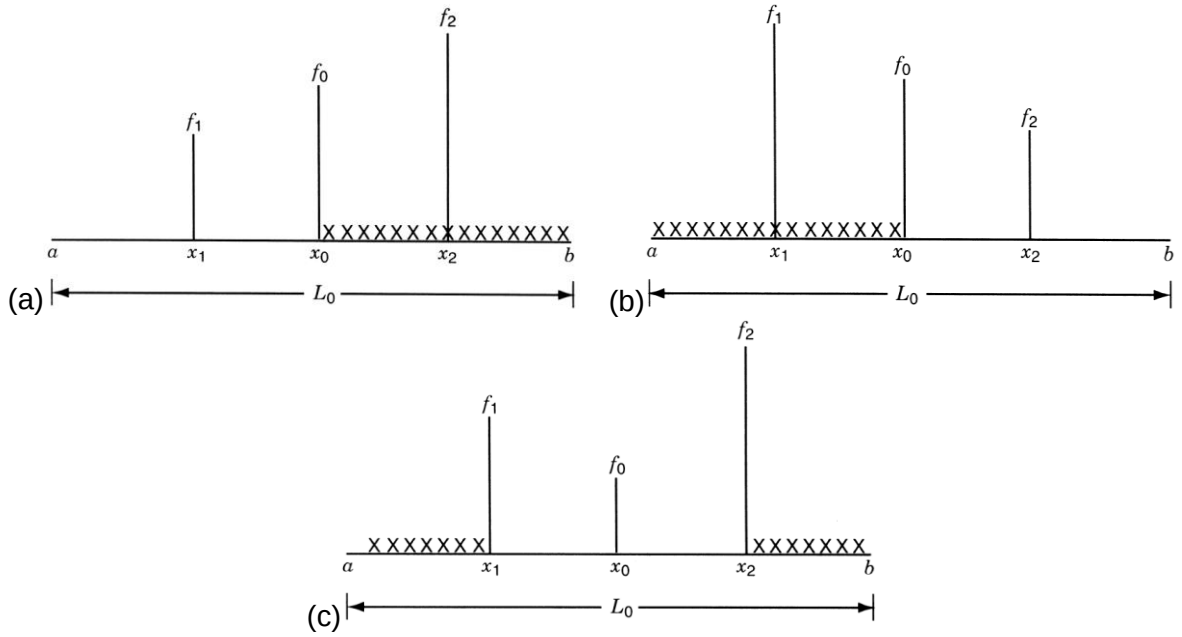


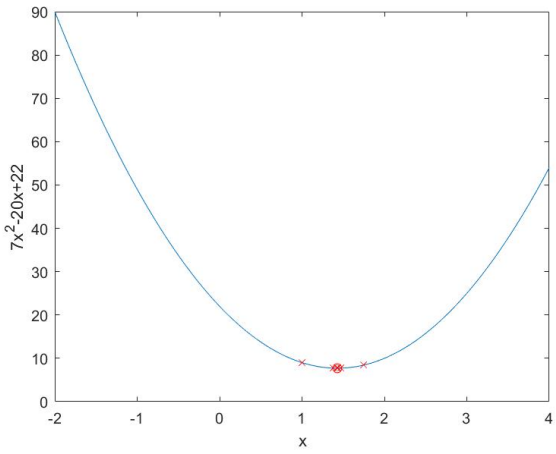
Figure 3.1: Possibilities in the interval halving method: (a) $f_2 > f_0 > f_1$; (b) $f_2 < f_0 < f_1$; (c) $f_1 > f_0$ and $f_2 > f_0$.

Bisector Example Problem 1: Problem 45 in Worked Examples

Minimize $f(x) = 7x^2 - 20x + 22$ for $-2 \leq x \leq 4$ with Bisector Search Parameters: xmin=-2, xmax=4, convergence tolerance = 0.01. Calculations:

iteration 1 of bisector search, domain length = 6.00000
a=-2.000 f(a)=90.000
x1=-0.500 f(x1)=33.750
x0= 1.000 f(x0)= 9.000
x2= 2.500 f(x2)=15.750
b= 4.000 f(b)=54.000
iteration 2 of bisector search, domain length = 3.00000
a=-0.500 f(a)=33.750
x1= 0.250 f(x1)=17.438
x0= 1.000 f(x0)= 9.000
x2= 1.750 f(x2)= 8.438
b= 2.500 f(b)=15.750
iteration 3 of bisector search, domain length = 1.50000
a= 1.000 f(a)= 9.000
x1= 1.375 f(x1)= 7.734
x0= 1.750 f(x0)= 8.438
x2= 2.125 f(x2)=11.109
b= 2.500 f(b)=15.750
iteration 4 of bisector search, domain length = 0.75000
a= 1.000 f(a)= 9.000
x1= 1.188 f(x1)= 8.121
x0= 1.375 f(x0)= 7.734
x2= 1.563 f(x2)= 7.840
b= 1.750 f(b)= 8.438
iteration 5 of bisector search, domain length = 0.37500
a= 1.188 f(a)= 8.121
x1= 1.281 f(x1)= 7.866
x0= 1.375 f(x0)= 7.734
x2= 1.469 f(x2)= 7.726
b= 1.563 f(b)= 7.840
iteration 6 of bisector search, domain length = 0.18750
a= 1.375 f(a)= 7.734
x1= 1.422 f(x1)= 7.715
x0= 1.469 f(x0)= 7.726

x2= 1.516 f(x2)= 7.767
b= 1.563 f(b)= 7.840
iteration 7 of bisector search, domain length = 0.09375
a= 1.375 f(a)= 7.734
x1= 1.398 f(x1)= 7.721
x0= 1.422 f(x0)= 7.715
x2= 1.445 f(x2)= 7.716
b= 1.469 f(b)= 7.726
iteration 8 of bisector search, domain length = 0.04688
a= 1.398 f(a)= 7.721
x1= 1.410 f(x1)= 7.717
x0= 1.422 f(x0)= 7.715
x2= 1.434 f(x2)= 7.714
b= 1.445 f(b)= 7.716
iteration 9 of bisector search, domain length = 0.02344
a= 1.422 f(a)= 7.715
x1= 1.428 f(x1)= 7.714
x0= 1.434 f(x0)= 7.714
x2= 1.439 f(x2)= 7.715
b= 1.445 f(b)= 7.716
iteration 10 of bisector search, domain length = 0.01172
a= 1.422 f(a)= 7.715
x1= 1.425 f(x1)= 7.714
x0= 1.428 f(x0)= 7.714
x2= 1.431 f(x2)= 7.714
b= 1.434 f(b)= 7.714
search completed after 10 iterations xmin= 1.42773 fmin= 7.71429

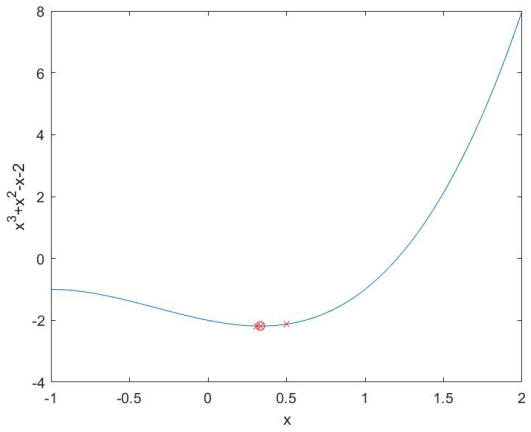


Bisector Example Problem 2: Problem 46 in Worked Examples

Minimize $f(x) = x^3 + x^2 - x - 2$ for $-1 \leq x \leq 2$ with Bisector Search Parameters: xmin=-1, xmax=2, convergence tolerance = 0.01. Calculations:

iteration 1 of bisector search, domain length = 3.00000
a=-1.000 f(a)=-1.000
x1=-0.250 f(x1)=-1.703
x0= 0.500 f(x0)=-2.125
x2= 1.250 f(x2)= 0.266
b= 2.000 f(b)= 8.000
iteration 2 of bisector search, domain length = 1.50000
a=-0.250 f(a)=-1.703
x1= 0.125 f(x1)=-2.107
x0= 0.500 f(x0)=-2.125
x2= 0.875 f(x2)=-1.439
b= 1.250 f(b)= 0.266
iteration 3 of bisector search, domain length = 0.75000

```
a= 0.125 f(a)=-2.107
x1= 0.313 f(x1)=-2.184
x0= 0.500 f(x0)=-2.125
x2= 0.688 f(x2)=-1.890
b= 0.875 f(b)=-1.439
iteration     4 of bisector search, domain length =    0.37500
a= 0.125 f(a)=-2.107
x1= 0.219 f(x1)=-2.160
x0= 0.313 f(x0)=-2.184
x2= 0.406 f(x2)=-2.174
b= 0.500 f(b)=-2.125
iteration     5 of bisector search, domain length =    0.18750
a= 0.219 f(a)=-2.160
x1= 0.266 f(x1)=-2.176
x0= 0.313 f(x0)=-2.184
x2= 0.359 f(x2)=-2.184
b= 0.406 f(b)=-2.174
iteration     6 of bisector search, domain length =    0.09375
a= 0.266 f(a)=-2.176
x1= 0.289 f(x1)=-2.181
x0= 0.313 f(x0)=-2.184
x2= 0.336 f(x2)=-2.185
b= 0.359 f(b)=-2.184
iteration     7 of bisector search, domain length =    0.04688
a= 0.313 f(a)=-2.184
x1= 0.324 f(x1)=-2.185
x0= 0.336 f(x0)=-2.185
x2= 0.348 f(x2)=-2.185
b= 0.359 f(b)=-2.184
iteration     8 of bisector search, domain length =    0.02344
a= 0.324 f(a)=-2.185
x1= 0.330 f(x1)=-2.185
x0= 0.336 f(x0)=-2.185
x2= 0.342 f(x2)=-2.185
b= 0.348 f(b)=-2.185
iteration     9 of bisector search, domain length =    0.01172
a= 0.330 f(a)=-2.185
x1= 0.333 f(x1)=-2.185
x0= 0.336 f(x0)=-2.185
x2= 0.339 f(x2)=-2.185
b= 0.342 f(b)=-2.185
search completed after     9 iterations xmin=   0.33301 fmin=  -2.18518
```



3.2 The Golden Section Method

The Golden Section Method is one of the better methods in the class of interval-reducing methods. The basic idea of the method is as follows: Evaluate the function at predetermined points, compare them to find the minimum. Then converge on the minimum point by systematically reducing the interval of uncertainty. The method uses fewer function evaluations to reach the minimum point compared with other similar methods.

The different steps of the Golden Search Algorithm used for the minimization of a single value function in the interval $[a,b]$ are as follow:

1) If this is the first iteration define the search interval $[a_0,b_0]$ to be the specified interval in which to search for the minimum, given by (3.2).

$$[a_0 = a, b_0 = b] \tag{3.2}$$

2) Calculate two intermediate points $[a_1,b_1]$ in the interval $[a_0,b_0]$, using (5.3) and (5.4):

$$a_1 = a_0 + \rho(b_0 - a_0) \tag{3.3}$$

$$b_1 = b_0 - \rho(b_0 - a_0) \tag{3.4}$$

$$where: \rho = \frac{3-\sqrt{5}}{2} = 0.38197$$

3) Evaluate the function at the two intermediate points $[a_1,b_1]$ to obtain $f(a_1)\ and\ f(b_1)$:

4) Determine the new search limit by the following comparisons:

a) If (3.5) is satisfied, the minimum point lies between a_0 and b_1, so the new search interval is then given by becomes (3.6). Go to step 5.

$$f(a_1) < f(b_1) \tag{3.5}$$

$$[a_0 = a_0, b_0 = b_1] \tag{3.6}$$

b) If (3.7) is satisfied, the minimum point lies between a_1 and b_0, then the new search interval becomes (5.8). Go to step 5.

$$f(a_1) > f(b_1) \tag{3.7}$$

$$[a_0 = a_1, b_0 = b_0] \tag{3.8}$$

c) If (3.9) is satisfied, the minimum point lies between a_1 and b_1, then the new search interval becomes (3.10). Go to step 5

$$f(a_1) = f(b_1) \quad (3.9)$$

$$[a_0 = a_1, b_0 = b_1] \quad (3.10)$$

5) Calculate the Interval of Uncertainty *(I)*, using (3.11).

$$I = b_0 - a_0 \quad (3.11)$$

6) Check limit to stop algorithm. If *(I)* is equal to, or smaller than the specified minimum range (3.12), using (3.13) calculate the optimal functional value and stop, else go to step 2.

$$I \leq Range \quad (3.12)$$

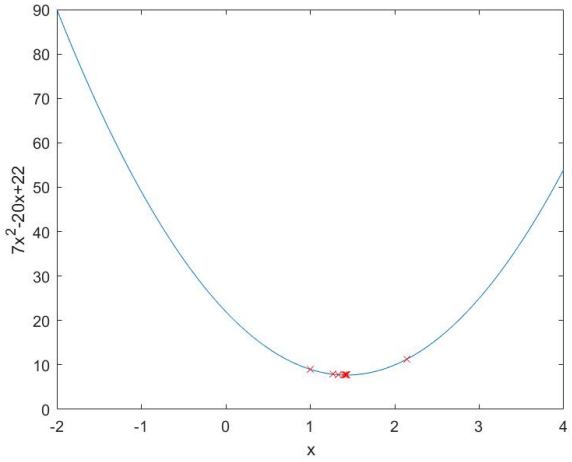
$$x_{Optimum} = \frac{a_0 + b_0}{2} \quad (3.13)$$

Golden Search Example Problem 1: Problem 45 in Worked Examples

Minimize $f(x) = 7x^2 - 20x + 22$ for $-2 \leq x \leq 4$ with Golden Search Parameters: xmin=-2, xmax=4, tol=0.01. Calculations:

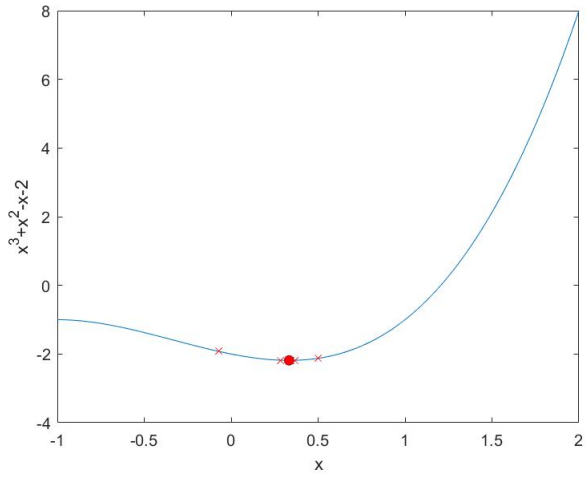
iteration 1 of golden section, domain length = 6.00000
a0=-2.000 f(a0)=90.000
a1= 0.292 f(a1)=16.760
b1= 1.708 f(b1)= 8.262
b0= 4.000 f(bo)=54.000
iteration 2 of golden section, domain length = 3.70820
a0= 0.292 f(a0)=16.760
a1= 1.708 f(a1)= 8.262
b1= 2.584 f(b1)=17.053
b0= 4.000 f(bo)=54.000
iteration 3 of golden section, domain length = 2.29180
a0= 0.292 f(a0)=16.760
a1= 1.167 f(a1)= 8.193
b1= 1.708 f(b1)= 8.262
b0= 2.584 f(bo)=17.053
iteration 4 of golden section, domain length = 1.41641
a0= 0.292 f(a0)=16.760
a1= 0.833 f(a1)=10.199
b1= 1.167 f(b1)= 8.193
b0= 1.708 f(bo)= 8.262
iteration 5 of golden section, domain length = 0.87539
a0= 0.833 f(a0)=10.199
a1= 1.167 f(a1)= 8.193
b1= 1.374 f(b1)= 7.735
b0= 1.708 f(bo)= 8.262
iteration 6 of golden section, domain length = 0.54102
a0= 1.167 f(a0)= 8.193
a1= 1.374 f(a1)= 7.735
b1= 1.502 f(b1)= 7.752
b0= 1.708 f(bo)= 8.262
iteration 7 of golden section, domain length = 0.33437
a0= 1.167 f(a0)= 8.193

a1= 1.295 f(a1)= 7.839
b1= 1.374 f(b1)= 7.735
b0= 1.502 f(bo)= 7.752
iteration 8 of golden section, domain length = 0.20665
a0= 1.295 f(a0)= 7.839
a1= 1.374 f(a1)= 7.735
b1= 1.423 f(b1)= 7.715
b0= 1.502 f(bo)= 7.752
iteration 9 of golden section, domain length = 0.12772
a0= 1.374 f(a0)= 7.735
a1= 1.423 f(a1)= 7.715
b1= 1.453 f(b1)= 7.718
b0= 1.502 f(bo)= 7.752
iteration 10 of golden section, domain length = 0.07893
a0= 1.374 f(a0)= 7.735
a1= 1.404 f(a1)= 7.719
b1= 1.423 f(b1)= 7.715
b0= 1.453 f(bo)= 7.718
iteration 11 of golden section, domain length = 0.04878
a0= 1.404 f(a0)= 7.719
a1= 1.423 f(a1)= 7.715
b1= 1.434 f(b1)= 7.715
b0= 1.453 f(bo)= 7.718
iteration 12 of golden section, domain length = 0.03015
a0= 1.423 f(a0)= 7.715
a1= 1.434 f(a1)= 7.715
b1= 1.441 f(b1)= 7.715
b0= 1.453 f(bo)= 7.718
iteration 13 of golden section, domain length = 0.01863
a0= 1.423 f(a0)= 7.715
a1= 1.430 f(a1)= 7.714
b1= 1.434 f(b1)= 7.715
b0= 1.441 f(bo)= 7.715
iteration 14 of golden section, domain length = 0.01152
a0= 1.423 f(a0)= 7.715
a1= 1.427 f(a1)= 7.714
b1= 1.430 f(b1)= 7.714
b0= 1.434 f(bo)= 7.715
Golden section completed after 14 iterations xmin= 1.43058 fmin= 7.71431



Golden Search Example Problem 2: Problem 46 in Worked Examples
Minimize $f(x) = x^3 + x^2 - x - 2$ for $-1 \leq x \leq 2$ with Golden Search Parameters: xmin=-1, xmax=2, convergence tolerance = 0.01. Calculations:
iteration 1 of golden section, domain length = 3.00000
a0=-1.000 f(a0)=-1.000
a1= 0.146 f(a1)=-2.122
b1= 0.854 f(b1)=-1.502
b0= 2.000 f(bo)= 8.000
iteration 2 of golden section, domain length = 1.85410
a0=-1.000 f(a0)=-1.000
a1=-0.292 f(a1)=-1.648
b1= 0.146 f(b1)=-2.122
b0= 0.854 f(bo)=-1.502
iteration 3 of golden section, domain length = 1.14590
a0=-0.292 f(a0)=-1.648
a1= 0.146 f(a1)=-2.122
b1= 0.416 f(b1)=-2.171
b0= 0.854 f(bo)=-1.502
iteration 4 of golden section, domain length = 0.70820
a0= 0.146 f(a0)=-2.122
a1= 0.416 f(a1)=-2.171
b1= 0.584 f(b1)=-2.044
b0= 0.854 f(bo)=-1.502
iteration 5 of golden section, domain length = 0.43769
a0= 0.146 f(a0)=-2.122
a1= 0.313 f(a1)=-2.184
b1= 0.416 f(b1)=-2.171
b0= 0.584 f(bo)=-2.044
iteration 6 of golden section, domain length = 0.27051
a0= 0.146 f(a0)=-2.122
a1= 0.249 f(a1)=-2.172
b1= 0.313 f(b1)=-2.184
b0= 0.416 f(bo)=-2.171
iteration 7 of golden section, domain length = 0.16718
a0= 0.249 f(a0)=-2.172
a1= 0.313 f(a1)=-2.184
b1= 0.353 f(b1)=-2.184

b0= 0.416 f(bo)=-2.171
iteration 8 of golden section, domain length = 0.10333
a0= 0.313 f(a0)=-2.184
a1= 0.353 f(a1)=-2.184
b1= 0.377 f(b1)=-2.181
b0= 0.416 f(bo)=-2.171
iteration 9 of golden section, domain length = 0.06386
a0= 0.313 f(a0)=-2.184
a1= 0.337 f(a1)=-2.185
b1= 0.353 f(b1)=-2.184
b0= 0.377 f(bo)=-2.181
iteration 10 of golden section, domain length = 0.03947
a0= 0.313 f(a0)=-2.184
a1= 0.328 f(a1)=-2.185
b1= 0.337 f(b1)=-2.185
b0= 0.353 f(bo)=-2.184
iteration 11 of golden section, domain length = 0.02439
a0= 0.328 f(a0)=-2.185
a1= 0.337 f(a1)=-2.185
b1= 0.343 f(b1)=-2.185
b0= 0.353 f(bo)=-2.184
iteration 12 of golden section, domain length = 0.01507
a0= 0.328 f(a0)=-2.185
a1= 0.334 f(a1)=-2.185
b1= 0.337 f(b1)=-2.185
b0= 0.343 f(bo)=-2.185
Golden section completed after 12 iterations xmin= 0.33282 fmin= -2.18518



3.3 Fibonacci Search Method

The Fibonacci sequence F_1, F_2, F_3,...,F_n is defined as follows. Starting with the following two values: $F_{-1} = 0$ and $F_0 = 1$

Then for values of $k \geq 0$ we have that (3.14) defined the Fibonacci sequence:

$$F_{k+1} = F_k + F_{k-1} \qquad (3.14)$$

The first 10 values of the Fibonacci sequence are then given in Table 5.1.

Table 5.1: Fibonacci Sequence

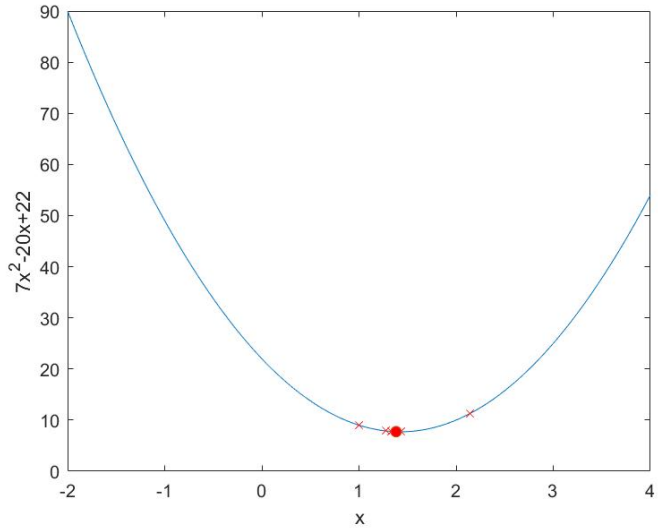
k	F_k
-1	0
0	1
1	1
2	2
3	3
4	5
5	8
6	13
7	21
8	34
9	55
10	89

Instead of using the a constant value of $\rho(=0.38197)$, as was the case for the Golden search method, for the Fibonacci search, the value of ρ_k changes with iteration number and is given by the sequence of (3.15). Note that the sequence of ρ_k is in reverse order to the Fibonacci number sequence, and that you always need one more Fibonacci sequence number than the number of iterations required!!

$$\begin{aligned}
\rho_1 &= 1-\frac{F_N}{F_{N+1}} \\
\rho_2 &= 1-\frac{F_{N-1}}{F_N} \\
&\vdots \\
\rho_k &= 1-\frac{F_{N-k+1}}{F_{N-k+2}} \\
&\vdots \\
\rho_N &= 1-\frac{F_1}{F_2}
\end{aligned} \tag{3.15}$$

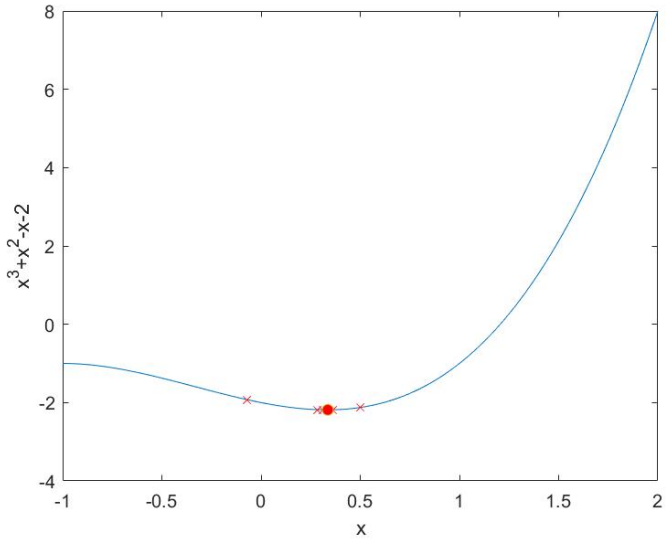
Fibonacci Search Example Problem 1: Problem 45 in Worked Examples
Minimize $f(x)=7x^2-20x+22$ for $-2\le x\le 4$ with Fibonacci Search Parameters: N=8 Fibonacci terms (i.e. 7 Fibonacci iterations), xmin=-2, xmax=4. Calculations:
iteration 1 of fibonacci, rho = 0.38182 domain length = 6.00000
a0=-2.000 f(a0)=90.000
a1= 0.291 f(a1)=16.774
b1= 1.709 f(b1)= 8.265
b0= 4.000 f(bo)=54.000
iteration 2 of fibonacci, rho = 0.38235 domain length = 3.70909
a0= 0.291 f(a0)=16.774
a1= 1.709 f(a1)= 8.265
b1= 2.582 f(b1)=17.024
b0= 4.000 f(bo)=54.000
iteration 3 of fibonacci, rho = 0.38095 domain length = 2.29091
a0= 0.291 f(a0)=16.774
a1= 1.164 f(a1)= 8.206

b1= 1.709 f(b1)= 8.265
b0= 2.582 f(bo)=17.024
iteration 4 of fibonacci, rho = 0.38462 domain length = 1.41818
a0= 0.291 f(a0)=16.774
a1= 0.836 f(a1)=10.169
b1= 1.164 f(b1)= 8.206
b0= 1.709 f(bo)= 8.265
iteration 5 of fibonacci, rho = 0.37500 domain length = 0.87273
a0= 0.836 f(a0)=10.169
a1= 1.164 f(a1)= 8.206
b1= 1.382 f(b1)= 7.730
b0= 1.709 f(bo)= 8.265
iteration 6 of fibonacci, rho = 0.40000 domain length = 0.54545
a0= 1.164 f(a0)= 8.206
a1= 1.382 f(a1)= 7.730
b1= 1.491 f(b1)= 7.741
b0= 1.709 f(bo)= 8.265
iteration 7 of fibonacci, rho = 0.33333 domain length = 0.32727
a0= 1.164 f(a0)= 8.206
a1= 1.273 f(a1)= 7.884
b1= 1.382 f(b1)= 7.730
b0= 1.491 f(bo)= 7.741
Fibonacci search completed with xmin= 1.38182 fmin= 7.72959



Fibonacci Search Example Problem 2: Problem 46 in Worked Examples
Minimize $f(x) = x^3 + x^2 - x - 2$ for $-1 \leq x \leq 2$ with with Fibonacci Search Parameters: N=8 Fibonacci terms (i.e. 7 Fibonacci iterations), xmin=-2, xmax=4. Calculations:
iteration 1 of fibonacci, rho = 0.38182 domain length = 3.00000
a0=-1.000 f(a0)=-1.000
a1= 0.145 f(a1)=-2.121
b1= 0.855 f(b1)=-1.500
b0= 2.000 f(bo)= 8.000
iteration 2 of fibonacci, rho = 0.38235 domain length = 1.85455
a0=-1.000 f(a0)=-1.000
a1=-0.291 f(a1)=-1.649
b1= 0.145 f(b1)=-2.121

```
b0= 0.855 f(bo)=-1.500
iteration     3 of fibonacci, rho =    0.38095 domain length =    1.14545
a0=-0.291 f(a0)=-1.649
a1= 0.145 f(a1)=-2.121
b1= 0.418 f(b1)=-2.170
b0= 0.855 f(bo)=-1.500
iteration     4 of fibonacci, rho =    0.38462 domain length =    0.70909
a0= 0.145 f(a0)=-2.121
a1= 0.418 f(a1)=-2.170
b1= 0.582 f(b1)=-2.046
b0= 0.855 f(bo)=-1.500
iteration     5 of fibonacci, rho =    0.37500 domain length =    0.43636
a0= 0.145 f(a0)=-2.121
a1= 0.309 f(a1)=-2.184
b1= 0.418 f(b1)=-2.170
b0= 0.582 f(bo)=-2.046
iteration     6 of fibonacci, rho =    0.40000 domain length =    0.27273
a0= 0.145 f(a0)=-2.121
a1= 0.255 f(a1)=-2.173
b1= 0.309 f(b1)=-2.184
b0= 0.418 f(bo)=-2.170
iteration     7 of fibonacci, rho =    0.33333 domain length =    0.16364
a0= 0.255 f(a0)=-2.173
a1= 0.309 f(a1)=-2.184
b1= 0.364 f(b1)=-2.183
b0= 0.418 f(bo)=-2.170
iteration     8 of fibonacci, rho =    0.50000 domain length =    0.10909
a0= 0.255 f(a0)=-2.173
a1= 0.309 f(a1)=-2.184
b1= 0.309 f(b1)=-2.184
b0= 0.364 f(bo)=-2.183
Fibonacci completed with xmin=   0.33636 fmin=  -2.18517
```



3.4 Steepest-Descent Method

The steepest-descent method is the simplest numerical method for unconstrained optimization. The aim of the method is to find the direction ***d***, at the current iteration, in which

the cost function ***f(x)*** decreases most rapidly, at least locally. The steepest-descent method is a first-order method since only the gradient of the cost function is calculated and used to evaluate the search direction.

The gradient of a scalar function ***f(x₁, x₂, . . ., xₙ)*** as the column vector is given by (3.16).

$$\mathbf{c} = \nabla \mathbf{f} = \left[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \cdots \frac{\partial f}{\partial x_n} \right]^T \tag{3.16}$$

The vector ***c*** is used to represent the gradient of the cost function ***f(x)***; represented by (3.17).

$$c_i = \frac{\partial f}{\partial x_i} \tag{3.17}$$

The point x_k at which this vector is then calculated is represented by (3.18).

$$c^{(k)} = c\left(x^{(k)}\right) = \left[\frac{\partial f\left(x^{(k)}\right)}{\partial x_i} \right]^T \tag{3.18}$$

The gradient at a point ***x*** points is the direction of *maximum increase* in the cost function. Thus the direction of *maximum decrease* is opposite to that, that is, negative of the gradient vector. Any small move in the negative gradient direction will result in the maximum local rate of decrease in the cost function. The negative gradient vector thus represents a direction of steepest descent for the cost function and is represented by (3.19).

$$\mathbf{d} = -\mathbf{c}, \textit{ or } d_i = -c_i = -\frac{\partial f}{\partial x_i}, i = 1 \textit{ to } n \tag{3.19}$$

Since ***d*** = **– *c*** , the descent condition of inequality is always satisfied due to (5.20).

$$\left(\mathbf{c} \bullet \mathbf{d}\right) = -\left\|\mathbf{c}\right\|^2 < 0 \tag{3.20}$$

The Steepest-Descent Algorithm is then given by the following 6 steps:

Step 1: Estimate a starting design $x^{(0)}$ and set the iteration counter $k = 0$. Select a convergence parameter $\varepsilon > 0$.

Step 2: Calculate the gradient of $f(x)$ at the current point $x^{(k)}$ as $c^{(k)} = \nabla f\left(x^{(k)}\right)$.

Step 3: Calculate the length of $c^{(k)}$ as $\left\|c^{(k)}\right\|$. If $\left\|c^{(k)}\right\| < \varepsilon$, then stop the iterative process because $x^* = x^{(k)}$ is a local minimum point. Otherwise, continue.

Step 4: Let the search direction at the current point $x^{(k)}$ be $\mathbf{d}^{(k)} = -\mathbf{c}^{(k)}$.

Step 5: Calculate a step size α_k that minimizes $f(\alpha) = f\left(x^{(k)} + \alpha \mathbf{d}^{(k)}\right)$ in the direction $\mathbf{d}^{(k)}$ Any one-dimensional search algorithm may be used to determine α_k.

Step 6: Update the design using $x^{(k+1)} = x^{(k)} + \alpha\, \mathbf{d}^{(k)}$. Set $k = k + 1$ and go to Step 2.

The basic idea of the steepest-descent method is quite simple. We start with an initial estimate for the minimum design. The direction of steepest descent is computed at that point. If the direction is nonzero, we move as far as possible along it to reduce the cost function. At the new design point, we calculate the steepest-descent direction again and repeat the entire process.

3.5 What is a Simplex

The simplex can be thought of as a polygon with $n + 1$ vertices. Where n is the number of design variables. So, if there are 2 design variables, $n = 2$, the simplex has 3 vertices and is a triangle, Figure 3.2.

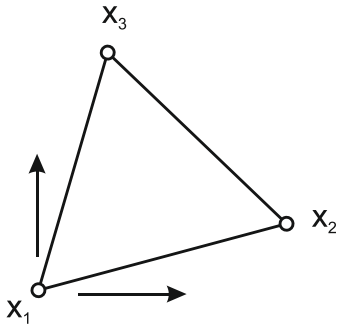


Figure 3.2: A Simplex which represents 2 design variables consists of 3 points and is a triangle.

If there are 3 design variables, then $n = 3$ and the simplex has 4 vertices and is now a tetrahedron, Figure 3.3.

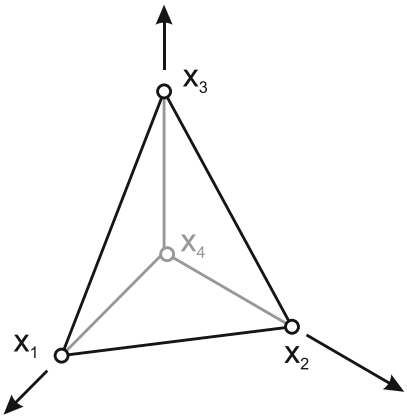


Figure 3.3: A Simplex which represents 3 design variables consists of 4 points and is a tetrahedron.

When the points are equidistant, the simplex is said to be regular. Basically, it has one more point than the number of design variables which represents the number of dimensions.

3.6 Nelder-Mead Simplex Method

The Nelder and Mead simplex method carries out a search in n^{th} dimensional space using heuristic ideas. Also known as nonlinear simplex.

The strengths of this method are:

1) Does NOT require derivatives of the Objective Function
2) The Objective Function does not have to be smooth.

The weakness of the method are that it is:

1) Not very efficient, particularly for problems with more than about 10 design variables;
2) Above *n = 10*, convergence becomes increasingly difficult

The basic idea in the simplex method is to compare the values of the objective function at the *n + 1* vertices of a general simplex. Then move the simplex gradually toward the optimum point using an iterative process with 5 simple operations. The sequence of operations is chosen based on the relative values of the objective function at each of the points.

The Steps of the Nelder-Mead Simplex Method are:

1) Find the n+1 points of the simplex
2) Evaluate and sort the points
3) Carry out the 5 Simplex Operations:
   i) Reflection,
   ii) Expansion,
   iii) Inside and Outside Contraction
   iv) Shrinking
   v) Convergence

We are now going to look at these 3 steps and 5 operations. Appreciate that each of the operations generates a new point. The sequence of operations carried out in each iteration depends on the value of the objective function at the new point relative to the other key points.

Let's now start with determining the initial Simplex, that is, the n+1 points

3.6.1 Step 1: Find the n+1 points of the simplex

The 1st step is to find the *n+1* points of the simplex from an initial guess starting position x_0. Then add a step size to each component of x_0 to generate *n+1* new points. Generating a simplex with equal length edges is preferable. Start by assuming that the length of all sides is defined as *c* and that the initial guess, x_0 is the *(n + 1)th* point. The remaining points, *i = 1...n* can be computed by adding a vector to x_0. With all components $\left(i = 1,2,\cdots i-1, i+1,\cdots n\right)$ equal to *b*, apart for the *ith* component which is *a*. The equations to calculate *a* and *b* are given in (5.21) and (3.22).

3 (3.21)

$$a = b + \frac{c}{\sqrt{2}} \tag{3.22}$$

where *n* is the number of design variables.

Let us assume that we have 2 design variables, x_1 and x_2. That means that we need to generate 3 points. Let us also assume that the point x_0 is given by an initial guess: $x_0 = \begin{bmatrix} x_{1_0} \\ x_{2_0} \end{bmatrix}$

And let's assume a step size of *c* units.

Since *n = 2*, then *b* and *a* become (i) and (ii):

$$b = \frac{c}{n\sqrt{2}}\left(\sqrt{n+1}-1\right) = \frac{c}{2\sqrt{2}}\left(\sqrt{2+1}-1\right) = \frac{c}{2\sqrt{2}}\left(\sqrt{3}-1\right) = 0.25882c \tag{i}$$

$$a = b + \frac{c}{\sqrt{2}} = 0.25882c + \frac{c}{\sqrt{2}} = 0.96593c \qquad \text{(ii)}$$

The two new points, x_1 and x_2 then become (iii) and (iv).

$$x_1 = \begin{bmatrix} x_{1_0} + a \\ x_{2_0} + b \end{bmatrix} = \begin{bmatrix} x_{1_0} + 0.96593c \\ x_{2_0} + 0.25882c \end{bmatrix} \qquad \text{(iii)}$$

$$x_2 = \begin{bmatrix} x_{1_0} + b \\ x_{2_0} + a \end{bmatrix} = \begin{bmatrix} x_{1_0} + 0.25882c \\ x_{2_0} + 0.96593c \end{bmatrix} \qquad \text{(iv)}$$

So, if c = 3 and $x_0 = \begin{bmatrix} 1.5 \\ 1 \end{bmatrix}$, then x_1 and x_2 become (v) and (vi).

$$x_1 = \begin{bmatrix} x_{1_0} + 0.96593c \\ x_{2_0} + 0.25882c \end{bmatrix} = \begin{bmatrix} 1.5 + 0.96593 \times 3 \\ 1 + 0.25882 \times 3 \end{bmatrix} = \begin{bmatrix} 4.398 \\ 1.776 \end{bmatrix} \qquad \text{(v)}$$

$$x_2 = \begin{bmatrix} x_{1_0} + 0.25882c \\ x_{2_0} + 0.96593c \end{bmatrix} = \begin{bmatrix} 1.5 + 0.25882 \times 3 \\ 1 + 0.96593 \times 3 \end{bmatrix} = \begin{bmatrix} 2.276 \\ 3.898 \end{bmatrix} \qquad \text{(vi)}$$

So the 3 points of the simplex are then given by (vii). And graphically, these 3 points generate the Simplex of Figure 3.4.

$$x_0 = \begin{bmatrix} 1.5 \\ 1 \end{bmatrix}; x_1 = \begin{bmatrix} 4.398 \\ 1.776 \end{bmatrix}; x_2 = \begin{bmatrix} 2.276 \\ 3.898 \end{bmatrix} \qquad \text{(vii)}$$

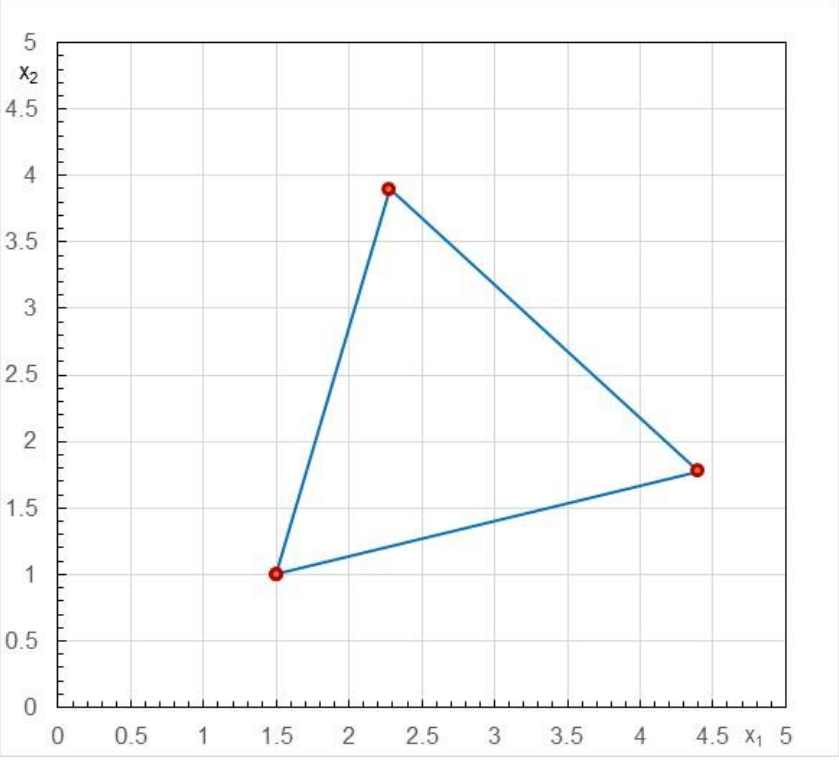


Figure 3.4: The triangular Simplex showing the generated points about x_0.

If we have a problem with 3 design variables, x_1, x_2 and x_3. That means that we need to generate 4 points.

If the initial guess point x_0 is $x_0 = \begin{bmatrix} x_{1_0} & x_{2_0} & x_{3_0} \end{bmatrix}^T$ and assuming a step size of c units, then since *n = 3*, then *b* and *a* become (viii) and (ix).

$$b = \frac{c}{n\sqrt{2}}\left(\sqrt{n+1}-1\right) = \frac{c}{3\sqrt{2}}\left(\sqrt{3+1}-1\right) = \frac{c}{3\sqrt{2}}\left(\sqrt{4}-1\right) = 0.23570c \qquad \text{(viii)}$$

$$a = b + \frac{c}{\sqrt{2}} = 0.23570c + \frac{c}{\sqrt{2}} = 0.94281c \qquad \text{(ix)}$$

The three new points, x_1, x_2 and x_3 then become (x).

$$x_1 = \begin{bmatrix} x_{1_0} + a \\ x_{2_0} + b \\ x_{3_0} + b \end{bmatrix} = \begin{bmatrix} x_{1_0} + 0.94281c \\ x_{2_0} + 0.23570c \\ x_{3_0} + 0.23570c \end{bmatrix}, x_2 = \begin{bmatrix} x_{1_0} + b \\ x_{2_0} + a \\ x_{3_0} + b \end{bmatrix} \text{and } x_3 = \begin{bmatrix} x_{1_0} + b \\ x_{2_0} + b \\ x_{3_0} + a \end{bmatrix} \qquad \text{(x)}$$

Table 3.2 gives the calculated values for ***a*** and ***b*** from equations (3.21) and (3.22) to calculate the vertices of the initial Simplex for problems of up to 10 design variables

Table 3.2: Values of a and b for up to 10 design variables

n	a	b
2	0.96593c	0.25882c
3	0.94281c	0.23570c
4	0.92561c	0.21851c
5	0.91210c	0.20499c
6	0.90106c	0.19395c
7	0.89181c	0.18470c
8	0.88388c	0.17678c
9	0.87699c	0.16988c
10	0.87092c	0.16381c

3.6.2 Step 2: Evaluate and Sort the points

After generating the initial simplex, the objective function needs to be evaluated at each of its vertices. Then all of the points need to be sorted from *best* (lowest valued) to *worst* (highest valued). Such that the points are arranged as per (3.23).

$$f\left(x_L\right) < \cdots < f\left(x_i\right) < \cdots < f\left(x_T\right) < f\left(x_H\right) \qquad (3.23)$$

Three points then need to be identified:

1) The point with the highest (worst) values of the objective function: (x_H)
2) The point which is next to the highest (worst) values of the objective function: (x_T)
3) The point with the lowest (best) values of the objective function: (x_L)

The following five (5) Simplex operations now need to be carried out:

3.6.3 Step 3: Carry out the 5 Simplex Operations

3.6.3.1 Reflection

Calculate the centroid of all the points x_i, excluding the worst point (when $i = H$) using equation (3.24). Graphically, for the case of n = 2, it looks like Figure 3.5. Note that for the Simplex with 2 design variables (n=3), x_2 is also the next to highest point x_T.

$$x_{Ce} = \frac{1}{n}\sum_{\substack{i=1 \\ i \neq H}}^{n+1} x_i \qquad (3.24)$$

After computing x_{Ce}, we know that the line from x_H to x_{Ce} is a descent direction, Figure 3.6. A new point can then be found on this line by reflection, which means that we reflect the distance from H to Ce about Ce.

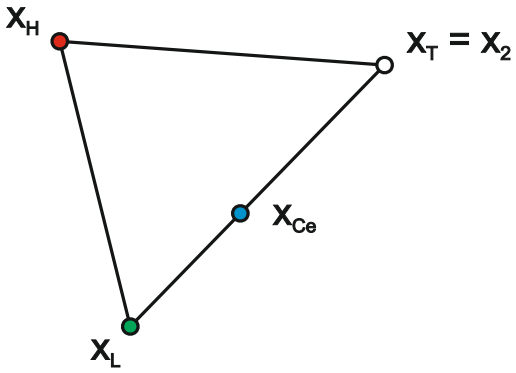


Figure 3.5: Simplex showing the position of the centroid point.

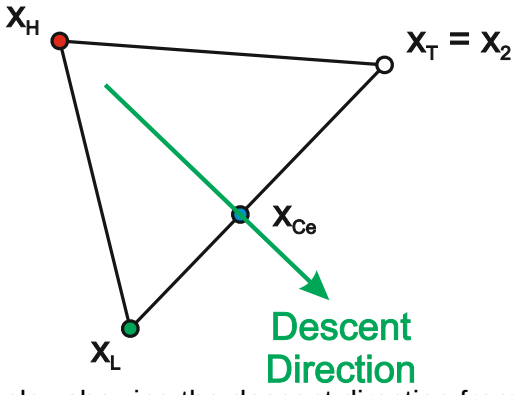


Figure 3.6: Simplex showing the descent direction from x_H towards x_{Ce}.

This gives the new point X_R, which is calculated using equation (3.25) and is shown in Figure 3.7.

$$x_R = (1+\alpha)x_{Ce} - \alpha x_H = x_{Ce} + \alpha(x_{Ce} - x_H) \qquad (3.25)$$

where: α is the reflection coefficient It always has a value greater than 0 but less than 1, although it is **usually given the value 1**. It is defined by (3.26).

$$\alpha = \frac{\text{distance between } x_R \text{ and } x_{Ce}}{\text{distance between } x_H \text{ and } x_{Ce}} \qquad (3.26)$$

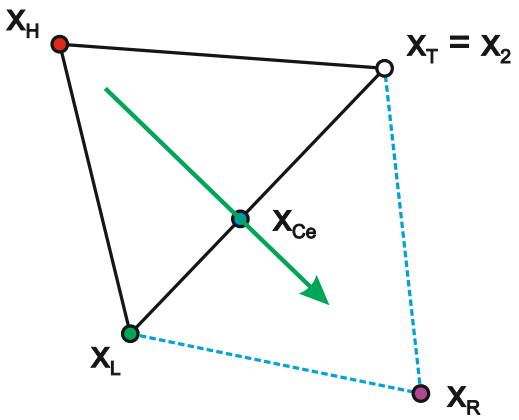


Figure 3.7: Simplex showing the reflected new point x_R.

Since the direction of movement of the simplex is always away from the worst result, we will be moving in a favourable direction.
Depending on the value of the objective function at the reflected point x_R, there are 4 choices which can be made of what to do next, these are:

1) If $f(x_R)$ lies between $f(x_T)$ and $f(x_L)$ *such that* $\{f(x_L) < f(x_R) < f(x_T)\}$, then x_H is replaced by x_R. A new simplex is started, and therefore we need to check for convergence. So go to Section 3.6.3.5 Convergence.
2) If $f(x_R)$ is less than $f(x_L)$ *such that* $\{f(x_R) < f(x_L)\}$, then the reflection produced a new minimum. This suggests that moving further in the same direction pointing from x_{Ce} to x_R might produce a new minimum. Which means we need to carry out the Expansion Operation, so go to Section 3.6.3.2 Expansion.
3) If $f(x_R)$ is greater than $f(x_T)$ and less than $f(x_H)$ such that $\{f(x_T) < f(x_R) < f(x_H)\}$. This suggests that the reflected point is *between the* ***two worst points***. Which means we need to carry out the Outside Contraction Operation, so go to Section 5.6.3.3 Outside Contraction.
4) If $f(x_R)$ is greater than $f(x_H)$ such that $\{f(x_R) > f(x_H)\}$, this means that the reflection produced a point *worse* than the ***worst***. This suggests that there might be a point inside the original points which might be better. Which means we need to carry out Inside Contraction Operation (iii), so go to Section 3.6.3.3a Inside Contraction

3.6.3.2 Expansion

To expand x_R to x_E , it is necessary to use equation (3.27), where the expansion process is shown in Figure 3.8.

$$x_E = \gamma x_R + (1-\gamma) x_{Ce} = x_{Ce} + \gamma (x_R - x_{Ce}) \qquad (3.27)$$

where: γ is the expansion coefficient It always has a value greater than 1, although it **is usually given the value 2**. It is defined by (3.28).

$$\gamma = \frac{\text{distance between } x_E \text{ and } x_{Ce}}{\text{distance between } x_R \text{ and } x_{Ce}} > 1 \qquad (3.28)$$

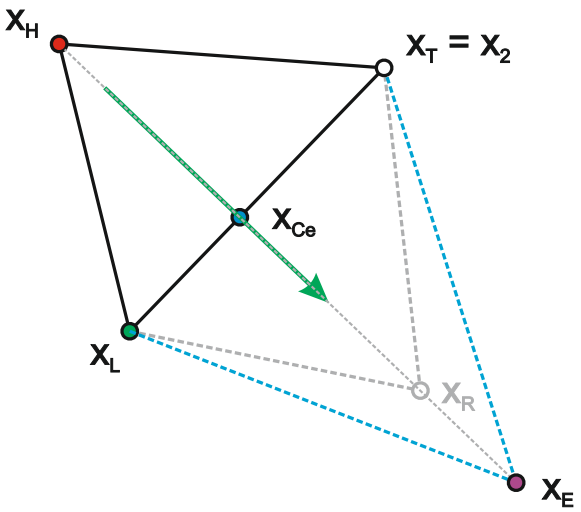


Figure 3.8: Simplex showing the expansion new point x_E.

Depending on the value of the objective function at the expansion point x_E, there are 2 choices which can be made of what to do next, these are:

1) If $f(x_E)$ is less than $f(x_R)$ such that $f(x_E) < f(x_R)$, then we need to replace x_H with x_E. A new simplex is started, and therefore we need to check for convergence. So go to Section 3.6.3.5 Convergence.
2) If $f(x_E)$ is greater than $f(x_R)$ such that $f(x_E) > f(x_R)$, this means that the Expansion process was NOT successful. So, need to replace x_H with x_R and. A new simplex is started, and therefore we need to check for convergence. So go to Section 3.6.3.5 Convergence.

3.6.3.3 Outside Contraction

To carry out Outside Contraction it is necessary to use equation (3.29), where the outside contraction process is shown in Figure 3.9.

$$x_{OC} = x_{Ce} + \beta\left(x_R - x_{Ce}\right) \tag{3.29}$$

where: β is the contraction coefficient It always has a value less than 1, although it is **usually given the value 0.5**. It is defined by (3.30).

$$\beta = \frac{\text{distance between } x_{OC} \text{ and } x_{Ce}}{\text{distance between } x_R \text{ and } x_{Ce}} < 1 \tag{3.30}$$

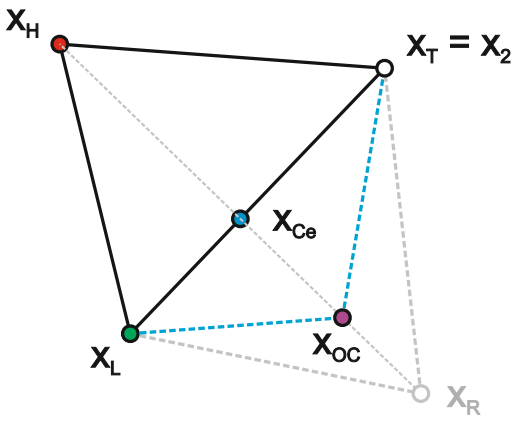


Figure 3.9: Simplex showing the outside contraction new point x_{OC}.

Depending on the value of the objective function at the Outside Contraction point x_{OC}, there are 2 choices which can be made of what to do next, these are:

1) If $f(x_{OC})$ is less than $f(x_R)$ such that $\{f(x_{OC}) < f(x_R)\}$, then replace x_H with x_{OC}. A new simplex is started, and therefore we need to check for convergence. So go to Section 3.6.3.5 Convergence.
2) Otherwise the Shrinking Operation needs to be carried out, so go to Section 3.6.3.4 Shrinking.

3.6.3.3a Inside Contraction

To carry out Inside Contraction it is necessary to use equation (3.31), where the inside contraction process is shown in Figure 3.10.

$$x_{IC} = x_{Ce} - \beta(x_R - x_{Ce}) \tag{3.31}$$

where: β is the contraction coefficient It always has a value less than 1, although it is **usually given the value 0.5**. It is defined by (3.32).

$$\beta = \frac{\text{distance between } x_{IC} \text{ and } x_{Ce}}{\text{distance between } x_R \text{ and } x_{Ce}} < 1 \tag{3.32}$$

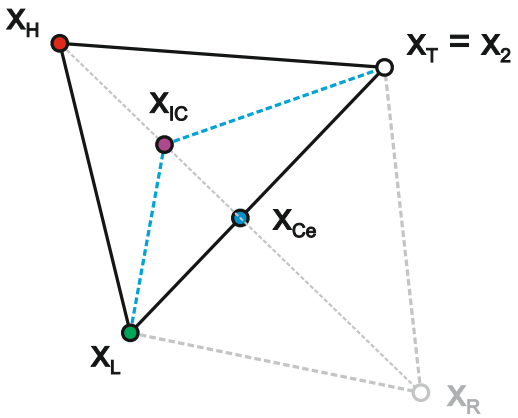


Figure 3.10: Simplex showing the inside contraction new point x_{IC}.

Depending on the value of the objective function at the Outside Contraction point x_{IC}, there are 2 choices which can be made of what to do next, these are:

1) If $f(x_{IC})$ is less than $f(x_H)$ $\{f(x_{IC}) < f(x_H)\}$, then replace x_H with x_{IC}. A new simplex is started, and therefore we need to check for convergence. So go to Section 3.6.3.5 Convergence.
2) Otherwise the Shrinking Operation needs to be carried out, so go to Section 3.6.3.4 Shrinking.

3.6.3.4 Shrinking

If *Reflection, Expansion* and both *Contractions* failed, it will be necessary to resort to the Shrinking operation. This operation retains the best point (x_L) and shrinks the Simplex about

that point. To shrink all points about the best point, (x_L) it is necessary to use equation (3.33) and this process is shown in Figure 3.11.

$$\text{For } 2 \le i \le n+1\text{: } x_i = x_L + \rho\left(x_i - x_L\right) \tag{3.33}$$

where: ρ is the shrinking coefficient It always has a value less than 1, although it is **usually given the value 0.5**. It is defined by (3.34).

$$\rho = \frac{\text{distance between } x_{i_{New}} \text{ and } x_L}{\text{distance between } x_H \text{ and } x_L} < 1 \tag{3.34}$$

As a new simplex is generated, we need to check for convergence. So go to Section 3.6.3.5 Convergence.

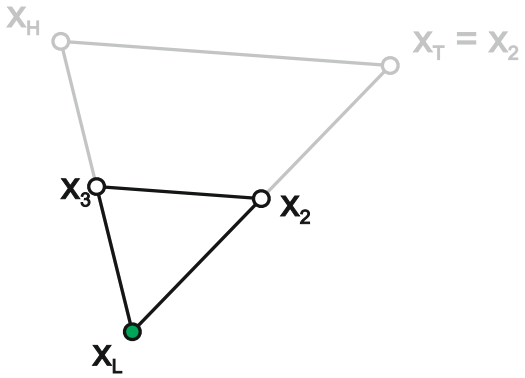


Figure 5.11: Simplex showing the shrinking new points x_i.

5.6.3.5 Convergence

Two convergence criteria can be used:

1) The size of the Simplex needs to be less than a tolerance $\left(\varepsilon_S\right)$, given by equation (3.35).

$$s = \sum_{i=1}^{n} \left| x_i - x_{i+1} \right| \le \varepsilon_S \tag{3.35}$$

2) The standard deviation of the function value in all vertices of the Simplex needs to be less than a small quantity $\left(\varepsilon_\sigma\right)$, given by equation (3.36).

$$\sigma = \frac{\sqrt{\sum_{i=1}^{n+1}\left(f\left(x_i\right) - f\left(x_{Ce}\right)\right)^2}}{n+1} \le \varepsilon_\sigma \tag{3.36}$$

To determine what to do next, 2 choices are available, these are:

1) If the convergence tolerance has been satisfied: $\left(s \le \varepsilon_S\right)$ or $\left(\sigma \le \varepsilon_\sigma\right)$, the solution has converged, so stop the algorithm.
2) The solution has ***not*** converged so need to sort point and start the cycle again, so go to Section 3.6.2 Evaluate and Sort the points

3.6.3 Example

Minimize the function (a) starting from the point (b) with a step size of c = 2

$$\textbf{Minimize:} \quad f(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 \tag{a}$$

$$x_0 = \begin{bmatrix} -1.2 \\ 1.0 \end{bmatrix} \tag{b}$$

Use the following parameters to solve this problem: $\{\alpha, \beta, \gamma, \rho\} = \{1, 0.5, 2, 0.5\}$

Step 1: Find the n+1 points of the simplex:

We have previously calculated these values to be (c) and (d).

$$x_1 = \begin{bmatrix} x_{1_0} + a \\ x_{2_0} + b \end{bmatrix} = \begin{bmatrix} x_{1_0} + 0.96593c \\ x_{2_0} + 0.25882c \end{bmatrix} \tag{c}$$

$$x_2 = \begin{bmatrix} x_{1_0} + b \\ x_{2_0} + a \end{bmatrix} = \begin{bmatrix} x_{1_0} + 0.25882c \\ x_{2_0} + 0.96593c \end{bmatrix} \tag{d}$$

Substituting for x_0 and c gives (e) and (f).

$$x_1 = \begin{bmatrix} -1.2 + 0.96593 \times 2 \\ 1.0 + 0.25882 \times 2 \end{bmatrix} = \begin{bmatrix} 0.732 \\ 1.518 \end{bmatrix} \tag{e}$$

$$x_2 = \begin{bmatrix} -1.2 + 0.25882 \times 2 \\ 1.0 + 0.96593c \times 2 \end{bmatrix} = \begin{bmatrix} -0.682 \\ 2.932 \end{bmatrix} \tag{f}$$

Therefore the 3 points of the simplex are given in (g), and plotted in Figure 3.12.

$$x_0 = \begin{bmatrix} -1.2 \\ 1.0 \end{bmatrix},\ x_1 = \begin{bmatrix} 0.732 \\ 1.518 \end{bmatrix} \text{and } x_2 = \begin{bmatrix} -0.682 \\ 2.932 \end{bmatrix} \tag{g}$$

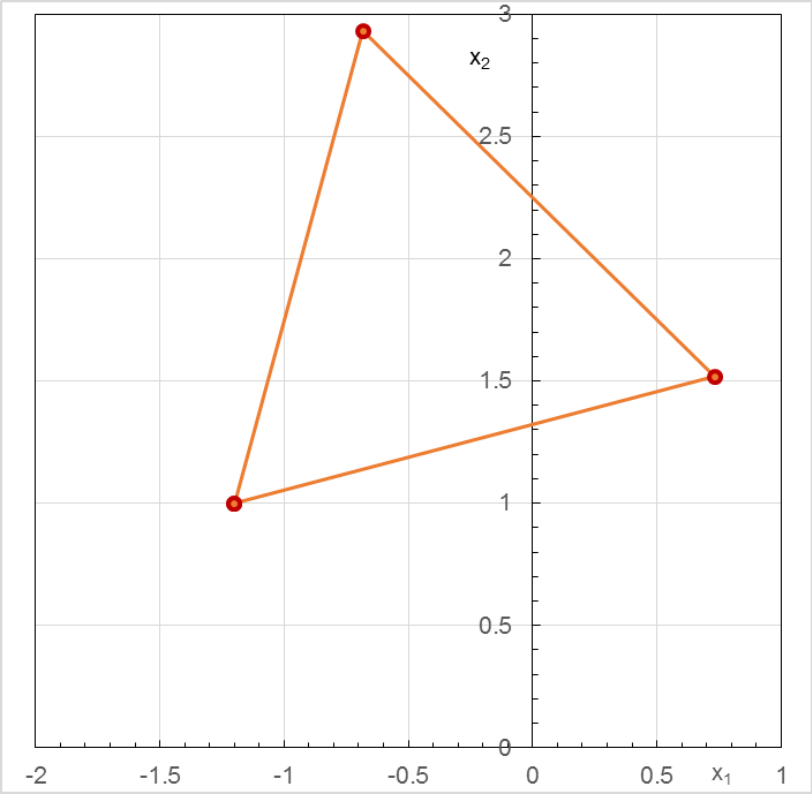


Figure 3.12: Initial Simplex for this problem.

Step 2: Evaluate and Sort the points

Substituting the 3 values from (g) into the function (a), gives the results of (h).

$f_{x_0}(-1.2,1.0) = 24.2$, $f_{x_1}(0.732,1.518) = 96.51$ and $f_{x_2}(-0.682,2.932) = 611.06$ (h)

Which coincidentally happen to be aligned in increasing order such that, the new names for the points are given in (i).

$$x_L = \begin{bmatrix} -1.2 \\ 1.0 \end{bmatrix},\ x_T = \begin{bmatrix} 0.732 \\ 1.518 \end{bmatrix} \text{ and } x_H = \begin{bmatrix} -0.682 \\ 2.932 \end{bmatrix} \quad \text{(i)}$$

So, now need to carry out the Reflection Operation

Step 3: Operation (i) Refection

Now need to calculate the centroid of all the points x_i, excluding the worst point. This is done by substituting x_L and x_1 into equation (3.24), which gives the coordinates in (j).

$$x_{Ce} = \frac{1}{n}\sum_{\substack{i=1 \\ i \neq H}}^{n+1} x_i = \begin{bmatrix} \frac{(-1.2+0.732)}{2} \\ \frac{(1.0+1.518)}{2} \end{bmatrix} = \begin{bmatrix} -0.234 \\ 1.259 \end{bmatrix} \quad \text{(j)}$$

Evaluate this point, gives (k).

$$f_{Ce}\left(-0.234,1.259\right)=146.49 \tag{k}$$

Calculate the reflection point using α = 1, using the points in (l).

$$x_{Ce}=\begin{bmatrix}-0.234\\1.259\end{bmatrix}\text{and } x_H=\begin{bmatrix}-0.682\\2.932\end{bmatrix} \tag{l}$$

Substituting these values into equation (3.25) gives (m).

$$\begin{aligned}&x_R=\left(1+\alpha\right)x_{Ce}-\alpha x_H=\left(1+1\right)\begin{bmatrix}-0.234\\1.259\end{bmatrix}-1\times\begin{bmatrix}-0.682\\2.932\end{bmatrix}=\begin{bmatrix}0.214\\-0.414\end{bmatrix}\\&\therefore x_R=\begin{bmatrix}0.214\\-0.414\end{bmatrix}\end{aligned} \tag{m}$$

Evaluate this point, gives (n).

$$f_R\left(0.214,-0.414\right)=21.79 \tag{n}$$

We now need to compare the value of $f(x_R)$ with those of the other points of the Simplex, and using the rules set in section 3.6.3.1 Reflection, we can then decide what needs to happen next.

Since $\left\{f\left(x_R\right)<f\left(x_L\right)\right\}$, the next step to follow is Step 3: (ii) Expansion.

Step 3: Operation (ii) Expansion

Calculate the expansion point using γ = 2, using the points in (o).

$$x_{Ce}=\begin{bmatrix}-0.234\\1.259\end{bmatrix}\text{and } x_R=\begin{bmatrix}0.214\\-0.414\end{bmatrix} \tag{o}$$

Substituting these values into equation (3.27) gives (p).

$$x_E=\gamma x_R+\left(1-\gamma\right)x_{Ce}=2\times\begin{bmatrix}0.214\\-0.414\end{bmatrix}+\left(1-2\right)\times\begin{bmatrix}-0.234\\1.259\end{bmatrix}=\begin{bmatrix}0.663\\-2.087\end{bmatrix} \tag{p}$$

Evaluate this point, gives (q).

$$f_E\left(0.663,-2.087\right)=638.26 \tag{q}$$

We now need to compare the value of $f(x_E)$ with those of the other points of the Simplex, and using the rules set in section 3.6.3.2 Expansion, we can then decide what needs to happen next.

Since $\{f(x_E) > f(x_L)\}$ this means that the Expansion process was **not** successful. So, replace x_H with x_R and generate the new Simplex (shown in Figure 3.13), and then go to Step 3: (v) Convergence

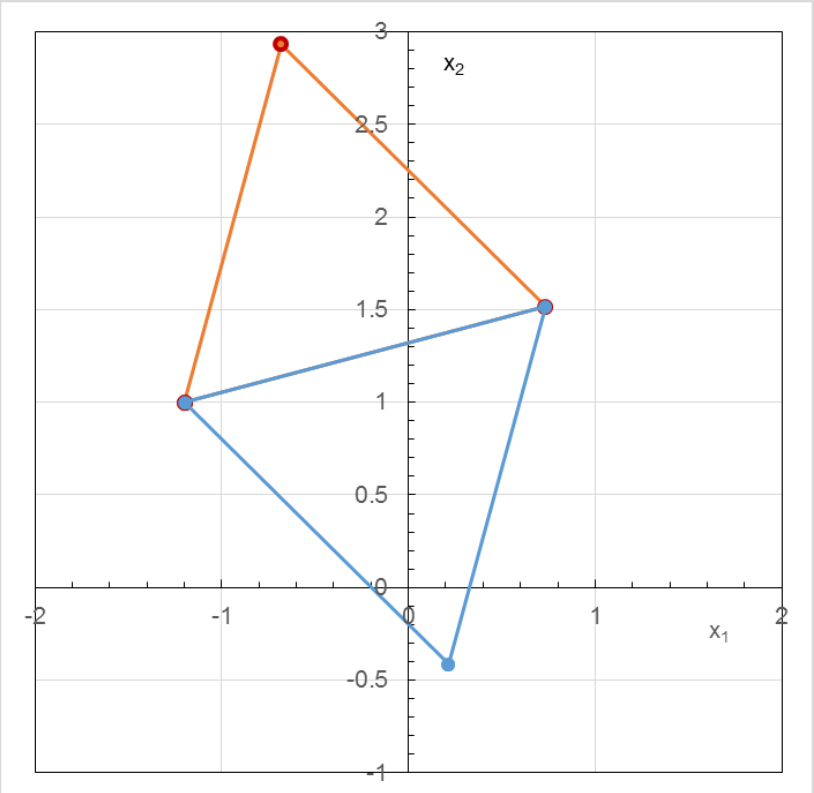


Figure 3.13: New Simplex generated by replacing x_H with x_R.

Step 3: Operation (v) Convergence

When putting the new Simplex together, arrange it in increasing order (r).

$$x_L = \begin{bmatrix} 0.214 \\ -0.414 \end{bmatrix},\ x_T = \begin{bmatrix} -1.2 \\ 1.0 \end{bmatrix} \text{ and } x_H = \begin{bmatrix} 0.732 \\ 1.518 \end{bmatrix} \tag{r}$$

$$f_L(0.214,-0.414) = 21.79,\ f_T(-1.2,1.0) = 24.20 \text{ and } f_H(0.732,1.518) = 98.28$$

Now need to calculate the standard deviation of the function value in all vertices of the Simplex using equation (3.36). But firstly, must calculate the centroid and function evaluation of the centroid point of the new Simplex using equation (3.24), the points x_L and x_1 which is calculated in (s).

$$x_{Ce} = \frac{1}{n}\sum_{\substack{i=1 \\ i\neq H}}^{n+1} x_i = \begin{bmatrix} \frac{(0.214-1.2)}{2} \\ \frac{(-0.414+1.0)}{2} \end{bmatrix} = \begin{bmatrix} -0.493 \\ 0.293 \end{bmatrix} \tag{s}$$

Evaluating this point gives (t).

$$f_{Ce}(-0.493,0.293) = 2.48 \tag{t}$$

Substituting all of these function values to (3.36) gives (u).

$$\begin{aligned}\sigma &= \frac{\sqrt{\sum_{i=1}^{n+1}\left(f\left(x_i\right)-f\left(x_{Ce}\right)\right)^2}}{n+1} \\ &= \frac{\sqrt{\left(21.79-2.48\right)^2+\left(24.20-2.48\right)^2+\left(98.28-2.48\right)^2}}{3} \\ \therefore \sigma &= 32.8\end{aligned} \qquad \text{(u)}$$

As we haven't specified a tolerance, at this stage we can't check if the problem has converged. So the next step, now that we have a new Simplex is to carry out another Reflection.

Step 3: Operation (i) Reflection

As we have already calculated the position of the centroid in the previous step. We can now calculate the reflection point using α = 1, using the points in (v).

$$x_{Ce} = \begin{bmatrix}-0.493\\0.293\end{bmatrix} \text{and } x_H = \begin{bmatrix}0.732\\1.518\end{bmatrix} \qquad \text{(v)}$$

Substituting these values into equation (3.25) gives (w).

$$\begin{aligned}x_R &= \left(1+\alpha\right)x_{Ce} - \alpha x_H = \left(1+1\right)\begin{bmatrix}-0.493\\0.293\end{bmatrix} - 1\times\begin{bmatrix}0.732\\1.518\end{bmatrix} \\ \therefore x_R &= \begin{bmatrix}-1.718\\-0.932\end{bmatrix}\end{aligned} \qquad \text{(w)}$$

Evaluate this point, gives (x).

$$f_R\left(-1.718,-0.932\right) = 1514.5 \qquad \text{(x)}$$

We now need to compare the value of $f(x_R)$ with those of the other points of the Simplex, and using the rules set in section 3.6.3.1 Reflection, we can then decide what needs to happen next.

Since $\left\{f\left(x_R\right) > f\left(x_H\right)\right\}$, need to carry out Operation (iii) Inside Contraction.

Step 3: Operation (iii) Inside Contraction

Calculate the Inside contraction using point using β = 0.5. Substituting the values of x_{CE} and x_R of (y) into equation (5.31) gives (z).

$$x_{Ce} = \begin{bmatrix}-0.493\\0.293\end{bmatrix} \text{and } x_R = \begin{bmatrix}-1.718\\-0.932\end{bmatrix} \qquad \text{(y)}$$

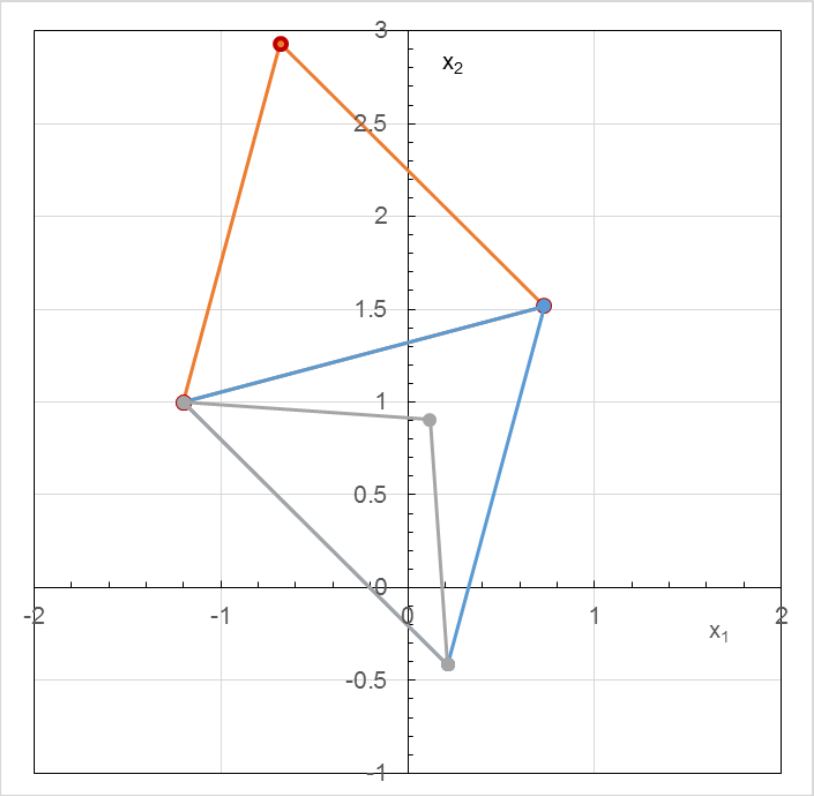


Figure 5.14: New Simplex generated by replacing x_H with x_{IC}.

$$\therefore x_{IC} = x_{Ce} - \beta\left(x_R - x_{Ce}\right) = \left(1+\beta\right)x_{Ce} - \beta x_R = \left(1+0.5\right)\begin{bmatrix}-0.493\\0.293\end{bmatrix} - 0.5\begin{bmatrix}-1.718\\-0.932\end{bmatrix} \quad \text{(z)}$$

$$x_{IC} = \begin{bmatrix}0.119\\0.905\end{bmatrix}$$

And evaluating this point, gives (aa).

$$f_{IC}\left(0.119, 0.905\right) = 80.16 \quad \text{(aa)}$$

We now need to compare the value of $f(x_{RIC})$ with those of the other points of the Simplex, and using the rules set in section 3.6.3.3a Inside Contraction, we can then decide what needs to happen next.

Since $\left\{f\left(x_{IC}\right) < f\left(x_H\right)\right\}$ then replace x_H with x_{IC} and generate the new Simplex (shown in Figure 3.14), and go to Step 3: (v) Convergence.... *and keep going*!

Figure 3.15, shows the plot of the function (a) superimposed on the Simplex of Figure 3.14.

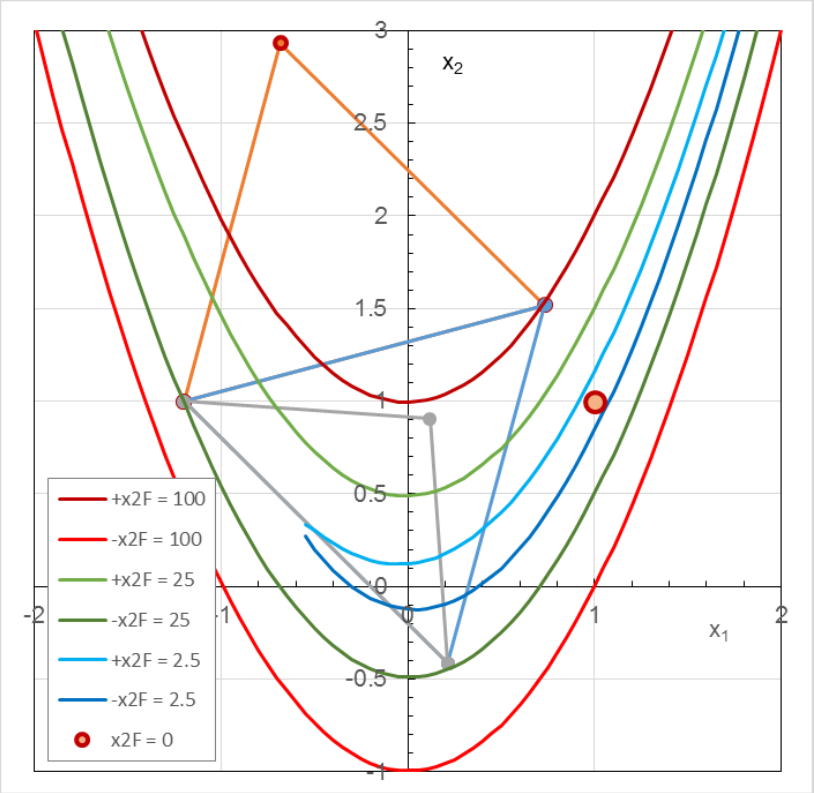


Figure 3.15: Plot of the contour of the function superimposed on the Simplex

Nelder-Mead Simplex Worked Example 1

Minimise $f(x_1, x_2) = 100\,(x_2 - x_1)^2 + (1 - x_1)^2$ using the following parameters:

Starting point $x^0 = [x_1 \quad x_2] = [-1.2 \quad 1.0]$

Initial step size c=2

$\alpha = 1.0, \quad \beta = 0.5, \quad \gamma = 2.0, \quad \rho = 0.5$

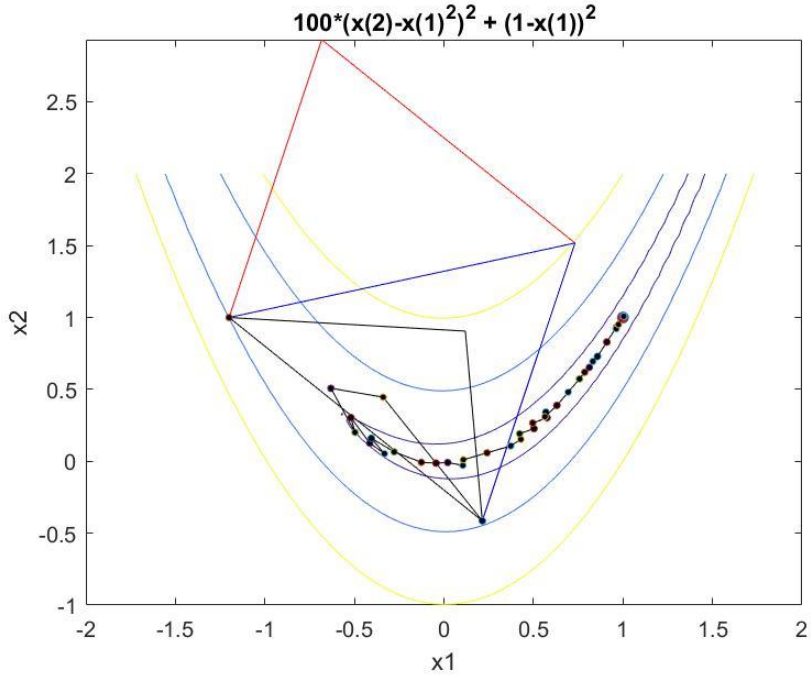
tol = 0.0001.

This takes a total of 70 iterations to converge. The calculations for the first 10 iterations give:

Initial simplex
xL= -1.200 1.000, fL=2.420e+01
xM= 0.732 1.518, fM=9.651e+01
xH= -0.682 2.932, fH=6.111e+02
After 1 iterations, simplex is given by:
xL= 0.214 -0.414, fL=2.179e+01
xM= -1.200 1.000, fM=2.420e+01
xH= 0.732 1.518, fH=9.651e+01
After 2 iterations, simplex is given by:
xL= 0.214 -0.414, fL=2.179e+01
xM= -1.200 1.000, fM=2.420e+01
xH= 0.119 0.905, fH=8.016e+01
After 3 iterations, simplex is given by:
xL= 0.214 -0.414, fL=2.179e+01
xM= -1.200 1.000, fM=2.420e+01
xH= -0.187 0.599, fH=3.324e+01
After 4 iterations, simplex is given by:
xL= -0.340 0.446, fL=1.272e+01
xM= 0.214 -0.414, fM=2.179e+01
xH= -1.200 1.000, fH=2.420e+01
After 5 iterations, simplex is given by:

xL= -0.631 0.508, fL=3.856e+00
xM= -0.340 0.446, fM=1.272e+01
xH= 0.214 -0.414, fH=2.179e+01
After 6 iterations, simplex is given by:
xL= -0.631 0.508, fL=3.856e+00
xM= -1.185 1.368, fM=4.913e+00
xH= -0.340 0.446, fH=1.272e+01
After 7 iterations, simplex is given by:
xL= -0.631 0.508, fL=3.856e+00
xM= -1.185 1.368, fM=4.913e+00
xH= -0.624 0.692, fH=1.179e+01
After 8 iterations, simplex is given by:
xL= -0.631 0.508, fL=3.856e+00
xM= -1.051 1.061, fM=4.386e+00
xH= -1.185 1.368, fH=4.913e+00
After 9 iterations, simplex is given by:
xL= -0.497 0.201, fL=2.448e+00
xM= -0.631 0.508, fM=3.856e+00
xH= -1.051 1.061, fH=4.386e+00
After 10 iterations, simplex is given by:
xL= -0.497 0.201, fL=2.448e+00
xM= -0.807 0.708, fM=3.580e+00
xH= -0.631 0.508, fH=3.856e+00

The evolution of the final solution is shown in the following figure**:**

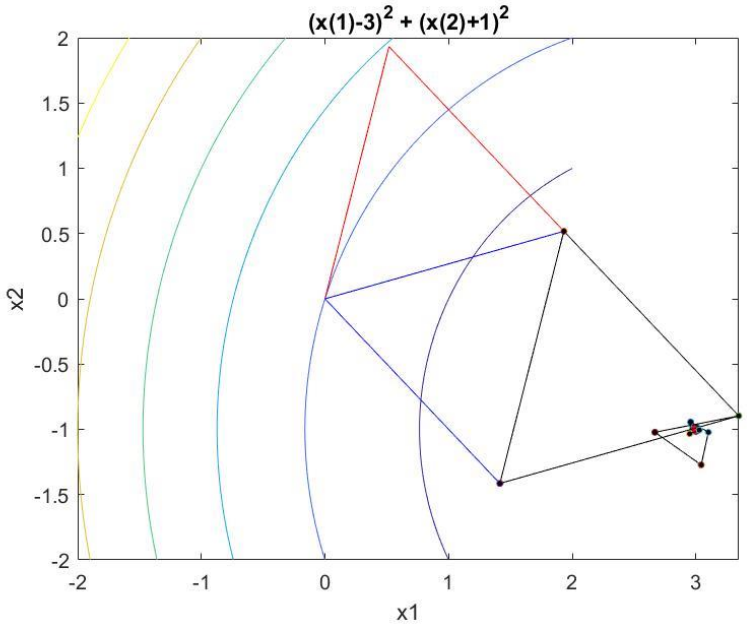


Plot of the Nelder-Mead Simplex solution from the calculations above.

Nelder-Mead Simplex Worked Example 2
Minimise $f(x_1, x_2) = (x_1 - 3)^2 + (x_2 + 1)^2$ using the following parameters:
Starting point $x^0 = [x_1 \;\; x_2] = [0 \;\; 0]$
Initial step size c=2
$\alpha = 1.0, \quad \beta = 0.5, \quad \gamma = 2.0, \quad \rho = 0.5$
tol = 0.001.

Initial simplex
xL= 1.932   0.518, fL=3.444e+00
xM= 0.000   0.000, fM=1.000e+01
xH= 0.518   1.932, fH=1.476e+01
After 1 iterations, simplex is given by:
xL= 1.414 -1.414, fL=2.686e+00
xM= 1.932   0.518, fM=3.444e+00
xH= 0.000   0.000, fH=1.000e+01
After 2 iterations, simplex is given by:
xL= 3.346 -0.897, fL=1.305e-01
xM= 1.414 -1.414, fM=2.686e+00
xH= 1.932   0.518, fH=3.444e+00
After 3 iterations, simplex is given by:
xL= 3.346 -0.897, fL=1.305e-01
xM= 2.604 -1.992, fM=1.140e+00
xH= 1.414 -1.414, fH=2.686e+00
After 4 iterations, simplex is given by:
xL= 3.346 -0.897, fL=1.305e-01
xM= 3.756 -1.459, fM=7.820e-01
xH= 2.604 -1.992, fH=1.140e+00
After 5 iterations, simplex is given by:
xL= 3.346 -0.897, fL=1.305e-01
xM= 3.078 -1.585, fM=3.482e-01
xH= 3.756 -1.459, fH=7.820e-01
After 6 iterations, simplex is given by:
xL= 2.668 -1.022, fL=1.107e-01
xM= 3.346 -0.897, fM=1.305e-01
xH= 3.078 -1.585, fH=3.482e-01
After 7 iterations, simplex is given by:
xL= 3.042 -1.272, fL=7.586e-02
xM= 2.668 -1.022, fM=1.107e-01
xH= 3.346 -0.897, fH=1.305e-01
After 8 iterations, simplex is given by:
xL= 3.101 -1.022, fL=1.060e-02
xM= 3.042 -1.272, fM=7.586e-02
xH= 2.668 -1.022, fH=1.107e-01
After 9 iterations, simplex is given by:
xL= 3.101 -1.022, fL=1.060e-02
xM= 2.870 -1.085, fM=2.413e-02
xH= 3.042 -1.272, fH=7.586e-02
After 10 iterations, simplex is given by:
xL= 2.957 -0.944, fL=5.041e-03
xM= 3.101 -1.022, fM=1.060e-02
xH= 2.870 -1.085, fH=2.413e-02
After 11 iterations, simplex is given by:
xL= 2.949 -1.034, fL=3.722e-03
xM= 2.957 -0.944, fM=5.041e-03
xH= 3.101 -1.022, fH=1.060e-02
After 12 iterations, simplex is given by:
xL= 3.027 -1.005, fL=7.441e-04
xM= 2.949 -1.034, fM=3.722e-03

xH= 2.957 -0.944, fH=5.041e-03
After 13 iterations, simplex is given by:
xL= 3.027 -1.005, fL=7.441e-04
xM= 2.972 -0.982, fM=1.104e-03
xH= 2.949 -1.034, fH=3.722e-03
After 14 iterations, simplex is given by:
xL= 3.027 -1.005, fL=7.441e-04
xM= 2.974 -1.014, fM=8.438e-04
xH= 2.972 -0.982, fH=1.104e-03
NM simplex converged with tol = 1.000e-03 after 14 iterations
Minimum f=7.441e-04 at x= 3.027 -1.005

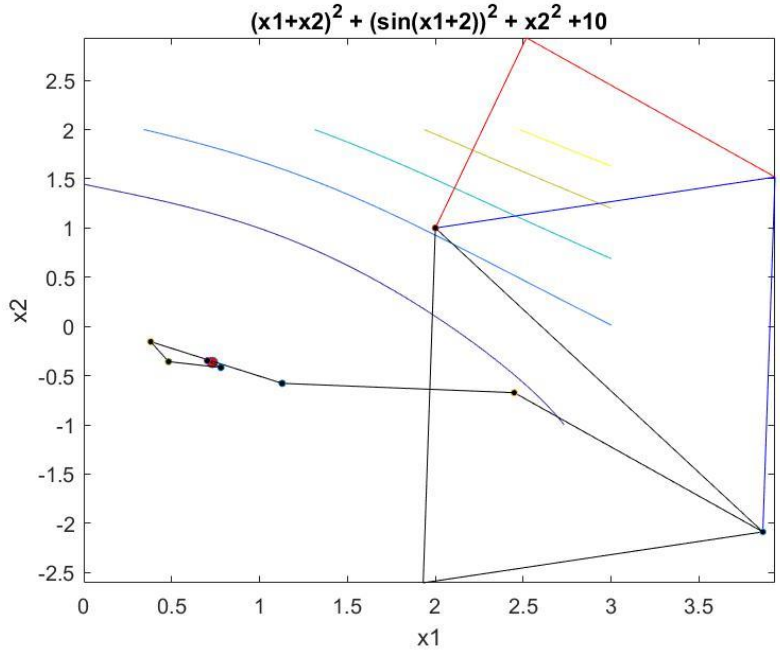


Plot of the Nelder-Mead Simplex solution from the calculations above.

Nelder-Mead Simplex Worked Example 3
Minimise $f(x_1, x_2) = (x_1 + x_2)^2 + sin^2(x_1 + 2) + x_2^2 + 10$ using the following parameters:
Starting point $x^0 = [x_1 \quad x_2] = [2.0 \quad 1.0]$
Initial step size c=2
$\alpha = 1.0, \quad \beta = 0.5, \quad \gamma = 2.0, \quad \rho = 0.5$
tol = 0.001.
Initial simplex
xL= 2.000 1.000, fL=2.057e+01
xM= 3.932 1.518, fM=4.212e+01
xH= 2.518 2.932, fH=4.926e+01
After 1 iterations, simplex is given by:
xL= 3.863 -2.087, fL=1.767e+01
xM= 2.000 1.000, fM=2.057e+01
xH= 3.932 1.518, fH=4.212e+01
After 2 iterations, simplex is given by:
xL= 3.863 -2.087, fL=1.767e+01
xM= 1.931 -2.605, fM=1.774e+01
xH= 2.000 1.000, fH=2.057e+01
After 3 iterations, simplex is given by:
xL= 2.448 -0.673, fL=1.454e+01
xM= 3.863 -2.087, fM=1.767e+01

xH= 1.931 -2.605, fH=1.774e+01
After 4 iterations, simplex is given by:
xL= 2.448 -0.673, fL=1.454e+01
xM= 2.543 -1.993, fM=1.524e+01
xH= 3.863 -2.087, fH=1.767e+01
After 5 iterations, simplex is given by:
xL= 1.129 -0.578, fL=1.064e+01
xM= 2.448 -0.673, fM=1.454e+01
xH= 2.543 -1.993, fH=1.524e+01
After 6 iterations, simplex is given by:
xL= 1.129 -0.578, fL=1.064e+01
xM= 1.034   0.741, fM=1.371e+01
xH= 2.448 -0.673, fH=1.454e+01
After 7 iterations, simplex is given by:
xL= 1.129 -0.578, fL=1.064e+01
xM= -0.285   0.836, fM=1.198e+01
xH= 1.034   0.741, fH=1.371e+01
After 8 iterations, simplex is given by:
xL= 1.129 -0.578, fL=1.064e+01
xM= -0.191 -0.484, fM=1.163e+01
xH= -0.285   0.836, fH=1.198e+01
After 9 iterations, simplex is given by:
xL= 1.129 -0.578, fL=1.064e+01
xM= 0.092   0.152, fM=1.084e+01
xH= -0.191 -0.484, fH=1.163e+01
After 10 iterations, simplex is given by:
xL= 1.129 -0.578, fL=1.064e+01
xM= 0.210 -0.348, fM=1.078e+01
xH= 0.092   0.152, fH=1.084e+01
After 11 iterations, simplex is given by:
xL= 0.381 -0.155, fL=1.055e+01
xM= 1.129 -0.578, fM=1.064e+01
xH= 0.210 -0.348, fH=1.078e+01
After 12 iterations, simplex is given by:
xL= 0.482 -0.358, fL=1.052e+01
xM= 0.381 -0.155, fM=1.055e+01
xH= 1.129 -0.578, fH=1.064e+01
After 13 iterations, simplex is given by:
xL= 0.780 -0.417, fL=1.043e+01
xM= 0.482 -0.358, fM=1.052e+01
xH= 0.381 -0.155, fH=1.055e+01
After 14 iterations, simplex is given by:
xL= 0.780 -0.417, fL=1.043e+01
xM= 0.882 -0.620, fM=1.052e+01
xH= 0.482 -0.358, fH=1.052e+01
After 15 iterations, simplex is given by:
xL= 0.780 -0.417, fL=1.043e+01
xM= 0.657 -0.438, fM=1.046e+01
xH= 0.882 -0.620, fH=1.052e+01
After 16 iterations, simplex is given by:
xL= 0.780 -0.417, fL=1.043e+01

xM= 0.637 -0.332, fM=1.044e+01
xH= 0.657 -0.438, fH=1.046e+01
After 17 iterations, simplex is given by:
xL= 0.780 -0.417, fL=1.043e+01
xM= 0.760 -0.311, fM=1.044e+01
xH= 0.637 -0.332, fH=1.044e+01
After 18 iterations, simplex is given by:
xL= 0.703 -0.348, fL=1.043e+01
xM= 0.780 -0.417, fM=1.043e+01
xH= 0.760 -0.311, fH=1.044e+01
After 19 iterations, simplex is given by:
xL= 0.703 -0.348, fL=1.043e+01
xM= 0.751 -0.347, fM=1.043e+01
xH= 0.780 -0.417, fH=1.043e+01
After 20 iterations, simplex is given by:
xL= 0.754 -0.382, fL=1.043e+01
xM= 0.703 -0.348, fM=1.043e+01
xH= 0.751 -0.347, fH=1.043e+01
NM simplex converged with tol = 1.000e-03 after 20 iterations
Minimum f=1.043e+01 at x= 0.754 -0.382



Plot of the Nelder-Mead Simplex solution from the calculations above.

3.7 Hooke-Jeeves Method

The Hooke-Jeeves method belongs to the class of search methods known as *pattern search*. Similarly, to the Nelder-Mead Simplex method, it carries out a pattern search of the design space without the need to calculate derivatives of the objective function, by only relying on the evaluation of the objective function at specific points. Although there is nothing stopping you from using this method on a function which can be differentiated!

The method requires two steps, an "*exploratory search*" to determine the best direction from the current location, and then a "*pattern move*" in that best direction.

3.7.1 Exploratory Search

The aim of this exploratory search step is to try to find a direction which improves the value of the objective function from the current point. In order to do this, the value of the current point is "*perturbed*" by a small amount (Δ) (called perturbation step) in the positive and negative direction, along each (design) variable, one at a time, and every time the objective function is evaluated to determine if the new point is better than the current point. At the end of each perturbation, the current point is replaced by the new point, provided it has a better objective function value.

Before carrying out the *exploratory search*, we need to specify the following four parameters:

1. The *initial* or *current point* about which the exploratory search will take place. This can take the form of the *current point* vector given by (3.37).

$$\mathbf{x}^0 = \left[x_1, x_2, \cdots, x_i, \cdots, x_n\right] \tag{3.37}$$

   where: *i* is the i^{th} design variable and *n* is the total number of design variables and also perturbation directions.

2. The size of the perturbation step along each direction, which is the same as saying the perturbation step for each design variable. This can take the form of the *perturbation vector* given by (5.38).

$$\mathbf{P}_0 = \left[\Delta x_1, \Delta x_2, \cdots, \Delta x_i, \cdots, \Delta x_n\right] \tag{3.38}$$

   where: Δx_i is the i^{th} design variable step size. Note that all perturbation step sizes are generally relatively small and do not have to be equal to each other.

3. The *step size reduction parameter* $\left(\eta > 1\right)$, typical values are: 2 or 10.

4. The *perturbation tolerance limit vector* $\left(\mathbf{T}\right)$, which defines the smallest possible perturbation for each design variable and which is used to stop the algorithm. This has the same form as the *perturbation vector* of (5.38) and is given by (3.39).

$$\mathbf{T} = \left[t_1, t_2, \cdots, t_i, \cdots, t_n\right] \tag{3.39}$$

The **<u>exploratory search steps</u>** are:

1. At the *current point* $\left(\mathbf{x}^0\right)$, calculate the objective function $f\left(\mathbf{x}^0\right)$ and copy these to the *best point vector* $\left(\mathbf{x}^{\text{best}}\right)$ and best function value $\left(f^{\text{best}}\right)$ respectively.
2. Copy the *perturbation vector* $\left(\mathbf{P}_0\right)$ into the *working perturbation vector* $\left(\mathbf{P}_w\right)$
3. Combine the *current point* vector $\left(\mathbf{x}^0\right)$ and the *working perturbation vector* $\left(\mathbf{P}_w\right)$ into the *search point vector* $\left(\mathbf{x}^1\right)$, which is of the form of (3.40).

$$\mathbf{x}_j^1 = \left[x_1 + \delta_{1j} d_k \Delta x_1, x_2 + \delta_{2j} d_k \Delta x_2, \cdots, x_i + \delta_{ij} d_k \Delta x_i, \cdots, x_n + \delta_{nj} d_k \Delta x_n\right] \tag{3.40}$$

where:

j: is the perturbation direction of the i^{h} design variable, starting with j = 1 up to n.

$x_{i(i=1 \text{ to } n)}$: are the design variables of the initial point from (3.37)

δ_{ij} : is the Kronecker delta given by equation (3.41), which allows to generate the perturbation step one design variable at a time

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \tag{3.41}$$

d_k : is the direction vector which allows the perturbation step to be carried out in the positive $(d_+ = 1)$ and negative $(d_- = -1)$ directions given by (3.42)

$$d_k = \begin{cases} 1 & \text{if } k = + \\ -1 & \text{if } \mathrm{k} = - \end{cases} \tag{3.42}$$

k: is the direction vector index which defines if the direction vector has a positive (k = +) or negative (k = –) perturbation direction

$\Delta x_{i(i=1 \text{ to } n)}$: are the perturbation step sizes from (3.38);

4. Set the perturbation direction to have the initial value of one $(j = 1)$ and set the direction vector index to positive $(k = +)$, so the search point vector goes from the format of equation (5.40) to that below:

$$\mathbf{x}_1^1 = \left[x_1 + d_k \Delta x_1, x_2, \cdots, x_i, \cdots, x_n\right]$$
$$\therefore \mathbf{x}_1^1 = \left[x_1 + \Delta x_1, x_2, \cdots, x_i, \cdots, x_n\right]$$

5. At the current *search point vector* $\left(\mathbf{x}_j^1\right)$, calculate the objective function $f\left(\mathbf{x}_j^1\right)$.

6. If $f\left(\mathbf{x}_j^1\right) < f^{best}$: Replace the best point with this one, such that $\mathbf{x}^{best} = \mathbf{x}_j^1$ and $f^{\text{best}} = f\left(\mathbf{x}_j^1\right)$, update *j* such that $(j = j + 1)$ and set the direction vector index to positive $(k = +)$ which then goes from the format of equation (3.40) to that below:

$$\mathbf{x}_{j+1}^1 = \left[x_1, x_2, \cdots, x_{j+1} + \delta_{j+1,j+1} d_+ \Delta x_{j+1}, \cdots, x_n\right]$$
$$\therefore \mathbf{x}_{j+1}^1 = \left[x_1, x_2, \cdots, x_{j+1} + \Delta x_{j+1}, \cdots, x_n\right]$$

However, if $(j > n)$ go to step 9, otherwise go to step 5.

7. If $f^{best} \leq f\left(\mathbf{x}_j^1\right)$ and the direction vector index is positive $(k = +)$, set the direction vector index to negative $(k = -)$, which then goes from the format of equation (5.40) to that below and go to step 5, otherwise go to step 8.

$$\mathbf{x}_j^1 = \left[x_1, x_2, \cdots, x_j + \delta_{jj} d_- \Delta x_j, \cdots, x_n\right]$$
$$\therefore \mathbf{x}_j^1 == \left[x_1, x_2, \cdots, x_j - \Delta x_j, \cdots, x_n\right]$$

8. If $f^{best} \leq f\left(\mathbf{x}_j^1\right)$ and the direction vector index is negative $(k = -)$, this perturbation has not improved the objective function, so it can be discarded. Update *j*, such that $(j = j + 1)$, set the direction vector index to positive $(k = +)$ and go to step 5, but if $(j > n)$ go to step 9.

9. Determine if the exploratory search has succeeded or failed.

   i) If $f^{best} < f\left(\mathbf{x}^0\right)$, the exploratory search has succeeded, so now go to step 11.

ii) Else if $f^{best} = f\left(\mathbf{x}^0\right)$ and $\mathbf{x}^{\text{best}} = \mathbf{x}^0$ the exploratory search has failed as a better value of the objective function could not be found. The following now needs to happen:

a) Reduce the size of the *working perturbation vector* $\left(\mathbf{P}_w\right)$ using (3.43).

$$\mathbf{P}_W = \frac{\mathbf{P}_W}{\eta} \tag{3.43}$$

b) If the *working perturbation vector* $\left(\mathbf{P}_w\right)$ is greater or equal to the *perturbation tolerance limit vector* $\left(\mathbf{T}\right)$, that is $\left(\mathbf{P}_w \geq \mathbf{T}\right)$, then go to step 3

c) If, however, the *working perturbation vector* $\left(\mathbf{P}_w\right)$ is less than the *perturbation tolerance limit vector* $\left(\mathbf{T}\right)$, that is $\left(\mathbf{P}_w < \mathbf{T}\right)$, go to step 10.

10. As the solution could not be improved any further, the final solution to the problem $\left(\mathbf{x}^{\text{best}}\right)$ and $\left(f^{\text{best}}\right)$ are given the *initial values of the problem* $\left(\mathbf{x}^0\right)$ and $f\left(\mathbf{x}^0\right)$. Then go to step 11..
11. **Exit** the *exploratory search* step.

3.7.2 Pattern Move

The *pattern move* step uses the *initial* or *current point* $\left(\mathbf{x}^0\right)$ and the *best point* $\left(\mathbf{x}^{\text{best}}\right)$ with an objective function value less than the current point $f^{best} < f\left(\mathbf{x}^0\right)$ in order to move in an improving direction. A *new point* $\left(\mathbf{x}^2\right)$ is created by moving from $\left(\mathbf{x}^0\right)$ to $\left(\mathbf{x}^2\right)$ using (3.44).

$$\mathbf{x}^2 = \mathbf{x}^0 + a\left(\mathbf{x}^{Best} - \mathbf{x}^0\right) \tag{3.44}$$

where: $\left(\mathbf{x}^{Best} - \mathbf{x}^0\right)$ is the *improving direction vector* and $\left(a\right)$ is a positive *accelerator factor*, which extends the length of direction vector. A typical value of the accelerator factor is two $\left(a = 2\right)$

3.7.3 The Hooke-Jeeves Pattern Search Algorithm

The Hooke-Jeeves Pattern Search Algorithm requires the following five parameters, four of which were specified in Exploratory Search. These are:

1) A starting point vector $\left(\mathbf{x}^0\right)$.
2) A perturbation step size vector $\left(\mathbf{P}_0\right)$.
3) The *perturbation tolerance limit vector* $\left(\mathbf{T}\right)$,
4) The *step size reduction parameter* $\left(\eta\right)$, and
5) The acceleration factor $\left(a\right)$

The seven steps for the Hooke-Jeeves Pattern Search Algorithm are as follow:

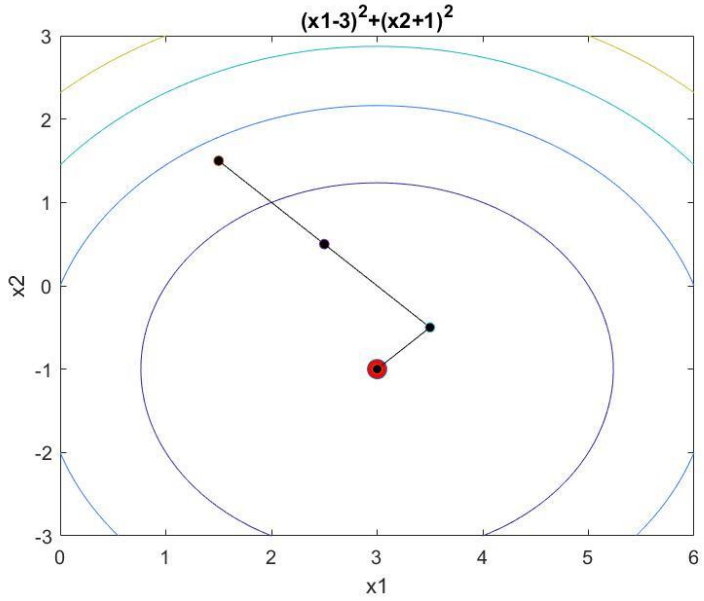
1) Specify the five parameters required for the algorithm to work: $\left(\mathbf{x}^0, \mathbf{P}^0, \mathbf{T}, \eta, a\right)$
2) Carry out an exploratory search around $\left(\mathbf{x}^0\right)$.
3) If the solution from the exploratory search is the same as the *initial problem* $f^{best} = f\left(\mathbf{x}^0\right)$, then this solution is the optimum. Go to step 7.
4) If the solution from the exploratory search is better than that of the *initial problem* $f^{best} < f\left(\mathbf{x}^0\right)$, then carry out the *Pattern Move step* and calculate $f\left(\mathbf{x}^2\right)$.
5) If $f^{Best} < f\left(\mathbf{x}^2\right)$, then copy the *best point* $\left(\mathbf{x}^{\text{best}}\right)$ to the *current point* $\left(\mathbf{x}^0\right)$ and go to step 2.
6) If $f\left(\mathbf{x}^2\right) \leq f^{Best}$, then copy the *new point* $\left(\mathbf{x}^2\right)$ to the *current point* $\left(\mathbf{x}^0\right)$ and go to step 2.
7) **Exit** the algorithm.

Hooke-Jeeves Worked Example 1
Minimise $f(x_1, x_2) = (x_1 - 3)^2 + (x_2 + 1)^2$ using the following parameters:

Starting point $x^0 = [x_1 \;\; x_2] = [1.5 \;\; 1.5]$
Perturbation step size $P_0 = [\Delta x_1 \; \Delta x_2] = [0.5 \;\; 0.5]$
Perturbation tolerance limit $T = [t_1 \; t_2] = [0.025 \;\; 0.025]$
Exploration accelerator factor a = 2
Step size reduction parameter η=2

Call 1 to exploratory search increment = 0.50000
Start x0 = 1.500000 1.500000 f0 = 8.500000
x = 2.000000 1.500000 f = 7.250000
x = 2.000000 2.000000 f = 10.000000
x = 2.000000 1.000000 f = 5.000000
Pattern search x2 = 2.500000 0.500000 f2 = 2.500000
Call 2 to exploratory search increment = 0.50000
Start x0 = 2.500000 0.500000 f0 = 2.500000
x = 3.000000 0.500000 f = 2.250000
x = 3.000000 1.000000 f = 4.000000
x = 3.000000 0.000000 f = 1.000000
Pattern search x2 = 3.500000 -0.500000 f2 = 0.500000
Call 3 to exploratory search increment = 0.50000
Start x0 = 3.500000 -0.500000 f0 = 0.500000
x = 4.000000 -0.500000 f = 1.250000
x = 3.000000 -0.500000 f = 0.250000
x = 3.000000 0.000000 f = 1.000000
x = 3.000000 -1.000000 f = 0.000000
Pattern search x2 = 2.500000 -1.500000 f2 = 0.500000
Call 4 to exploratory search increment = 0.50000
Start x0 = 3.000000 -1.000000 f0 = 0.000000
x = 3.500000 -1.000000 f = 0.250000
x = 2.500000 -1.000000 f = 0.250000
x = 3.000000 -0.500000 f = 0.250000
x = 3.000000 -1.500000 f = 0.250000
search increment reduced to 0.25000

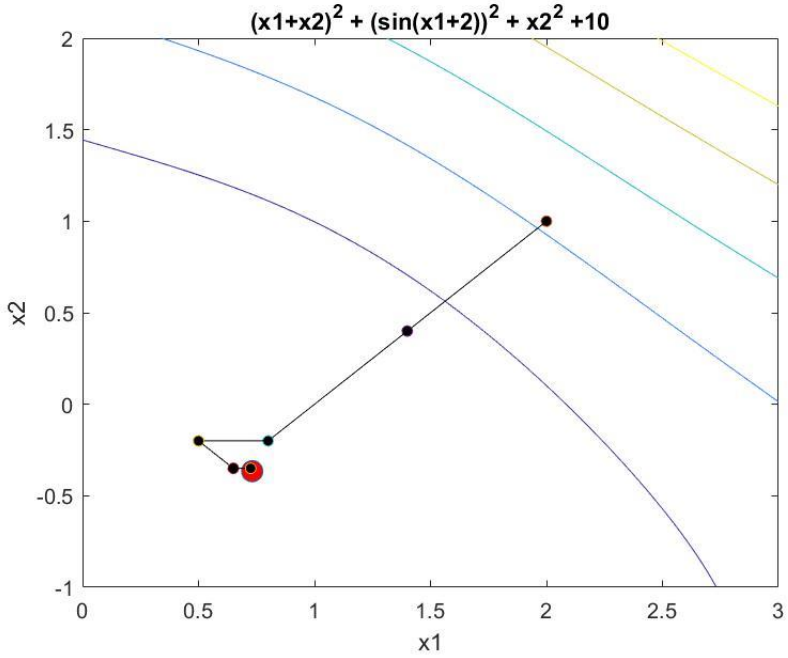
Call 5 to exploratory search increment = 0.25000
Start x0 = 3.000000 -1.000000 f0 = 0.000000
x = 3.250000 -1.000000 f = 0.062500
x = 2.750000 -1.000000 f = 0.062500
x = 3.000000 -0.750000 f = 0.062500
x = 3.000000 -1.250000 f = 0.062500
search increment reduced to 0.12500
search completed - increment 0.12500 below tolerance 0.25000
search completed xbest= 3.00000 -1.00000 fbest = 0.00000



Plot of the Hooke-Jeeves solution from the calculations above.

Hookes-Jeeves Worked Example 2
Minimise $f(x_1, x_2) = (x_1 + x_2)^2 + sin^2(x_1 + 2) + x_2^2 + 10$ using the following parameters:
Starting point $x^0 = [x_1 \;\; x_2] = [2.0 \;\; 1.0]$
Perturbation step size $P_0 = [\Delta x_1 \; \Delta x_2] = [0.3 \;\; 0.3]$
Perturbation tolerance limit $T = [t_1 \; t_2] = [0.025 \;\; 0.025]$
Exploration accelerator factor a = 2
Step size reduction parameter η=2
Call 1 to exploratory search increment = 0.30000
Start x0 = 2.000000 1.000000 f0 = 20.572750
x = 2.300000 1.000000 f = 22.729360
x = 1.700000 1.000000 f = 18.570726
x = 1.700000 1.300000 f = 20.970726
x = 1.700000 0.700000 f = 16.530726
Pattern search x2 = 1.400000 0.400000 f2 = 13.465301
Call 2 to exploratory search increment = 0.30000
Start x0 = 1.400000 0.400000 f0 = 13.465301
x = 1.700000 0.400000 f = 14.850726
x = 1.100000 0.400000 f = 12.411729
x = 1.100000 0.700000 f = 13.731729
x = 1.100000 0.100000 f = 11.451729
Pattern search x2 = 0.800000 -0.200000 f2 = 10.512217
Call 3 to exploratory search increment = 0.30000
Start x0 = 0.800000 -0.200000 f0 = 10.512217

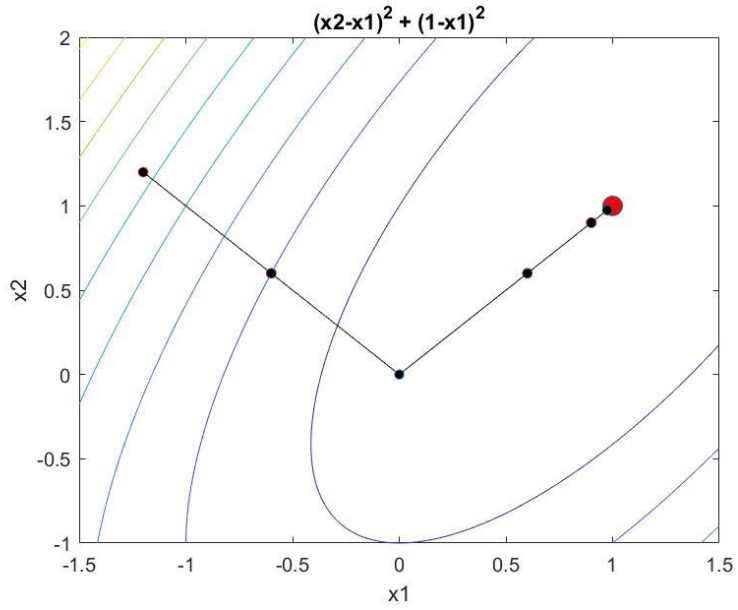
```
x =   1.100000  -0.200000 f =  10.851729
x =   0.500000  -0.200000 f =  10.488169
x =   0.500000   0.100000 f =  10.728169
x =   0.500000  -0.500000 f =  10.608169
Pattern search x2 =   0.200000  -0.200000 f2 =  10.693666
Call     4 to exploratory search increment =    0.30000
Start x0 =   0.500000  -0.200000 f0 =  10.488169
x =   0.800000  -0.200000 f =  10.512217
x =   0.200000  -0.200000 f =  10.693666
x =   0.500000   0.100000 f =  10.728169
x =   0.500000  -0.500000 f =  10.608169
search increment reduced to    0.15000
Call     5 to exploratory search increment =    0.15000
Start x0 =   0.500000  -0.200000 f0 =  10.488169
x =   0.650000  -0.200000 f =  10.465313
x =   0.650000  -0.050000 f =  10.585313
x =   0.650000  -0.350000 f =  10.435313
Pattern search x2 =   0.800000  -0.500000 f2 =  10.452217
Call     6 to exploratory search increment =    0.15000
Start x0 =   0.650000  -0.350000 f0 =  10.435313
x =   0.800000  -0.350000 f =  10.437217
x =   0.500000  -0.350000 f =  10.503169
x =   0.650000  -0.200000 f =  10.465313
x =   0.650000  -0.500000 f =  10.495313
search increment reduced to    0.07500
Call     7 to exploratory search increment =    0.07500
Start x0 =   0.650000  -0.350000 f0 =  10.435313
x =   0.725000  -0.350000 f =  10.426864
x =   0.725000  -0.275000 f =  10.441864
x =   0.725000  -0.425000 f =  10.434364
Pattern search x2 =   0.800000  -0.350000 f2 =  10.437217
Call     8 to exploratory search increment =    0.07500
Start x0 =   0.725000  -0.350000 f0 =  10.426864
x =   0.800000  -0.350000 f =  10.437217
x =   0.650000  -0.350000 f =  10.435313
x =   0.725000  -0.275000 f =  10.441864
x =   0.725000  -0.425000 f =  10.434364
search increment reduced to    0.03750
Call     9 to exploratory search increment =    0.03750
Start x0 =   0.725000  -0.350000 f0 =  10.426864
x =   0.762500  -0.350000 f =  10.429614
x =   0.687500  -0.350000 f =  10.428818
x =   0.725000  -0.312500 f =  10.431552
x =   0.725000  -0.387500 f =  10.427802
search increment reduced to    0.01875
search completed - increment    0.01875 below tolerance    0.02500
search completed xbest=    0.72500   -0.35000 fbest =   10.42686
```



Plot of the Hooke-Jeeves solution from the calculations above.

Hookes-Jeeves Worked Example 3
Minimise $f(x_1, x_2) = (x_2 - x_1)^2 + (1 - x_1)^2$ using the following parameters:
Starting point $x^0 = [x_1 \;\; x_2] = [-1.2 \;\; 1.2]$
Perturbation step size $P_0 = [\Delta x_1 \; \Delta x_2] = [0.3 \;\; 0.3]$
Perturbation tolerance limit $T = [t_1 \; t_2] = [0.05 \;\; 0.05]$
Exploration accelerator factor a = 2
Step size reduction parameter η=2
Call 1 to exploratory search increment = 0.30000
Start x0 = -1.200000 1.200000 f0 = 10.600000
x = -0.900000 1.200000 f = 8.020000
x = -0.900000 1.500000 f = 9.370000
x = -0.900000 0.900000 f = 6.850000
Pattern search x2 = -0.600000 0.600000 f2 = 4.000000
Call 2 to exploratory search increment = 0.30000
Start x0 = -0.600000 0.600000 f0 = 4.000000
x = -0.300000 0.600000 f = 2.500000
x = -0.300000 0.900000 f = 3.130000
x = -0.300000 0.300000 f = 2.050000
Pattern search x2 = 0.000000 -0.000000 f2 = 1.000000
Call 3 to exploratory search increment = 0.30000
Start x0 = 0.000000 -0.000000 f0 = 1.000000
x = 0.300000 -0.000000 f = 0.580000
x = 0.300000 0.300000 f = 0.490000
Pattern search x2 = 0.600000 0.600000 f2 = 0.160000
Call 4 to exploratory search increment = 0.30000
Start x0 = 0.600000 0.600000 f0 = 0.160000
x = 0.900000 0.600000 f = 0.100000
x = 0.900000 0.900000 f = 0.010000
Pattern search x2 = 1.200000 1.200000 f2 = 0.040000
Call 5 to exploratory search increment = 0.30000
Start x0 = 0.900000 0.900000 f0 = 0.010000

```
x =   1.200000   0.900000 f =   0.130000
x =   0.600000   0.900000 f =   0.250000
x =   0.900000   1.200000 f =   0.100000
x =   0.900000   0.600000 f =   0.100000
search increment reduced to    0.15000
Call     6 to exploratory search increment =    0.15000
Start x0 =   0.900000   0.900000 f0 =   0.010000
x =   1.050000   0.900000 f =   0.025000
x =   0.750000   0.900000 f =   0.085000
x =   0.900000   1.050000 f =   0.032500
x =   0.900000   0.750000 f =   0.032500
search increment reduced to    0.07500
Call     7 to exploratory search increment =    0.07500
Start x0 =   0.900000   0.900000 f0 =   0.010000
x =   0.975000   0.900000 f =   0.006250
x =   0.975000   0.975000 f =   0.000625
Pattern search x2 =   1.050000   1.050000 f2 =   0.002500
Call     8 to exploratory search increment =    0.07500
Start x0 =   0.975000   0.975000 f0 =   0.000625
x =   1.050000   0.975000 f =   0.008125
x =   0.900000   0.975000 f =   0.015625
x =   0.975000   1.050000 f =   0.006250
x =   0.975000   0.900000 f =   0.006250
search increment reduced to    0.03750
search completed - increment    0.03750 below tolerance    0.05000
search completed xbest=    0.97500    0.97500 fbest =    0.00062
```



Plot of the Hooke-Jeeves solution from the calculations above.

Chapter 4

Search Methods of Optimization

4.0 Introduction

There are many search methods which can be used to find an optimum. In this course, he following methods will be introduced:

1. Ant Colony Optimization (ACO)
2. Differential Evolution (DE) Algorithm
3. Genetic Algorithms (GA) – Excel Add-in provided
4. Particle Swarm Optimization (PSO) – Excel Add-in provided
5. Random Search Method – can just use Excel, very simple!
6. Simulated Annealing (SA)

4.1 Random Search Method

Pure Random Search is the simplest stochastic[1] method for global optimization, and most other stochastic methods are variations of it. Be aware that ***it is very inefficient!***

Pure random search consists only of a global phase of two steps:

1) Evaluate ***f(x)*** at *N* sample points from a random uniform distribution over the set S_b.
2) The smallest function value found is the candidate global minimum for *f(x)*.

Pure random search is asymptotically guaranteed to converge, in a probabilistic sense, to the global minimum point. It is quite inefficient because of the large number of function evaluations required to provide such a guarantee.

A simple extension of the method is so-called *single start*. In *single start*, a single local search is performed (if the problem is continuous) starting from the best point in the sample set at the end of pure random search.

4.1.1 Multistart Method

The Multistart method is one of several extensions of pure random search where a local phase is added to the global phase to improve efficiency. In Multistart, each sample point is used as a starting point for the local minimization procedure. The best local minimum point found is a candidate for the global minimum X_G^*. The method is reliable, but it is not efficient since many sample points will lead to the same local minimum.

[1]Stochastic optimization (SO) methods are those that use random numbers for their operations.

The algorithm consists of three simple steps:

1) Take a random point *x(0)* from a uniform distribution over the set S_b.
2) Start a local minimization procedure from *x(0)*.
3) Return to Step 1 unless a stopping criterion is satisfied.

Once the stopping criterion is satisfied, the local minimum with the smallest function value is taken as the global minimum x_G^*.

To calculate the value of a design variable from a random number using the upper and lower limits for the design variable, use equation (6.1).

$$x_i = x_{iL} + r_i \left(x_{iU} - x_{iL} \right); \quad i = 1 \, to \, n \tag{4.1}$$

where:

L: lower limit;
U: upper limit,
i: i^{th} design point;
0: 0^{th} generation;
r_i: uniformly distributed random number between 0 and 1

Please note: Equation (4.1) is used by all stochastic methods to calculate the value of the design variables from random numbers.

Random Search Example:

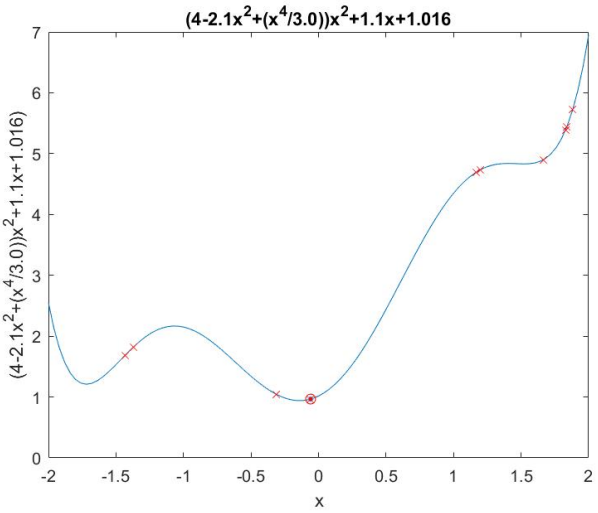
Use the Random Search Method to solve the minimization of the following function after **10** full iterations of the algorithm, between the limits of -2 < x < 2.

$$f(x) = \left(4 - 2.1x^2 + \frac{x^4}{3} \right) x^2 + 1.1x + 1.0164$$

The random points and optimum solution are given by:

For Random Search with 10 points and xmin= -2.0000 xmax = 2.0000
For random point i xrand= 0.1576 x= -1.3695 obj= 1.8241
For random point i xrand= 0.9706 x= 1.8824 obj= 5.7235
For random point i xrand= 0.9572 x= 1.8287 obj= 5.3856
For random point i xrand= 0.4854 x= -0.0585 obj= 0.9657
For random point i xrand= 0.8003 x= 1.2011 obj= 4.7385
For random point i xrand= 0.1419 x= -1.4325 obj= 1.6864
For random point i xrand= 0.4218 x= -0.3130 obj= 1.0441
For random point i xrand= 0.9157 x= 1.6629 obj= 4.8970
For random point i xrand= 0.7922 x= 1.1688 obj= 4.6973
For random point i xrand= 0.9595 x= 1.8380 obj= 5.4361
Optimum from Random Search with 10 random points x= -0.0585 obj= 0.9657.

These are shown on the following figure:



NOTE: since the random search is stochastic, these points will change every time the algorithm is run.

4.2 Simulated Annealing (SA)

Simulated annealing (SA) is a stochastic approach for locating a good approximation to the global minimum of a function. The name comes from the annealing process in metallurgy, which involves heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. At high temperatures, the atoms become loose from their initial configuration and move randomly to reach a configuration having absolute minimum energy. The cooling process should be slow, and enough time needs to be spent at each temperature, giving more chance for the atoms to find configurations of lower internal energy. If the temperature is not lowered slowly and enough time is not spent at each temperature, the process can become trapped in a local minimum for the internal energy. The resulting crystal may have many defects or the material may even become glass with no crystalline order.

The Simulated Annealing method for optimization of systems emulates this process. Given a long enough time to run, an algorithm based on this concept finds global minima for continuous-discrete-integer variable nonlinear programming problems.

The basic procedure is to generate random points in the neighbourhood of the current best point and evaluate the problem functions there. If the cost (or penalty) function value is smaller than its current best value, the point is accepted and this becomes best function value. If the function value is higher than the best value known so far, the point is sometimes accepted and sometimes rejected.

The point's acceptance is based on the value of the probability density function of the Bolzman-Gibbs distribution. If this probability density function has a value greater than a random number, then the trial point is accepted as the best solution even if its function value is higher.

A parameter called the Temperature (T) is used to calculate the probability density function. For the optimization problem, this temperature can be a target value for the optimum value of the cost function. Initially, a larger target value is selected. As the trials progress, the target value (temperature) is reduced (called the cooling schedule), and the process is terminated after a large number of trials.

The acceptance probability steadily decreases to zero as the temperature is reduced. So, in the initial stages, the method sometimes accepts worse designs, while in the final stages the worse designs are almost always rejected. This strategy avoids getting trapped at a local minimum point. The SA method requires evaluation of a cost and constraint functions only. Continuity and differentiability of functions are not required. So the method can be useful for non-differentiable problems, and problems where gradients cannot be calculated or are too expensive to calculate.

The (SA) algorithm is simple and easy to program. The following five steps give the basic ideas of the algorithm:

1) Choose an initial temperature $T^{(1)}$ (Section 4.2.1) and a feasible trial point $\left(x^{(1)}\right)$. Compute $f\left(x^{(1)}\right)$. Select a limit on the number of iterations *(M)* to reach the expected minimum value. Initialize the iteration counter *(k = 1)*.
2) Generate a new point $x^{(k+1)}$ randomly in a neighbourhood of the current point $x^{(1)}$ using (4.2). If the point is infeasible, generate another random point until feasibility is satisfied. Calculate $f\left(x^{(k+1)}\right)$ and $\Delta f = f\left(x^{(k+1)}\right) - f\left(x^{(1)}\right)$.

$$x_i^{(k+1)} = x_i^{(1)} + \left[\left(x_i^{\max} - x_i^{(1)}\right) r_i^{(k)} - \left(x_i^{(1)} - x_i^{\min}\right) s_i^{(k)}\right] \frac{T^{(k)}}{T^{(1)}} \qquad (4.2)$$

where: *i* is the *i*th design variable in the range $(i = 1,...,n)$, *n* is the number of design variables, $r_i^{(k)}, s_i^{(k)}$ are random numbers for the *i*th design variable in the *k*th iteration in the range $[0,1]$

3) If $\Delta f < 0$ then accept $x^{(k+1)}$ as the new best point $\left(x^{(1)}\right)$, set $f\left(x^{(1)}\right) = f\left(x^{(k+1)}\right)$ go to Step 4. Otherwise, calculate the probability density function (6.3).Generate a random number *(z)* uniformly distributed in [0,1]. If $z < p\left(\Delta f\right)$, then accept $x^{(k+1)}$ as the new best point $x^{(1)}$ and go to Step 4. Otherwise go to Step 2.

$$p\left(\Delta f\right) = \exp\left(\frac{-\Delta f}{T^{(k)}}\right) = e^{\left(\frac{-\Delta f}{T^{(k)}}\right)} \qquad (4.3)$$

4) If k < M, then k = k+ 1 and go to Step 5, else if k > M and one of the stopping criteria explained below is satisfied, then stop.
5) Update the temperature $T^{(k)}$, (Section 4.2.2); go to Step 2.

In order to implement this algorithm, the following three points need to be considered:

1) In Step 2 only one point is generated at a time within a certain neighbourhood of the current point. Thus, although SA randomly generates design points without the need for function or gradient information, it is not a pure random search within the entire design space. At the early stage, a new point can be located far away from the current point to speed up the search process and to avoid being trapped at a local minimum point. Once the temperature gets low, the new point is usually created nearby in order to focus on the local area. This can be controlled by defining a step size procedure.
2) In Step 2, the newly generated point needs to be feasible. If it is not, another point is generated until a point in the feasible region is obtained. Another method for treating constraints is to use the penalty function approach; that is, the constrained problem is converted to an unconstrained one. The cost function is replaced by the penalty function in the algorithm. Therefore, the feasibility requirements are not imposed explicitly in Step 2.
3) The following stopping criteria are suggested for Step 4:
    a. The algorithm stops if change in the best function value is less than some specified value for the last *j* number of consecutive iterations.
    b. The algorithm stops if *k* reaches a specified number of iterations by the user.

4.2.1 Selecting the Initial Temperatures $T^{(1)}$

A suitable initial temperature is one that results in an acceptance probability of value close to 1, which means that there is an almost 100% chance that a change which increases the objective function will be accepted. The value of initial temperature will clearly depend on the objective function and, hence, be problem-specific. It can be estimated by conducting an initial search in which all increases are accepted (i.e., the fixed number of iterations of simulated annealing in which all perturbed solutions are unconditionally accepted) and calculating the maximum objective increase observed Δf. Then, the initial temperature $T^{(1)}$ is given by (4.4).

$$T^{(1)} = \frac{-\Delta f}{\ln(p)} \tag{4.4}$$

where: *p* is a probability close to 1 (e.g. 0.8 – 0.9).

4.2.2 Decreasing the Temperature $T^{(k)}$

In the SA algorithm, the temperature is decreased gradually such that (4.5) and (4.6) are satisfied.

$$T^{(k)} > 0 \tag{4.5}$$

$$\lim_{k\to\infty} T^{(k)} = 0 \tag{4.6}$$

There is always a compromise between the quality of the obtained solutions and the speed of the cooling scheme. If the temperature is decreased slowly, better solutions are obtained but with a more significant computation time. The temperature T can be updated using one of four different schemes: 1) Linear, 2) Geometric, 3) Logarithmic, and 4) Modified logarithmic. These are explained next.

4.2.2.1 Linear Temperature Update Scheme

The Linear temperature update scheme consists of (4.7), where the temperature T is

$$T^{(k)} = T^{(1)} - (k-1)\Delta T \tag{4.7}$$

where: ΔT is a specified constant value which decreases the temperature equally in each iteration, *k* is the iteration number.

The value of ΔT can be calculated with (4.8), where M is the number of trials (iterations)

$$\Delta T = \frac{T^{(1)} - T^{(Final)}}{M} \tag{4.8}$$

4.2.2.2 Geometric Temperature Update Scheme

The Geometric temperature update scheme consists of equation (6.9) where the temperature in each iteration is a multiple of the previous temperature.

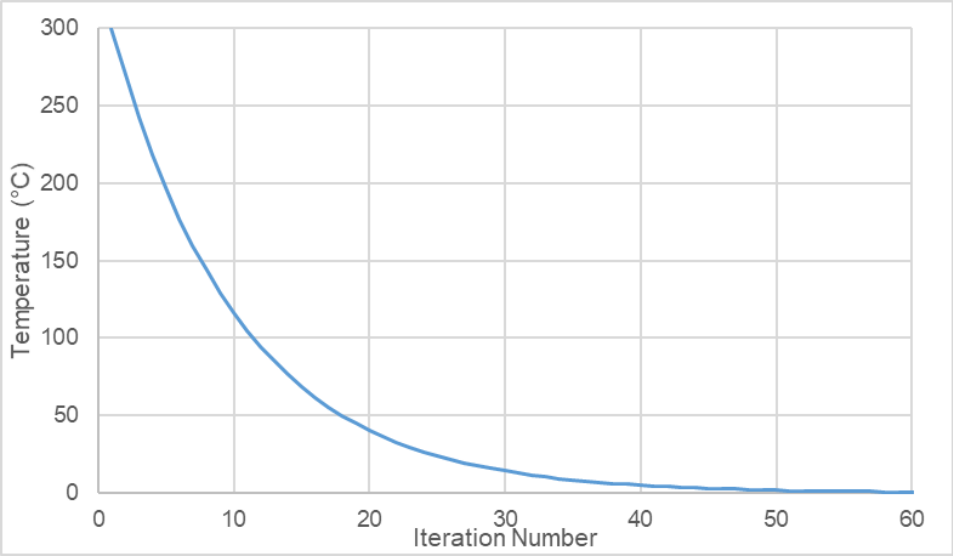


Figure 4.1: Temperature change during the search process for $T^{(1)}$ = 300 °C and α =0.9.

$$T^{(k+1)} = \alpha T^{(k)} \tag{4.9}$$

where α is a value between 0 and 1. The smaller the value of α the faster that the temperature reaches 0, the larger the value of α the more number of iterations before reaching a solution. Figure 4.1 shows how the temperature decreases during the search process for an initial temperature of $T^{(1)}$ = 300 °C and the multiplier with a value of α =0.9.

4.2.2.3 Logarithmic Temperature Update Scheme

The Logarithmic temperature update scheme consists of (4.10), where the initial temperature is divided by the logarithm of the current iteration number (k).

$$T^{(k)} = \frac{T^{(1)}}{\ln(k)} or \frac{T^{(1)}}{\log(k)} \tag{4.10}$$

This scheme is too slow to be applied in practice but has been proven to have the property of convergence to a global optimum.

Figure 4.2 shows how the temperature decreases during the search process for an initial temperature of T_0 = 300 °C and 20,000 iterations.

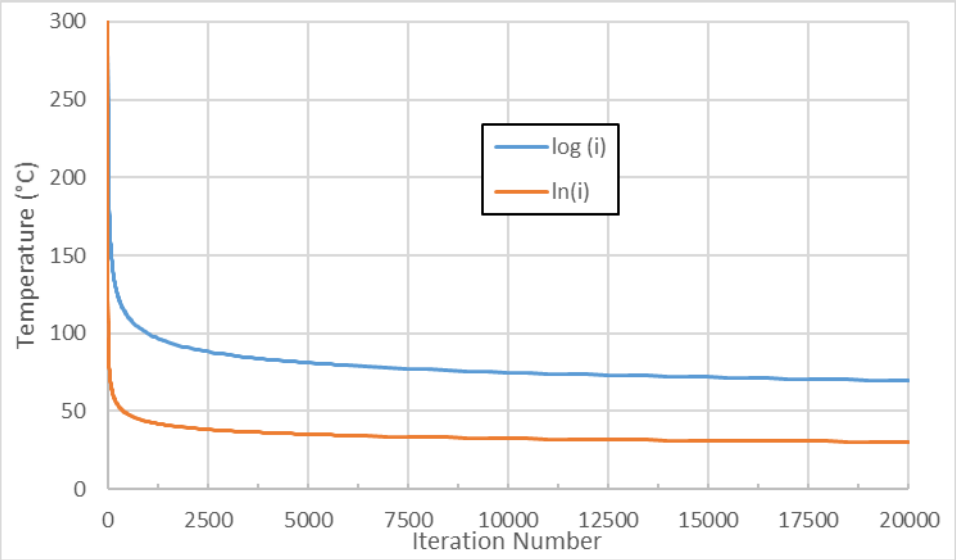


Figure 4.2: Temperature change during the search process for $T^{(1)}$ = 300 °C and 20,000 iterations.

4.2.2.4 Modified Logarithmic Temperature Update Scheme

The main trade-off in a cooling scheme is the use of a large number of iterations at a few temperatures or a small number of iterations at many temperatures. The Modified logarithmic temperature update scheme consists of (4.11), which is a very slow decreasing function, Figure 4.3.

$$T^{(k+1)} = \frac{T^{(k)}}{1+\alpha T^{(k)}} \tag{4.11}$$

where α is a very small constant parameter with values of approximately $\alpha = 10^{-4}$. Figure 4.12 shows how the temperature decreases during the search process for an initial temperature of $T^{(1)}$ = 300 °C, $\alpha = 10^{-4}$ and 20,000 iterations.

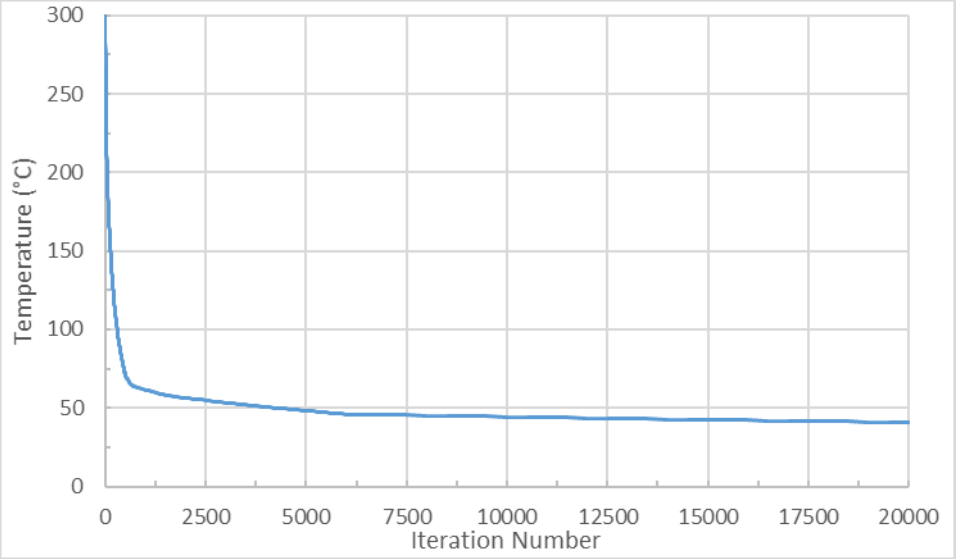


Figure 4.3: Temperature change during the search process for $T^{(1)}$ = 300 °C, $\alpha = 10^{-4}$ and 20,000 iterations.

Simulated Annealing Worked Example

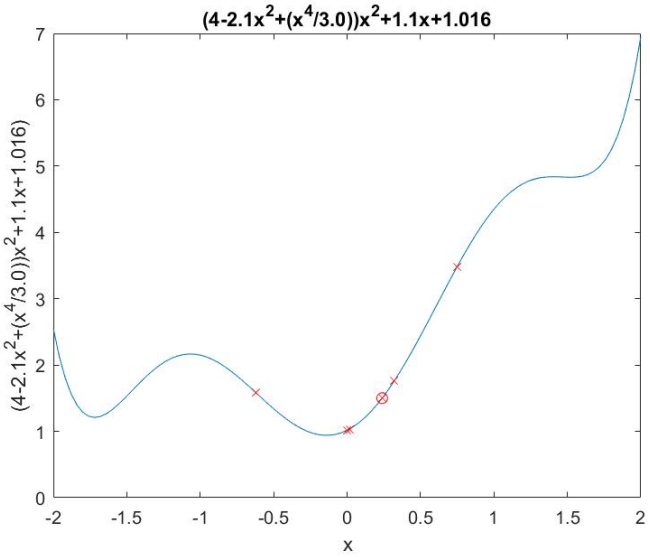
The aim is to minimise $f(x) = \left(4 - 2.1x + \frac{x^3}{3}\right)x^2 + 1.1x + 1.0164$ in the x-interval $-2 < x < 2$ using 6 iterations of the SA algorithm.

Number	Random Values	Rand Used	k			f(x)		T_K
1	0.02850	1	1	x_1 =	-1.8860	1.6014		350
2	0.34773	2,3		x_2 =	-0.6213	1.5833		
3	0.75934			**Δf =**	**$f(x_2) - f(x_1)$=**	**-0.01816**	**<0**	**Accept**
4	0.73739							
5	0.86296		2	x_1 =	-0.6213	1.5833		300
6	0.69197	4,5		x_3 =	0.01569	1.0346		
7	0.69055			**Δf =**	**$f(x_3) - f(x_1)$=**	**-0.54862**	**<0**	**Accept**
8	0.72874							
9	0.44761		3	x_1 =	0.01569	1.0346		250
10	0.00820	6,7		x_4 =	0.00222	1.0189		
11	0.80004			**Δf =**	**$f(x_4) - f(x_1)$=**	**-0.01577**	**<0**	**Accept**
12	0.15108							
13	0.22558		4	x_1 =	0.0022	1.0189		200
14	0.45510	8,9		x_5 =	0.3220	1.7632		
15	0.85492			**Δf =**	**$f(x_5) - f(x_1)$=**	**0.744335**	**>0**	**Check**
					p(Δf) =	0.99629		
		10		Next Rand Number z =		0.0082		
					z < p(Δf)	**True**		**Accept**
			5	x_1 =	0.3220	1.7632		150
		11,12		x_6 =	0.7470	3.4742		
				Δf =	**$f(x_6) - f(x_1)$=**	**1.711002**	**>0**	**Check**
					p(Δf) =	0.98866		
		13		Next Rand Number z =		0.22558		
					z < p(Δf)	**True**		**Accept**
			6	x_1 =	0.7470	3.4742		100
		14,15		x_7 =	0.2389	1.5008		
				Δf =	**$f(x_7) - f(x_1)$=**	**-1.97338**	**<0**	**Accept**

The numerical solution is given by the following calculations:

```
Initial random number =  0.02850 xmin=-2.00000 xmax= 2.00000 initial x=-1.88600
Iteration   1 T=350.000 xmin=-2.000 xmax= 2.000
x1=-1.88600 f1= 1.60142
r= 0.34773 s= 0.75934 probrand= 0.00000
x2=-0.62129 f2= 1.58326
Iteration   2 T=300.000 xmin=-2.000 xmax= 2.000
x1=-0.62129 f1= 1.58326
r= 0.73739 s= 0.86296 probrand= 0.00000
x2= 0.01569 f2= 1.03464
Iteration   3 T=250.000 xmin=-2.000 xmax= 2.000
x1= 0.01569 f1= 1.03464
r= 0.69197 s= 0.69055 probrand= 0.00000
x2= 0.00222 f2= 1.01887
Iteration   4 T=200.000 xmin=-2.000 xmax= 2.000
x1= 0.00222 f1= 1.01887
r= 0.72874 s= 0.44761 probrand= 0.00820
x2= 0.32202 f2= 1.76320
Iteration   5 T=150.000 xmin=-2.000 xmax= 2.000
x1= 0.32202 f1= 1.76320
r= 0.80004 s= 0.15108 probrand= 0.22558
x2= 0.74701 f2= 3.47420
Iteration   6 T=100.000 xmin=-2.000 xmax= 2.000
x1= 0.74701 f1= 3.47420
r= 0.45510 s= 0.85492 probrand= 0.00000
x2= 0.23894 f2= 1.50082
SA search after   6 iterations completed: x=     0.23894 f =     1.50082
```

The solution is shown on the following figure:



4.3 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a population based search algorithm based on the simulation of the social behaviour of birds in a flock, Figure 4.4. PSO is stochastic and mimics the flock's behaviour as it adjusts its movement to avoid predators to seek food sources. Individual particles exchange information about their position, velocity and fitness. The position of each individual particle represents a candidate solution to the optimization problem.

The particle swarm optimisation algorithm (PSOA) uses a fitness function to evaluate the optimality of each solution.

Figure 4.4: Swarming of starlings

This function enables each solution to be compared and ranked against each other. If the objective function is required to be minimised, a solution with the smallest value will have a higher fitness value. The sharing of information between particles is fundamental to the PSOA as it offers an evolutionary advantage. The act of exchanging information influences the behaviour of the flock, which adapts by returning to regions of high fitness already discovered and searching for better positions with each time step. The real-valued particle swarm optimization method works like this:

Assume that the search space has (*d*) dimensions (*d is the number of design variables*). The i^{th} particle of a swarm with N_p particles can be represented by the *d-dimensional* position vector of (6.12).

$$X_i = (x_{i1}, x_{i2}, \cdots, x_{id}) \tag{4.12}$$

The velocity of the particle is denoted by the vector of (4.13).

$$V_i = (v_{i1}, v_{i2}, \cdots, v_{id}) \tag{4.13}$$

In order for PSO to work, it is also necessary to consider both the best-visited position of the i^{th} particle (4.14) and the best global position explored so far by the entire swarm (4.15).

$$P_{i,best} = (p_{i1}, p_{i2}, \cdots, p_{id}) \tag{4.14}$$

$$P_{g,best} = (p_{g1}, p_{g2}, \cdots, p_{gd}) \tag{4.15}$$

The velocity of the particle which is then used to calculate its position at the *(t + 1)* iteration is given by (4.16) and (4.17) respectively.

$$V_i(t+1) = wV_i(t) + c_1\varphi_1(P_{i,best} - X_i) + c_2\varphi_2(P_{g,best} - X_i) \tag{4.16}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{4.17}$$

where:

c_1: A positive constant called the cognitive parameter, a typical value is 0.72

c_2: A positive constant called the social parameter, a typical value is 1.44

φ_1: random variables with uniform distribution between 0 and 1

φ_2: random variables with uniform distribution between 0 and 1

w: is the inertia weight which shows the effect of the velocity vectors on the new vector. This value is generally constant for the entire optimization, but could be made to vary between two values with decreasing effect as the solution evolves using equation (6.18)

$$w_t^d = w_{\max} - \frac{(w_{\max} - w_{\min})t}{t_{\max}} \tag{4.18}$$

t: t^{th} iteration

t_{max}: Maximum number of iterations, specified by the user.

d: d^{th} design variable

$w_{\min}$: Minimum inertia value

$w_{\max}$ Maximum inertia value, used at the start of the optimization to allow a wide search space. The maximum inertia value should be less than ~50% of the design variable range:
$w_{\max} V \le 0.5 \times \left(x_d^U - x_d^L\right)$

w_t^d Actual inertia value at t^{th} iteration

An upper bound is placed on the velocity in all dimensions V_{max} .This limitation prevents the particle from moving too rapidly from one region in search space to another. This value is usually initialized as a function of the range of the problem. For example, if the range of all X_{ij} is [-50,50] then V_{max} is proportional to 50.

$P_{i,best}$ for each particle is updated in each iteration when a better position for the particle or for the whole swarm is obtained. PSO is driven by social interaction. Individuals (particles) within the swarm learn from each other, and based on the knowledge obtained then move to become similar to their “better” previously obtained position and also to their “better” neighbours. Individuals within a neighbourhood communicate with one other. Based on the communication of a particle within the swarm different neighbourhood topologies are defined. Each particle can communicate with every other individual, forming a fully connected social network. In this case each particle is attracted toward the best particle (best problem solution) found by any member of the entire swarm. Each particle therefore imitates the overall best particle. So the value of $P_{g,best}$ is updated when a new best position within the whole swarm is found.

The algorithm for the PSO consists of the following 7 steps:

1) Initialize the swarm X. The positions of the N_p particles are randomly initialized within the hypercube of feasible space.
2) Evaluate the performance F of each particle, using its current position X(t).
3) Compare the performance of each individual to its best performance so far. If $F(X_i) < F(P_{i,best})$ then $F(P_{i,best}) = F(X_i)$, $P_{i,best} = X_i$

4) Compare the performance of each particle to the global best particle. If $F(X_i) < F(P_{g,best})$ then $F(P_{g,best}) = F(X_i), P_{g,best} = X_i$
5) Change the velocity of the particle based on the value calculated using equation (1).
6) Move each particle to a new position using equation (2).
7) Go to step 2, and repeat until convergence.

Particle Swarm Optimisation Worked Example

The aim is to minimise $f(x) = \left(4 - 2.1x + \frac{x^3}{3}\right)x^2 + 1.1x + 1.0164$ in the x-interval $-2 < x < 2$ using the PSO algorithm. The following parameters are used with 3 iterations:

6

c_1	0.75	Cognitive Parameter C_1
c_2	1.5	Social Parameter C_2
N_p	2	Number of Particles
i_{max}	1	Max Number of Iterations
ω_x	1	Inertia Value
$v_{Initial}$	0.5	Initial velocity

Table 1: Random Numbers in the range of 0 to 1

Number	Random Values
1	0.457
2	0.818
3	0.285
4	0.373
5	0.609
6	0.251
7	0.678
8	0.778
9	0.964
10	0.073
11	0.429
12	0.572
13	0.219
14	0.891

The calculations result in:

PSO: c1= 0.750 c2= 1.500 Np= 2 Niter= 3 w_x= 1.000 v_initial= 0.500 xmin=-2.000 xmax= 2.000

Random numbers used throughout PSO calculations:

Particle initialisation: for particle 1 xrand= 0.45700, particle 2 xrand= 0.81800
For k=1 particle 1: thi1= 0.28500 thi2 = 0.37300 particle 2: thi1= 0.60900 thi2 = 0.25100
For k=2 particle 1: thi1= 0.67800 thi2 = 0.77800 particle 2: thi1= 0.96400 thi2 = 0.07300
For k=3 particle 1: thi1= 0.42900 thi2 = 0.57200 particle 2: thi1= 0.21900 thi2 = 0.89100

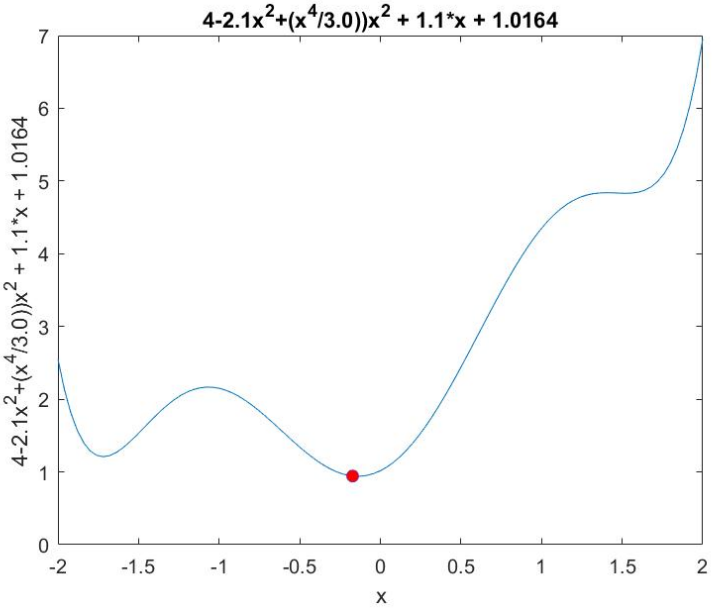
Iteration 1: Initial x1= -0.17200 x2= 1.27200 f1= 0.94371 f2= 4.80190

Initial x1best= -0.17200 f1best= 0.94371 x2best= 1.27200 f2best= 4.80190
Initial xgbest= -0.17200 fgbest= 0.94371
For particle 1 thi1= 0.28500 thi2= 0.37300 For particle 2 thi1= 0.60900 thi2= 0.25100
Initial veocity particle 1= 0.50000 particle 2= 0.50000

Start of iteration 2 xgbest= -0.17200 fgbest= 0.94371

Before iteration x1= -0.17200 x2= 1.27200 f1= 0.94371 f2= 4.80190
Before iteration x1best= -0.17200 f1best= 0.94371 x2best= 1.27200 f2best= 4.80190
For particle 1 thi1= 0.28500 thi2= 0.37300 For particle 2 thi1= 0.60900 thi2= 0.25100
After iteration: For particle 1 velocity= 0.50000 particle 2 velocity= -0.04367
After iteration x1= 0.32800 x2= 1.22833 f1= 1.78365 f2= 4.76708

```
After iteration x1best=  -0.17200 f1best=   0.94371 x2best=   1.22833 f2best=   4.76708
After iteration   2 xgbest=  -0.17200 fgbest=   0.94371
-------------------------------------------------------------

Start of iteration   3 xgbest=  -0.17200 fgbest=   0.94371
-------------------------------------------------------------
Before iteration x1=   0.32800 x2=   1.22833 f1=   1.78365 f2=   4.76708
Before iteration x1best=  -0.17200 f1best=   0.94371 x2best=   1.22833 f2best=   4.76708
For particle 1 thi1=   0.67800 thi2=   0.77800  For particle 2 thi1=   0.96400 thi2=   0.07300
After iteration: For particle 1 velocity=  -0.33775 particle 2 velocity=  -0.19700
After iteration x1=  -0.00975 x2=   1.03133 f1=   1.00606 f2=   4.43074
After iteration x1best=  -0.17200 f1best=   0.94371 x2best=   1.03133 f2best=   4.43074
After iteration   3 xgbest=  -0.17200 fgbest=   0.94371
-------------------------------------------------------------
 PS search completed xgbest=   -0.17200 fgbest =    0.94371
```



4.4 Differential Evolution (DE)

The differential evolution (DE) algorithm works with a population of designs. At each iteration (called generation), a new design is generated using some current designs and random operations. If the new design is better than a preselected parent design, then it replaces that design in the population; otherwise, the old design is kept and the process is repeated. Compared to genetic algorithms (GA), DE algorithms are easier to implement, and don't require binary number coding and encoding.

The basic DE algorithm consists of the following four steps, which will be explained next.

1) Generate an initial population of designs.
2) Mutation with difference of vectors to generate a donor design vector.
3) Crossover/recombination to generate a design vector.
4) Selection (accept/reject) the trial design vector using the fitness function.

4.4.1 Generation of the Initial Population

It is necessary to generate an initial population of *Np* design points. Np is usually a large number, between five (*5n*) and ten times (1*0n*): the number of design variables (*n*), where each design variable is called a chromosome. The initial designs are generated using equation (6.1), which for DE has the form of (6.19).

$$x_j^{(i,0)} = x_{jL} + r_{ij}\left(x_{jU} - x_{jL}\right), \quad j = 1 \, to \, n \tag{4.19}$$

where:

- *L:* lower limit;
- *U:* upper limit,
- *I:* ith design point;
- *0:* 0th generation;
- *J:* jth component of the population
- r_{ij}: Uniformly distributed random number between 0 and 1, generated for each component of the design point.

4.4.2 Generate a Donor Design

The donor design point is created by changing a design point from the current population. This change is done by combining the design vector with the difference between two other vectors of the population, all randomly selected. The generated design vector is called the donor design/vector. Mutation implies changing all components of a design vector. So, at the k^{th} generation, to generate the donor design vector, the following steps are required:

1) Randomly select three design points from the current population, represented by these 3 variables: $\left\{x^{(r1,k)}, x^{(r2,k)}, x^{(r3,k)}\right\}$: r_1, r_2, and r_3 are three different designs.
2) Select a fourth point $x^{(p,k)}$, called the parent/target design point; *p* means parent design.
3) Generate a Donor design vector using (6.20)

$$V^{(p,k)} = x^{(r1,k)} + F \times \left(x^{(r2,k)} - x^{(r3,k)}\right) \tag{4.20}$$

where:

- F: is a scale factor, with values typically between 0.4 and 1;
- $V^{(p,k)}$: Donor design vector at the k^{th} generation/iteration associated with p^{th} parent design

4.4.3 Crossover Operation to Generate Trial Design

The crossover operation is carried out using (4.21).

$$U_j^{(p,k)} = \begin{cases} V_j^{(p,k)}, & if \; r_{pj} \le Cr \; or \; j = j_r \\ x_j^{(p,k)}, & otherwise \end{cases}; \; j = 1 \, to \, n \tag{4.21}$$

where:

- r_{pj}: is a uniformly distributed random number between 0 and 1;
- j_r: is a randomly generated index between 1 and *n* that ensures that $U^{(p,\,k)}$ receives at least one component from $V^{(p,k)}$;
- Cr: Crossover rate (value of 0.9 commonly used).

4.4.4 Acceptance/Rejection of the Trial Design

Check if the trial design $U^{(p,k)}$ is better than the parent design $x^{(p,k)}$.If it is, it replaces the parent design (to keep population size constant). This is represented by (4.22).

$$x^{(p,k+1)} = \begin{cases} U^{(p,k)}, & \text{if } f\left(U^{(p,k)}\right) \le f\left(x^{(p,k)}\right) \\ x^{(p,k)}, & \text{otherwise} \end{cases} \tag{4.22}$$

If the cost function value for the trial design point is less than for the parent design, it replaces the parent design point in the next generation; otherwise, the parent design is retained. The population then gets better or remains the same but doesn't deteriorate. Note that the parent design is replaced by the trial design even if both produce the same cost function value. This allows the design vectors to move over a flat fitness landscape.

4.4.5 The DE Algorithm

The DE algorithm only requires three parameters: *Np, F,* and *Cr*, and its flow chart is given in Figure 4.5. The termination criteria consist of the following three steps:

1) Specify a limit k_{max} on the number of generations.
2) The best fitness/cost function value of the population does not change appreciably for several generations.
3) A specified value for the cost function is reached.

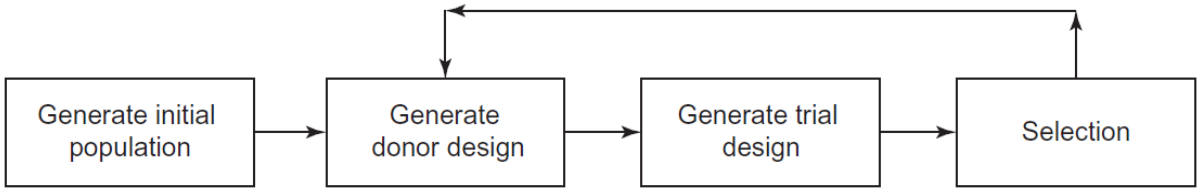


Figure 4.5: Main steps of the differential evolution algorithm..

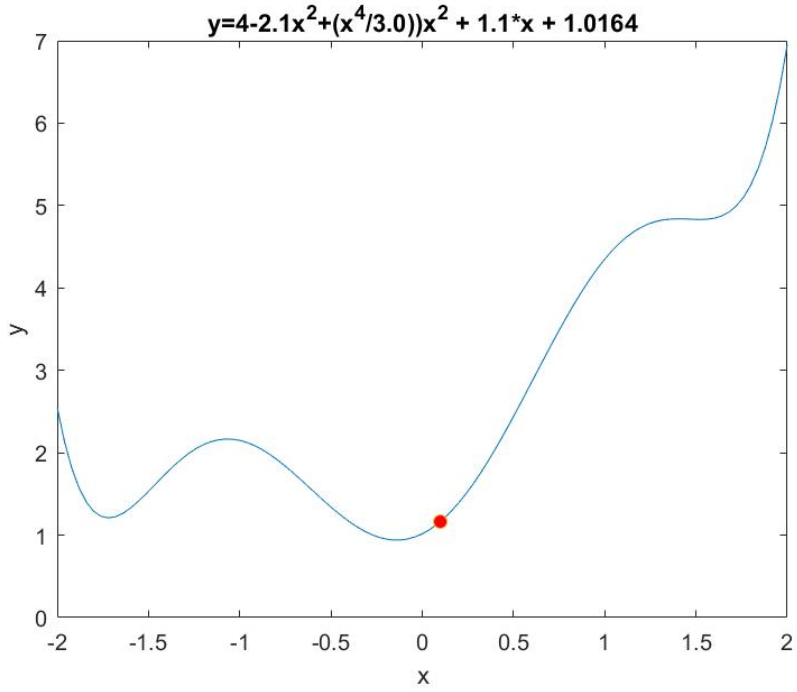
Differential Evolution Worked Example

The aim is to minimise $f(x) = \left(4 - 2.1x + \frac{x^3}{3}\right)x^2 + 1.1x + 1.0164$ in the x-interval $-2 < x < 2$ using the DE algorithm. The following parameters are used:

Table 1: Random Numbers in 0 to 1 range	
Number	Random Values
1	0.6582
2	0.7161
3	0.2503
4	0.7816
5	0.8839
6	0.5247
7	0.1934
8	0.5037
9	0.0998
10	0.4530
11	0.7599
12	0.0387
13	0.6920
14	0.2269
15	0.8592
16	0.3498
17	0.6921
18	0.3651

n	1	Number of Design variables
Np	6	Total number of design points (6n)
F	0.4	Scale accelerator
x_{1L}	-2	Lower limit of design variable
x_{1U}	2	Upper limit of design variable
r^{i1}		Random number from table
Cr	0.9	Cross over rate
k_{max}	1	Maximum number of generations

```
Differential Evolution: n=1 Np=6 Ngen=2 xmin= -2.0000 xmax=  2.0000 F=  0.4000 Cr=  0.9000
-------------------------------------------------------------------------------------------

Random numbers for initialisation of design points:
---------------------------------------------------
Point 1=  0.6582 2=  0.7161 3=  0.2503 4=  0.7816 5=  0.8839 6=  0.5247

Random numbers for generating donor and target points at each iteration:
------------------------------------------------------------------------
Generation 1 random numbers:   0.1934   0.5037   0.0998   0.4530
Generation 1 donor designs:   2   4   1 and target design  3

At start of generation   1:
---------------------------
x1=  0.6328 x2=  0.8644 x3= -0.9988 x4=  1.1264 x5=  1.5356 x6=  0.0988
f1=  2.9989 f2=  3.9226 f3=  2.1491 f4=  4.6308 f5=  4.8315 f6=  1.1639
Minimum function value=  1.1639 at x=  0.0988
For generation   1 donation use r1=  2 r2=  4 r3=  1 for target rp=  3 with potential x=  1.0618

At end of generation   1:
---------------------------
x1=  0.6328 x2=  0.8644 x3= -0.9988 x4=  1.1264 x5=  1.5356 x6=  0.0988
f1=  2.9989 f2=  3.9226 f3=  2.1491 f4=  4.6308 f5=  4.8315 f6=  1.1639
Minimum function value=  1.1639 at x=  0.0988

DE search completed xbest=    0.09880 fbest =    1.16393
```



The following example solves the same problem but with a different set of random numbers and a larger number of generations:

```
Differential Evolution: n=1 Np=6 Ngen=10 xmin= -2.0000 xmax=  2.0000 F=  0.4000 Cr=  0.9000
-------------------------------------------------------------------------------------------

Random numbers for initialisation of design points:
---------------------------------------------------
Point 1=  0.9106 2=  0.8006 3=  0.7458 4=  0.8131 5=  0.3833 6=  0.6173
```

Random numbers for generating donor and target points at each iteration:

Generation 1 random numbers: 0.5755 0.0871 0.8990 0.4106
Generation 1 donor designs: 4 1 5 and target design 3
Generation 2 random numbers: 0.5301 0.8021 0.6259 0.9843
Generation 2 donor designs: 4 5 4 and target design 6
Generation 3 random numbers: 0.2751 0.9891 0.1379 0.9456
Generation 3 donor designs: 2 6 2 and target design 6
Generation 4 random numbers: 0.2486 0.0669 0.2178 0.6766
Generation 4 donor designs: 2 1 2 and target design 4
Generation 5 random numbers: 0.4516 0.9394 0.1821 0.9883
Generation 5 donor designs: 3 6 2 and target design 6
Generation 6 random numbers: 0.2277 0.0182 0.0418 0.7668
Generation 6 donor designs: 2 1 1 and target design 5
Generation 7 random numbers: 0.8044 0.6838 0.1069 0.3367
Generation 7 donor designs: 5 4 2 and target design 3
Generation 8 random numbers: 0.9861 0.7837 0.6164 0.6624
Generation 8 donor designs: 6 5 4 and target design 4
Generation 9 random numbers: 0.0300 0.5341 0.9397 0.2442
Generation 9 donor designs: 1 4 6 and target design 2

At start of generation 1:
---
x1= 1.6423 x2= 1.2022 x3= 0.9834 x4= 1.2525 x5= -0.4668 x6= 0.4691
f1= 4.8751 f2= 4.7397 f3= 4.3039 f4= 4.7879 f5= 1.2782 f6= 2.3146
Minimum function value= 1.2782 at x= -0.4668
For generation 1 donation use r1= 4 r2= 1 r3= 5 for target rp= 3 with potential x= 2.0961

At end of generation 1:
---
x1= 1.6423 x2= 1.2022 x3= 0.9834 x4= 1.2525 x5= -0.4668 x6= 0.4691
f1= 4.8751 f2= 4.7397 f3= 4.3039 f4= 4.7879 f5= 1.2782 f6= 2.3146
Minimum function value= 1.2782 at x= -0.4668
At start of generation 2:
---
x1= 1.6423 x2= 1.2022 x3= 0.9834 x4= 1.2525 x5= -0.4668 x6= 0.4691
f1= 4.8751 f2= 4.7397 f3= 4.3039 f4= 4.7879 f5= 1.2782 f6= 2.3146
Minimum function value= 1.2782 at x= -0.4668
For generation 2 donation use r1= 4 r2= 5 r3= 4 for target rp= 6 with potential x= 0.5648

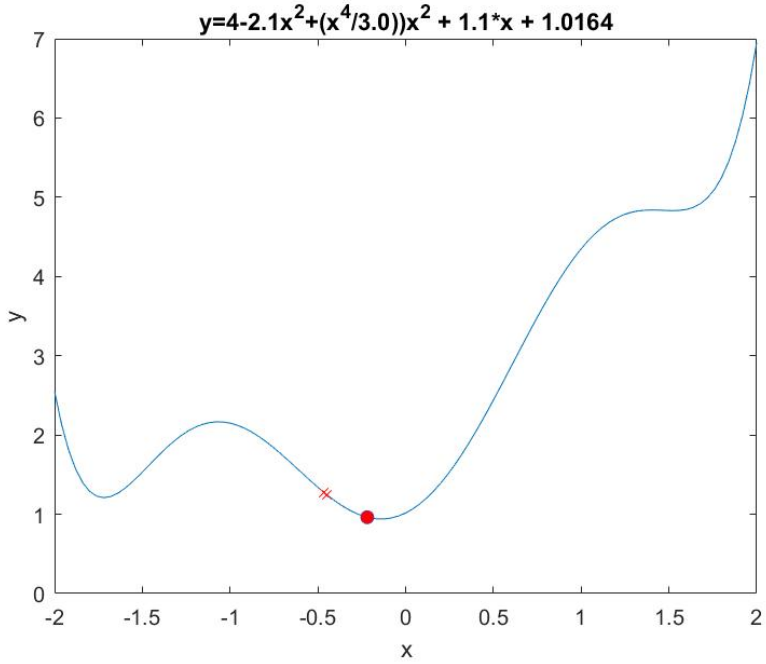
At end of generation 2:
---
x1= 1.6423 x2= 1.2022 x3= 0.9834 x4= 1.2525 x5= -0.4668 x6= 0.4691
f1= 4.8751 f2= 4.7397 f3= 4.3039 f4= 4.7879 f5= 1.2782 f6= 2.3146
Minimum function value= 1.2782 at x= -0.4668
At start of generation 3:
---
x1= 1.6423 x2= 1.2022 x3= 0.9834 x4= 1.2525 x5= -0.4668 x6= 0.4691
f1= 4.8751 f2= 4.7397 f3= 4.3039 f4= 4.7879 f5= 1.2782 f6= 2.3146
Minimum function value= 1.2782 at x= -0.4668
For generation 3 donation use r1= 2 r2= 6 r3= 2 for target rp= 6 with potential x= 0.9090

At end of generation 3:
---
x1= 1.6423 x2= 1.2022 x3= 0.9834 x4= 1.2525 x5= -0.4668 x6= 0.4691
f1= 4.8751 f2= 4.7397 f3= 4.3039 f4= 4.7879 f5= 1.2782 f6= 2.3146
Minimum function value= 1.2782 at x= -0.4668

At start of generation 4:
---
x1= 1.6423 x2= 1.2022 x3= 0.9834 x4= 1.2525 x5= -0.4668 x6= 0.4691
f1= 4.8751 f2= 4.7397 f3= 4.3039 f4= 4.7879 f5= 1.2782 f6= 2.3146

```
Minimum function value=  1.2782 at x= -0.4668
For generation   4 donation use r1=  2 r2=  1 r3=  2 for target rp=  4 with potential x=  1.3783

At end of generation   4:
---------------------------
x1=  1.6423 x2=  1.2022 x3=  0.9834 x4=  1.2525 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  4.3039 f4=  4.7879 f5=  1.2782 f6=  2.3146
Minimum function value=  1.2782 at x= -0.4668

At start of generation   5:
---------------------------
x1=  1.6423 x2=  1.2022 x3=  0.9834 x4=  1.2525 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  4.3039 f4=  4.7879 f5=  1.2782 f6=  2.3146
Minimum function value=  1.2782 at x= -0.4668
For generation   5 donation use r1=  3 r2=  6 r3=  2 for target rp=  6 with potential x=  0.6901

At end of generation   5:
---------------------------
x1=  1.6423 x2=  1.2022 x3=  0.9834 x4=  1.2525 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  4.3039 f4=  4.7879 f5=  1.2782 f6=  2.3146
Minimum function value=  1.2782 at x= -0.4668

At start of generation   6:
---------------------------
x1=  1.6423 x2=  1.2022 x3=  0.9834 x4=  1.2525 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  4.3039 f4=  4.7879 f5=  1.2782 f6=  2.3146
Minimum function value=  1.2782 at x= -0.4668
For generation   6 donation use r1=  2 r2=  1 r3=  1 for target rp=  5 with potential x=  1.2022

At end of generation   6:
---------------------------
x1=  1.6423 x2=  1.2022 x3=  0.9834 x4=  1.2525 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  4.3039 f4=  4.7879 f5=  1.2782 f6=  2.3146
Minimum function value=  1.2782 at x= -0.4668

At start of generation   7:
---------------------------
x1=  1.6423 x2=  1.2022 x3=  0.9834 x4=  1.2525 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  4.3039 f4=  4.7879 f5=  1.2782 f6=  2.3146
Minimum function value=  1.2782 at x= -0.4668
For generation   7 donation use r1=  5 r2=  4 r3=  2 for target rp=  3 with potential x= -0.4467

At end of generation   7:
---------------------------
x1=  1.6423 x2=  1.2022 x3= -0.4467 x4=  1.2525 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  1.2422 f4=  4.7879 f5=  1.2782 f6=  2.3146
Minimum function value=  1.2422 at x= -0.4467

At start of generation   8:
---------------------------
x1=  1.6423 x2=  1.2022 x3= -0.4467 x4=  1.2525 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  1.2422 f4=  4.7879 f5=  1.2782 f6=  2.3146
Minimum function value=  1.2422 at x= -0.4467
For generation   8 donation use r1=  6 r2=  5 r3=  4 for target rp=  4 with potential x= -0.2186

At end of generation   8:
---------------------------
x1=  1.6423 x2=  1.2022 x3= -0.4467 x4= -0.2186 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  1.2422 f4=  0.9623 f5=  1.2782 f6=  2.3146
Minimum function value=  0.9623 at x= -0.2186

At start of generation   9:
```

```
---------------------------
x1=  1.6423 x2=  1.2022 x3= -0.4467 x4= -0.2186 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  1.2422 f4=  0.9623 f5=  1.2782 f6=  2.3146
Minimum function value=  0.9623 at x= -0.2186
For generation   9 donation use r1=  1 r2=  4 r3=  6 for target rp=  2 with potential x=  1.3672

At end of generation   9:
---------------------------
x1=  1.6423 x2=  1.2022 x3= -0.4467 x4= -0.2186 x5= -0.4668 x6=  0.4691
f1=  4.8751 f2=  4.7397 f3=  1.2422 f4=  0.9623 f5=  1.2782 f6=  2.3146
Minimum function value=  0.9623 at x= -0.2186
DE search completed xbest=   -0.21857 fbest =    0.96231
```



4.5 Genetic Algorithms (GA)

Genetic algorithms (GA) are a class of stochastic algorithms which simulate natural inheritance and Darwin's survival of the fittest concept. They belong to the class of probabilistic algorithms; but are very different from random ones, as they combine elements of directed and stochastic search. They are superior to hill-climbing methods, since at any time a GA provides for both exploitation of the best solutions, and exploration of the search space.

GA performs a multi-directional search by maintaining a population of potential solutions and encourages information formation and exchange between these directions. This population undergoes a simulated evolution: at each generation the relatively "good" solutions reproduce, while the relatively "bad" solutions die. One of the main differences between GA and other stochastic methods is in the way that the design variables are stored and manipulated. GA attempts to simulate numerically biological genetics and evolution. So, the design variables are represented not as a vector of values, but as a binary string of 0's and 1's called a chromosome. This means that this vector of design variables: $x = \left(x_1, x_2, \cdots, x_n\right)$ is represented by this binary string: $x = 10101001110011110010011000100101010010010 1$. But instead of calling this the design variables, this binary string is called a chromosome (6.23).

$$chromosome = \left[1010100111\ 0011110010\ 0110001001\ 0101001001\ 01\right] \quad (4.23)$$

A chromosome (also called an *Individual* or *string*) is a binary string which holds all of the design variables and which needs to be optimized. Since the design variables are all embedded inside of the chromosome, then all of the segments which make up each individual design variable are the genes, (4.24).

$$chromosome = \left[\underbrace{1010100111}_{gene\ 1}\underbrace{0011110010}_{gene\ 2}\cdots\underbrace{0100100101}_{gene\ N_{var}}\right] \quad (4.24)$$

The genes then represent the design variables inside of the chromosome. In the chromosome of (4.24), the number of design variables is N_{var}, and each one has been coded using 10 bits, i.e. N_{gene} = 10. The length of this chromosome is then given by (4.25).

$$N_{bits} = N_{gene} \times N_{var} = 10 \times N_{var}\ bits \quad (4.25)$$

Each individual bit of a chromosome is called an *Allele*, (4.26).

$$chromosome = \left[\underbrace{1010100111}_{gene\ 1}\underbrace{0011110010}_{gene\ 2}\cdots\underbrace{0100100101}_{gene\ N_{var}}\right] \quad (4.26)$$

A *Population* consists of a group of individuals that interact (breed) together and looks like (4.27).

$$Population = \begin{Bmatrix} 1010100111001111001001100010010101001001 01 \\ 0010010011110001010010100100101010000000 10 \\ \vdots \\ 1111100100010110010100100101100100100 11 \end{Bmatrix} \quad (4.27)$$

A population of individuals can then be manipulated and combined by using the following six genetic operators, each of these operators will be explained in the sections which follow.

1) Evaluation
2) Selection
3) Reproduction (Cross-over)
4) Mutation
5) Elitism
6) Extermination

4.5.1 Evaluation

In conventional optimization, the objective function is used to provide a measure of optimality for the values of the design variables. In GA, the fitness of an individual is used to determine its relative performance (fitness) compared to the other individuals of a population. Fitness Function (FF): is the function used to calculate the fitness, (4.28).

$$F(x) = g(f(x)) \tag{4.28}$$

where:

f: is the objective function;
g: transforms the value of the objective function to a non-negative number
F: is the resulting relative fitness.

4.5.2 Selection

The process of selection consists of selecting two chromosomes (individuals) from the mating pool (population) in order to carry out reproduction to produce two new offspring. The selection process is biased toward fit members of the current population. Using the fitness value F_i for each individual of the population, its probability of being selected is calculated using (4.29).

$$P_i = \frac{F_i}{Q}, \quad Q = \sum_{j=1}^{N_p} F_j \tag{4.29}$$

The members with the higher fitness have the largest probability of selection. To explain the process of selection, consider the roulette wheel with a handle, Figure 4.6. The wheel has *Np* segments to cover the entire population. The size of the i^{th} segment is proportional to the probability P_i. A random number *w* is generated between 0 and 1. The wheel is then rotated clockwise, with the rotation proportional to the random number *w*.

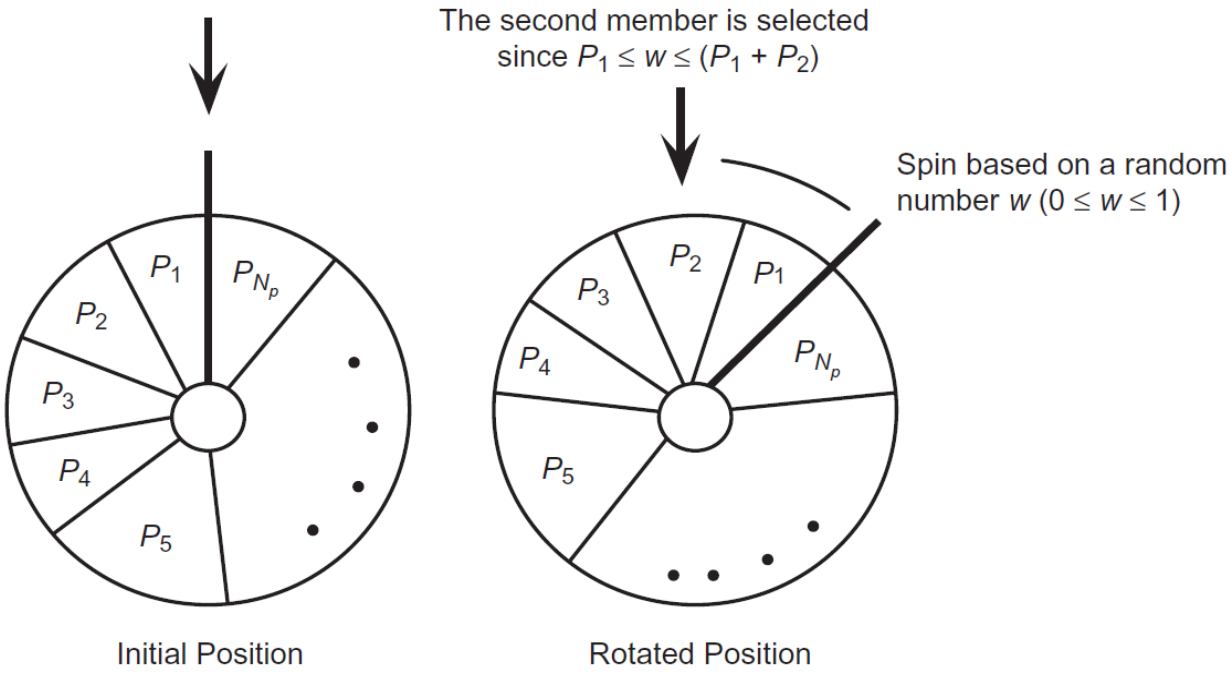


Figure 4.6: Roulette wheel process for selection of designs for new generation (reproduction). After spinning the wheel, the member pointed by the arrow at the starting location is selected for inclusion in the next generation. In Figure 4.6, the 2nd individual is selected as a parent for reproduction. Since the segments on the wheel are sized according to the probabilities P_i, the selection process is biased toward fitter members of the current population.

Note that a member selected for mating remains in the current population for further selection. Therefore, the new population may contain identical members and may not contain some of the less fit individual members from the current population. This guarantees that the average fitness of the new population is increased. The three of the most common selection methods are:

1) Roulette Wheel Selection
2) Linear-rank selection
3) Tournament selection:

4.5.2.1 Roulette Wheel Selection

Roulette Wheel Selection or Proportional Selection was the original selection process. As was just explained, each individual of the population is represented by a space proportional to its fitness. By repeatedly spinning the wheel, individuals are chosen using random sampling with replacement.

4.5.2.2 Linear-Rank Selection

The individuals of the population are ordered according to their fitness. Copies are assigned in such a way that the best individual receives a pre-determined multiple of the number of copies the worst one receives. Rank selection implicitly reduces the dominating effects of "super individuals" in populations (i.e., individuals that are assigned a significantly better fitness value than all other individuals). However, it warps the difference between close fitness values, thus increasing the selection pressure in stagnant populations.

4.5.2.3 Tournament Selection

There are a number of variants on this theme. The most common one is k-tournament selection where k individuals are selected from a population. The fittest individual of the k selected ones is considered for reproduction. In this variant, selection pressure can be scaled quite easily by choosing an appropriate number for k.

4.5.3 Reproduction (Cross-over)

The operator to produce new individuals is call *reproduction*. Like nature, reproduction is carried out by crossover, which produces new individuals with some parts from the genetic material of both parents. There are 3 common forms of cross over, these are:

1) Single-point Crossover
2) Multi-point Crossover
3) Uniform Crossover

4.5.3.1 Single-point Crossover

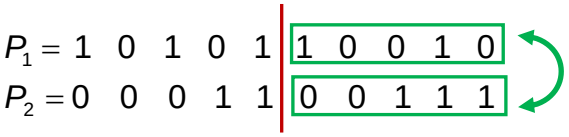
Consider the following two binary strings as two parents:

$$P_1 = 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0$$
$$P_2 = 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1$$

A position, *i*, is selected randomly between 1 and the string length, *l*. Assume that for this problem, the position i = 5 was randomly selected. The two strings are then cut between the 5th and 6th allele.

$$P_1 = 1\ 0\ 1\ 0\ 1 \mid 1\ 0\ 0\ 1\ 0$$
$$P_2 = 0\ 0\ 0\ 1\ 1 \mid 0\ 0\ 1\ 1\ 1$$

The segments of the strings to the right of the cut are then swapped.



To produce these two children

$$O_1 = 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1$$
$$O_2 = 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0$$

4.5.3.2 Multi-point Crossover

For multi-point crossover, *m* crossover positions are chosen at random with no duplicates and these are then sorted into ascending order, (4.30).

$$k_i \in \{1,3,7,12,\cdots,l-1\} \tag{4.30}$$

where:

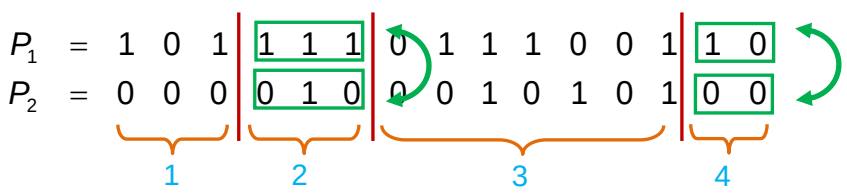
k_i: are the crossover points
l: is the length of the chromosome

Then, the bits (alleles) between successive crossover points are exchanged between the two parents to produce two new offspring

Consider the following two binary strings as two parents:

$$P_1 = 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0$$
$$P_2 = 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0$$

If the number of cross overs (*m*) was set to m = 3. With the 3 positions randomly selected to be $k_i \in \{3,6,13\}$, this then creates the 4 segments shown below. But only segments 2 and 4 take part in the crossover. These two segments are cut are then swapped.



To produce these two children

$$O_1 = 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0$$
$$O_2 = 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0$$

4.5.3.3 Uniform Crossover

Uniform crossover generalises this scheme to make every locus a potential crossover point. A crossover mask (the same length as the chromosome) is randomly created with values of 0 or 1. The value of the bits in the mask indicates which parent supplies each offspring with their information. Consider the following two parents and the randomly generated crossover mask.

$$P_1 = 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0$$
$$P_2 = 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0$$
$$mask = 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0$$

The first offspring, O_1, is produced by taking the bit from P_1 if the corresponding mask bit is 1 or the bit from P_2 if the corresponding mask bit is 0. Offspring O_2 is created using the inverse of the mask. For the two parents and mask shown above, the two offspring become:

$$O_1 = 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0$$
$$O_2 = 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0$$

4.5.4 Mutation

In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure. In GA, mutation is randomly applied with a probability of between 0.001 and 0.01 (0.1% to 1%). Mutation takes place after the reproduction (cross-over) stage and modifies individual alleles in the chromosome by inverting their value. Mutation provides a guarantee that all possible strings will be searched. It acts as a safety net to recover good genetic material which may have be lost through selection and crossover

Consider that the 2nd of these two offspring was randomly selected for mutation. It was then randomly found, that the 7th position was going to be mutated.

$$O_1 = 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0$$
$$O_2 = 1\ 0\ 0\ 0\ 1\ 0\ \boxed{0}\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0$$

Since the current value is 0, this is then mutated to have a value of 1.

$$O_1 = 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0$$
$$O_2 = 1\ 0\ 0\ 0\ 1\ 0\ \boxed{1}\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0$$

4.5.5 Elitism

A very small proportion of the individuals (chromosomes) with the best fitness are carried over from one generation to the next. Elitism guarantees that the solution quality obtained by the

GA does not decrease from one generation to the next. A reasonable proportion of the population to be considered for elitism is 10%.

4.5.6 Extermination

It's reasonable to exterminate a certain percentage (%) of the population of lowest fitness. A reasonable proportion of the population to be exterminated is between 10 and 25%.

4.5.7 Termination Condition

The GA process can go forever, unless there is some sort of termination condition. There are normally a couple of ways to achieve this:

1) Maximum number of iterations (called generations) reached . A typical value is 150 generations
2) Fitness function conversion. If the fitness function value does not improve by more than a minimum relative threshold (typically) 0.0001% (0.000001) over the previous n generations (typically 10) the process is terminated.

4.5.8 The Genetic Algorithm Procedure

The virtual code of Figure 4.7 shows how the GA procedure works.

Genetic Algorithm Procedure
begin
t = 1
Initialize *P*(*t*) - *Randomly*
Evaluate *P*(*t*)
while (**not** termination-condition) **do**
begin
Extermination *P(t)*
Selection from *P*(*t*)
Elitism *P*(*t+1*) from *P*(*t*)
Reproduction *P*(*t+1*)
Mutation *P*(*t+1*)
Evaluate *P*(*t+1*)
t = t+ 1
end
end

Figure 4.7: GA virtual code

4.6 Ant Colony Optimization (ACO)

Ant colony optimization (ACO) emulates the food searching behaviour of ants. The method was originally developed by Dorigo (1992) to search for the optimal path for a problem represented by a graph. It was based on the behaviour of ants seeking the shortest path

between their colony and a food source. ACO falls into the metaheuristics[2] and swarm intelligence methods class. It is a stochastic technique for solving computational problems, which can be used to find optimal paths.

4.6.1 How Real Ants Work

Ants are able to deal with complex tasks by acting collectively. This collective behaviour is supported by the release of a chemical substance, named pheromone. During their movement, ants deposit pheromone in their followed paths.

The presence of pheromone in a path attracts other ants. In this way, pheromone plays a key role in the information exchange between ants, allowing them to accomplish several important tasks. A classic example is the selection of the shortest path between their nest and a food source.

Consider four ants (A_1, A_2, A_3 and A_4), and two possible paths, P_1 and P_2, Figure 6.8. These two paths link a nest (NE) to a food source (FS), and for this explanation, it is assumed that path P_1 is longer than path P_2, hence $P_2 < P_1$.

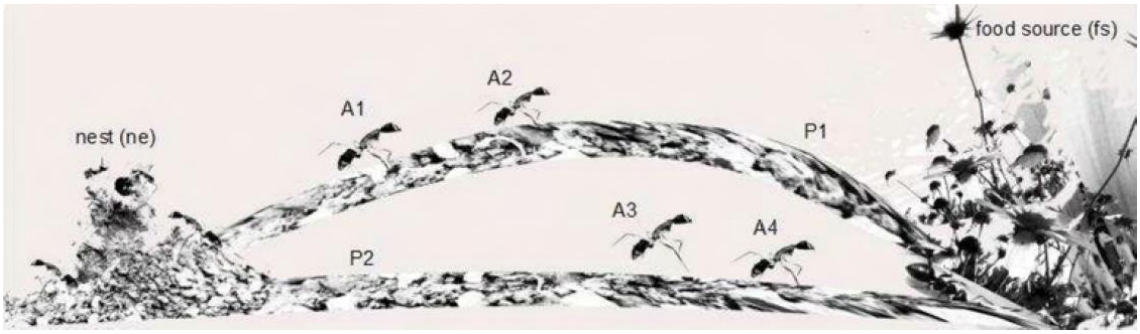


Figure 4.8: Two paths between a nest and food source with four ants.

Initially, all the ants (A_1, A_2, A_3 and A_4) are in N_E and must choose between the paths P_1 and P_2 to arrive to F_S.

At the N_E, the four ants don't know the localization of the food source (F_S). Randomly they choose between P_1 and P_2, with the same probability. So assume that ants A_1 and A_2 choose P_1, and ants A_3 and A_4 choose P_2, Figure 4.8.

As the ants travel by P_1 and P_2, they leave a certain amount of pheromone on the paths, τ_1 and τ_2, respectively. Since $P_2 < P_1$, A_3 and A_4 arrive to F_S before A_1 and A_2. At that moment, $\tau_2 = 2$, but $\tau_1 = 0$ since A_1 and A_2 have not arrived to F_S, Figure 4.9.

[2] Metaheuristics: Makes few assumptions about the optimization problem being solved, making it usable for a variety of problems.

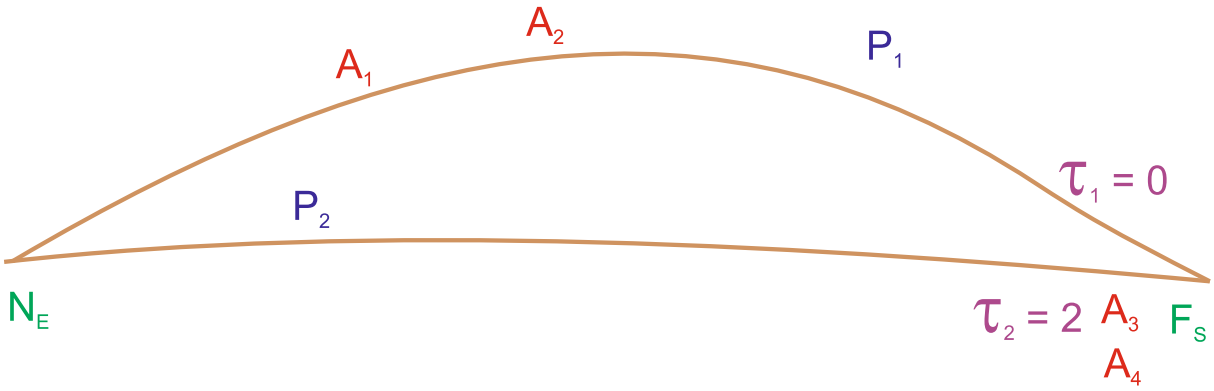


Figure 4.9: Two ants follow path P_1 and the other two paths P_2. Now at the food source, the ants need to decide which path to take.

In order to come back to N_E, A_3 and A_4 must choose again between P_1 and P_2. At the F_S, $\tau_2 > \tau_1$, the probability of these ants choosing P_2 is higher. Assume then, that A_3 and A_4 choose P_2, so that as they travel back to N_E $\tau_2 = 4$.

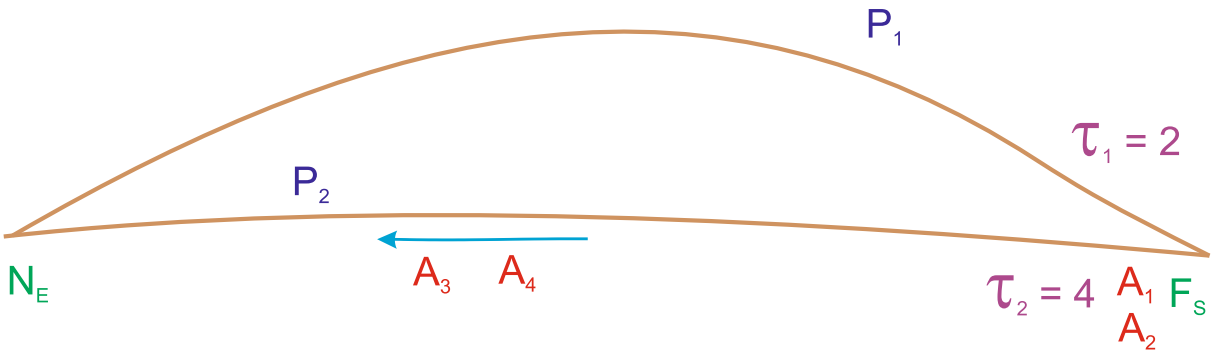


Figure 4.10: With ants A_3 and A_4 already along path P_2, ants A_1 and A_2 need to decide which path to take.

When A_1 and A_2 arrive at the F_S, then $\tau_2 = 4$ and $\tau_1 = 2$, Figure 4.10. When A_1 and A_2 decide to go back to N_E, since $\tau_2 > \tau_1$, then the probability that they choose to return via P_2 becomes higher. As they do, then $\tau_2 = 6$. When all ants are back at N_E, then $\tau_2 = 6$ and $\tau_1 = 2$, so in the future P_2, will have highest probability of being selected, Figure 6.11.

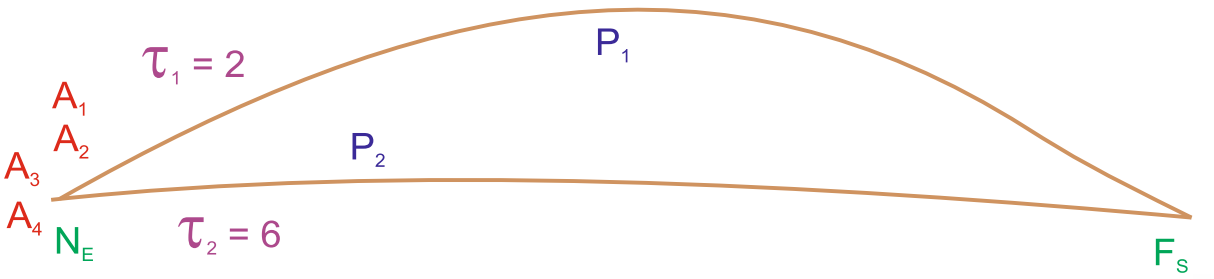


Figure 4.11: Final pheromone composition in the two paths after all ants are back at the nest.

When there is no pheromone, an ant looking for food randomly will choose between P_1 and P_2 with a probability of 0.5 (50% possibility of choosing each path). When there is pheromone

on at least one of the paths, the probability of selecting a given path is proportional to the amount of pheromone on it. Thus, paths with a higher concentration of pheromone have a higher probability of being selected.

To understand how to use ant colonies to solve problems, it is necessary to understand the problem of foraging for food and how ants solve it. Each location (nest, food source, etc.) is represented by a node and each path by an edge in a graph, Figure 4.12.

To solve a problem using ant colony optimization the domain needs to be able to be represented, as a graph and the goal will then be to find the best path.

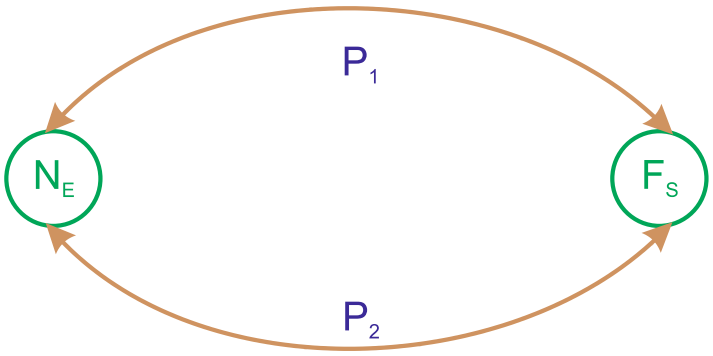


Figure 4.12: Two nodes representing the nest and food source connected by two paths.

4.7 How to Solve Constrained Optimization Problems

The most common approach for solving optimization problems which have constraints (particularly, inequality constraints) with any of the stochastic methods mentioned in this chapter as well as pattern search methods of chapter 3 (sections 3.6 and 3.7) is to use Penalty Functions.

The idea of this method is to change a constrained-optimization problem into an unconstrained problem; by adding a value to the objective function based on the amount of constraint violation present in the solution. The modified objective function with the penalty terms is given by equation (4.31).

$$F(x) = f(x) + r_k \sum_{j=1}^{p} h_j^2(x) + r_k \sum_{i=1}^{m} \langle g(x) \rangle^2 \tag{4.31}$$

where:

r_k is (> 0) and has to be appropriately selected. A possible equation for it is given by equation (4.32). It needs to have a small value at the start but then increase to a larger value for the purpose of tightening the constraints.

$\langle g(x) \rangle$ is the function of equation (4.33) which has a value of zero if the inequality constraint is not violated or the value of how much the constraint is violated.

$$r_k = \max\left[1, \frac{1}{\left|\langle g_i(x) \rangle \quad h_j(x)\right|}\right] \tag{4.32}$$

$$\langle g_i(x) \rangle = \max\left[0, g_i(x)\right] \tag{4.33}$$

In case constraints are satisfied $g_i(x) \leq 0$, then $\langle g_i(x) \rangle$ will be zero and there will be no penalty on the objective function. In case constraints are violated $g_i(x) > 0$ then $\langle g_i(x) \rangle$ will be a positive value resulting in a penalty on the objective function. The penalty will be higher for higher infeasibility of the constraints. The function F(x) can be optimized using the algorithms for unconstrained problems. The penalty function method of this form is called the *exterior penalty function* method.

The main advantages of the penalty function method are that:

a) It can be started from an infeasible point.
b) Unconstrained optimization methods can be directly used.

The main disadvantages of the penalty function method are that:

a) The function becomes ill-conditioned as the value of the penalty terms is increased. Owing to abrupt changes in the function value, the gradient value may become large and the algorithm may show divergence.
b) As this method does not satisfy the constraints exactly, it is not suitable for optimization problems where feasibility must be ensured in all iterations.

Only exterior penalty function method was presented, which can be started even from an infeasible point. Some problems require feasibility to be maintained in all iterations. In such cases, the *interior penalty function* methods also called *barrier function* methods can be used.

Chapter 5

Design of Experiments (DoE)

5.0 Introduction

A Design of Experiment (DoE) [42, 48, 123] is a procedure for selecting the values of the input variables for the experiment from the predetermined parameter domain. DoE methods are not always relevant for certain physical experiments, such as data values dictated by geographical locations, however they are ideal for computer simulations. Theoretically there are an infinite number of possible design choices, practically however the domain is discretised into a finite number of possible data points either based upon the computer paradigm and/or engineering requirements. The values for the design parameters chosen by the DoE are members of this finite set and are also known as the training data for the resulting approximation model. The quantity of training data points required for any given situation is problem dependent and can be viewed as an optimisation in its own right, how to obtain the best (most representative) results for the least amount of work.

Whilst the overall aim is to use computer simulations as a complement, or even as a replacement, for physical experiments, the techniques are illustrated using standard analytical optimisation test problems. However, a possible drawback with this approach is that for practical engineering problems we do not usually know how smooth the actual response surface is. The addition of a small (< 10%) amount of normally distributed random errors ('noise') to the true analytical response value can mimic real life engineering applications slightly better, whilst remaining computationally cheap to analyse and potentially highlighting any possible pitfalls with the surrogate models, such as over-fitting. Hueng *et a*l. [61] use this approach for several analytical test functions, with varying amounts of noise per function, normally distributed about the true values.

5.0.1 Test Functions

Two different analytical test functions with different properties have been chosen for illustrative and testing purposes. The first is the Six-Hump Camel-Back (SHCB) function [102, 151] which has three pairs of local minima, of which one pair are global minima, within the specified domain. The second test function, the Rosenbrock 'Banana' (RB) [143], presents different challenges with large response values at the domain extrema and a notoriously difficult to find global minimum [6] located within a banana shaped groove.

The SHCB function is described by (8.1) with the global minima having values of $f = -1.0316$ at $(0.0898, -0.7127)$ and $(-0.0898, 0.7127)$. Contours of the function are shown in Figure 5.1. The contours shown are evenly spaced at increments of $f = 0{:}25$.

$$f(x_1, x_2) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4,$$
$$\text{for } x_1 \in [-2,2],\ x_2 \in [-1,1]. \qquad (5.1)$$

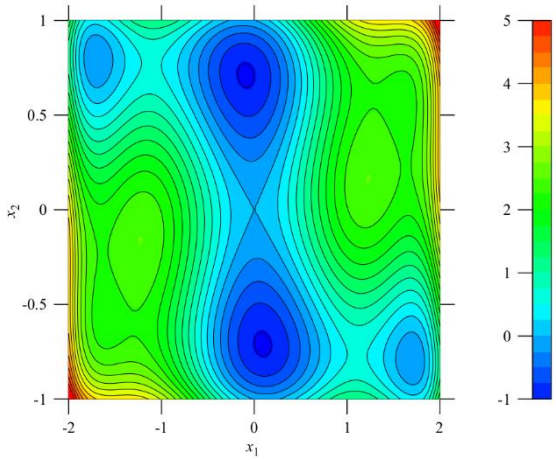


Figure 8.1: Contour plot for the Six-Hump Camel-Back analytical function.

Whilst the SHCB function has multiple minima, it only accepts two-dimensional training data. The two-dimensional RB function given by (5.2a), shown in Figure 5.2, can be extended to p dimensions, as in equation (5.2b).

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2, \qquad (5.2a)$$
$$f(\vec{x}) = \sum_{i=1}^{p}((1 - x_1)^2 + 100(x_{i+1} - x_i^2)^2). \qquad (5.2b)$$

For the two dimensional case, the single global minimum lies at $(1,1)$ with a value of $\mathrm{f} = 0$. For higher dimensions [6, 79], there are multiple minima (two for $4 \leq n \leq 7$, with the global located at $x_i = 1, i = 1, \ldots, p$ at a value of $f = 0$.
The contour plot for the two dimensional case is shown in Figure 5.2, with contours at $f = 0.5, 1, 2, 3, 4, 5, 10, 25, 50, 100, 250, 500, 1000$ and 2000.

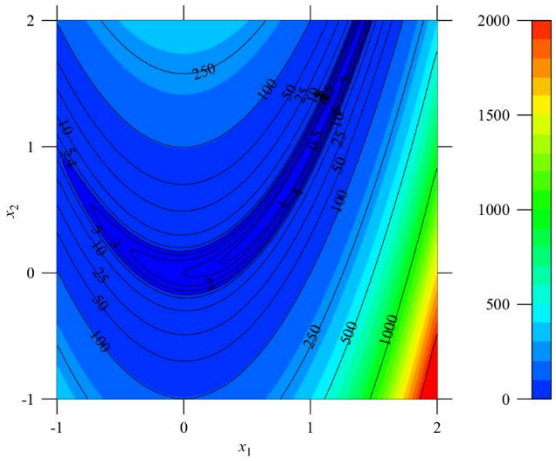


Figure 5.2: Contour plot for the 2D Rosenbrock Banana function.

5.0.2 DoE Notation

Each DoE contains n training data points $\vec{x}_i \left(1 \leq i \leq n\right)$ located within the parameter domain, where each data point is represented by a p dimensional coordinate $\overrightarrow{x_i} \epsilon R^p$. As such, each DoE can be represented as either a vector of vectors or a matrix, (8.3), which may be used to determine the quality of the experimental design. Some DoE techniques require that each design parameter be discretised into levels (intervals) to aid with allocation of the parameter values for the n data points.

$$X = \left(\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_n\right) = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{p1} & \cdots & x_{pn} \end{bmatrix} \tag{5.3}$$

Whilst in the context of CFD and engineering, each training data point corresponds to a particular set of values for the design parameters, for the purposes of this report (and following literature [92, 103]) a 'design' can be taken to mean a particular set of n training data points that constitute a given DoE and may be interchanged accordingly. The design parameters p are the variables for both the DoE and the resulting *n* simulations, hence the terms design parameters and parameters are equally valid in both instances.

Following this notation, a population of DoEs contains *Q* individual designs, each design containing *n* training data points in *p* dimensions, $\overrightarrow{x_i} \epsilon R^p$ for $1 \leq i \leq n$.

5.1 Experimental Designs for Simulations

Theoretically the exact input-output relationship for computer simulations is already known. Furthermore, plotting the computer response for the entire parameter domain would provide an accurate response (hyper-) surface. However as this may be too computationally expensive to realise, numerous methods (known as *surrogate models* or often termed *metamodels*) for mimicking the behaviour of a given hyper-surface have been developed [25, 54, 69, 86, 123, 129, 151] based on a reduced (hopefully minimum) number of strategically chosen input data points.

Regardless of the ultimate experimental objective, there exist several different and well-documented methods for obtaining the initial training data. Recent years have seen an increase in the development of such techniques for computational experiments [42, 73, 125], tailoring the methods to the specific requirements, many of which are straightforward to implement for regular parameter domains. As irregular domains may be problematic [73] and affect properties of the method, they will not be considered in this thesis. However, constraints may be added to a regular domain [73], after DoE selection and surrogate model construction, as part of the optimisation process. Although this may incur additional computational efforts, it simplifies the groundwork stages enormously.

5.1.1 DoE Requirements for Simulations

An obvious requirement is that given any set of design parameters and number of training data points, the DoE technique yields different sets of training data whose resulting surrogate models have predictive (quantitative and qualitative) qualities, thus allowing accurate comparison of different surrogate modelling techniques. If different designs produce significantly different hyper-surfaces when implementing the same surrogate model, this indicates that the initial design may not have contained enough data points. Alternatively, it may suggest that the surrogate model was inappropriate and another may provide better results. There may even be a necessity to customise a DoE such that it is suitable for a certain surrogate model.

As uniform designs provide a good basis for an average surrogate model representation across the parameter domain [125], a sensible strategy could be to get basic approximation with a space-filling design and then tailor the surrogate model further with exploratory techniques for allocating additional data points. As such, the initial focus is concerned with space-filling designs. Situations that require additional training data that also fulfil the space-filling requirement are also considered, for example when the initial number of data points is insufficient.

For most practical engineering problems, it is a reasonable assumption that the cost of the calculation of the response value at each training data point is computationally expensive, especially in comparison with the cost associated with that of generating a suitable DoE, which may be small, or even negligible, to that of a single data point evaluation. Therefore the time taken to obtain a good set of input points can potentially avoid wasted simulation time.

5.1.2 Quantifying and Comparing Spatial Coverage

A population of distinct DoE designs can be generated from any DoE method for a given number of training data points in any specified parameter domain. Each member of the population can be evaluated according to the spatial coverage it provides over the parameter domain and the most appropriate DoE design employed for the subsequent simulation. There exist several documented measurements based on the Euclidean distance between the data locations in parameter space to quantify spatial coverage and compare different experimental designs. The methods are equally applicable for comparisons of different designs in a population or modification to an individual design to produce a new design, and hence determining whether the modification has yielded any improvement. Two of the simplest quantifiers are the maxi-min and mini-max methods, mathematical details for both can be found in [125] and references therein.

The maxi-min method ensures that in any individual design no two training data points are too ‘close’ in parameter space. The first stage calculates the Euclidean distances between all the data points in a design and determines the minimum. If another design, be it a modification or another member of the population, has a greater minimum Euclidean distance, then the

second design is better under this criterion. Similarly the mini-max method guarantees that the data points are not too far apart by finding the minimum of the maximum Euclidean distances. Used in tandem, the mini-max and maxi-min methods can provide bounds for ascertaining the 'best' design in a population. However, these methods may not provide the optimum design as they do not consider the distribution of data points in each design as a whole, only the extrema.

The potential energy analogy is shown to overcome this barrier [11, 13, 14, 88] by effectively assigning a unit of 'charge' to each of the n data points and calculating the resulting 'energy', *U*, in the whole design:

$$U = \sum_{i=1}^{n} \sum_{j=i+1}^{n} r_{ij}^{-2}, \quad for\ i \neq j \tag{5.4}$$

where: r_{ij} is the Euclidean distance between data points *i* and *j* The 'best' (as in the most uniformly distributed) DoE of any given set of DoE designs is the one with the lowest potential energy *U*.

5.2 Overview of Various Design of Experiment Techniques

Traditionally experiments were carried out by varying one design factor at a time and holding the others constant. Classical DoE techniques came in 1920 when Fisher published his strategy for experimenting with several design variables [148], initially applying his Full Factorial designs to agricultural problems. The next developmental wave came in the 1950s with Box and Wilson's work on Response Surface Methodology (RSM), with application in the chemical industry [148]. Since then, classical DoE methods have traditionally been used with physical experimentation [92], however they are not always applicable to computational simulations. Newer methods, such as Hammersley Sampling and Latin Hypercube Sampling, have been developed to fulfil the requirements, and exploit the additional flexibility, of computational experimentation.

The desired outcome of the experiment can also influence the choice of technique used. The authors of [103] advise that a sequence of smaller experiments can provide better results than one large experiment. An example of which is a screening DoE to determine the significance of the design variables with a view to reducing the final set of experimental variables to the most significant. A strategy, therefore, would be to consider a low resolution design for screening the main effects followed by a high resolution design for investigating input-output relationships in detail. They also state that the 'effect' of any given design factor be determined by the change in the average response over the m levels (intervals dividing the domain). This is in stark contrast to the Taguchi viewpoint which advocates a large experiment including all the main factors to highlight the interactions, including noise parameters [148]. Taguchi methodology has been fundamental in the development of robust design in industry over the last thirty years.

Training data can be chosen from the parameter domain randomly, uniformly or in an exploratory manner. Santner *et al*. [125] recommend that for computer simulations the experiment be designed so as to obtain a surrogate model which provides estimates of the computer response at unsampled locations over the entire experimental domain. They also suggest that the DoE (set of training data points) allows for a range of surrogate models. Both of these objectives can be achieved using uniformly distributed training data points, also known as space-filling designs.

5.2.1 Classical Designs

Full factorial designs [92,103] became popular in the 1920s as an alternative to the existing costly and time-consuming methods [148]. They contain $n = m^p$ points for m level designs, where m is typically 2 or 3, increasing too rapidly with increasing m or p to be viable for most problems. A two level design can be represented by strings (or matrices) of -1 and 1, or even simply $+$ and $-$, whilst a three level design typically includes the centre point and is represented by 0. Fractional factorial designs [92] were developed in the 1930s and 1940s [148] as a cost effective alternative to full factorial designs. They contain a selected fraction of the full factorial, typically $\frac{1}{2}$ or $\frac{1}{3}$, however n still rises rapidly with m and (especially) p.

Box-Wilson Central-Composite designs (CCD) [103] and Box-Behnken designs (BBD) [20,103] are a direct consequence of the RSM development in the 1950s. CCD typically contain five levels for each variable, with full or fractional factorial designs (with centre points) embedded in the design space. BBD is more efficient than CCD [157], containing three levels for each design variable, including the central point but omitting the corners, thereby requiring fewer data points n than CCD. However unlike CCD, Box-Behnken designs do not contain embedded full or fractional factorial designs. Both methods are used to generate quadratic response surfaces.

5.2.2 Non-Classical Designs

Sacks et al. [123] note that a common feature of classical DoEs is to account for random, non-repeatable, errors in physical experiments, which therefore make them inappropriate for deterministic computer experiments [73]. Simpson et al. [130] find that DoEs with a more uniform coverage of the design domain produce more accurate approximations, irrespective of the sample size (n). Several computer-aided algorithms have been developed to generate and evaluate the design matrices, including Monte Carlo methods, orthogonal arrays and Latin Hypercube sampling.

Monte Carlo (MC) methods [30, 73] are statistical sampling techniques based on randomly generated numbers. Basic MC methods may over sample some areas of the design space, leaving others inadequately sampled. Stratified MC [70, 73] divide the domain into hypercubes of equal probability to ensure a more even coverage of data points.

Orthogonal arrays aim to separate the effects of various design parameters from other factors [18], with an equal number of levels (design intervals) in each column of the array, including

domain corner points [73, 130]. Taguchi proposed a set of orthogonal arrays for robust design, where noise accounts for variation in response values [7, 30]. Fractional designs and Latin hypercubes are also subsets of orthogonal arrays [30].

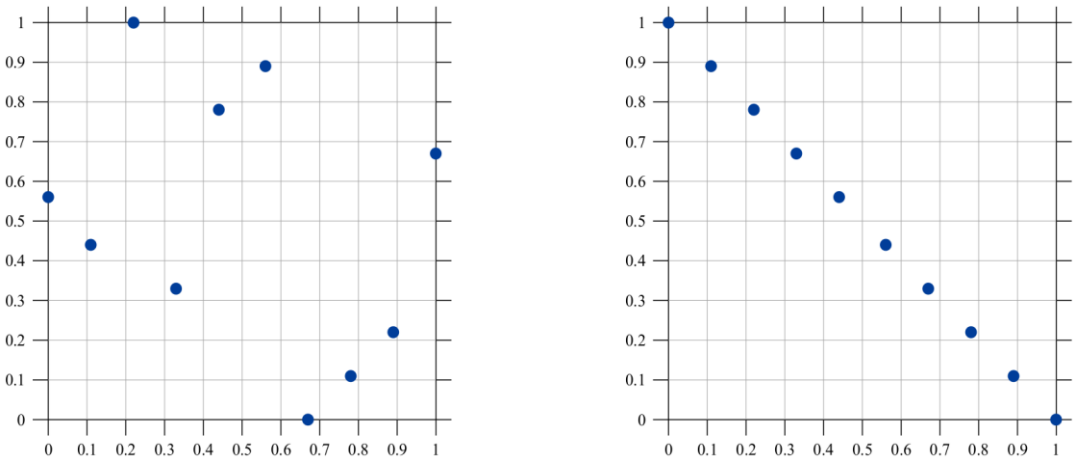
For a regular parameter domain in Latin Hypercube Sampling (LHS) [73, 130], for n samples and p parameters, the domain is divided into n^p hypercubes of equal probability (where each variable is discretised into n levels). The n samples are chosen such that no two data points lie in the same hypercube or share any coordinate values. Whilst random LHS (RLHS) evenly samples each design parameter, it may not sample the parameter domain evenly, although there exist techniques to obtain LHS which do.

5.3 Latin Hypercube Sampling

LHS can be described for normalized design parameters [73] by (5.5). Here $\vec{\kappa}_i$ is a vector of independent, random permutations of the sequence of integers $\{0, \dots, n-1\}$ and $\vec{\tau}_i$ is generally a vector of uniform random numbers in the interval $[0,1]$ although this can differ between texts; $\vec{\tau}_i$ may also be set as a constant, 0.5 say [73, 125], for all *i*. Equally valid, is to discretise the parameter domain from 1 to *n* and force the data points onto the nodes. The latter method ensures fewer DoE permutations which can be of benefit when searching for uniform designs. Use of random number generators to determine the design matrix makes it highly unlikely for the process to produce the same results twice.

$$\vec{x}_i = \frac{\vec{k}_i + \vec{\tau}_i}{n}, \ \forall\ 1 \le i \le n \tag{5.5}$$

One of the main shortcomings of RLHS is illustrated in Figures 5.3a and 5.3b which shows two equally likely DoEs in two dimensions for ten training data points, highlighting that spatial coverage cannot be guaranteed with random LHS.

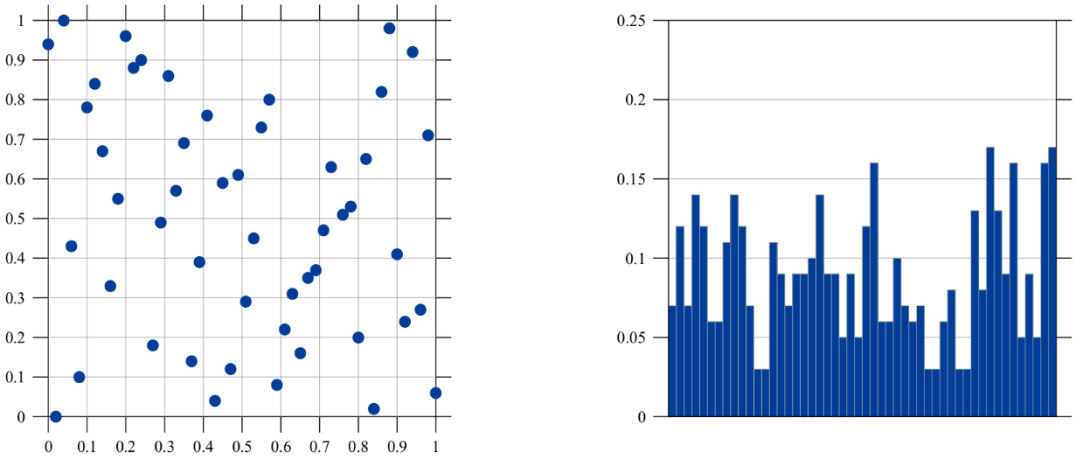


(A) Non-uniform spatial coverage (B) Worst case scenario.

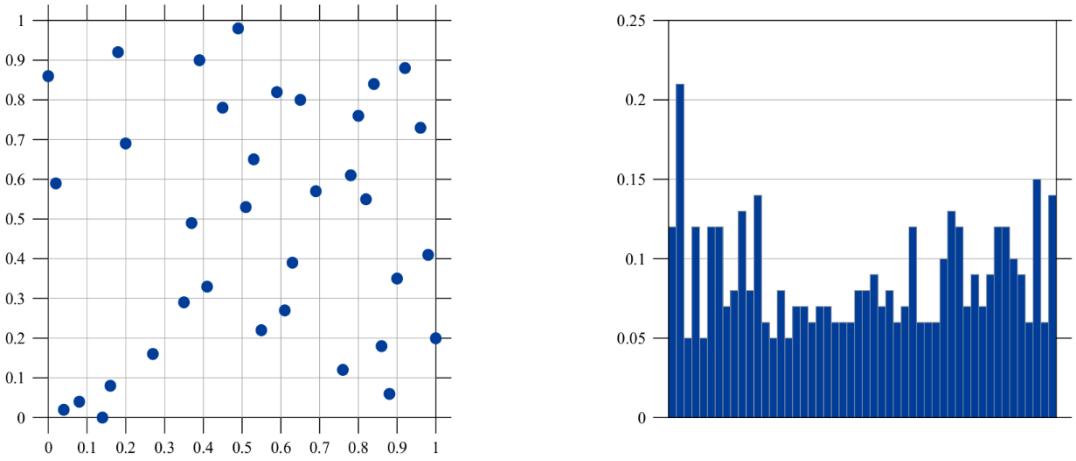
Figure 5.3: Two sample 10 point normalised RLHS in 2D.

As with all random sampling methods, there is a tendency for uniformity to increase with larger numbers of design points. Figures 5.4a to 5.5 show the distribution for three *n* = 50 and one *n*

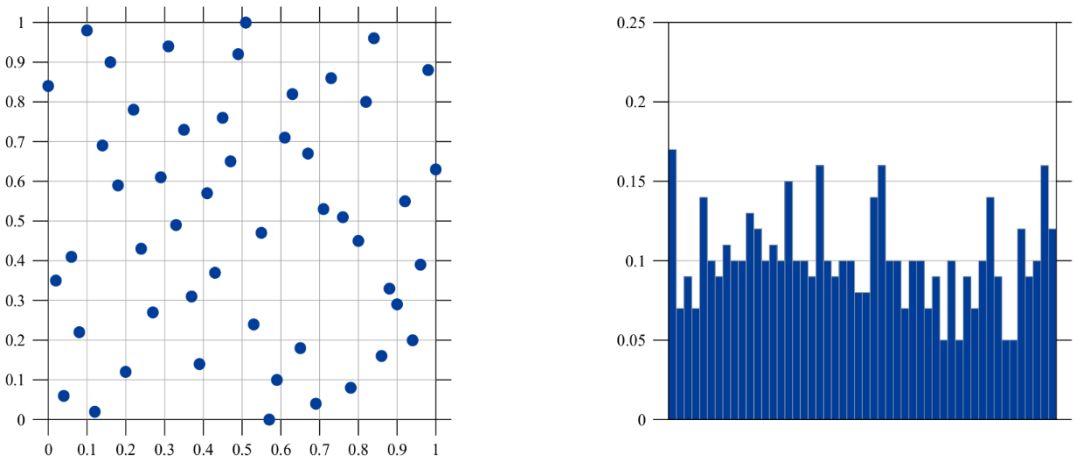
= 100 RLHS DoEs respectively, for two design parameters in a normalised domain, and their corresponding minimum distance plots (from one DoE point to its nearest neighbour). It can be seen that the minimum distances between neighbouring data points varies drastically, which would not be the case if the designs were uniformly distributed over the parameter domain. However, it is important to note that although the minimum distance plots are a useful visual aid (as in [150]), they can also be misleading as would be the case for Figure 5.3b which would show a small constant minimum distance for all the data points.



(A) Example A.



(B) Example B.



(C) Example C.

Figure 5.4: Three sample 50 point normalised RLHS in 2D. Left: Point distributions. Right: Minimum distances.

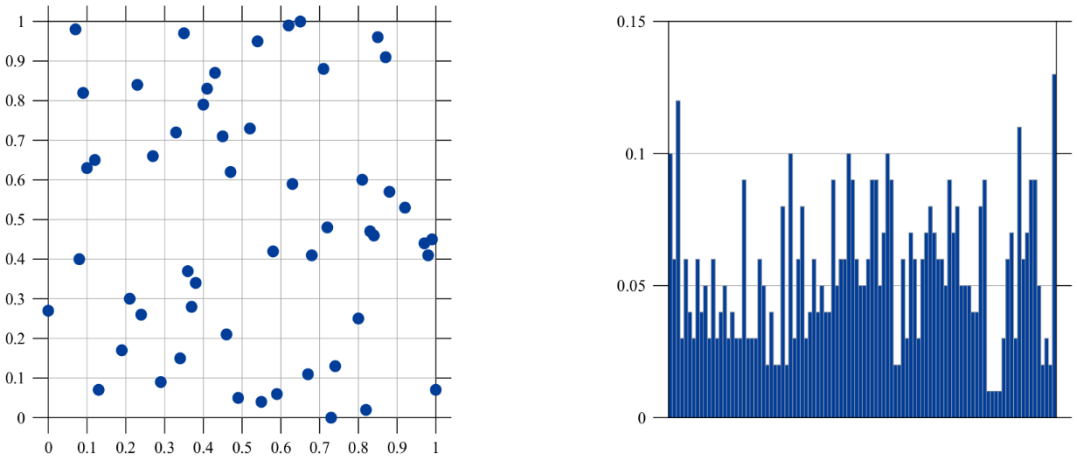


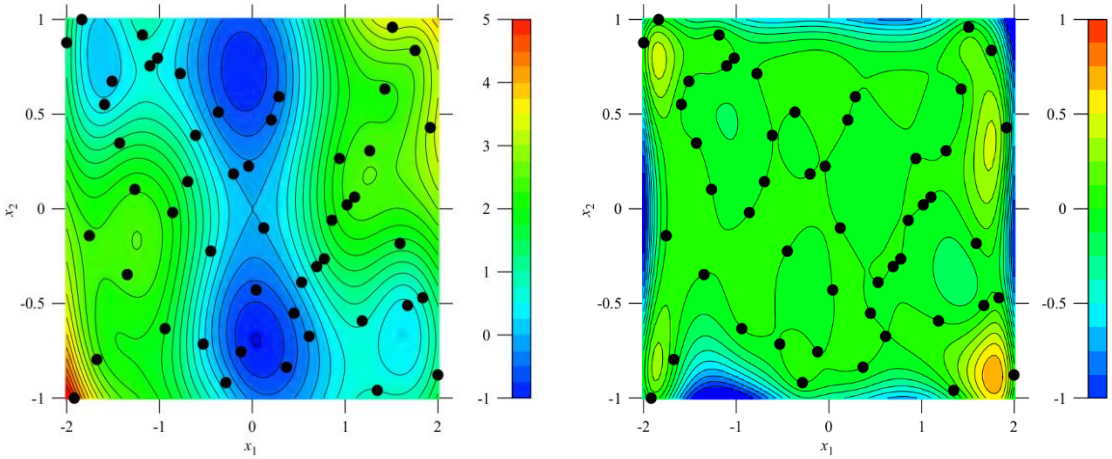
Figure 5.5: A sample 100 point normalised RLHS in 2D. Left: Point distribution. Right: Minimum distances.

As discussed in section 5.1.2 there are various methods to quantify the spatial distribution of the DoE. Using the minimum Euclidean distances as in [150], a mean μ and standard deviation σ can be assigned to these distances. As demonstrated in Figure 5.3 above, the mean and standard deviation of the minimum distances is a necessary, but certainly not sufficient, condition to guarantee uniform spatial coverage of the design space as only one extreme is considered. When comparing DoEs, a better indication of the entire spatial distribution of data points can be achieved by considering the Potential Energy analogy given by (5.4).

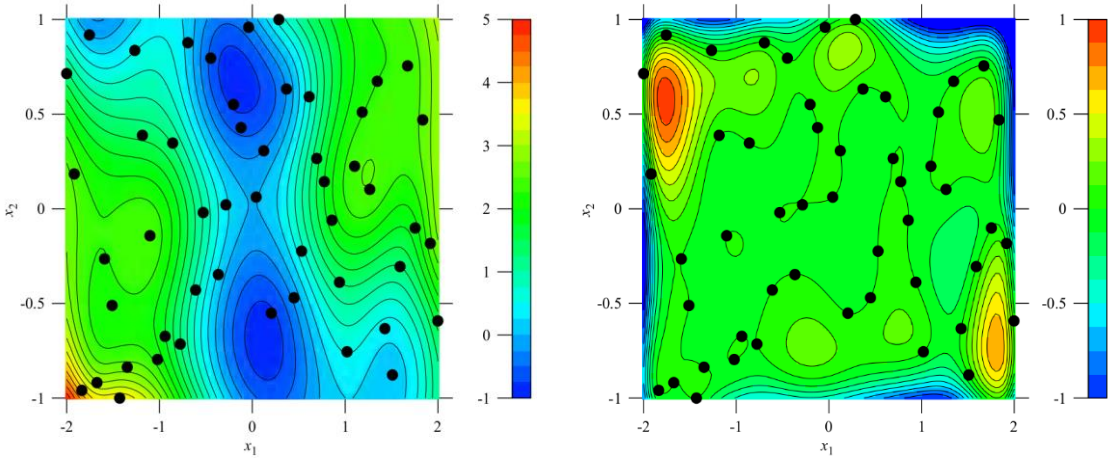
To illustrate the need for uniform spatial coverage of the design space, the three two-dimensional 50 point DoEs presented in Figure 5.4 are used as training data for cubic Radial Basis Function (RBF) surrogate models of the Six-Hump Camel-Back function, Figure 5.6, and the Rosenbrock Banana function, Figure 5.7. RBF methods will be discussed in greater detail in the next chapter. Both figures show the contours of the three resultant surrogate models and the contours of the errors produced by direct comparison of the surrogate models with the analytical function, where a positive error value indicates that the surrogate model has over-predicted the true function and a negative value under-predicts. The relevant DoEs are superimposed on all contour plots.

For the SHCB function in Figure 5.6, the contours are evenly spaced at intervals of $f = 0.25$ for the surrogate models and $f = 0.125$ for the error plots. The RBF however has dramatically different scales and evenly spaced contours are not appropriate. The contours for the surrogate models include those shown in the analytical function depicted in Figure 3.2, however some additional negative values are also required: $f = \pm 0.5, \pm 1, \pm 2, \pm 3, \pm 4, \pm 5, \pm 10, \pm 25, \pm 50, 100, 250, 500, 1000$ and 2000. The error contours are at $f = 0, \pm 5, \pm 10, \pm 25, \pm 50$ and ± 100.

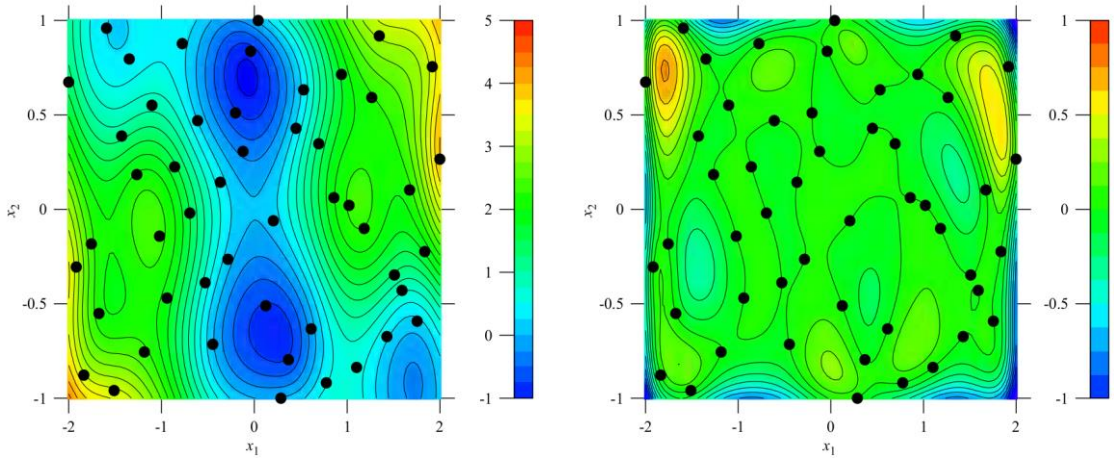
Although the number of training data points, the choice of surrogate model and the choice of analytical function is arbitrary for this illustration, several conclusions can still be drawn. Perhaps the most obvious is that all the training data points lie on contours of zero error due to the interpolation surrogate model used. An approximation model would have minimum errors close to the data points, but not necessarily zero. Another obvious trend is that the maximum errors are on the edges of the domain where there are fewer data points to influence the surrogate model, highlighting one drawback of surrogate modelling. Naturally, errors are larger where there is a lower concentration of points. Clearly 50 data points are sufficient to capture the essence of these functions, if somewhat inaccurately.



(A) Example A. Left: prediction. Right: error.

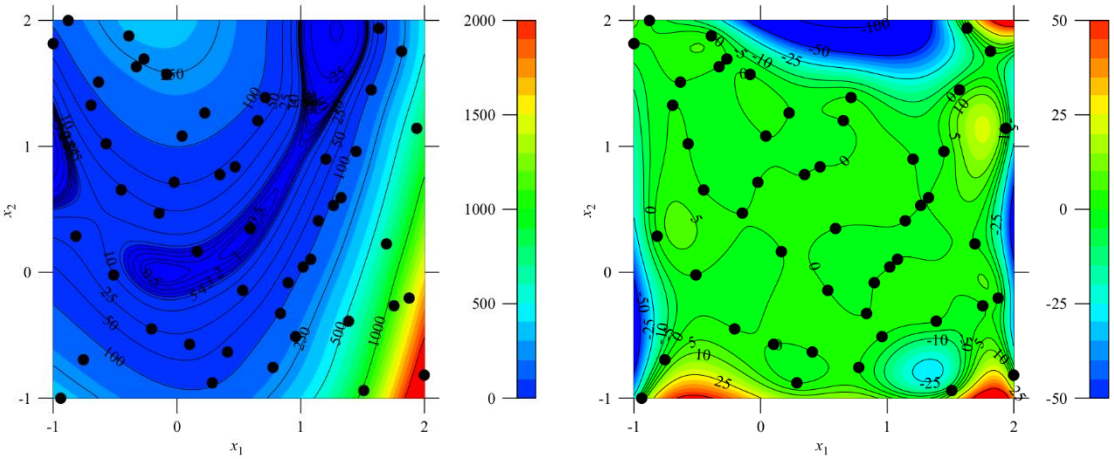


(A) (B) Example B. Left: prediction. Right: error.

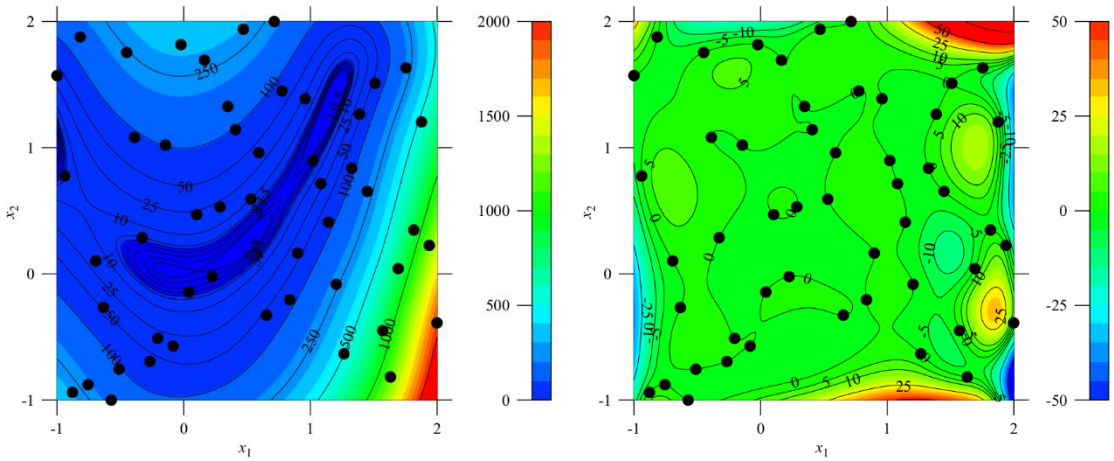


(A) (C) Example C. Left: prediction. Right: error.

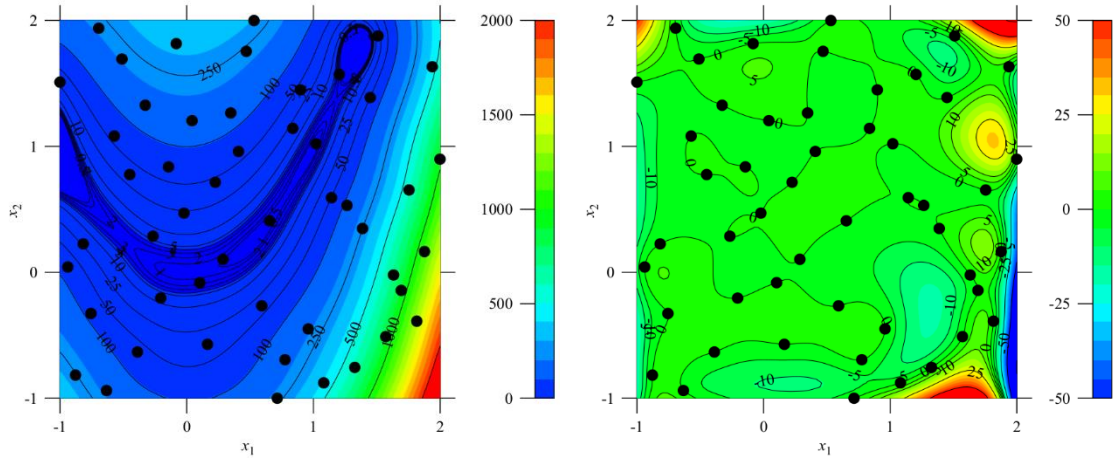
Figure 5.6: Surrogate model and error contour plots based on three 50 point RLHS predicting the SHCB function.



(A) Example A. Left: prediction. Right: error.



(B) Example B. Left: prediction. Right: error.



(C) Example C. Left: prediction. Right: error.

Figure 5.7: Surrogate model and error contour plots based on three 50 point RLHS predicting the RB function.

Despite having the lowest potential of the three DoEs, the surrogate model based on DoE Example C is the only model to over-predict the number of local minima for the SHCB, incorrectly showing seven whereas the other two examples both show six reasonably close to the true locations. The surrogate model based on DoE Example A, on the other hand, vastly under-predicts the global minimum for the RB.

To produce the contour plots, each coordinate axis is divided into 100, with the values of the surrogate model and associated errors determined at 100^2 locations, effectively creating an output lattice of data points. Clearly this method is adequate for graphical purposes, adjusting the size of this lattice as required, but it is severely limited for optimisation. For example, increasing the accuracy of the coordinates in the output lattice by one decimal place requires that the total number of evaluations increases by a factor of 10^p, quickly making this infeasible. Also worthy of consideration is the fact that discretising the domain then performing an optimisation routine may find a different minimum than optimising on a continuous domain. Further, for domains that are unequal in size (in the coordinate directions) and have not been normalised, as with the SHCB case, the Euclidean distances between output points are not equal (in the coordinate directions).

5.3.1 Optimising the Latin Hypercube

The aim of optimising the LH is to obtain a uniform DoE, otherwise known as an Audze-Eglais Latin Hypercube (AELH) [11, 13]. Intuitively, this can be achieved simply by rearranging the coordinates of the design points [88] and calculating the value of the designated spatial quantifier, U say. It is easily seen that 'brute-force' systematic rearranging of the design is not a feasible option for larger numbers of design parameters p (or larger numbers of design points n) as this method rapidly becomes too expensive [13]. For 2 design variables p and 5 training data points n, there are 52 possible positions for the first data point, 42 for the second and so forth, leading to a total of $5!^2$ (or $14{,}400$) possible designs. This generalises to $n!^2$ which rapidly becomes infeasibly large. Even if one of the parameters is fixed [13], systematic checking of

each and every design combination is not an option for large numbers of design variables or data points as the equation is only reduced to $n!^{p-1}$.

Another option is to use an optimisation algorithm to find a 'better', if not necessarily the best, DoE. One method of achieving this is to generate an initial population of designs, encode them, and then use a Genetic Algorithm (GA) [13, 14, 26, 59, 60] to find a better configuration. Genetic Algorithms follow four basic steps: initialisation, selection, reproduction and termination. Once a population size Q has been chosen, Q random LH designs can be generated. The fitness for each individual design is assessed using the potential energy U, (5.4). Designs which do not satisfy a criterion based on the fitness value are discarded whilst the remainder are selected to go through to the next generation and to reproduce further designs so that every generation has Q individual designs. The algorithm is terminated either when a specified number of generations is met or the designs meet a designated criterion.

Two possible methods for elite selection criteria are only allowing designs which are within a user specified percentage (10% say) of the minimum fitness value to pass through to the next generation or using the average fitness for the generation as a kill criteria. The latter method is based on a modified version of that presented in [102]: a design is allowed through to the next generation if it has a fitness value less than the average for the current generation. The fitness U_j for an individual design j is determined from the potential energy U, as defined in (5.4), whilst the average fitness for a generation U_{ave} is calculated from the sum of the individual finesses:

$$U_{ave} = \frac{1}{Q}\sum_{j=1}^{Q} U_j \tag{5.6}$$

Once the selected designs are through to the next generation, individual designs are chosen at random to become 'parents'. Each pair of parents produce one 'child' which is added to the generation. Thus each generation, apart from the initial one, consists entirely of parents from the previous generation and their children. When applying a GA to a LH DoE, it has been found, that due to the nature of the LH, permutations to the design are more suitable than encoding the design and using a binary method with penalties [14], as permutations ensure that the LH criteria are fulfilled at every stage, making the resulting GA inherently more efficient.

5.3.1.1 The 'PermGA' Method

The potential energy, U, of the DoE designs is to be minimised using a permutation GA. Permutating parent designs can be accomplished through mutation of a single design or swapping values between two parents. Bates *et al*. [14] found that 'cycle crossover' used in conjunction with a mutating 'inversion' provided the speediest solutions for the GA. The permutation technique is known as permGA [14, 102].

Cyclic Crossover Two parents are chosen from a population of designs. For example for a problem with 1 design parameter and 6 training points we could have:

Parent A = $[2\ 5\ 4\ 6\ 1\ 3]$

Parent B = $[1\ 4\ 5\ 3\ 6\ 2]$.

The first entry in Parent A becomes the first entry in Child A:

Child A = $[2 * * * * *]$.

Next, the value from the first entry in Parent B is located in Parent A and inserted into Child A. Here the value is 1 and is the fifth entry in Parent A (and consequently in Child A):

Child A = $[2\ * * *\ 1\ *]$.

In this example, the fifth entry in Parent B has a value of 6, which is the fourth entry in Parent A. Similarly, the fourth entry in Parent B is 3, which is the sixth entry in Parent A, giving Child A as:

Child A = $[2 * *\ 6\ 1\ 3]$.

The sixth entry in Parent B has a value of 2, which is located in the first entry of Parent A, thereby ending the cycle. The remaining entries in Child A are filled with the corresponding entries from Parent B. The LH criteria are fulfilled as the cycle ensures that no data points are repeated in any given dimension. A second child can be generated using this method with the first entry from Parent B as a starting point.

Child A = $[2\ 4\ 5\ 6\ 1\ 3]$,

Child B = $[1\ 5\ 4\ 3\ 6\ 2]$.

This crossover method is applied to each design variable independently. There may be instances where the first entry for a design variable for each of the parents is identical, in which case no crossover occurs and Child A = Parent A (similarly Child B = Parent B) for that design variable. The example below shows a two design variables and six training data points, where the LH values for the first design variable are identical for both parents (the second variables take the values described in the previous example):

Parent A = $\begin{bmatrix}1\ 2\ 3\ 4\ 5\ 6\\2\ 5\ 4\ 6\ 1\ 3\end{bmatrix}$ and Parent B = $\begin{bmatrix}1\ 2\ 3\ 4\ 5\ 6\\1\ 4\ 5\ 3\ 6\ 2\end{bmatrix}$

Child A = $\begin{bmatrix}1\ 2\ 3\ 4\ 5\ 6\\2\ 4\ 5\ 6\ 1\ 3\end{bmatrix}$ and Child B = $\begin{bmatrix}1\ 2\ 3\ 4\ 5\ 6\\1\ 5\ 4\ 3\ 6\ 2\end{bmatrix}$

It need not be the case that all the values for the first design variables are identical, if just the primary entries for each design variable are identical in both parents, then no cycle crossover occurs. As such, this method cannot guarantee that the children produced are different from their parents.

Inversion Mutation (IM) Two ‘cut-off’ points are randomly chosen, inverting the values in the parent design to produce the single offspring. For cut-off points 2 and 5 say, represented by $||$ and applied to Parent A, we obtain

Parent A = $[2\ 4||5\ 6\ 1||3]$,

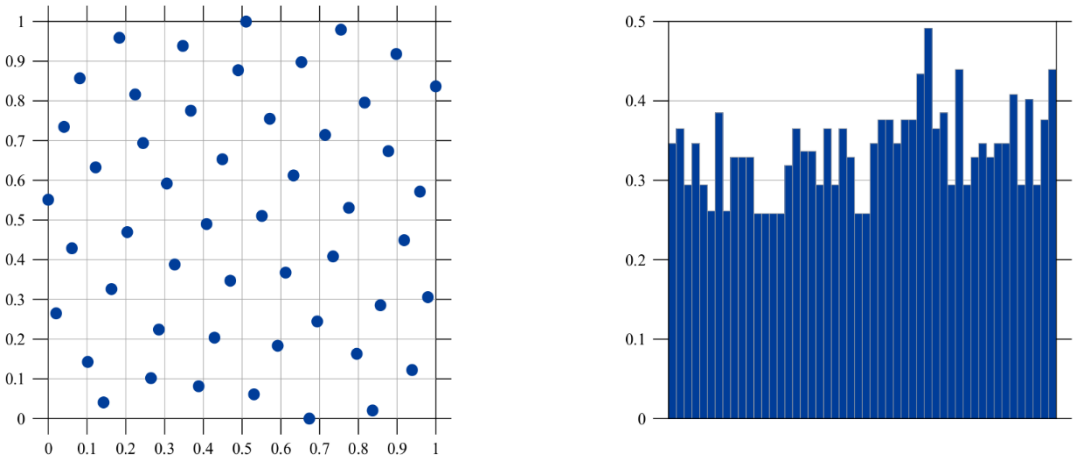
Child C = $[2\ 4||1\ 6\ 5||3]$.

In practice, this is applied not to a parent, but as a transformation applied to a child generated from the cycle crossover described above. The cut-off points are allocated randomly for each design variable, with the restriction that the cut-off points are not in the same location. Hence this method does guarantee that an identical design is not obtained as a result of inverting the original design.

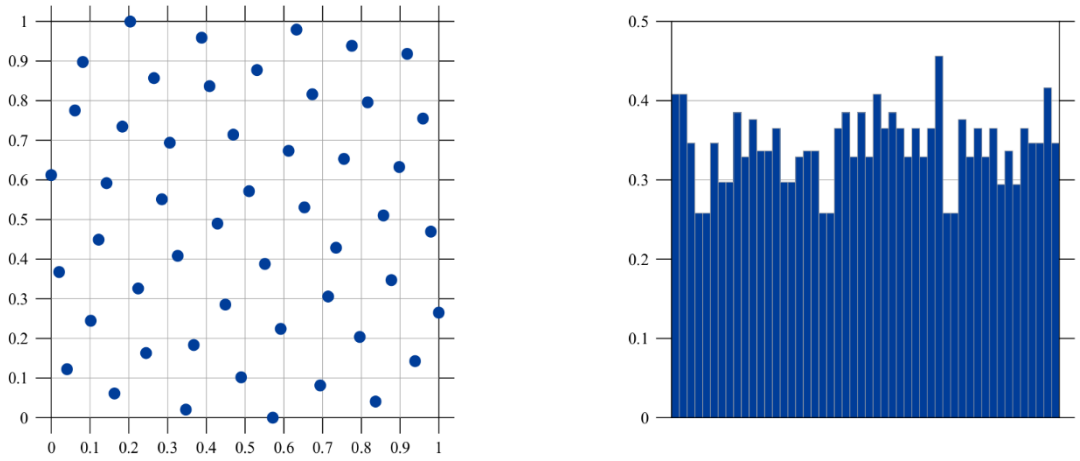
5.3.1.2 Effect of Uniform Spatial Coverage

In addition to running the permGA for the validation parameters, near optimal DoEs were also generated for $p = 2; n = 50$ for direct comparison with the RLHS shown in Figure 5.4. The point distributions and minimum distance plots for these three examples are shown in Figure 5.5. Visual comparisons between Figures 5.4 and 5.13 show that a greater uniformity of spatial coverage has been achieved using the permGA method than any of the original three RLHS.

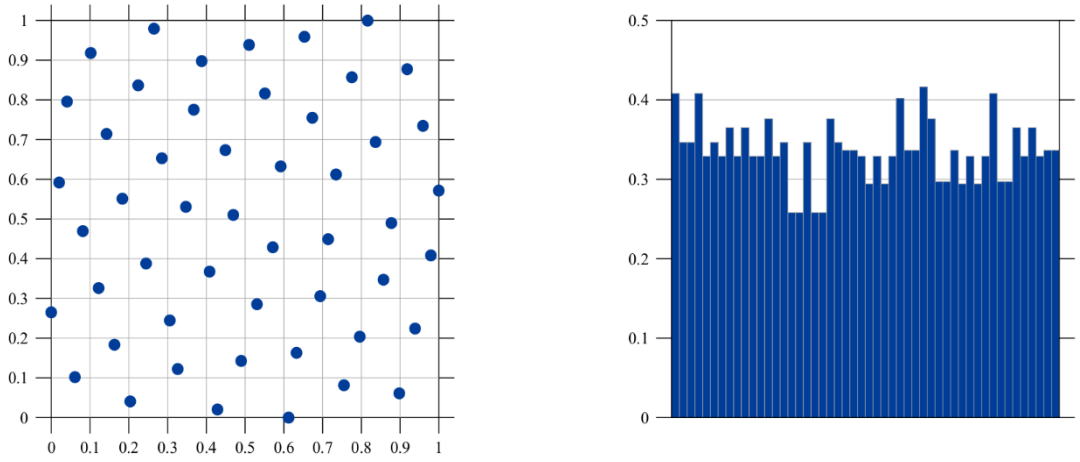
As before, a cubic based RBF was used to build a surrogate model approximation for both the SHCB function and RB function. Clearly both test functions have different requirements, in particular the RB approximations would greatly benefit from iterative improvement in the trough area. However, a more uniform distribution of training data points improves all the approximations for the two test functions. Other factors such as choice of surrogate model (including parameters) and number of data points also influence the final approximation.



(A) Example A.

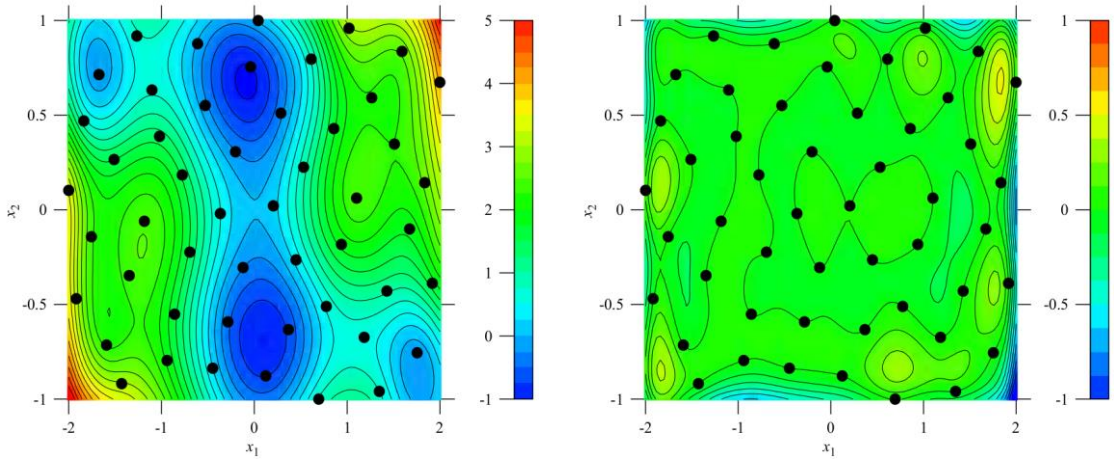


(B) Example B.

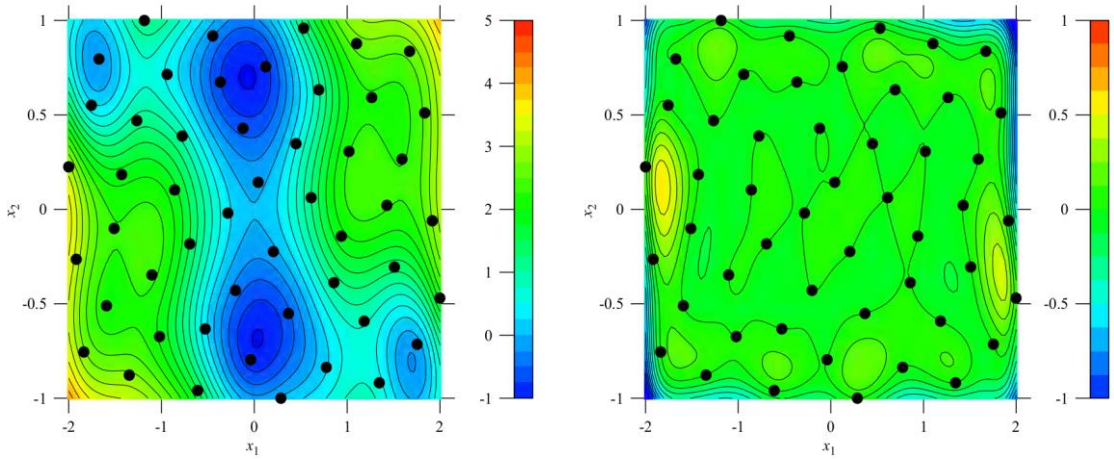


(C) Example C.

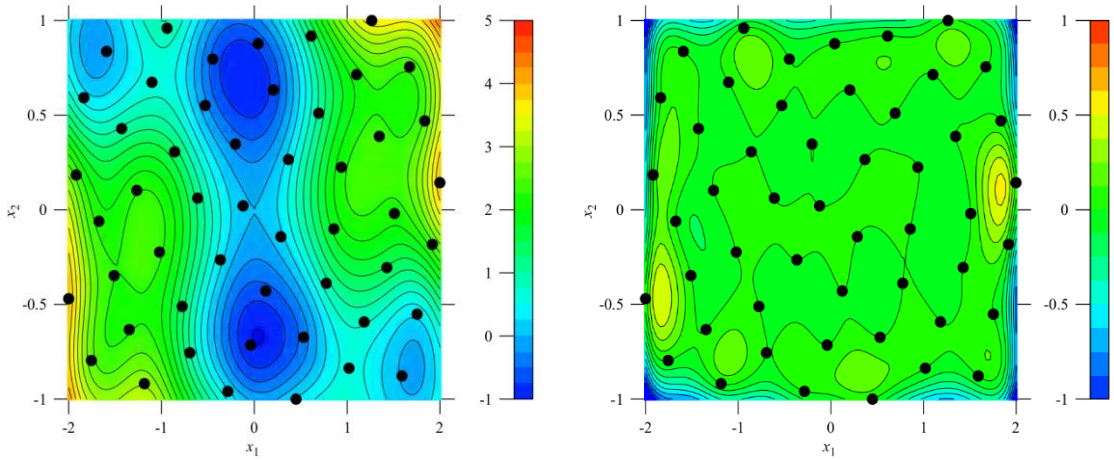
Figure 5.8: Optimised LHS DoE with 50 data points in 2 dimensions and corresponding minimum distance plots.



(A) Example A. Left: prediction. Right: error.

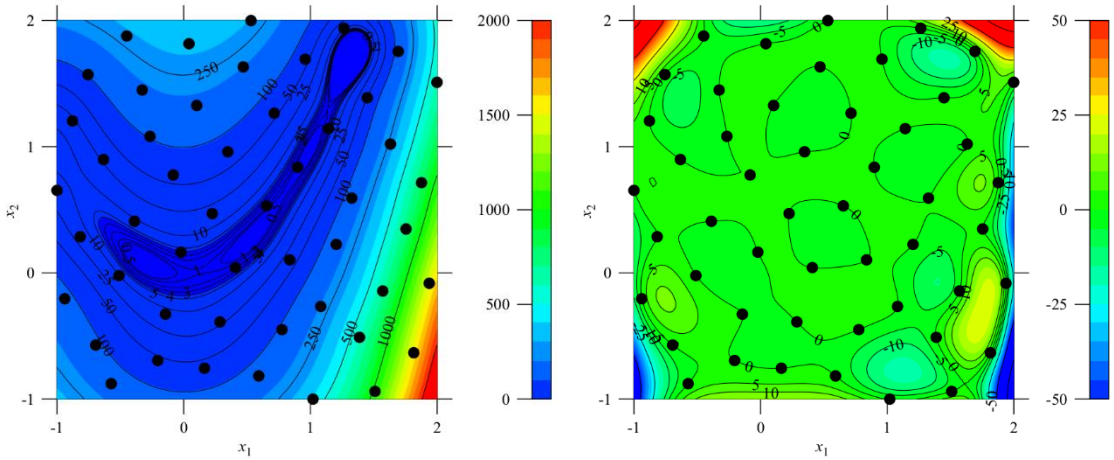


(B) Example B. Left: prediction. Right: error.

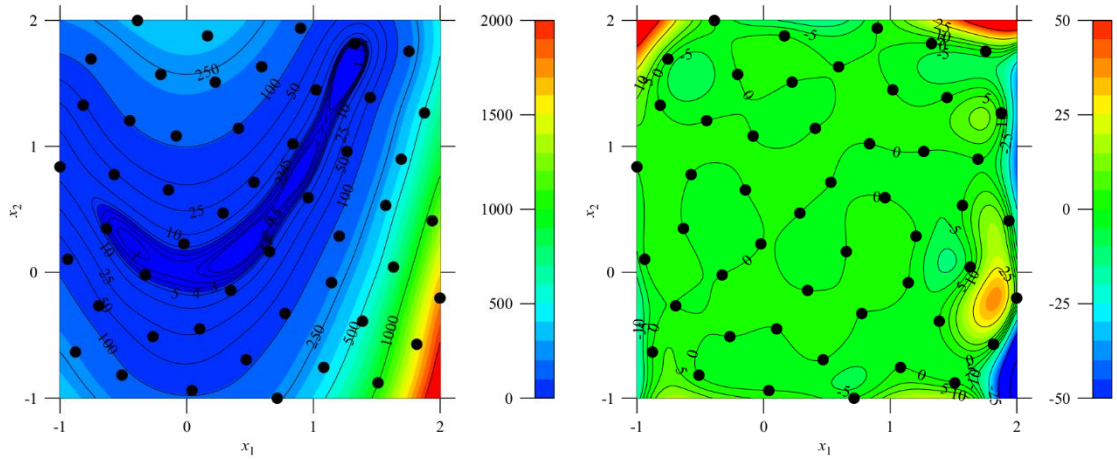


(C) Example C. Left: prediction. Right: error.

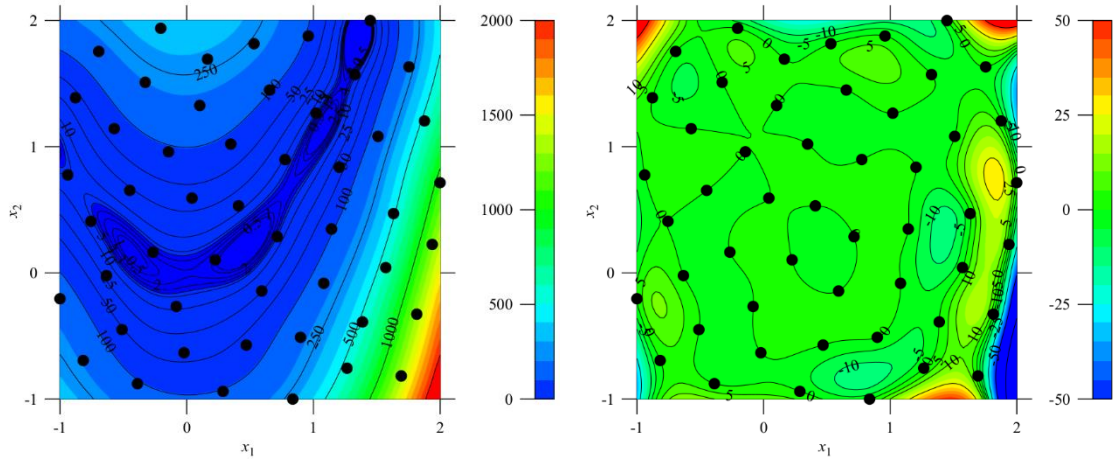
Figure 5.9: Surrogate model and error contour plots based on three 50 point OLHS predicting the SHCB function.



(A) Example A. Left: prediction. Right: error.



(B) Example B. Left: prediction. Right: error.



(C) Example C. Left: prediction. Right: error.

Figure 5.10: Surrogate model and error contour plots based on three 50 point OLHS predicting the RB function.

5.3.2 Simultaneous Generation of Initial and Validation OLH DoEs

Analytical test functions are known *a priori* making validation of surrogate models trivial, as the test functions can be evaluated at any number of data points in the parameter domain. Clearly this is not the case for more expensive functions where a more sophisticated method is required. In practical engineering applications, the process of surrogate model fitting includes the initial build stage and validation of the model. The validation is usually problem specific, dependent on the required accuracy, and must be valid throughout the design parameter domain [102]. As such, in addition to an OLH DoE for the initial build points, the validation points should also meet OLH DoE criteria. Further, subsequent to a successful validation exercise, a refined surrogate model based on the combined build and validation DoE points requires that the merged DoEs also exhibit space filling properties, as proposed by Narayanan et al. [102].

The total number of levels, n, in the DoE is split into build, b, and validation, v, levels such that $n = b + v$. The number of validation levels can be varied according to the problem in question. Figure 3.16 illustrates a simple case with a total of $n = 7$ for $p = 2$, where the build

and validation levels are $b = \{1,3,5,7\}$ and $v = \{2,4,6\}$ respectively. The (blue) build points are free to allocate any of the intersections of the solid lines, whilst the (red) validation points are confined to the intersections of the dashed lines. The build and validation sections of the chromosomes are generated separately for the initial population. The fitness of a design, U_{fit}, is given as a multi-objective function of the individual fitness of the build, validation and merged DoEs:

$$U_{fit} = f\left(U_b, U_v, U_m\right) \tag{5.7}$$

Each generation is ranked according to the multi-objective fitness function, those designs whose overall fitness are less than the average fitness for that generation become the elite designs for the next, as with the single objective fitness function for the basic permGA and in line with strategy presented in [102]. The parents are chosen in the same way as in the basic permGA with tournament selection and a weighted roulette wheel. The cycle crossover for generating children preserves the build and validation levels and are therefore applied independently to each.

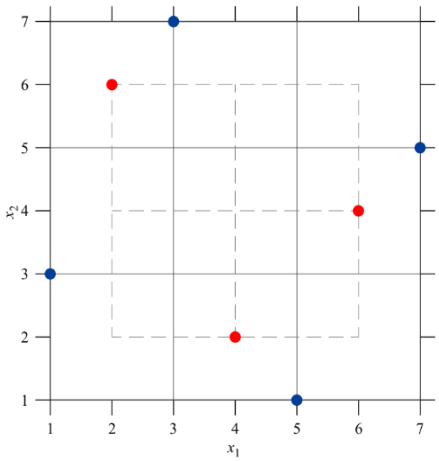
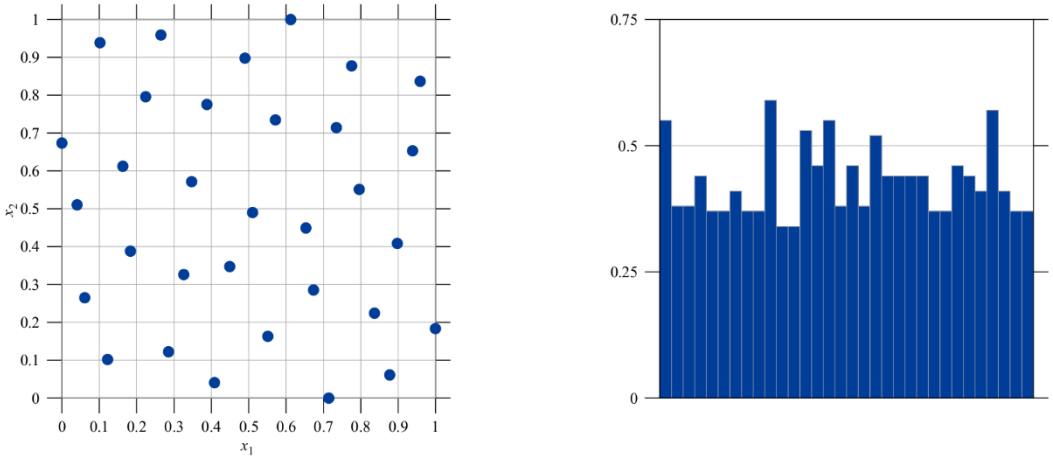
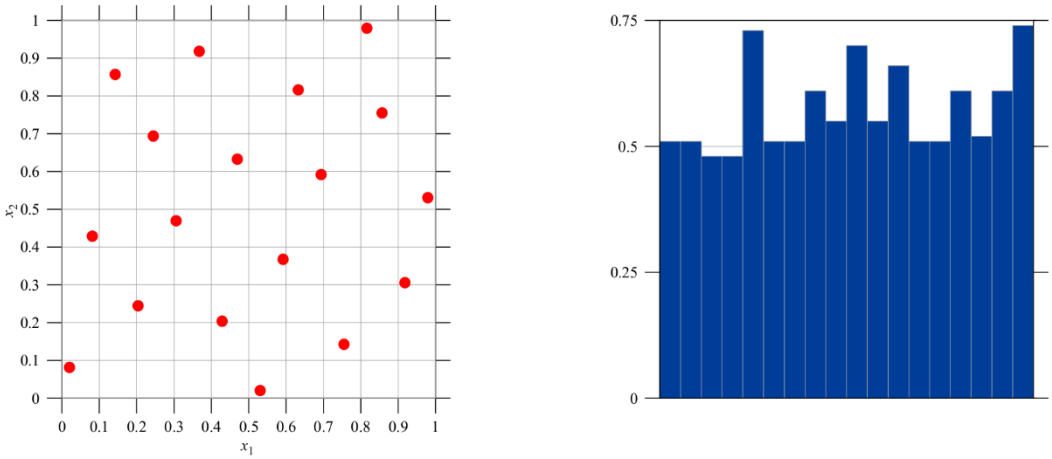


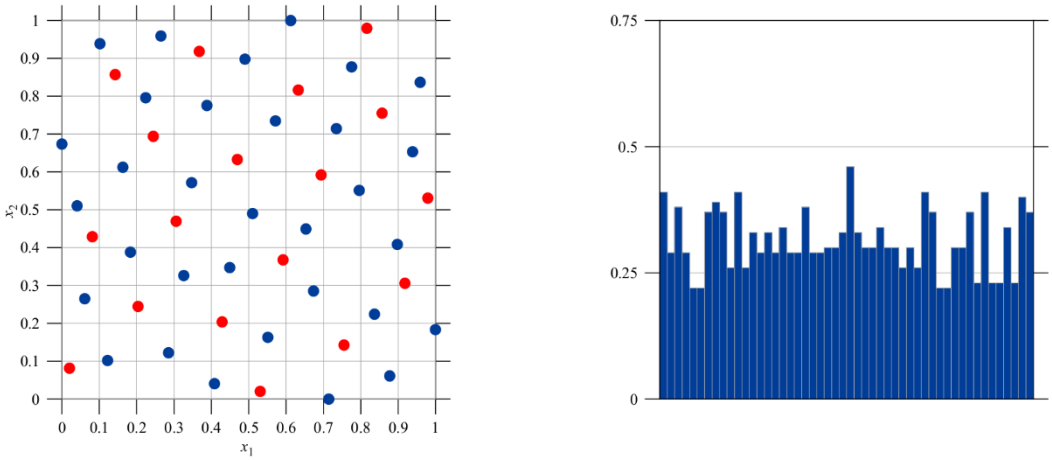
Figure 5.11: Illustration of build and validation levels.



(A) Build DoE.



(B) Validation DoE.



(C) Merged DoE.

Figure 5.12: Example BVM with b = 32 and v = 18. Left: Point distributions. Right: Minimum distances.

5.3.3 Inclusion of Corner Points

Section 5.2 introduces various sampling techniques for obtaining training data points whose computer responses can be used with a surrogate model to provide an approximation of the response surface. Whilst some methods do not meet the space-filling criteria required for DoE for surrogate models they can provide a cheap overview for determining which of the design variables warrant further investigation, which will inevitably lead to data points with useful response information. Due to the computational expense of acquiring the response data it would be desirable to reuse this information by incorporating it into the final design, however, this is likely to mean that the final design is not technically a Latin Hypercube. Toropov et al. [150] simply call these types of designs ‘Extended Latin Hypercubes’ (ELH).

A weakness of LHS is that it is not possible to have training data points located at each ‘corner’ of the domain [150]. Simpson et al. [130] find that orthogonal arrays, which have points located in the corners of the design domain, have lower values of maximum errors [73]. This can be seen in Figures 5.6, 5.7, 5.9 and 5.10 where the largest errors are at the edges and corner points of the domain. Leary et al. [86] find that the training data points need to go to edge of

domain, otherwise surrogate models methods fail as they are not designed for extrapolation. Thus, including the corner points should help to bound the surrogate model and hence improve accuracy. The technique used in [150] is to allow the fixed data points to be included in the calculation of the 'potential energy' objective function, but to exclude the points from the design variable set which is being modified to minimise the objective function.

One method to achieve this is to divide the domain into the extremities and the interior, where the LH requirement is relaxed on the boundaries only. The inner LH is self-contained, but optimised subject to the potential for the domain as a whole. Figure 5.13 shows an example in two dimensions with $n = 10$ training data points. The corners require four of these points ($n_c = 4$), shown in red, leaving a further six data points within the inner LH ($n_{LH} = 6$), shown in blue. To ensure equal spacing between the levels, the domain is divided into $(6 + 2)^2$ intersections. For the multidimensional case, the number of corner points increase rapidly as $n_c = 2^p$ whilst the number of points in the inner LH decrease as $n_{LH} = n - 2^p$ and the domain is divided into $(n - 2^{p-1})^p$ hypercubes. Due to the number of corner points rising rapidly, this method is restricted to $n_c < n_{LH}$ to ensure adequate spatial coverage.

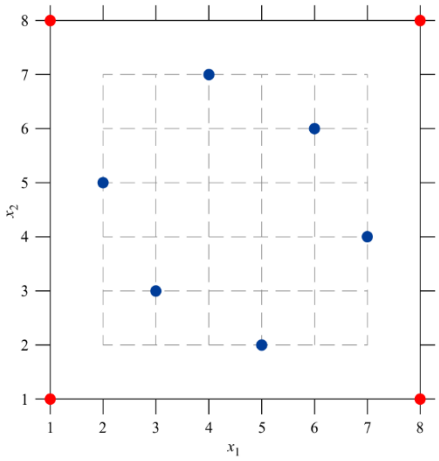


Figure 5.13: Illustration of fixed corner points and inner DoE in two dimensions.

Chapter 6

Surrogate Modelling Techniques

6.0 Introduction

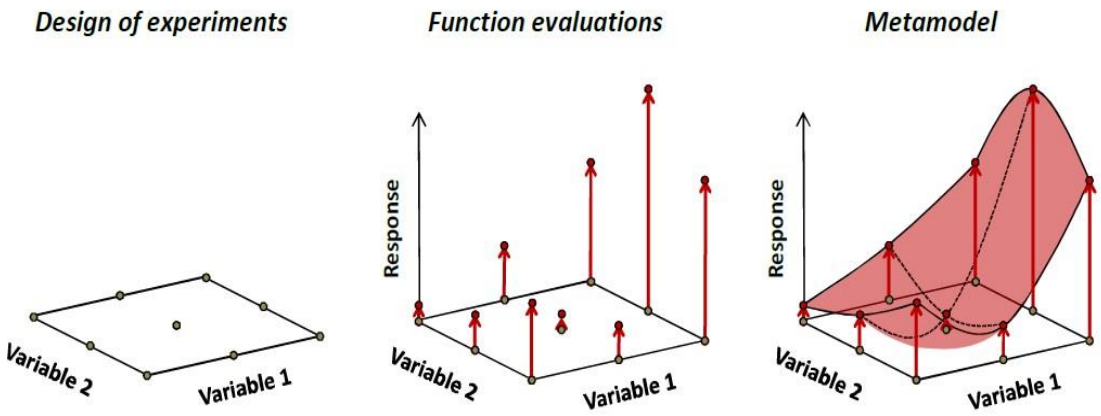
A **surrogate model** (which is also often called a **metamodel** or a **response surface model**) aims to mimic the response surface of some important output metric over the entire parameter domain based on the location and response information provided from a small (ideally minimal) number of experimentally- and/or computationally-generated training data points chosen from the domain. Whilst metamodels are not optimizing methods in themselves, they form a cheaper alternative to direct optimization on problems when obtaining data points is expensive, as is often the case with experiments or using a high fidelity computational simulations such as a CFD solve and post process. The goal is to produce a metamodel that is much faster to compute than the original function, but is still sufficiently accurate away from the known data points. This enables the optimization procedure to be carried out using the metamodel and the optimal result to be validated subsequently using experiments or a high fidelity computational evaluation.

This general approach is often referred to as **surrogate-based optimisation (SBO)**, whereas if the optimization is carried out using both the surrogate model and data obtained from the experiments or high fidelity simulations, this is termed **surrogate-assisted optimisation**.

The first step in creating the surrogate model is to generate the high fidelity data at a series of sampling, or Design of Experiment (DoE) points. Once sampling has been performed we have a list of data points called the training data $\{x^i, f^i\}$ for i=1,..,n, where x^i is the ith DoE point, f^i contains the corresponding high fidelity output at x^i and n is the number of DoE points.

Surrogate models can be based on i**nterpolation** or **regression**. Interpolation builds surrogate models that exactly matches the training data. Regression methods do not try to match training points exactly – they minimise the error between a smooth trend function and the training data. To fit a surface by means of regression, the criteria of passing the surface exactly through the data points is relaxed. The use of regression techniques (such as least squares methods discussed below) in determining surrogate surfaces can be explained by the origins of Response Surface Methods, which lie in the interpretation of experimental data [73]. Data obtained from physical experiments is noisy since all observations are subject to measurement error. Regression techniques allow for this noise as the surrogate surface does not pass through the data points, only close by. Exactly how close depends on tuneable parameters in the model, which will be referred to generically as **hyper-parameters** of the surrogate model, with larger parameters allowing for noisier data.

The surrogate modelling process for two design variables is illustrated in the following figure



6.1 Surrogate Model Validation

To ensure confidence, any surrogate model requires that its accuracy and quality be checked. One method is to validate against extra (potentially expensive) data. Other methods rely on **cross-validation** methods where the original DoE dataset is split up (often randomly to avoid bias errors) into training and testing sub-sets. Such approaches are needed to avoid **over-fitting** the surrogate model to the training dataset, as this results in the surrogate model being inaccurate at points not contained in the training dataset. In practice, each surrogate modelling technique will have a range of hyper-parameters associated with it and cross-validation methods enable the most appropriate hyper-parameters to be obtained.

Note that obtaining the hyper-parameters is often an extremely challenging optimization problem in itself!

6.1.1 Holdout Dataset Cross-Validation

One popular approach is to use most of the DoE data to train, or fit, the surrogate model and use the rest of the DoE points to test/validate the accuracy of the generated surrogate model. A common approach would be to use typically 70%-80% of the DoE dataset, chosen randomly from the full DoE dataset, to train the surrogate model and then use the surrogate model to predict the value of the output metric, f say, at the remaining DoE points that have not been used to construct the surrogate model.

The points used to train the surrogate model are referred to as the training dataset and those at which the accuracy of the surrogate model is determined are referred to as the testing dataset. If the full DoE dataset $\{x^i, f^i\}$ =1,…,n is split into *ntrain* training points $\{x^i_{train}, f^i_{train}\}$ and *ntest* testing points $\{x^i_{test}, f^i_{test}\}$ where n=ntrain+ntest, then the accuracy of the surrogate model at the testing data points can be quantified using metrics such as the Root Mean Square Error, RMSE, defined by

$$RMSE = \sqrt{\frac{1}{ntest}\sum_{i=1}^{ntest}(\tilde{f}^i - f_{test}^i)^2}$$

where $\tilde{f}^i$ is the value of the output from the surrogate model at the ith testing point and f_{test}^i is the (actual) value of the output at the DoE testing point. Clearly, smaller values of the RMSE indicate that the surrogate model is more accurate at the testing datapoints.

Note that, if possible, this process would be repeated a number of times to decrease bias errors resulting from specific train/test data splits.

The hyper-parameters of the surrogate model would be determined by minimising the RMSE during the training/testing cross-validation process.

6.1.2 Leave-One-Out Cross-Validation (LOOCV)

LOOCV is particularly useful when the DoE dataset is small so that it is possible to remove too many points from the surrogate model's training dataset. In this approach, the first DoE point is removed from the training dataset and the surrogate model is trained on the remaining (n-1) DoE points. The square of the difference between the surrogate model at this first DoE point $\tilde{f}^1$ and the (actual) value of the output at the first DoE point, f^1, $\left(\tilde{f}^1 - f^1\right)^2$ is stored. This process is repeated consecutively at each of the n DoE points and the total RMSE calculated via

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\tilde{f}^i - f^i)^2}$$

Once again, the hyper-parameters of the surrogate model would be determined by minimising the RMSE during the training/testing cross-validation process.

6.1.3. k-fold Cross-Validation

This process is similar to LOOCV but this time the DoE dataset is split randomly into k roughly equally-sized data subsets. The first subset is removed from the full DoE dataset and used as the first testing dataset. The surrogate model is then trained on the remaining (k-1) subsets and the RMSE calculated for the first testing dataset. This process is repeated over each of the k data subsets and an average value of the RMSE is calculated.

Once again, if possible, this k-fold randomisation process would be repeated a number of times to decrease bias errors resulting from specific train/test data splits. The hyper-parameters of the surrogate model would then be determined by minimising the RMSE during the training/testing cross-validation process.

6.1.4. Adding Extra Points into the DoE sample

Sometimes we find that the surrogate model is not accurate enough so it is necessary to include additional points in the DoE sample and re-construct the surrogate model – a process called **infill.** It is claimed that for an accurate estimate of a global optimum to be found, the surrogate surface must converge to the true surface at every point in the domain [69, 149]. Thus, any method for selecting additional data points to evaluate must also incorporate samples from untested regions in parameter space so the entire domain is adequately represented [54]. There are a number of methods for updating the DoE points. Whilst general surrogate modelling methods often have to use empirical approaches, or computationally expensive Bayesian techniques, some methods such as Kriging/Gaussian Process Regression can provide simple methods for estimating the errors in the surrogate model which can be used as a convenient guide for choosing the next DoE point to evaluate.

6.2 Least Squares Regression Surrogate Modelling

As noted above, regression can be very useful when the data is noisy since interpolation models may produce undesirable oscillations when filtering the noise.

6.2.1. Linear Least Squares Regression

If we have n DoE points giving a response f as a function of ndv design variables x_1, x_2, x_3,…, x_{ndv}, we can look for a polynomial fit of the data.

For example, a linear fit for three design variables we would fit the data to a hyper-plane of the form: $f = c_1+c_1x_1+c_2x_2+c_3x_3$. The goal is then to find the regression coefficients c_1, c_2, c_3, c_4. The regression coefficients are the *hyper-parameters* of the surrogate model in this case.

Least squares regression analysis seeks to minimise the sum of the squares of the differences (Square Errors, SE) between the data points and the fitted curve. For this example

$$SE = \sum_{i=1}^{n} \left(f^i - c_1 - c_2\, x_1^i - c_3 x_2^i - c_4 x_3^i\right)^2$$

where f^i is the response at the ith DoE point.

To find the regression coefficients c_i which minimise the SE we need to satisfy

$$\frac{\partial SE}{\partial c_1} = -2 \sum_{i=1}^{n} \left(f^i - c_1 - c_2\, x_1^i - c_3 x_2^i - c_4 x_3^i\right) = 0$$

$$\frac{\partial SE}{\partial c_2} = -2 \sum_{i=1}^{n} \left(f^i{}_i - c_1 - c_2\, x_1^i - c_3 x_2^i - c_4 x_3^i\right) x_1^i = 0$$

$$\frac{\partial SE}{\partial c_3} = -2 \sum_{i=1}^{n} \left(f^i - c_1 - c_2\, x_1^i - c_3 x_2^i - c_4 x_3^i\right) x_2^i = 0$$

$$\frac{\partial SE}{\partial c_4} = -2 \sum_{i=1}^{n} \left(f^i - c_1 - c_2\, x_1^i - c_3 x_2^i - c_4 x_3^i\right) x_3^i = 0$$

These lead to the four regression equations (a linear system):

$$n\, c_1 + c_2 \sum_{i=1}^{n} x_1^i + c_3 \sum_{i=1}^{n} x_2^i + c_4 \sum_{i=1}^{n} x_3^i = \sum_{i=1}^{n} f^i$$

$$c_1\sum_{i=1}^{n}x_1^i + c_2\sum_{i=1}^{n}\left(x_1^i\right)^2 + c_3\sum_{i=1}^{n}x_1^i x_2^i + c_4\sum_{i=1}^{n}x_1^i x_3^i = \sum_{i=1}^{n}f^i x_1^i$$

$$c_1\sum_{i=1}^{n}x_{2,i} + c_2\sum_{i=1}^{n}x_1^i x_2^i + c_3\sum_{i=1}^{n}\left(x_2^i\right)^2 + c_4\sum_{i=1}^{n}x_2^i x_3^i = \sum_{i=1}^{n}f^i x_2^i$$

$$c_1\sum_{i=1}^{n}x_3^i + c_2\sum_{i=1}^{n}x_1^i x_3^i + c_3\sum_{i=1}^{n}x_2^i x_3^i + c_4\sum_{i=1}^{n}\left(x_3^i\right)^2 = \sum_{i=1}^{n}f^i x_3^i$$

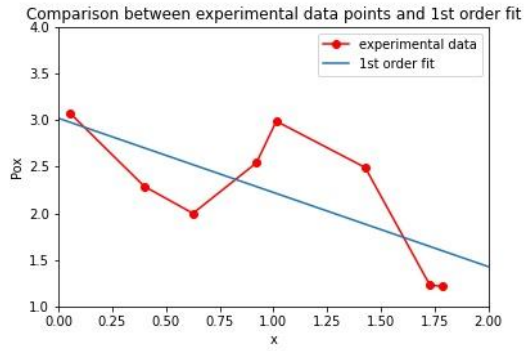
We solve these equations to obtain the regression coefficients c_1, c_2, c_3, c_4. this leads to the surrogate model: $f = c_1+c_2x_1+c_3x_2+c_4x_3$.

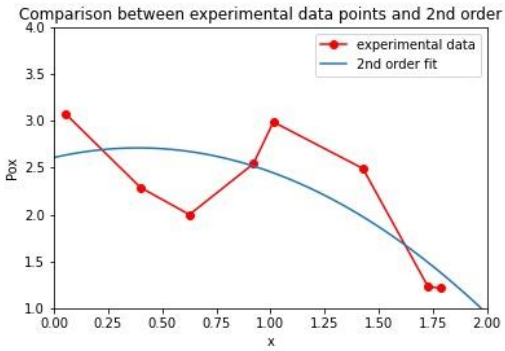
6.2.2. Engineering Example

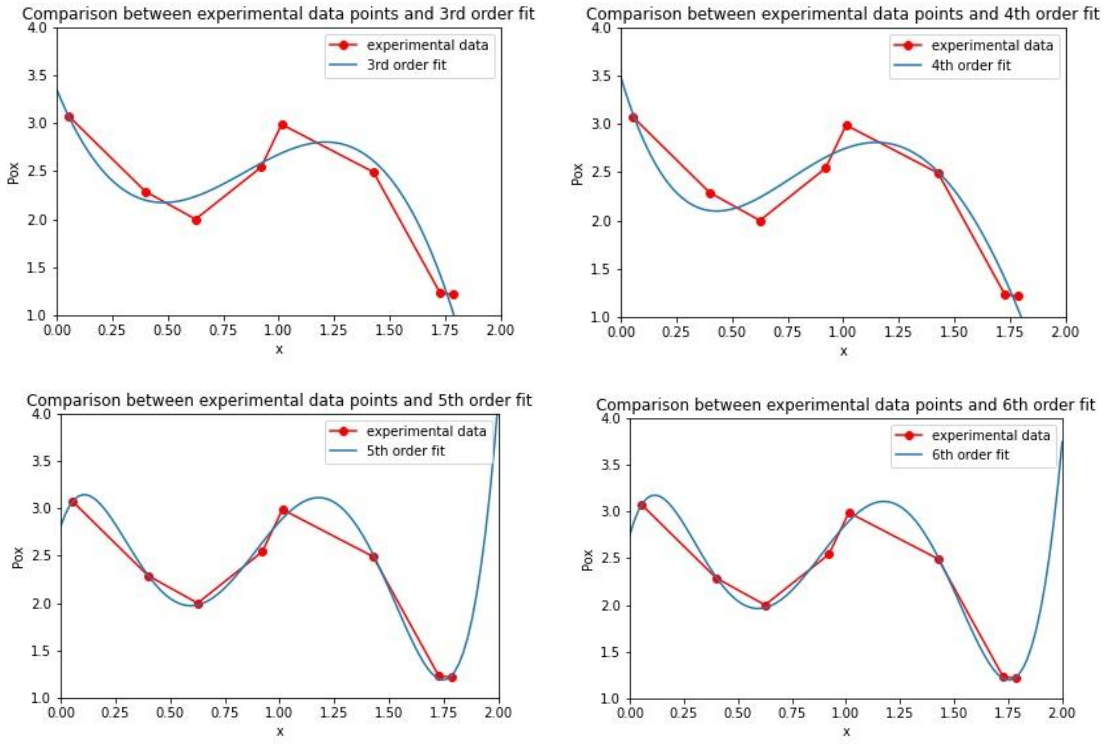
The following example explores the effect of the order of the polynomial regression model for representing data from an engineering company that produces algal photo-bioreactors to harvest phosphorous from wastewater. The company is trying to optimise the system so that they can extract the maximum amount of phosphorous (kg) which depends on the amount of light intensity ($watts$), the concentration of oxygen ($\frac{kg}{m\text{^}3}$) and biomass ($\frac{kg}{m^3}$). It is assumed that the phosphorus mass $P(ox)$ depends only on the concentration of oxygen ox. The dataset of Phosphorus mass as a function of oxygen concentration obtained from experiments is given by:

ox	P(ox)
0.054848447	3.070101933
0.399361417	2.288275883
0.625771597	1.998910871
0.920604333	2.538819248
1.013368979	2.985745038
1.426680967	2.489943212
1.727360442	1.230933862
1.786319956	1.215692053

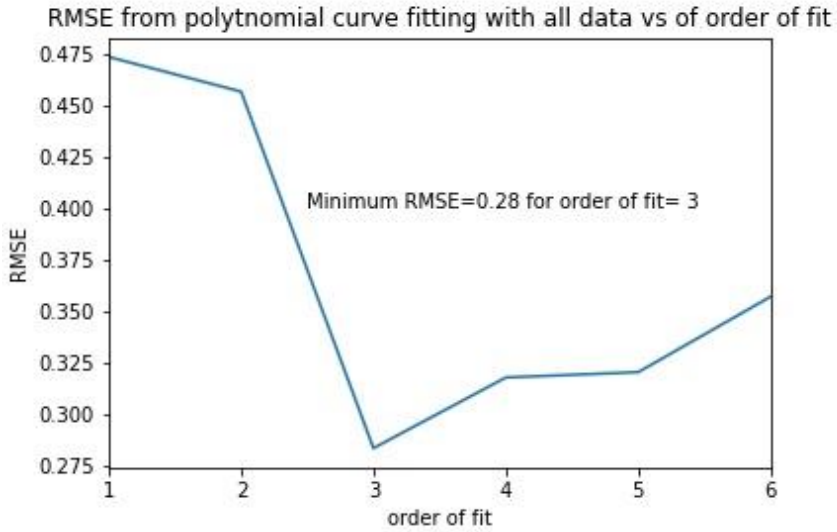
Least squares Regression fitting of P vs ox leads to the following regression curves:







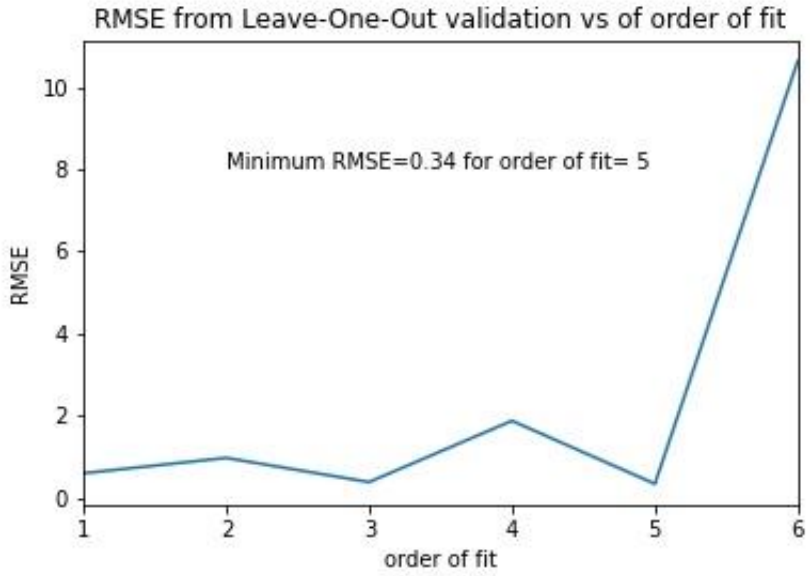
Calculating the RMSE between the predicted points, using the polynomial curve fitting, and the actual data points points yields the following:



These results are obtained by using the Python program, **using_alldata.py** on the **alldata** directory. These results suggest using a third order polynomial fit as this leads to the smallest RMSE error. However, this approach is not recommended and it is much more effective to use a cross-validation approach that splits the data into training and testing datasets in some way. This avoid the problem of **over-fitting** which can lead to the surrogate modelling being inaccurate at points not in the training dataset.

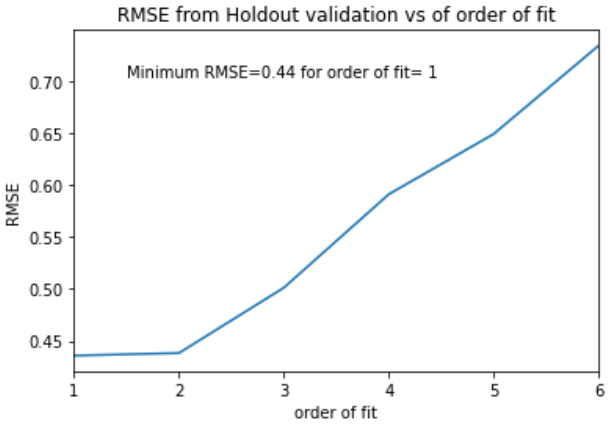
In this case, the only hyper-parameter of the surrogate model is the order of the polynomial. How do we choose the order of regression to use? If we use Leave-One-Out Cross Validation

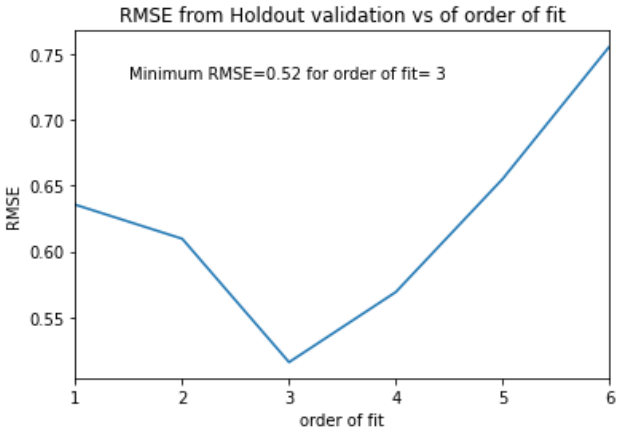
and calculate the RMSE for each regression order we obtain the following using the program **using_leaveoneout.py** on the **leave-one-out directory**, we obtain:



On the basis of LOOCV we would use a 5th order polynomial regression model for to represent P vs ox.

Suppose now that we use a holdout approach, where we divide the full dataset into a testing dataset consisting of two points selected randomly from the dataset of Phosphorus mass as a function of oxygen concentration obtained from experiments, given above, and a training dataset of the remaining six points. This algorithm is implemented in **using_holdout.py** on the **holdout** directory. Since the points are selected randomly, the solutions change every time the program is run. Here are two examples of results obtained by running using_holdout.py:





The first result on the left suggests a first order polynomial approximation is best, whereas the second result suggests a third order polynomial is needed. How should be proceed? The main problem is that our dataset is so small. A more effective approach for smaller datasets is to use **k-fold cross validation** described in section 6.1.3 above. You will explore k-fold cross validation in your optimization assignment described below.

OPTIMIZATION ASSIGNMENT

In this assignment you are asked to modify the codes you have been given above to explore what happens when you use 4-fold cross validation where you randomly selected 4 sets of 2 points from the experimental dataset given above. You will combine 3 of these sets of 2 points into a training dataset with 6 points with the remaining points forming the testing dataset.

You should:

a. Write a Python program that calculates the RMSE at the testing datapoints for polynomial curves of orders 1 to 6 and determines which polynomial order leads to the smallest RMSE. Use your program to explore how much variability there is in the selected polynomial order of fit.

b. Summarise your findings.

Only if you have the time and are interested in doing so:

c. Extend your Python program so that the 4-fold cross validation process is repeated a specified number of times. Explore how the variability in the selected polynomial order of fit depends on the number of times you repeat the 4-fold cross-validation.

Good luck!

Note: Once you have implemented your programs you can compare your findings with the **using-kfold1.py** and **using_kfold2.py** programs on the **kfold** directory.

You have now finished your assignments for this brief, introductory course. The following content is for information only, if you are interested in finding out more about surrogate modelling!

6.2.3. Second Order (Quadratic) Regression: example with 2 design variables

This uses the surrogate model:

$$\hat{f}(x) = c_1 + c_2 x_1 + c_3\, x_2 + c_4 x_1^2 + c_5 x_1 x_2 + c_6 x_2^2.$$

The Least Squares (LS) regression coefficients c_1, c_2, c_3, c_4, c_5 and c_6 at the output point $\{x^j\} = \{x_1^j,\ x_2^j\}$ are obtained by minimising the sum of the least squares, SE_j, over all the sampling points (x_1^i, x_2^i) defined by

$$SE_j = \sum_{i=1}^{n} \left(f^i - c_1 - c_2 x_1^i - c_3\, x_2^i - c_4 \left(x_1^i\right)^2 - c_5 x_1^i x_2^i - c_6 \left(x_2^i\right)^2 \right)^2$$

The LS coefficients are obtained by requiring that

$$\frac{\partial SE_j}{\partial c_1} = \frac{\partial SE_j}{\partial c_2} = \frac{\partial SE_j}{\partial c_3} = \frac{\partial SE_j}{\partial c_4} = \frac{\partial SE_j}{\partial c_5} = \frac{\partial SE_j}{\partial c_6} = 0.$$

$$\frac{\partial SE_j}{\partial c_1} = \sum_{i=1}^{n} \left(f^i - c_1 - c_2 x_1^i - c_3 x_2^i - c_4 \left(x_1^i\right)^2 - c_5 x_1^i x_2^i - c_6 \left(x_2^i\right)^2 \right) = 0$$

$$\frac{\partial SE_j}{\partial c_2} = \sum_{i=1}^{n} x_{1,i} \left(f^i - c_1 - c_2 x_1^i - c_3 x_2^i - c_4 \left(x_1^i\right)^2 - c_5 x_1^i x_2^i - c_6 \left(x_2^i\right)^2 \right) = 0$$

$$\frac{\partial SE_j}{\partial c_3} = \sum_{i=1}^{n} x_{2,i}\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

$$\frac{\partial SE_j}{\partial c_4} = \sum_{i=1}^{n} x_{1,i}^2\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

$$\frac{\partial SE_j}{\partial c_5} = \sum_{i=1}^{n} x_{1,i}x_{2,i}\left(f^i{}_i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

$$\frac{\partial SE_j}{\partial c_6} = \sum_{i=1}^{n} x_{2,i}^2\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

Solve equations for c_1, c_2, c_3, c_4, c_5 and c_6 and obtain the LS approximation

$$\hat{f}\left(x^j\right) = c_1 + c_2x_1^j + c_3x_2^j + c_4\left(x_1^j\right)^2 + c_5x_1^jx_2^j + c_6\left(x_2^j\right)^2$$

of f, at the output point $\{x_1^j\} = \{x_1^j,\ x_2^j\}$.

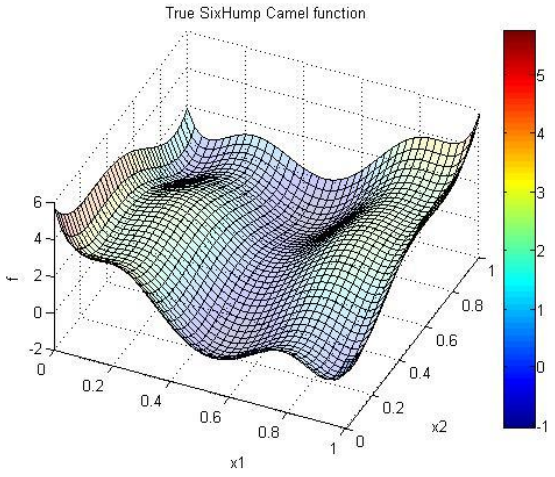
6.2.4. Six Hump Camel Back Function Examples

In the remainder of this chapter the surrogate modelling methods are demonstrated for the Six Hump Camel Back (SHCB) function defined in the previous chapter:

$$X_1 = 4x_1 - 2;\ X_2 = 2x_2 - 1;\ 0 \le x_1, x_2 \le 1$$

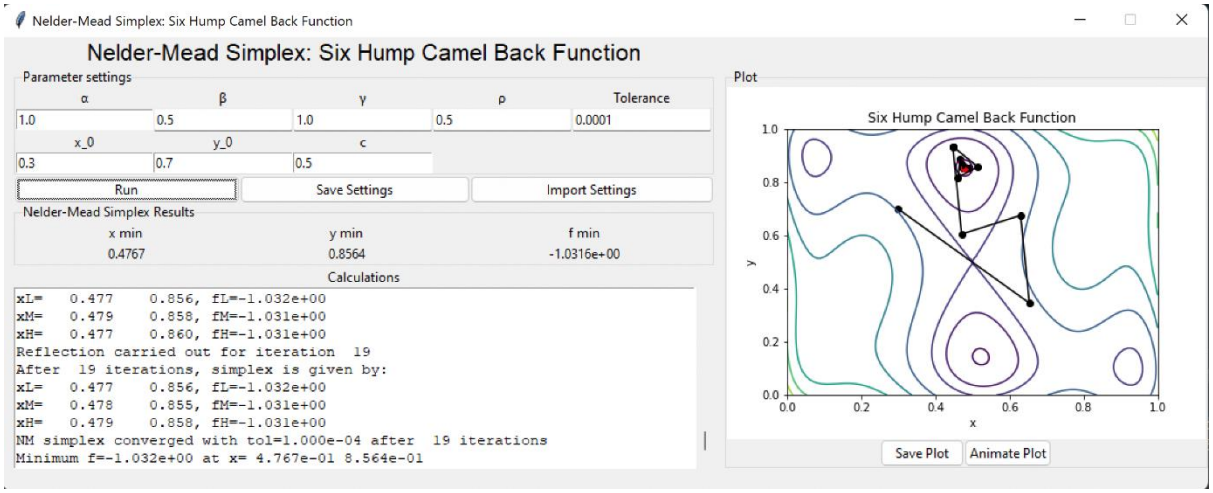
$$f(x_1, x_2) = (4 - 2.1{X_1}^2 + {X_1}^4/3)\ {X_1}^2 + X_1X_2 + (-4 + 4{X_2}^2)\ {X_2}^2$$

with the global minima having values of $f = -1.0316$ at $(0.0898, -0.7127)$ and $(-0.0898, 0.7127)$. The SHCB surface is given below:

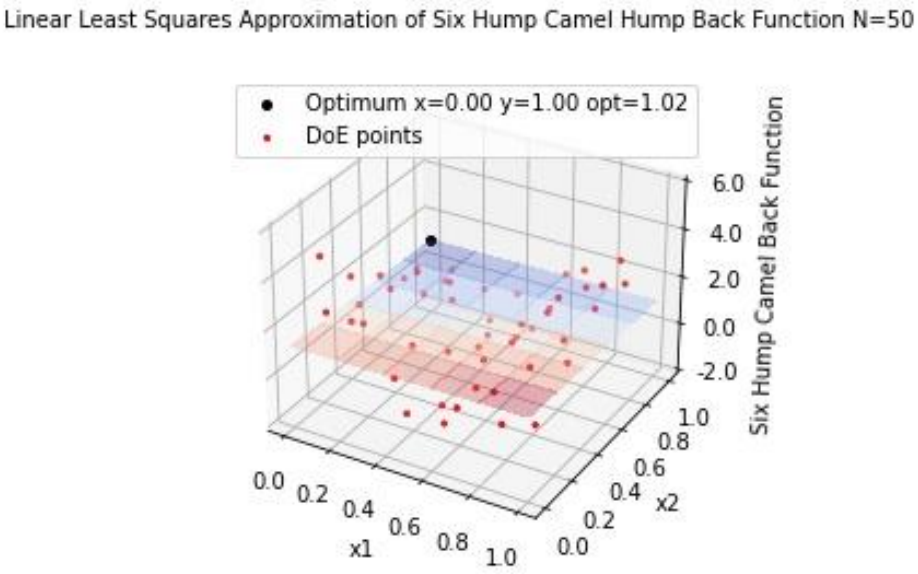


In the following examples, the surrogate modelling is used within an optimisation algorithm to determine the global optimum. For example, if the analytical solution is used within a Nelder-Mead Simplex optimisation algorithm in Python the following convergence to the optimum is observed when the initial point $x_0 = (0.3, 0.7)$, $\alpha = 1.0, \beta = 0.5, \gamma = 1.0, \rho = 0.5, c = 0.5$ and a

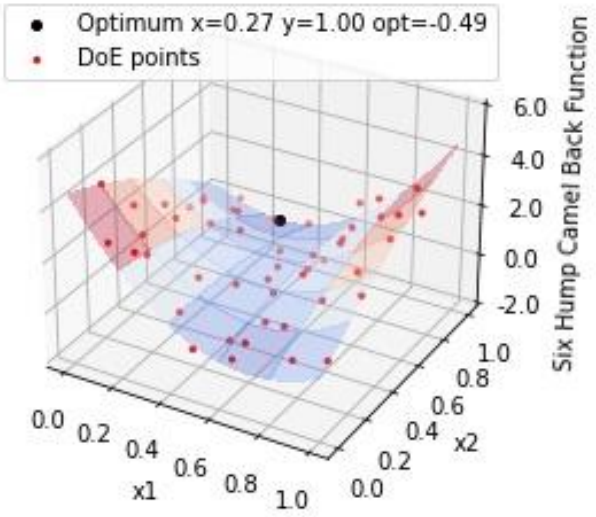
convergence tolerance of 0.0001. The path of the optimisation algorithm is shown as the solution meanders from (0.3,0.7) towards the optimum at (0.48,0.86) where the function value is -1.032.



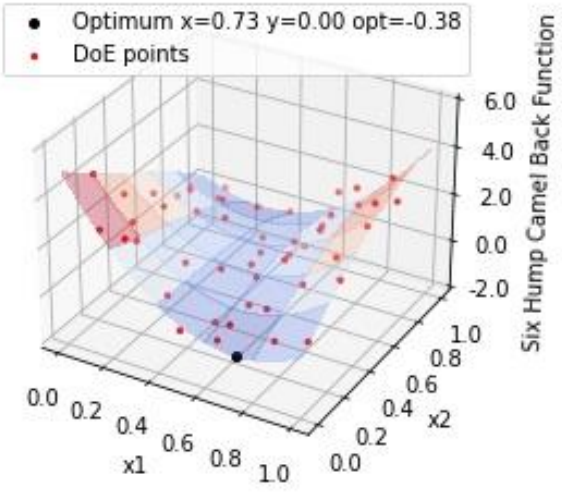
Examples of using Least Squares Regression for the SHCB function are given below:

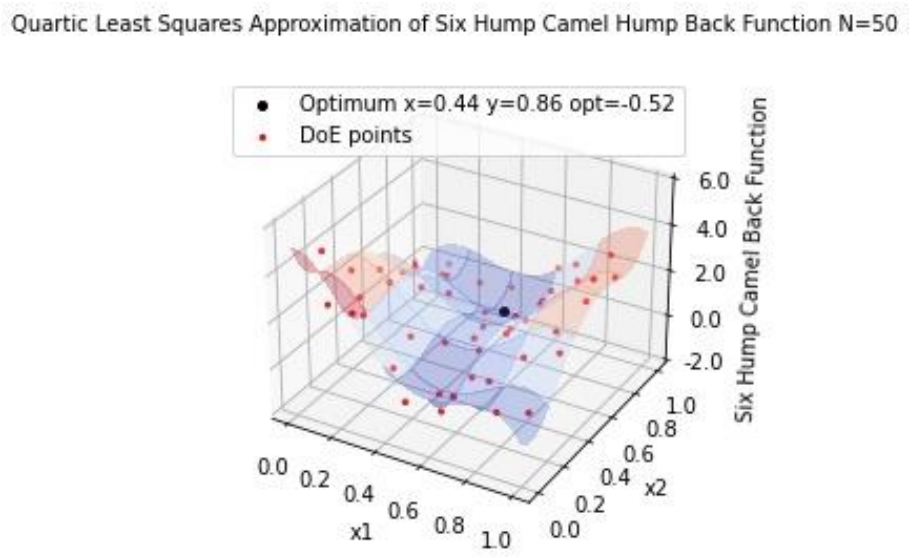


Quadratic Least Squares Approximation of Six Hump Camel Hump Back Function N=50



Cubic Least Squares Approximation of Six Hump Camel Hump Back Function N=50

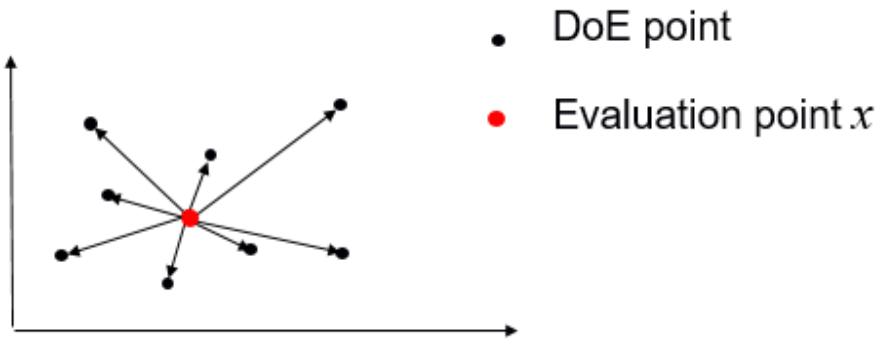




It can be seen that Least Squares Regression is not very effective in these examples.

6.3 Moving Least Squares Surrogate Modelling

Least Squares Regression can be extended to the **Moving Least Squares Method**. In this case, weights are applied which are functions of the Euclidian distance r_k from a *k*-th DoE sampling point to a point *x* where the surrogate model is evaluated.



One possible approach is to create a surrogate model estimate in the form

$$\hat{f}(x) = \sum_{i=1}^{n} w_i(\|x - x^i\|) f^i$$

where $r_i = \|x - x^i\|$ is the Euclidean norm between the point at which the surrogate model is being evaluated and the ith DoE sampling point.

6.3.1. First Order (Linear) Regression: example with 3 design variables
MLSM used to estimate the response, f, using a first order (linear) interpolation fit to the sampling points with three design variables.

This uses the surrogate modelling estimate $\hat{f}(x) = c_1 + c_2 x_1 + c_3\, x_2 + c_4 x_3$.

The MLSM regression coefficients c_1, c_2, c_3, c_4 at the output point $\{x^j\} = \{x_1^j,\ x_2^j,\ \ldots,\ x_{ndv}^j\}$ are obtained by minimising the sum of the least squares, SE_j, over all the sampling points $\{x^i\} = \{x_1^i,\ x_2^i,\ \ldots,\ x_{ndv}^i\}$, defined by

$$SE_j = \sum_{i=1}^{n} w_{ij}\left(f^i - c_1 - c_2 x_1^i - c_3\, x_2^i - c_4 x_3^i\right)^2$$

w_{ij} is the weight decay function between output point j and sample point i. For or a Gaussian weight decay function $w_{ij} = e^{-\beta r_{ij}^2}$ where $r_{ij} = \|x^j - x^i\|$ and β is the only hyper-parameter for the model. The MLS coefficients are obtained by requiring that

$$\frac{\partial SE_j}{\partial c_1} = \frac{\partial SE_j}{\partial c_2} = \frac{\partial SE_j}{\partial c_3} = \frac{\partial SE_j}{\partial c_4} = 0.$$

$$\frac{\partial SE_j}{\partial c_1} = 0 \Rightarrow 0 = \sum_{i=1}^{n}\left[-2w_{ij}\left(f^i - c_1 - c_2 x_1^i - c_3 x_2^i - c_4 x_3^i\right)\right] \Rightarrow$$

$$c_1 \sum_{i=1}^{n} w_{ij} + c_2 \sum_{i=1}^{n} w_{ij} x_1^i + c_3 \sum_{i=1}^{n} w_{ij} x_2^i + c_4 \sum_{i=1}^{n} w_{ij} x_3^i = \sum_{i=1}^{n} w_{ij} f^i$$

$$\frac{\partial SE_j}{\partial c_2} = 0 \Rightarrow 0 = \sum_{i=1}^{n}\left[-2w_{ij} x_1^i\left(f^i - c_1 - c_2 x_1^i - c_3 x_2^i - c_4 x_3^i\right)\right] \Rightarrow$$

$$c_1 \sum_{i=1}^{n} w_{ij} x_1^i + c_2 \sum_{i=1}^{n} w_{ij}\left(x_1^i\right)^2 + c_3 \sum_{i=1}^{n} w_{ij} x_1^i x_2^i + c_4 \sum_{i=1}^{n} w_{ij} x_1^i x_3^i = \sum_{i=1}^{n} w_{ij} f^i\, x_1^i$$

$$\frac{\partial SE_j}{\partial c_3} = 0 \Rightarrow 0 = \sum_{i=1}^{n}\left[-2w_{ij} x_2^i\left(f^i - c_1 - c_2 x_1^i - c_3 x_2^i - c_4 x_3^i\right)\right] \Rightarrow$$

$$c_1 \sum_{i=1}^{n} w_{ij} x_2^i + c_2 \sum_{i=1}^{n} w_{ij} x_1^i x_2^i + c_3 \sum_{i=1}^{n} w_{ij}\left(x_2^i\right)^2 + c_4 \sum_{i=1}^{n} w_{ij} x_2^i x_3^i = \sum_{i=1}^{n} w_{ij} f^i\, x_2^i$$

$$\frac{\partial SE_j}{\partial c_4} = 0 \Rightarrow 0 = \sum_{i=1}^{n}\left[-2w_{ij} x_3^i\left(f^i - c_1 - c_2 x_1^i - c_3 x_2^i - c_4 x_3^i\right)\right] \Rightarrow$$

$$c_1 \sum_{i=1}^{n} w_{ij} x_3^i + c_2 \sum_{i=1}^{n} w_{ij} x_1^i x_3^i + c_3 \sum_{i=1}^{n} w_{ij} x_2^i x_3^i + c_4 \sum_{i=1}^{n} w_{ij}\left(x_3^i\right)^2 = \sum_{i=1}^{n} w_{ij} f^i\, x_3^i$$

We then solve these equations for c_1, c_2, c_3 and c_4 and obtain the MLSM surrogate model estimate at design point

$$\hat{f}\left(x^j\right) = c_1 + c_2 x_1^j + c_3\, x_2^j + c_4 x_3^j$$

6.3.2. Second Order (Quadratic) Regression: example with 2 design variables
This uses the surrogate model:

$$\hat{f}(x) = c_1 + c_2 x_1 + c_3\, x_2 + c_4 x_1^2 + c_5 x_1 x_2 + c_6 x_2^2.$$

The MLS regression coefficients c_1, c_2, c_3, c_4, c_5 and c_6 at the output point $\{x^j\} = \{x_1^j,\ x_2^j\}$ are obtained by minimising the sum of the least squares, SE_j, over all the sampling points $(x_1^i,\ x_2^i)$ defined by

$$SE_j = \sum_{i=1}^{n} w_{ij}\left(f^i - c_1 - c_2 x_1^i - c_3\, x_2^i - c_4\left(x_1^i\right)^2 - c_5 x_1^i x_2^i - c_6\left(x_2^i\right)^2\right)^2$$

where w_{ij} is the weight decay function between output point j and sample point i. The MLS coefficients are obtained by requiring that

$$\frac{\partial SE_j}{\partial c_1} = \frac{\partial SE_j}{\partial c_2} = \frac{\partial SE_j}{\partial c_3} = \frac{\partial SE_j}{\partial c_4} = \frac{\partial SE_j}{\partial c_5} = \frac{\partial SE_j}{\partial c_6} = 0.$$

$$\frac{\partial SE_j}{\partial c_1} = \sum_{i=1}^{n} -2w_{ij}\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

$$\frac{\partial SE_j}{\partial c_2} = \sum_{i=1}^{n} -2w_{ij}x_{1,i}\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

$$\frac{\partial SE_j}{\partial c_3} = \sum_{i=1}^{n} -2w_{ij}x_{2,i}\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

$$\frac{\partial SE_j}{\partial c_4} = \sum_{i=1}^{n} -2w_{ij}x_{1,i}^2\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

$$\frac{\partial SE_j}{\partial c_5} = \sum_{i=1}^{n} -2w_{ij}x_{1,i}x_{2,i}\left(f^i{}_i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

$$\frac{\partial SE_j}{\partial c_6} = \sum_{i=1}^{n} -2w_{ij}x_{2,i}^2\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2\right) = 0$$

Solve equations for c_1, c_2, c_3, c_4, c_5 and c_6 and obtain the MLSM approximation

$$\hat{f}\left(x^j\right) = c_1 + c_2x_1^j + c_3x_2^j + c_4\left(x_1^j\right)^2 + c_5x_1^jx_2^j + c_6\left(x_2^j\right)^2$$

of f, at the output point $\{x_1^j\} = \{x_1^j,\ x_2^j\}$.

6.3.3. Higher Order Regression

For higher order regression, the number of regression coefficients increases rapidly. e.g. with two design variables $\{x\} = \{x_1,\ x_2\}$ the third order MLSM builds an approximation of the form:

$$\hat{f}\left(x^j\right) = c_1 + c_2x_1^j + c_3x_2^j + c_4\left(x_1^j\right)^2 + c_5x_1^jx_2^j + c_6\left(x_2^j\right)^2 + c_7\left(x_1^j\right)^3 + c_8\left(x_1^j\right)^2x_2^j + c_9x_1^j\left(x_2^j\right)^2 + c_{10}\left(x_2^j\right)^3$$

at the output point $\{x_1^j\} = \{x_1^j,\ x_2^j\}$. c_1-c_{10} are determined by minimising SE, summed over all n sampling points $(x_1^i,\ x_2^i)$:

$$SE_j = \sum_{i=1}^{n}\left(f^i - c_1 - c_2x_1^i - c_3x_2^i - c_4\left(x_1^i\right)^2 - c_5x_1^ix_2^i - c_6\left(x_2^i\right)^2 - c_7\left(x_1^i\right)^3 - c_8\left(x_1^i\right)^2x_2^i - c_9x_1^i\left(x_2^i\right)^2 - c_{10}\left(x_2^i\right)^3\right)^2$$

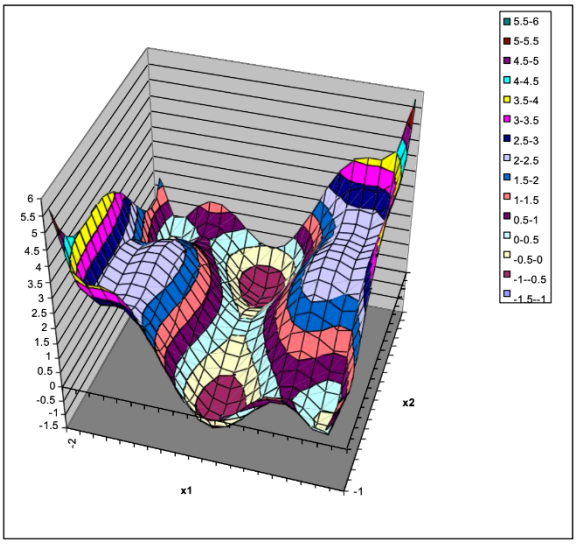
For two design variables $\{x\} = \{x_1,\ x_2\}$ the fourth order MLSM builds an approximation of the form:

$$\hat{f}\left(x^j\right) = c_1 + c_2x_1^j + c_3x_2^j + c_4\left(x_1^j\right)^2 + c_5x_1^jx_2^j + c_6\left(x_2^j\right)^2 + c_7\left(x_1^j\right)^3 + c_8\left(x_1^j\right)^2x_2^j + c_9x_1^j\left(x_2^j\right)^2 + c_{10}\left(x_1^j\right)^3 + c_{11}\left(x_1^j\right)^4 + c_{12}\left(x_1^j\right)^3x_2^j + c_{13}\left(x_1^j\right)^2\left(x_2^j\right)^2 + c_{14}x_1^j\left(x_2^j\right)^3 + c_{15}\left(x_2^j\right)^4$$

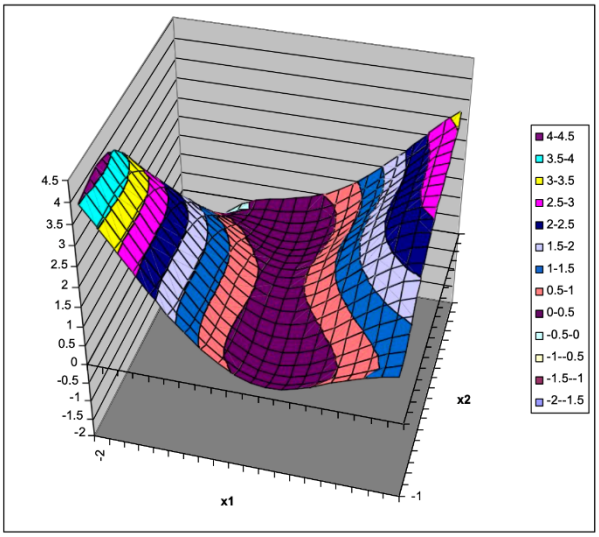
at the output point $\{x_1^j\} = \{x_1^j,\ x_2^j\}$. The regression coefficients c_1-c_{15} are determined by minimising the Moving Least Squares expression, summed over the n sampling points $(x_1^i,\ x_2^i)$ etc.

6.3.4. MLS surrogate modelling of the Six Hump Camel Back Function

Analytical Function

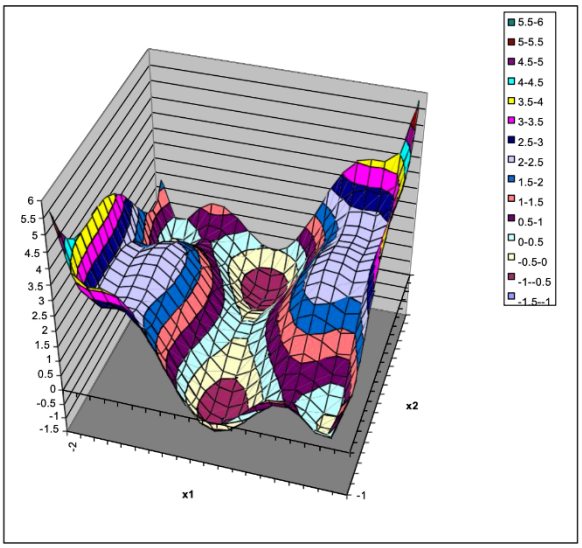


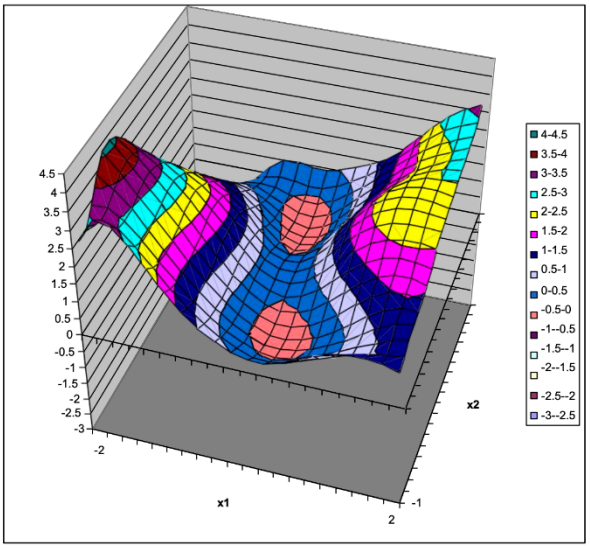
20 DoE points, β=10.0



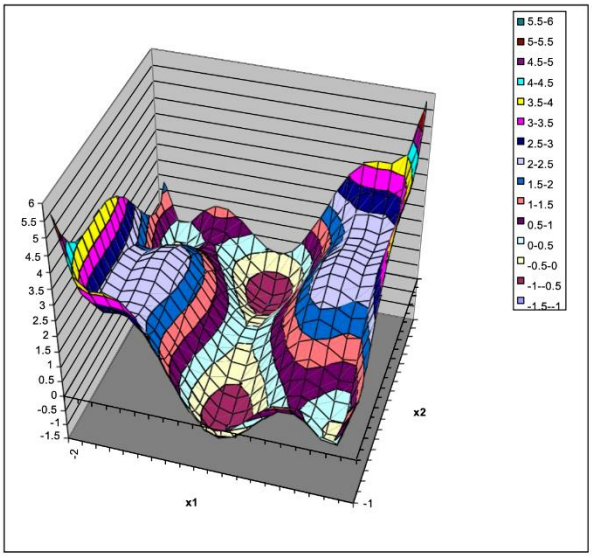
Analytical Function

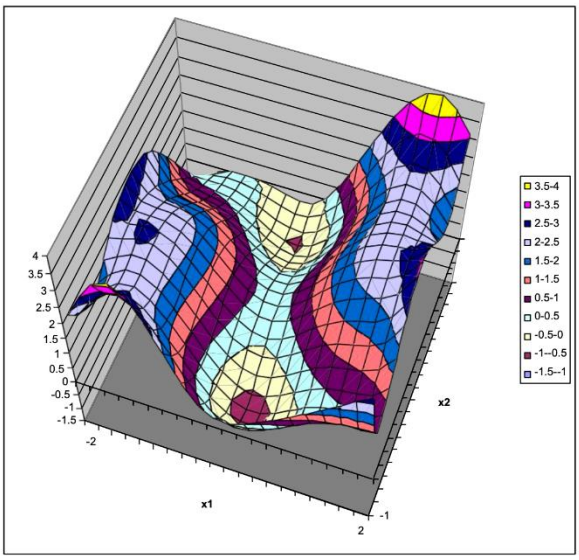
20 DoE points, β=20.0



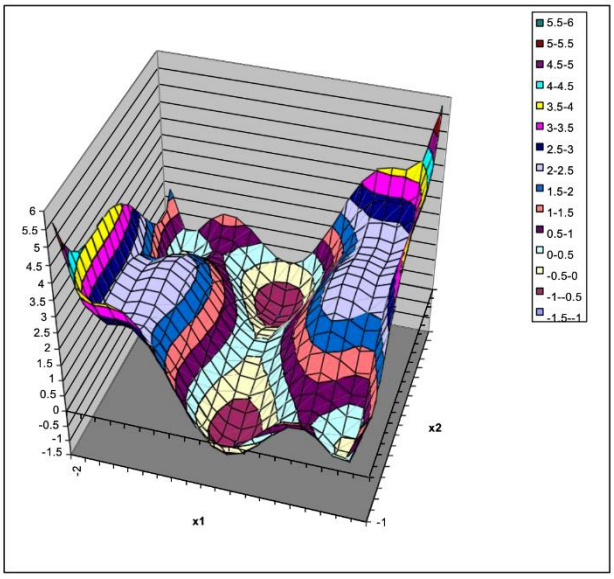


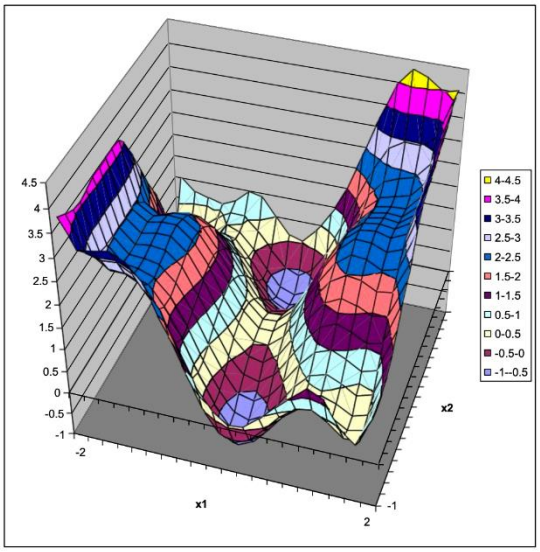
Analytical Function *50 DoE points, β=20.*



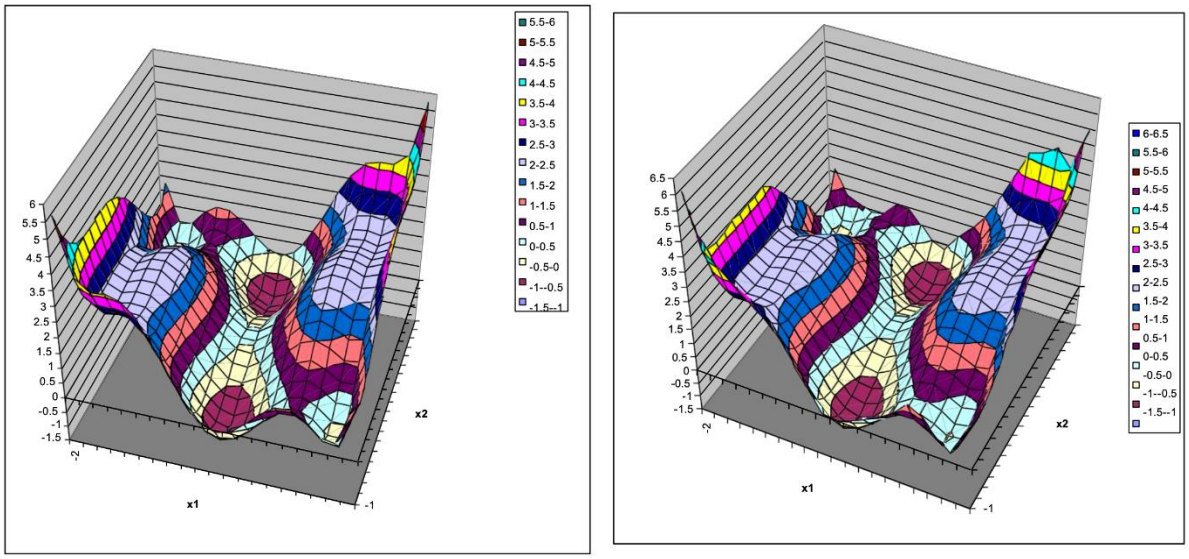


Analytical Function *100 DoE points, β=60.*





Analytical Function *100 DoE points, β=120.*



These show that the MLS method can represent the SHCB function accurately with ~50 DoE points and that the hyper-parameter β is also very influential.

6.4 Radial Basis Functions

In addition to surrogate modelling, radial basis functions (RBFs) are used in other areas, for example in computer graphics and mesh deformation. Initially requiring $O(n^3)$ calculations, a further $O(n)$ calculations are required per prediction [113].

As discussed earlier, the RBF surrogate model uses the data at the Design of Experiment points to create an approximation that is accurate everywhere in the design space. If there are *ndv* design variables and a total of *n* DoE points $\{x^i\} = \{x_1^i,\ x_2^i,\ \ldots,\ x_{ndv}^i\}$, for i=1,…,n at which the output response takes the values y^i for i=1,…,n then the RBF approximation to y_{rbf} at any point $x = \{x_1, x_2 \ldots, x_{ndv}\}$ with the design space is given by the RBF approximation

$$y_{rbf}(x) \approx \sum_{i=1}^{n} \lambda_i \, \psi\|x - x^i\| = \sum_{i=1}^{n} \lambda_i \, \psi(r_i) =$$

where $\{x^i\} = \{x_1^i,\ x_2^i,\ \ldots,\ x_{ndv}^i\}$ is the ith DoE point. The norm $r_i = \|x - x^i\|$ is often taken to be the Cartesian distance between the points, given by

$$r_i = \sqrt{\sum_{j=1}^{ndv} \left(x_j - x_j^i\right)^2}$$

where ndv is the number of design variables.

RBF basis functions ψ can take several different forms. Common examples include:

$$\psi(r) = r;\ \psi(r) = r^3;\ \psi(r) = r^2 \ln r$$

Gaussian: $\psi(r) = exp(-\beta r^2)$

Multi-quadratic: $\psi(r) = (r^2 + \beta^2)^{1/2}$.

Inverse multi-quadratic: $\psi(r) = (r^2 + \beta^2)^{-1/2}$.

where β is the single hyper-parameter. RBFs are normally used in *interpolating* mode which means that the weights λ_i are chosen so that the RBF approximation is exact at each of the DoE points, i.e. $y_{rbf}(x^i) = y^i$ for i=1,…,n. In this case, the λ_i are obtained by solving the linear matrix equation:

$$\lambda = \Psi^{-1} y$$

where Ψ is the Gram matrix defined such that $\Psi_{i,j} = \psi\|x^i - x^j\|$. If the Cartesian norm is used, then

$$\|x^i - x^j\| = r_{ij} = \sqrt{\sum_{k=1}^{ndv} \left(x_k^i - x_k^j\right)^2}$$

which is the Cartesian distance between the ith nd jth DoE point.

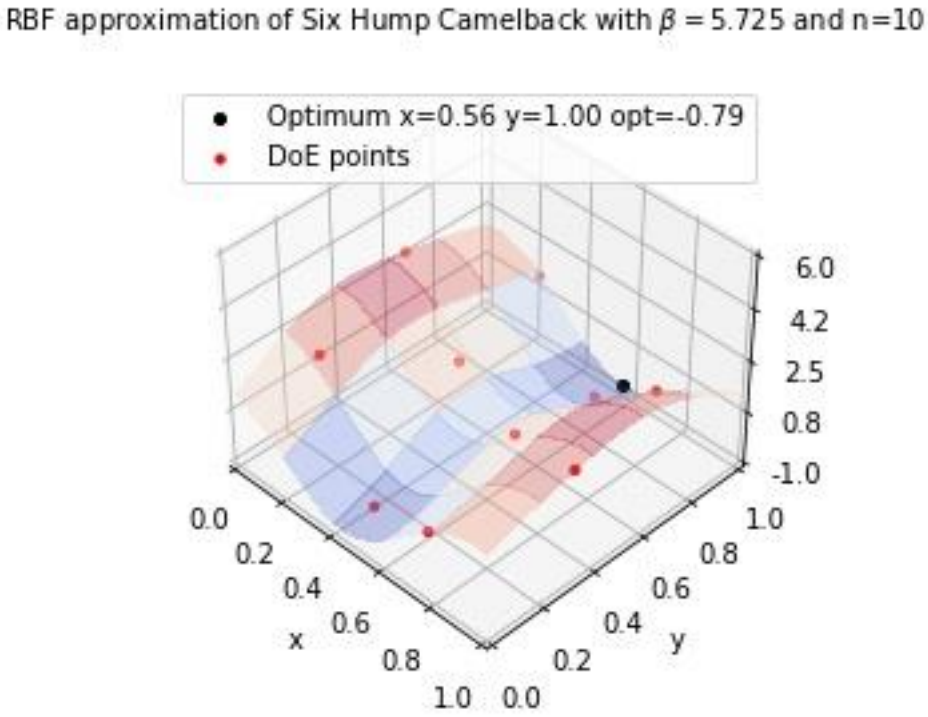
Note: If the responses y^i are corrupted by numerical noise, this may lead to overfitting of the data – this does not discriminate between the underpinning response and the noise. We can introduce a regularization parameter, *r*, added to the main diagonal of the Gram matrix:

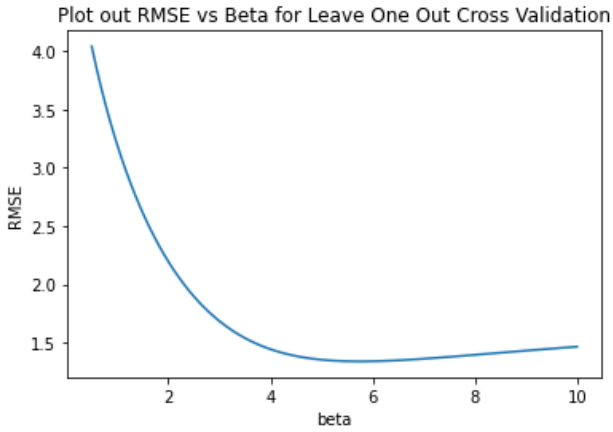
$$w = (\Psi + rI)^{-1} y$$

where *r* is a (usually small) number and I is the unit nxn matrix with 1s on the leading diagonal and zeros elsewhere.

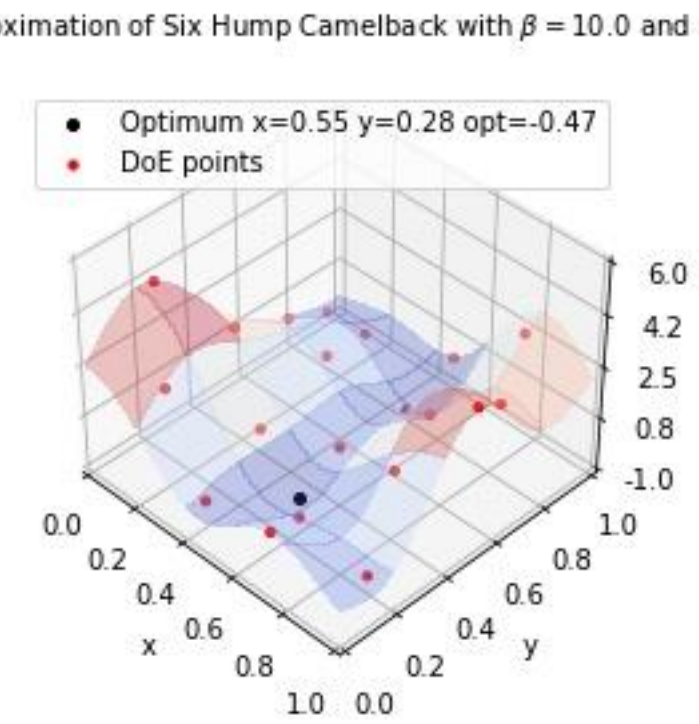
The following results illustrate RBF surrogate modelling of the Six Hump Camel Back function as a function of the number of DoE points. In each case the hyper-parameter β is obtained using LOOCV for $0.5 \leq \beta \leq 10$ and the global minimum is obtained using a Nelder-Mead simplex optimisation algorithm available in Python. The optima can be compared to the 'true' value of -1.031 obtained from the analytical form of the Six Hump Camel Back function.

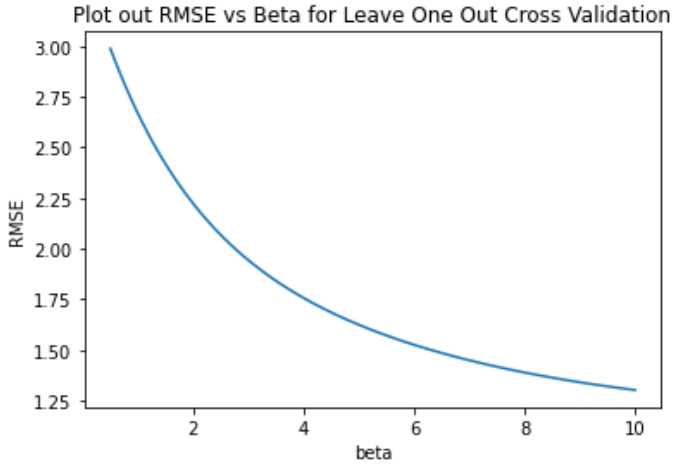
Case 1: n=10 DoE points



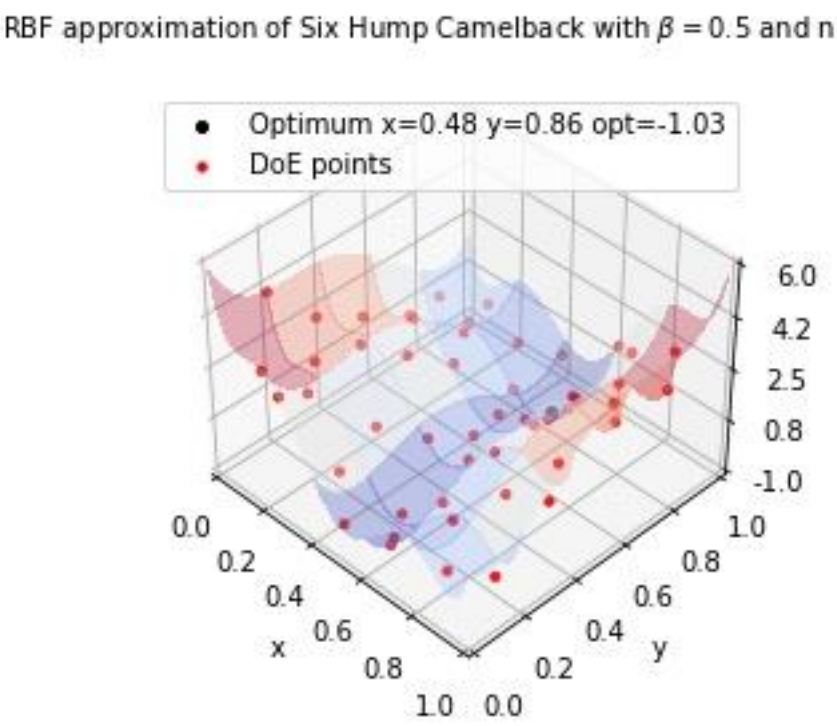


Case 2: n=20 DoE points

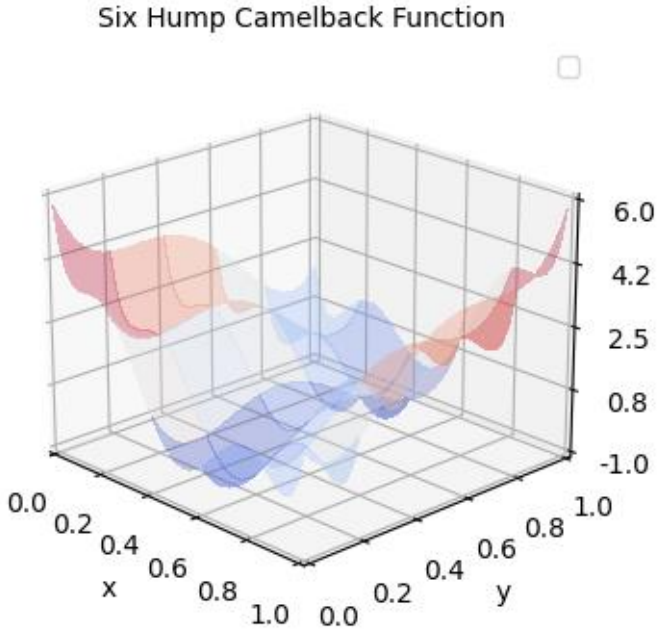


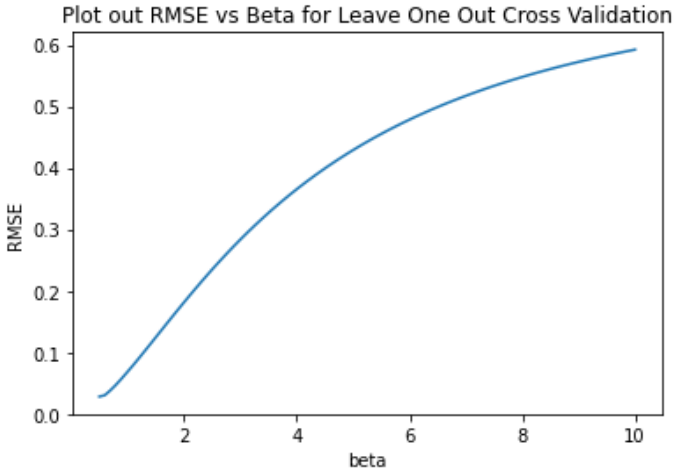


Case 3: n=50 DoE points



Analytical Function





The above results show that n=50 DoE points is sufficient to get a reasonable RBF representation of the functional surface and good agreement with the global optimum value.

6.5 Random Forests

Random Forests is a surrogate modelling method (also termed a **supervised machine learning method**) which clusters data points into functional groups. It is very effective for avoiding the problem of over-fitting the DoE data to the resultant surrogate model and is well suited to creating surrogate models for optimisation problems where the design variables are mixed continuous/categorical in nature. A categorical design might be for example, a particular manufacturing method or a metal being used, i.e. a variable which does not have a clear link to a numerical value. Random Forests are based on Decision Trees.

6.5.1 Decision Trees

In a Decision Tree the DoE dataset is split up according the values of the input variables into a series of smaller subsets whose output values have similar values. The similarity in the output variables if often measured in terms of a standard deviation/variance about the average output value in the subset. Each split in the input DoE data is like a branch in a tree and each data subset is called a leaf. The data are progressively split until some convergence condition is satisfied. The convergence criterion could be based on

- The maximum number of splits that have been performed
- The standard deviation in the output values of the subset is below a specified tolerance

The prediction from the Decision Tree for a specific set of input variables would then be based on the average value associated with the final leaf with which it is associated and which will not be split any further. The general idea can be explained by a simple example.

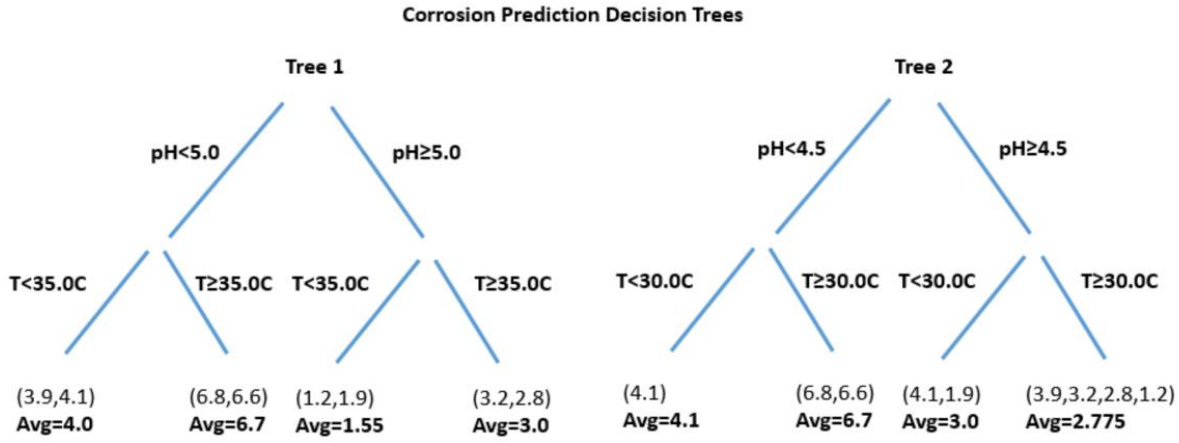
6.5.2. Example

Suppose that a Decision Tree is used to predict the corrosion rate of a metal in a corrosive liquid (in mm/year) as a function of 2 input variables – the pH and the temperature of the liquid in ^{o}C. A series of experiments are carried out and the experimental (DoE) data is given by the following table:

pH	T (ºC)	Corrosion Rate (mm/year)
4.1	45.2	6.8
3.8	41.5	6.6
4.8	32.5	3.9
4.6	28.0	4.1
5.1	36.0	3.2
6.5	38.0	2.8
6.3	33.0	1.2
5.5	29.5	1.9

The following figure shows two possible Decision Trees that could be used. Tree 1 uses the following splits. Data is split up first of all according to whether pH<5.0 or pH≥5.0 (the first branch in Tree 1) and then according to whether T<35ºC or T≥35ºC (the second branch). The leaf associated with the branches with pH<5.0 and T<35ºC results in the leaf with corrosion rates (3.9,4.1). Hence for the specific case with pH=4.0 and T=29.0ºC, the Decision Tree would predict the average of these two values: 4.0 mm/year. Other design variables would result in the other averages shown.

Suppose now there is a second tree, Tree 2, where data is split up according to whether pH<4.5 or pH≥4.5 (the first branch in Tree 2) and then according to whether T<30ºC or T≥30ºC (the second branch).



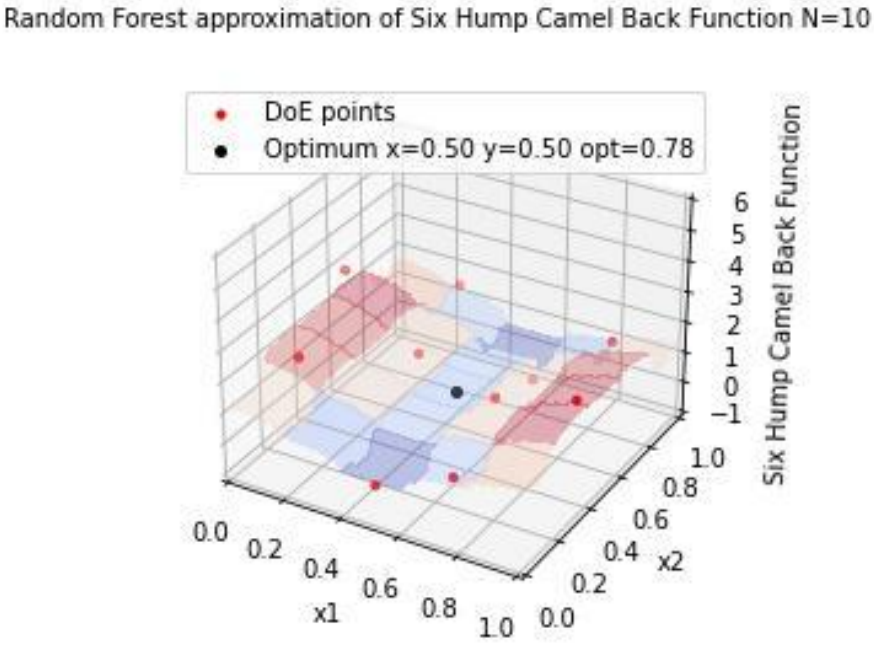
For the specific case with pH=4.0 and T=29.0ºC, Tree 2 has only one output value, 4.1, in the associated leaf (obtained from the branches pH<4.5 and T<30ºC) hence Tree 2 predicts the corrosion rate to be 4.1 mm/year.

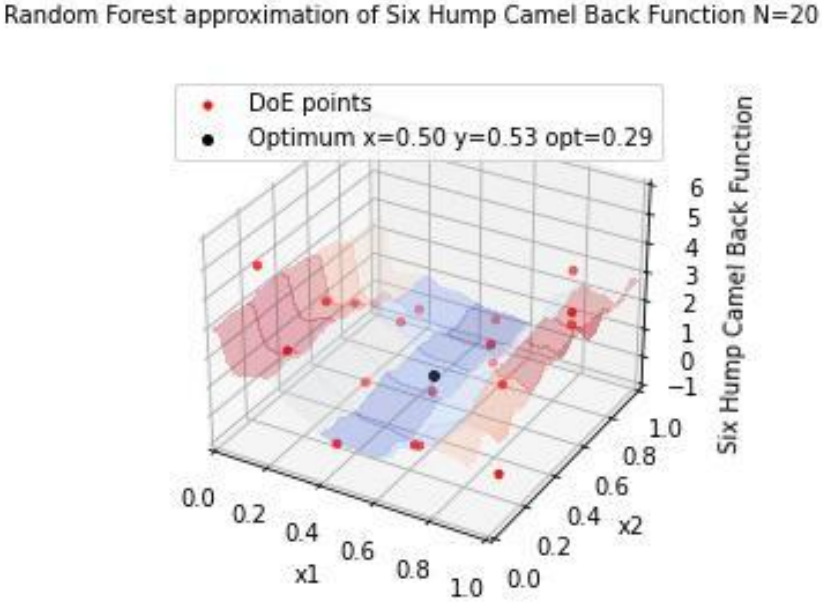
In practice, the power of the Random Forest method is based on using a number of trees where the branching criteria are specified randomly and then taking the averages over all these Decision Trees. This is an example of an **Ensemble Learning** method which uses the outcomes of many different models. If the results of Tree 1 and Tree 2 are combined into a Random Forest with these two Decision Trees, the corrosion rate prediction for the case with pH=4.0 and T=29.0ºC would be given by the weighted sum of the two leaves. This would be the average of 3.9, 4.1 and 4.1 or 4.03 mm/year.

The Random Forest method has a number of hyper-parameters which have to be optimised during the training and validation process. These include the number of trees, the number of decision levels and the convergence criteria.

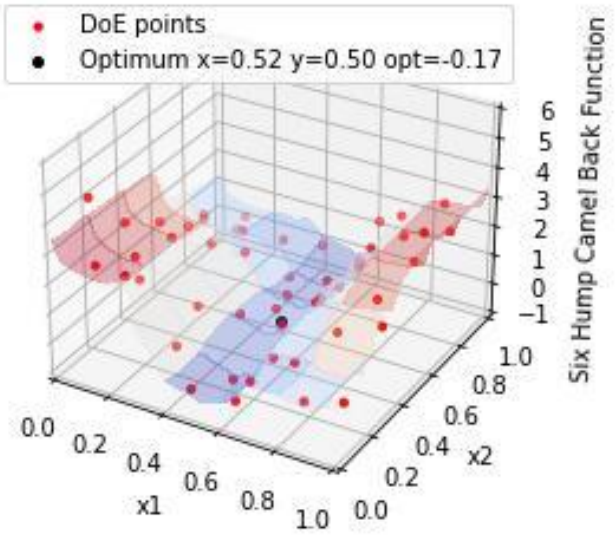
Examples of using Random Forest surrogate modelling of the Six Hump Camel Back Function are given below.

Six Hump Camel Back Function: Random Forests

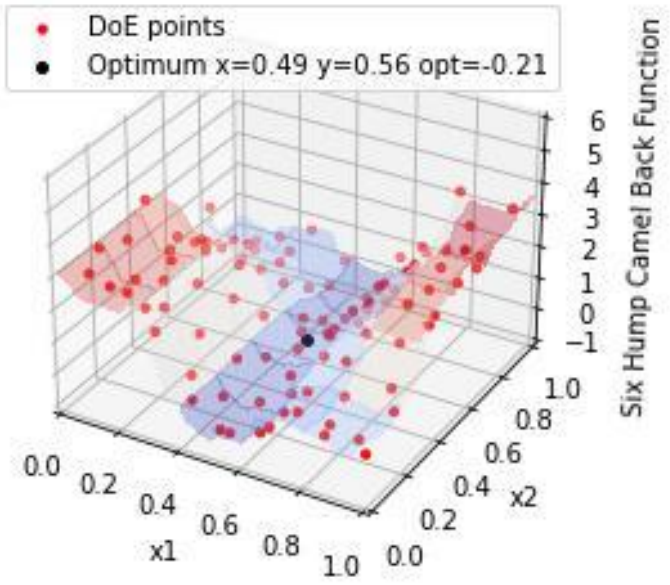




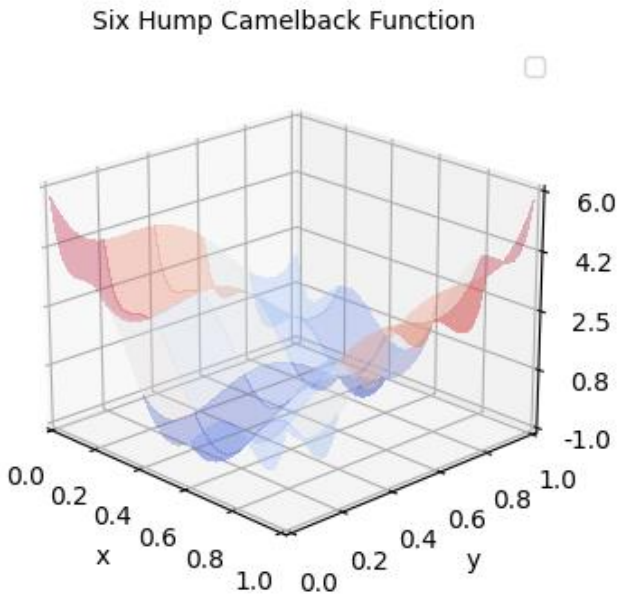
Random Forest approximation of Six Hump Camel Back Function N=50



Random Forest approximation of Six Hump Camel Back Function N=100



Actual Function



It can be seen that 100 DoE points are needed for the Random Forest to generate a surface similar to that of the actual function. However, even for 100 DoE points, the optimal solution (-0.21) is very inaccurate compared to the actual optimum (-1.03).

6.6 Gaussian Process Regression

Gaussian Process Regression (GPR) models are widely used in surrogate modelling and machine learning due to their ability to represent complex, nonlinear relationships between input and output variables. Their assumption of Gaussian/Normal behaviour enables equations for quantifying uncertainty to be obtained in an easy to calculate form. The background to GPR models is rather mathematical but they are widely available in packages such as Matlab and Python.

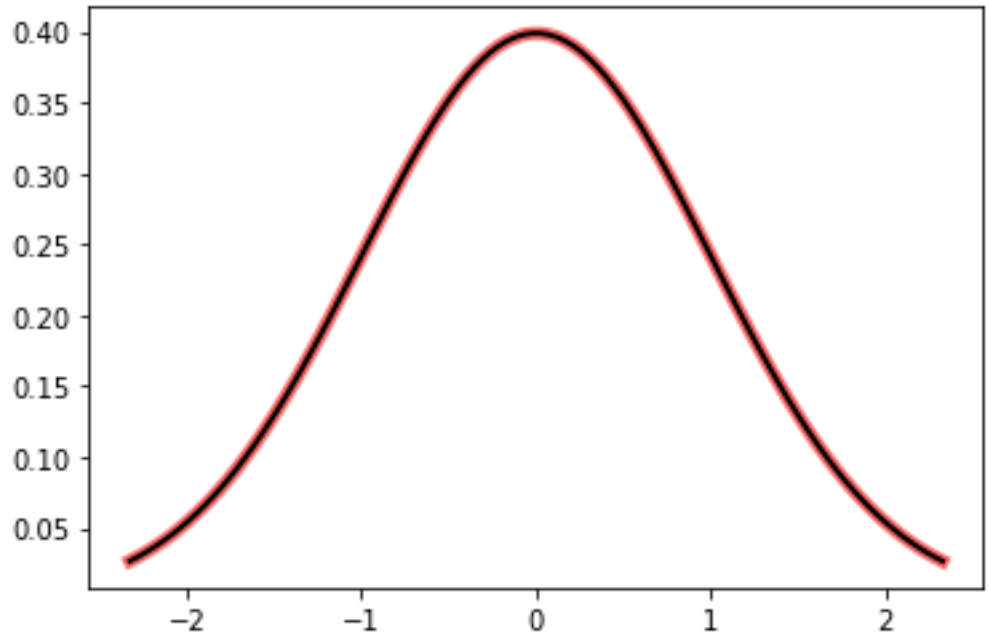
Regressions models formulate a function that represents observed data and uses this function to predict values at new data points. There are an infinite number of ways this function can be formulated and GPR models use a probability distribution over this infinite number of functions to determine which is the most likely given the DoE data provided.

6.6.1. Gaussian/Normal distribution

If a single random variable X is Gaussian – or normally – distributed with mean μ and variance σ^2, its probability density function (pdf) is given by

$$P_X(s) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(s-\mu)^2}{2\sigma^2}\right)$$

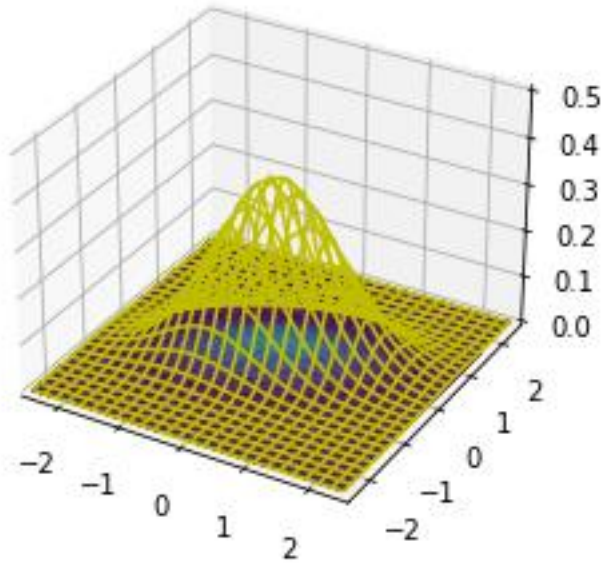
This is the well-known *bell-shaped* curve.



This can be generalised to the Multivariate Normal Distribution (MVN) for several design variables $x = (x_1, \dots, x_{ndv})$

$$P_x(s|\mu, \Sigma) = \frac{1}{(2\pi)^{ndv/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(s-\mu)^T \Sigma^{-1}(s-\mu))$$

where $\mu = E(x)$ is the mean vector and Σ is the (ndv x ndv) covariance matrix. The MVN pdf can be visualised for 2 design variables. An example is shown below



6.6.2 Covariance Matrix/Kernels

The covariance function between the design variables is also called the Kernel function and encapsulates prior knowledge about the functions we are trying to represent. The squared

exponential (SE) kernel, also known as the Gaussian or Radial Basis Function (RBF) kernel, is widely used

$$Cov(x^i, x^j) = \sigma_f^2 \exp(-\frac{r_{ij}^2}{2l})$$

where r_{ij} is the Cartesian distance between points x^i and x^j and σ_f^2 and l are hyper-parameters of the Covariance matrix. The hyper-parameters in a GPR model are generally associated with the Correlation matrix/kernel function and with the noise levels in the data. The hyper-parameters are found be solving a separate optimisation problem that maximises the likelihood of obtaining the observed set of DoE data.

The key limitations of GPR models are that:

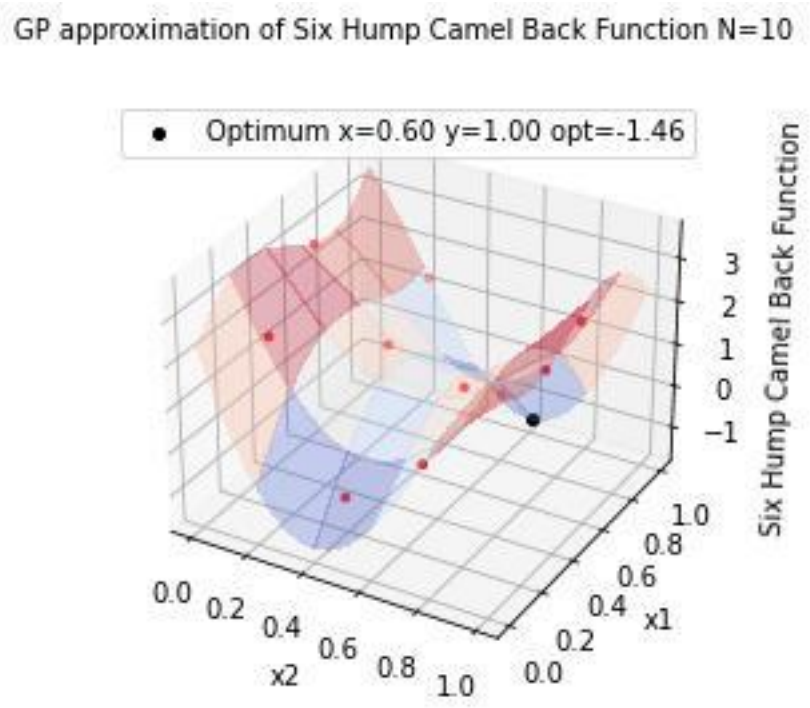
(i) The computational complexity is $O(ndv^3)$ where ndv is the number of design variables

(ii) The memory requirements are $O(ndv^2)$

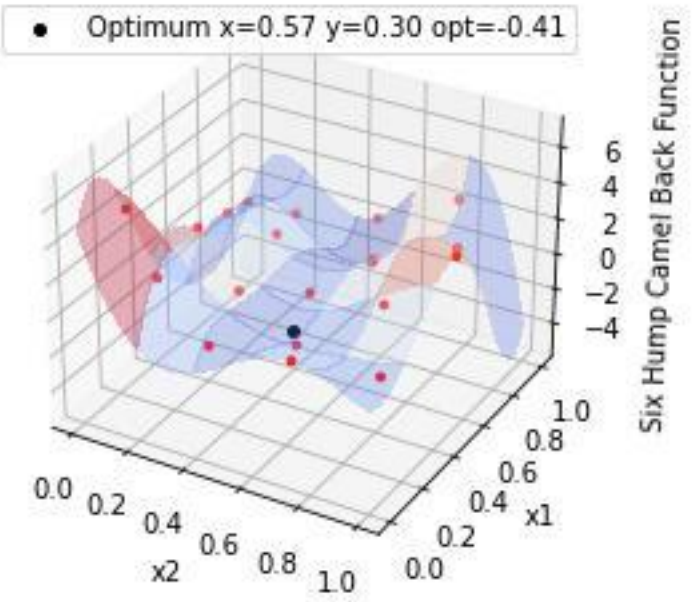
This means that GPR models are impractical for data sets with large numbers of design variables. In such cases, sparse GPR models are used instead.

6.6.3 GPR Surrogate Modelling of Six Hump Camel Back Function

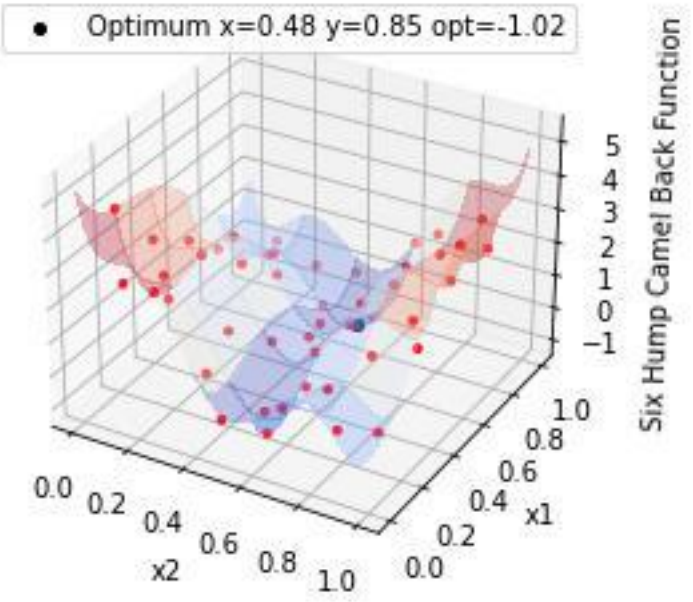
Examples are given below.

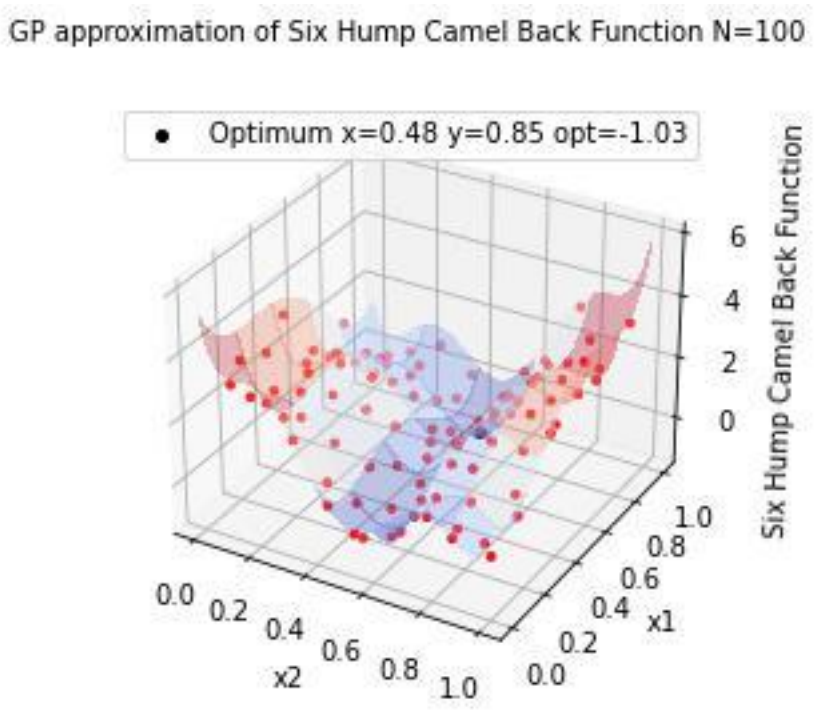


GP approximation of Six Hump Camel Back Function N=20



GP approximation of Six Hump Camel Back Function N=50





It can be seen that the surrogate model and optimum value being predicted are accurate for N>50 DoE points.

6.7 Neural Networks

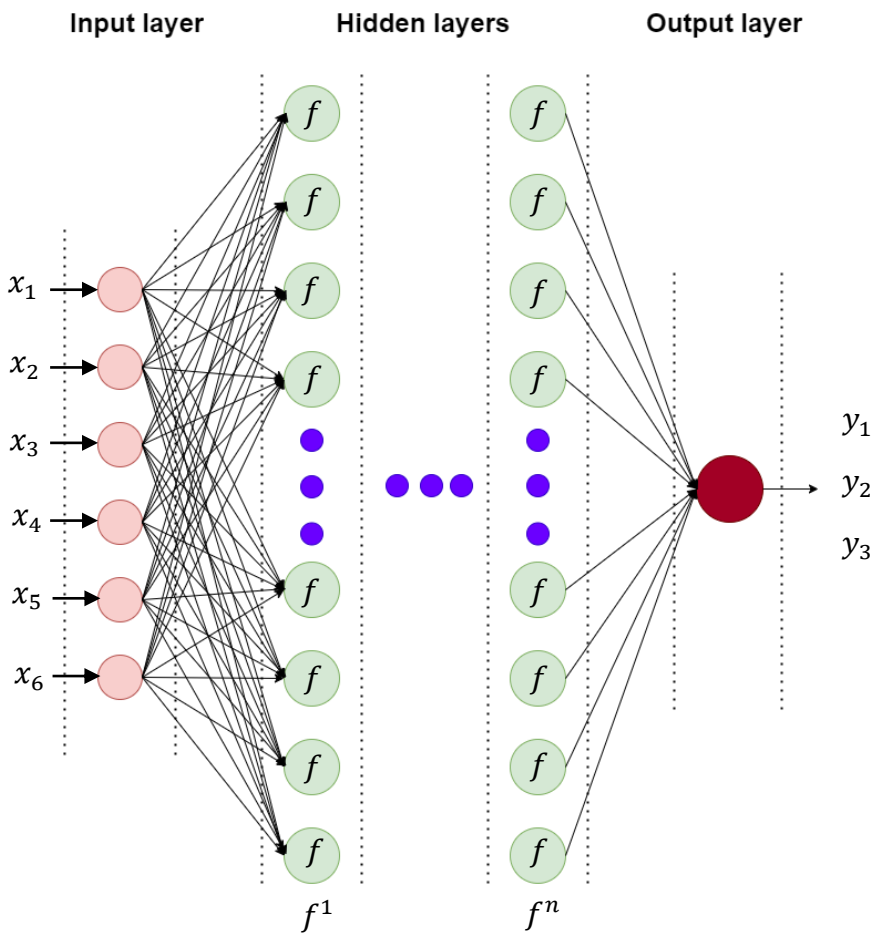
This section is based on the excellent book by Martins & Ning (2022) listed as one of the module's recommended textbooks. Interest in Neural Networks (NNs) has exploded in recent years due to their ability to approximate highly non-linear relationships between input and output variables. In addition to their use in optimisation, NNs are used in a wide range of AI applications, including Large Language Models (LLMs), Machine Vision and medical diagnostic devices.

NNs are simplified models based on the brain, with its enormous network of neurons and in NNs each neuron is a node that represents the value from a simple function. The power of the NN comes from its definition of chains of simple functions into composite functions which are able to model much more complex, non-linear behaviours. For example, if we have four simple functions f^1, f^2, f^3 and f^4, these can be chained together into the composite functions or network:

$$f(x) = f^4(f^3\left(f^2\left(f^1(x)\right)\right))$$

the composite function $f(x)$ can model very complex behaviour.

Most NNs are *feedforward* ones where information flows from the inputs x to the outputs $f(x)$. *Recurrent* NNs also have important elements of feedback throughout the network. The figure below (*due to Muhammad Raihan*) shows a diagram of a NN, where each node represents neuron. The neurons are joined between consecutive layers to form the network.



The first layer is called the input layer and the last one is the output layer. The layers between these two are the hidden layers. The total number of layers is called the network's depth. *Deep Neural Networks* have many layers, enabling very complex behaviour to be represented accurately. The first and last layers can be considered to be the inputs and outputs of the surrogate model. Each nodes in the hidden layers represents a function. The output from the NN can be represented by a vector, x. In the example above, the output is

$$x = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

More generally, the vector of values for layer k is x^k, the value for the ith neuron in layer k is x_i^k, there being n_k neurons in layer k. A neuron in layer k is connected to many neurons from the previous layer, (k-1). We can select functions for each neuron in layer k that takes values from layer (k-1) as inputs. If only linear functions were used then all the functions would be linear and only linear relationships could be modelled. Hence, some layers have to use non-linear functions. A common approach is to have hidden layers with a layer of linear functions followed by a layer with nonlinear functions. A neuron in the linear layer produces the intermediate value

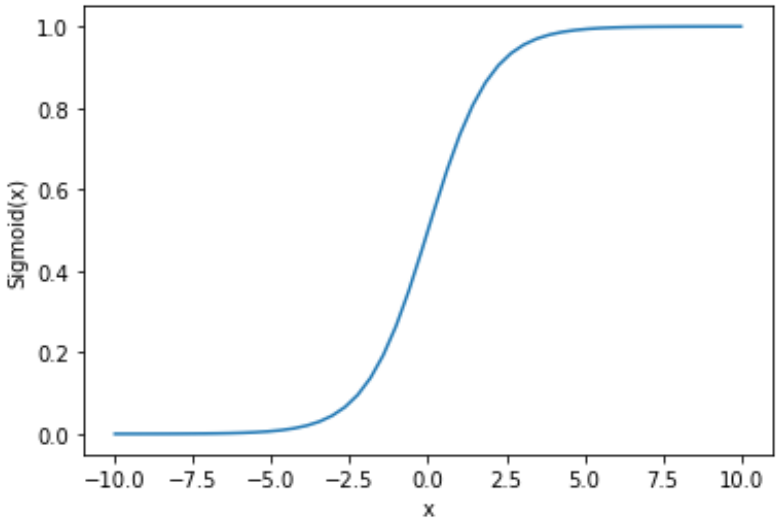
$$z = \sum_{j=1}^{n_{k-1}} w_j x_j^{(k-1)} + b$$

where n_{k-1} is the number of neurons in layer (k-1), w_j are weights for layer (k-1) and b is called the bias term which scales the significance of the overall output.
This can be written more conveniently using vector notation as

$$z = w^T x^{(k-1)} + b$$

which is a linear function of the neurons in the previous layer (k-1). The next key step is to pass the value z through an activation function, $a(z)$. In the past one of the most common activation functions was the *sigmoid function*:

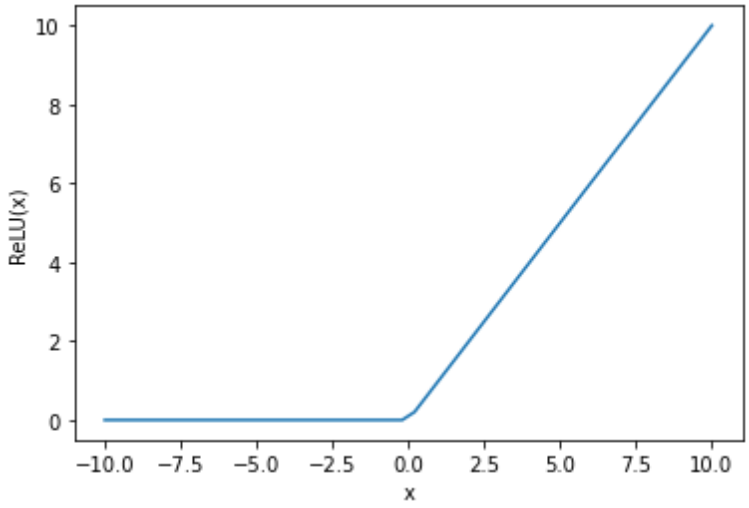
$$a(z) = \frac{1}{1 + e^{-z}}$$



This produces values between 0 and 1 so that large negative outputs are insignificant while large outputs results in values close to 1.

The *Rectified Linear Unit* (ReLU) activation function is now much more common:

$$a(z) = \max(0, z)$$



As ReLU eliminates negative inputs, the bias term is a threshold defining what is a significant value. The output from the ith neuron is obtained by combining the linear function with the activation function:

$$x_i^k = a(w^T x^{(k-1)} + b_i)$$

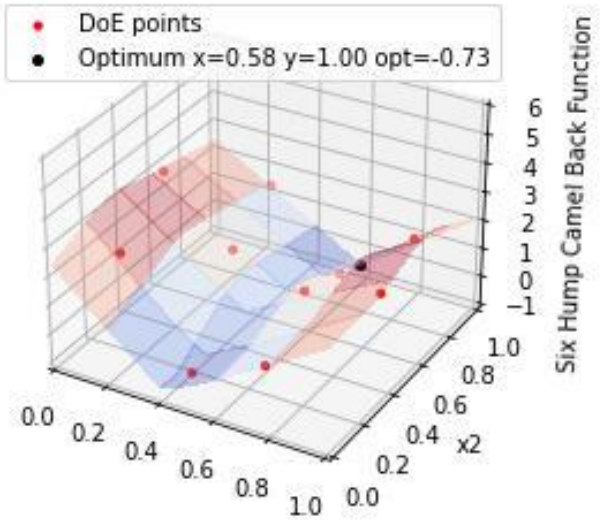
As a result of the above, the NN is now parametrised in terms of the weight and bias parameters. These are all hyper-parameters of the NN and like all surrogate models, a separate optimisation problem has to be solved to determine the optimal value of these parameters. This is called *Training the Network*. If we consider the NN in the figure above with 6 input values/neurons and then have the first hidden layer with 10 neurons, the second hidden layer with 8 neurons and 3 output neurons then there would be a total of (6x10 + 10x8 + 8x3) weights and 10+8+3 bias parameters giving a total of 185 variables. Note this is a very small NN – large NNs, for example with Large Language Models, will have several million such variables.

Since the optimisation problem that needs to be solved to train the network is very large, gradient-based optimisation methods are used. These require derivatives to be determined for all of the optimisation variables. These are obtained using *reverse-mode algorithmic differentiation* (AD) – also known as *backpropagation*. The derivatives are commonly used with specialist steepest descent methods which are referred to as *stochastic gradient descent* methods. In practical problems the goal is not to obtain the absolute minimum but to find a good enough solution quickly. The stochastic gradient descent method does not perform a line search. Instead a step-size, called the *learning rate*, is used and this is usually a pre-selected value. These algorithmic developments have been crucial in enabling NNs to be applied in increasing numbers of important AI and optimisation applications.

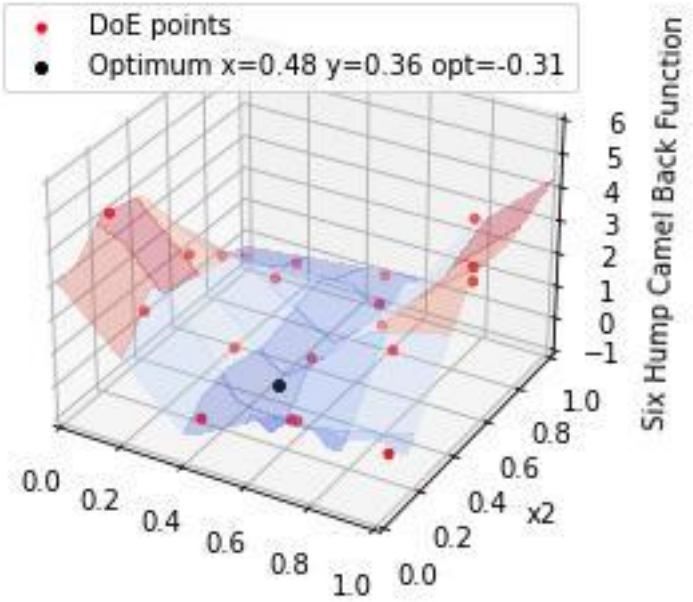
6.7.1 NN Surrogate Modelling of Six Hump Camel Back Function

Examples are given below.

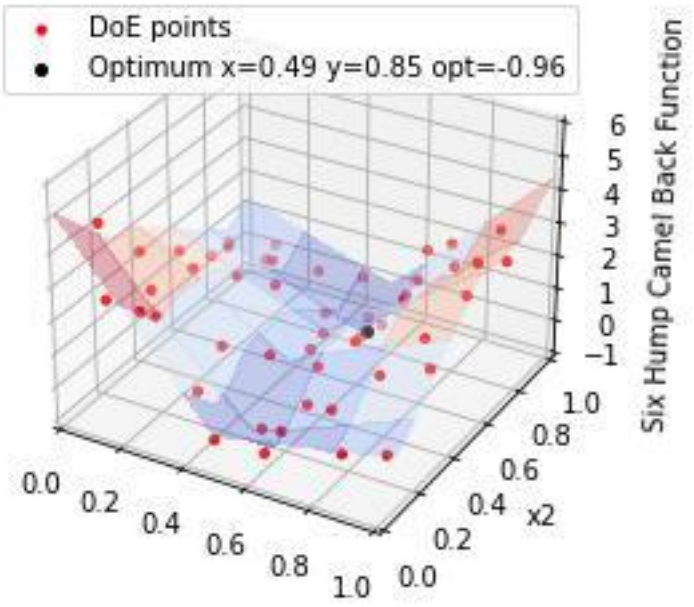
ANN approximation of Six Hump Camel Back Function N=10



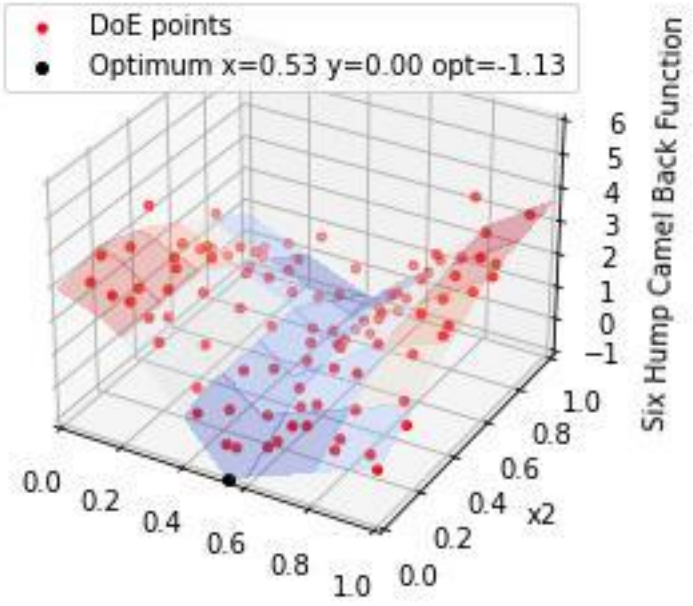
ANN approximation of Six Hump Camel Back Function N=20



ANN approximation of Six Hump Camel Back Function N=50



ANN approximation of Six Hump Camel Back Function N=100



It can be seen that the accuracy generally improves for larger values of DoE points, although the performance is not as good as seen for the GPR surrogate models. Note that NNs generally perform much better than GPRs for larger numbers of design variables and much bigger training datasets. The Six Hump Camel Back function is too small to demonstrate their capabilities fully.

6.8 Discussion and Engineering Example

The future development of many different complex products and processes will be based on a systematic optimisation process where design optimisation methods are used extensively. The data used within these optimisation methods can come from a variety of sources, including experiments, simple mathematical models and/or physics-based computer simulations. The latter type of data, in particular, is increasingly being used to solve a wide variety of challenging design problems in science and industry. This approach is very well established for structural design problems and is now used routinely to minimise the weight of automotive components or design composite wings for aircraft. Although there have been comparatively few studies which have used Computational Fluid Dynamics (CFD) to optimise complex flow problems, interest in CFD-enabled design optimisation methods is now also growing rapidly.

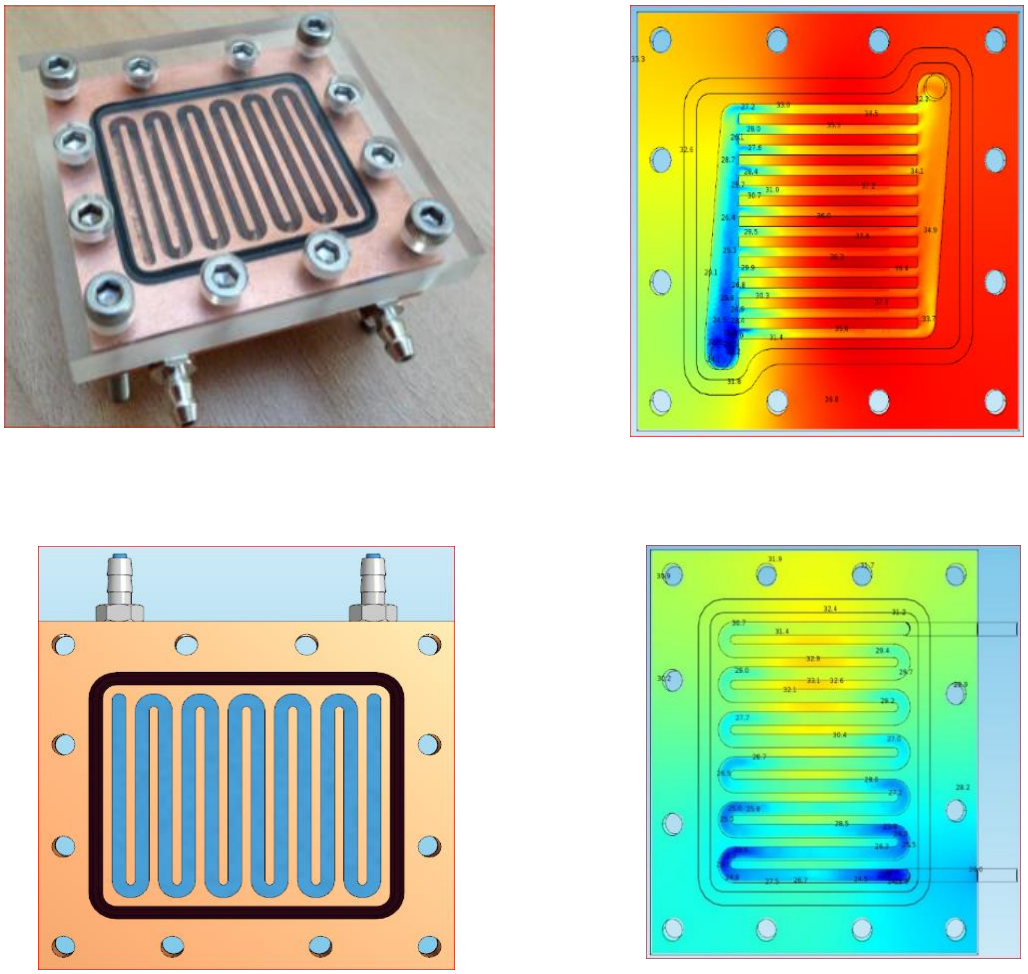
There has been rapid progress in reducing computational times for both gradient-free and gradient-based optimisation methods. Gradient-free optimisation methods can be very effective for up to 100 design variables, whereas for larger design problems, with > 100 design variables, gradient-based methods, powered by rapid advances in adjoint methods, have solved problems where the number of design variables is in the 1000s or even millions. Other key improvements that have driven these advances include adaptive Design of Experiments methods, which can provide an appropriate balance between exploration and exploitation, and multi-fidelity modelling which enables most of the computational work to be done on cheaper, lower fidelity models. Both of these enable the number of expensive, high fidelity computer simulations used in the optimisation process to be kept to a minimum. Significant progress has also been made in multi-disciplinary design optimisation (MDO) methods which coordinate simulations of the individual disciplines affecting a design (e.g. fluid mechanics, structural mechanics, ...) toward a system design that is optimal as a whole, taking into account the competing objectives.

There are several exciting research directions that will enable design optimisation methods to have even greater impact in the future. The ACARE Beyond 2020 Vision (European Commission, 2019), for example, predicts that effective MDO methods will be a key enabling technology for the future development of environment-friendly aircraft and that these aircraft will be designed virtually, using computer-based simulations, by 2050. For these and other safety-critical applications (in for example the nuclear industry), there will be increasing demands for the development of robust simulation-based optimisation methods that can ensure that product and/or process performance does not degrade significantly due to unavoidable variations in manufacturing tolerances, operating conditions, etc. It is also likely that the growing interest in using Machine Learning, for example to tune parameters in turbulence models, and the increasing trend of combining physics-based and data-driven flow simulations will widen the both the power and scope of simulation-based design optimisation methods in the very near future.

6.8.1 Electronics Cooling using Heat Sinks

This example is based on a recent research project which analysed the cooling potential of liquid cooled heat sinks for high-density electronics cooling. Heat sinks are used to take the heat away from the electronics as efficiently as possible while at the same time ensuring the energy required to pump the liquid through the heat sinks is minimal. The work is published

in the article: A.F. Al-Neama et al., ‘An experimental and numerical investigation of the use of liquid flow in serpentine microchannels for microelectronics cooling’, *Applied Thermal Engineering*, **116**, 709-723, 2017. The following figure shows the physical design (top left), CAD model (bottom left); heat sink temperature distributions (top and bottom right).



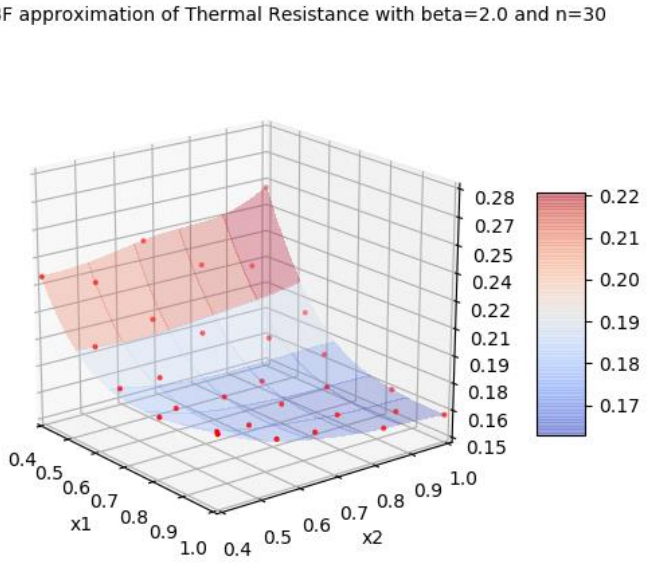
There are two objectives that need to be minimised for electronics cooling: the thermal resistance (which measures the resistance to dissipating heat from the electronics) and the pressure drop (which is directly related to energy losses in the system). Here we use the following data at 30 Design of Experiments data points:

0.4	0.4	0.2269708	32295.88
1	0.4	0.1862377	12553.08
0.4	1	0.2472688	38679.57
1	1	0.1618792	13867.9
0.4	0.664	0.2334668	35215.58
0.424	0.52	0.2195144	31543.47
0.448	0.784	0.2176468	31986.3
0.472	0.904	0.2128178	31044.3
0.496	0.616	0.1995214	27282.88
0.52	0.448	0.1951516	24729.56
0.544	0.712	0.1902447	25240.69
0.568	0.976	0.1894705	25538.35
0.592	0.856	0.1832713	23768.87
0.616	0.544	0.1800919	21209.4
0.64	0.424	0.1825142	19769.97

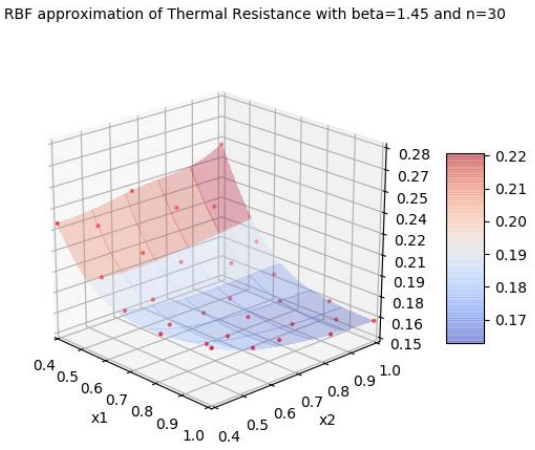
0.664	0.952	0.1739714	21385.96
0.688	0.76	0.1713519	19779.7
0.712	0.64	0.1711744	18578.62
0.736	0.496	0.1749193	17376.79
0.76	0.88	0.1666564	18166.45
0.784	0.736	0.1670559	17080.95
0.808	0.4	0.1808273	15481.67
0.832	1	0.1636113	16821.3
0.856	0.592	0.1693082	15178.61
0.88	0.808	0.1641188	15338.31
0.904	0.472	0.176308	14047.45
0.928	0.928	0.1625802	14806.84
0.952	0.688	0.167041	13858.26
0.976	0.568	0.1721966	13244.67
1	0.832	0.1643945	13500.18

Here the first two columns relate to two geometrical design parameters which are in the range $0.4 \leq x_1 \leq 1.0$ and $0.4 \leq x_2 \leq 1.0$, the third column is the thermal resistance and the fourth the pressure drop in Pascal. We will be focussing here on creating surrogate models for the thermal resistance in the heat sink system.

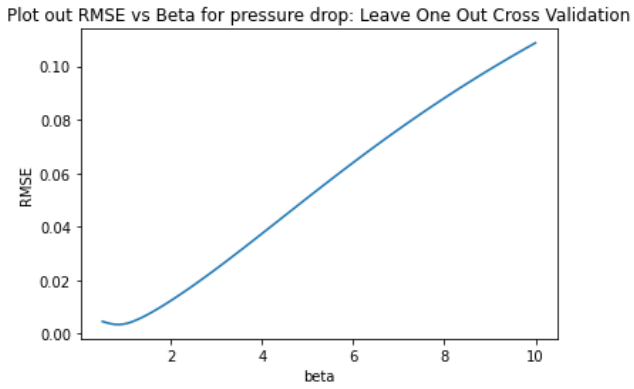
The surrogate modelling is carried out using Gaussian Radial Basis Functions. For example, using the hyper-parameter β=2.0 creates the following surrogate model of the thermal resistance:

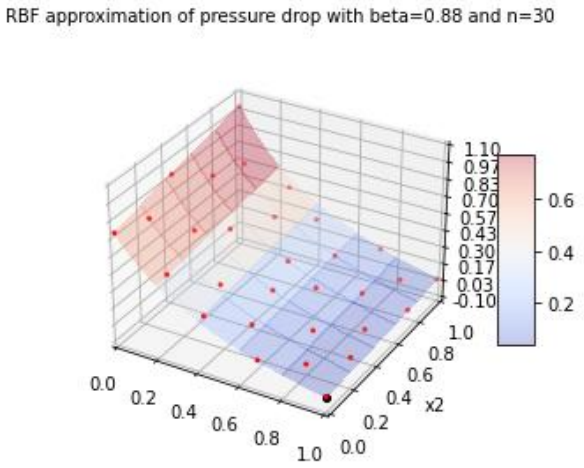


Using Leave One Out Cross Validation for the thermal resistance leads to the following figures, with β=1.45 leading to the smallest RMSE.

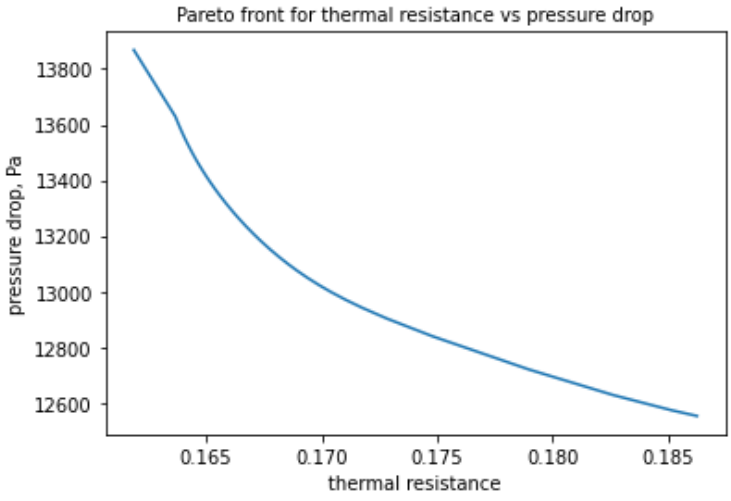


Using Leave One Out Cross Validation for the pressure drop leads to the following figures, with β=0.88 leading to the smallest RMSE.





The last figure shows the Pareto front that results from multi-objective optimisation of the thermal resistance and pressure drop (both suitably scaled). It shows the compromises that can be struck between minimising each of the objectives. For example. Reducing thermal resistance below 0.17 will results in pressure drops > 13,000.



REFERENCES for CHAPTERS 1-4

1. Arora J., Introduction to Optimum Design, 3rd Ed. Elsevier (2011).
2. Asghar Bhatti M., Practical Optimization Methods: With Mathematica® Applications, Springer; 2000 edition, (2012), DOI: https://doi.org/10.1007/978-1-4612-0501-2, ISBN: 978-1-4612-6791-1.
3. Charalambous C., Conn A.R., An efficient method to solve the minimax problem directly, SIAM J. Numer. Anal., 15 (1) (1978), pp. 162-187.
4. Das, B. C., Effect of graphical method for solving mathematical programming problem. Daffodil International University Journal of Science and Technology, [S.l.], v. 5, n. 1, p. 29-36, Feb. 2010. ISSN 2408-8498. doi: http://dx.doi.org/10.3329/diujst.v5i1.4379.
5. Gürdal Z, Haftka R. T., Elements of Structural Optimization 3rd Ed., Solid Mechanics and its Applications, Volume 11 Series Editor: G. M. L. Gladwell, ISBN 0-7923-1505-7 .(1992).
6. Hooke, R.; Jeeves, T.A. (1961). "Direct search" solution of numerical and statistical problems". Journal of the ACM. 8 (2): 212–229. doi:10.1145/321062.321069.
7. Nelder, John A.; R. Mead (1965). "A simplex method for function minimization". Computer Journal. 7 (4): 308–313. doi:10.1093/comjnl/7.4.308.
8. Olhoff N, Multicriterion structural optimization via bound formulation and mathematical programming, Structural optimization, 1, 11 – 17, (1989).
9. Querin O.M., Victoria M., Alonso Gordoa C., Ansola R., Martí P., Topology Design Methods for Structural Optimization, Elsevier (2017), ISBN: 978-0-08-100916-1
10. Rao S., Engineering Optimization: Theory and Practice, Fourth Edition, John Wiley & Sons, Inc. (2009)
11. Spendley, W.; Hext, G. R.; Himsworth, F. R. (1962). "Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation". Technometrics. 4 (4): 441–461. doi:10.1080/00401706.1962.10490033.
12. Spillers W.R., MacBain K.M., Structural Optimization, Springer (2009), ISBN 978-0-387-95864-4
13. Xie Y. M., Steven G.P., Evolutionary Structural Optimization, Springer, 1997

REFERENCES for CHAPTERS 5&6

6) ANFD. SPREVAK, J. M. M. Introduction to Unconstrained Optimization. A Computer Illustrated Text. Institute of Physics Publishing, 1990. ISBN 0750300256.

7) ANTONY, J. Taguchi or Classical Design of Experiments: A Perspective from a Practitioner. Sensor Review 26, 3 (2006), 123–160.

11) AUDZE, P., AND EGLAIS, V. New Approach for Planning Out of Experiments. Problems of Dynamics and Strengths 35 (1977), 104 – 107.

13) BATES, S. J., SIENZ, J., AND LANGLEY, D. S. Formulation of the Audze–Eglais Uniform Latin Hypercube Design of Experiments. Advances in Engineering Software 34, 8 (2003), 493–506.

14) BATES, S. J., SIENZ, J., AND TOROPOV, V. V. Formulation of the Optimal Latin Hypercube Design of Experiments Using a Permutation Genetic Algorithm. AIAA-2004-2011 (2004).
18) BLUE, G., AND LAUNSBY, R. Design for Six Sigma. McGraw Hill, 2003. ISBN 0071413766.
20) BOX, G. E. P., AND BEHNKEN, D. W. Some New Three Level Designs for the Study of Quantitative Variables. Technometrics 2, 4 (1960), 455–475.
25) CHOI, K., YOUN, B., AND YANG, R. Moving Least Square Method for Reliability-Based Design Optimization. The Fourth World Congress of Structural and Multidisciplinary Optimization (2001).
26) CHUNG, H. S., AND ALONSO, J. J. Multiobjective Optimization Using Approximation Model-Based Genetic Algorithms. AIAA-2004-4325 (2004).
28) CLARK, I. Practical Geostatistics. Ecosse, 1979. ISBN 0954891198.
30) COX, D. R., AND REID, N. Theory of DoE. Chapman and Hall, 2000. ISBN 158488195X.
31) CRESSIE, N. A. C. Statistcs for Spatial Data. Wiley, 1993. ISBN 0471002550.
42) FANG, K.-T., ZE LI, R., AND SUDJIANTO, A. Design and Modeling for Computer Experiments. Chapman & Hall, 2006. ISBN 1584885467.
46) FORRESTER, A., SOBESTER, A., AND KEANE, A. Engineering Design Via Surrogate Modelling: A Practical Guide. WileyBlackwell, 2008. ISBN 0470060689.
47) FORRESTER, A. I. J., KEANE, A. J., AND BRESSLOFF, N. W. Design and Analysis of “Noisy” Computer Experiments. AIAA Journal 44, 10 (2006), 2331–2339.
48) GHOSH, S., AND RAO, C. R. Design and Analysis of Experiments. Handbook of Statistics, vol. 13. Elsevier, 1996. ISBN 0444820612.
54) GUTMANN, H. M. A Radial Basis Function Method for Global Optmization. Journal of Global Optimization 19 (2001), 201–227.
59) HOLLAND, J. H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press, 1975. ISBN 0262581116.
60) HOLLAND, J. H. Genetic Algorithms. www.econ.iastste.edu/tesfatsi/holland.GAIntro.htm, 2005.
61) HUANG, D., ALLEN, T. T., NOTZ, W. I., AND ZHENG, N. Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models.
69) JONES, D. R. A Taxonomy of Global Optimization Methods Based on Response Surfaces. Journal of Global Optimization 21 (2001), 345–383.
70) JURECKA, F. Robust Design Optimization Based on Metamodeling Techniques. Ph.D. Thesis, Technique Universit¨at M¨uchen, 2007.
72) KEANE, A. J. Wing Optimization Using Design of Experiment, Response Surface and Data Fusion Methods. Journal of Aircraft 40, 4 (2003), 741–750.
73) KEANE, A. J., AND NAIR, P. B. Computational Approaches for Aerospace Design: The Pursuit of Excellence. J. Wiley, 2005. ISBN 0470855401.
78) KITANIDIS, P. K. Introduction to Geostatistics. Cambridge University Press, 1997. ISBN 0521583128.
79) [KOK, S., AND SANDROCK, C. Locating and Characterizing the Stationary Points of the Extended Rosenbrock Function. Evolutionary Computation 17, 3 (2009), 437 – 453.

80) KRIGE, D. G. A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand. Journal of the Chemical, Metallurgical and Mining Society of South Africa 52, 6 (1951), 119–139.
81) KRIGE, D. G. Letter to the Editor. Mathematical Geology 18, 5 (1986), 501–502.
82) KRISHNAMURTHY, T. Response Surface Approximation with Augmented and Compactly Supported Radial Basis Functions. In Proceedings of the 44th AIAA/ASME/ASCE/AHS/ASC Conference (2003). AIAA-2003-1748.
84) LAVENUE, A. M., AND PICKENS, J. F. Application of a Coupled Adjoint Sensitivity and Kriging Approach to Calibrate a Groundwater Flow. Water Resources Research 28, 6 (1992), 1543–1569.
85) LEARY, S., BASKAR, A., AND KEANE, A. J. A Knowledge-based Approach to Response Surface Modelling in Multifidelity Optimization. Journal of Global Optimization 26 (2003), 297–319.
86) LEARY, S., BASKAR, A., AND KEANE, A. J. Global Approximation and Optimization Using Adjoint Computational Fluid Dynamics Codes. AIAA Journal 42, 3 (2004), 631–641.
87) LEE, K.-H., AND LANG, D.-H. Structural Optimization of an Automotive Door Using the Kriging Interpolation Method. Proc. IMechE Part D. Journal of Automobile Engineering. 221, 12 (2007), 1525–1534.
88) LIEFVENDAHLA, M., AND STOCKIB, R. A study on Algorithms for Optimization of Latin Hypercubes. Journal of Statistical Planning and Inference 136 (2006), 3231 – 3247.
92) MASON, R. L., GUNST, R. F., AND HESS, J. L. Statistical Design and Analysis of Experiments, with Applications to Engineering and Science. John Wiley & Sons, 1989. http://www.itl.nist.gov/div898/handbook/.
93) MATHERON, G. Trait´e de G´eostatistique appliqu´ee. Tome I: M´emores du Bureau de Recherches Gologiques et Mini´eres, 14 (1962).
98) MYERS, D. E. Letter to the Editor. Mathematical Geology 18, 7 (1986), 699–700.
102) NARAYANAN, A., TOROPOV, V., WOOD, A. S., AND CAMPEAN, I. F. Simultaneous Model Building and Validation with Uniform Designs of Experiments. Engineering Optimization 39, 5 (2007), 497–512.
103) NIST/SEMATECH. e-Handbook of Statistical Methods. 2006. http://www.itl.nist.gov/div898/handbook/.
106) PAPALAMBROS, P. Y., AND WILDE, D. J. Principles Of Optimal Design. Modeling and Computation., 2nd ed. Cambridge University Press, 2000. ISBN 0521627273.
108) PARK, K., OH, P.-K., AND LIM, H.-J. The Application of the CFD and Kriging Method to an Optimization of Heat Sink. International Journal of Heat and Mass Transfer 49 (2006), 3439–3447.
111) PHILIP, G. M., AND WATSON, D. F. Matheronian Geostatistics - Quo Vadis? Mathematical Geology 18, 1 (1986), 93–117.
113) PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. Numerical Recipes in C++ : The Art of Scientific Computing, 3 ed. Cambridge University Press, 2007. ISBN 9780521866088.
123) SACKS, J., WELCH, W. J., MITCHELL, T. J., AND WYNN, H. P. Design and Analysis of Computer Experiments. Statistical Science 4, 4 (1989), 409–423.
125) SANTNER, T. J., WILLIAMS, B. J., AND NOTZ, W. I. The Design and Analysis of Computer Experiments. Springer, 2003. ISBN 0387954201.

129) SHEPARD, D. A Two Dimensional Interpolation Function for Irregularly Spaced Data. In Proceedings of the 23rd National Conference, Association for Computing Machinery (1968), pp. 517–523.
130) SIMPSON, T., LIN, D., AND CHEN, W. Sampling Strategies for Computer Experiments: Design and Analysis. International Journal of Reliability and Application 2, 3 (2001), 209–240.
131) SIMPSON, T. W., MAUERY, T. M., KORTE, J. J., AND MISTREE, F. Comparison of Response Surface and Kriging Models for Multidisciplinary Design Optimization. In Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization (1998). AIAA-98-4755.
132) SIMPSON, T. W., MAUERY, T. M., KORTE, J. J., AND MISTREE, F. Kriging Models for Global Approximation in Simulation-Based Multidisciplinary Design Optimization. AIAA Journal 39, 12 (2001), 2233–2241.
142) STEIN, M. L. Interpolation of Spatial Data. Some Theory for Kriging. Springer, 1999. ISBN 0387986294.
143) SUN, W., AND XIANG YUAN, Y. Optimization Theory and Methods: Nonlinear Programming. Springer, 2006. ISBN 0387249753.
144) SWAN, A. R. H., AND SANDILANDS, M. Introduction to Geological Data Analysis. Blackwell Science, 1995. ISBN 0632032243.
148) TANCO, M., VILES, E., AND POZUETA, L. Comparing Different Approaches for Design of Experiments. In Advances in Numerical Engineering and Computational Science. Lecture Notes in Electrical Engineering (2009), vol. 139, Springer, pp. 611–621. ISBN 9789048123117.
149) TORN, A., AND ZILINSKAS, A. Global Optmization. Springer, 1987. ISBN ?
150) TOROPOV, V. V., BATES, S. J., AND QUERIN, O. M. Generation of Extended Uniform Latin Hypercube Design of Experiments. Submitted (2007).
151) [151] TOROPOV, V. V., SCHRAMM, U., SAHAI, A., JONES, R. D., AND ZEGUER, T. Design Optimization and Stochastic Analysis based on the Moving Least Squares Method. 6th World Congress on Structural and Multidisciplanary Optimization (2005).
157) WHITTINGHILL, D. C. A Note on the Robustness of Box-Behnken Designs to the Unavailability of Data. Metrika 48, 1 (1998), 305–325.