Chapter 2

Basic Optimization and Numerical Methods

This Chapter deals with basic definition of optimality, the general non-linear programming problem, analytical and numerical optimization methods used in various mathematical models are briefly discussed in this Chapter.

2.1 Introduction

The non-linear programming techniques are used routinely and particularly efficient in solving optimal control problems. In the case of a discrete control problem i.e., when the controls are exerted at discrete points, the problem can be directly stated as a non-linear programming problem. In a continuous control problem i.e., when the controls are functions exerted over a prescribed planning horizon and approximate solution can be found by solving the non-linear programming problem. The general structure of optimization problem is shown in Figure 2.1.

\[
\min_{x \in X} f(x)
\]

subject to

\[g(x) \leq 0,\]

\[h(x) = 0.\]

where, \( X \) is a subset of \( \mathbb{R}^{n_x} \), \( x \) is a vector of \( n_x \) components \( x_1, x_2, \ldots, x_{n_x} \) and \( f : X \to \mathbb{R}, g : X \to \mathbb{R}^{n_g} \) and \( h : X \to \mathbb{R}^{n_h} \) are defined on \( X \).
The function \( f \) is usually called the objective function or the criterion function. Each of the constraints \( g_i(x) \leq 0, \; i = 1, 2, \ldots, n_g \), is called an inequality constraint, and each of the constraints \( h_i(x) = 0, \; i = 1, 2, \ldots, n_h \), is called the equality constraint. A vector \( x \in X \) satisfying all the constraints is called a feasible solution to the problem; the collection of all such points form the feasible region. The non-linear programming problem, then, is to find a feasible point \( x^* \) such that \( f(x) \geq f(x^*) \) for each feasible point \( x \). A non-linear programming problem can be stated as a maximization problem and the inequality constraints can be written in the form \( g(x) \geq 0 \).

### 2.2 Basic Definitions of Optimality

Let \( S \in \mathbb{R} \) be nonempty set.

#### 2.2.1 Infimum and Supremum

The infimum of \( S \), denoted as \( \inf S \), provided it exists, is greater lower bound for \( S \).

i.e., a number \( \alpha \) satisfying:

(i) \( z \geq \alpha \quad \forall \; z \in S \),

(ii) \( \forall \bar{\alpha} > \alpha \quad \exists z \in S \) such that \( z < \bar{\alpha} \) Similarly, the supremum of \( S \), denoted
as \( \text{Sup } S \), provided it exists, is the least upper bound for \( S \), i.e., a number \( \alpha \) satisfying:

(i) \( z \leq \alpha \quad \forall \ z \in S \),

(ii) \( \forall \alpha < \alpha \exists z \in S \) such that \( z > \alpha \).

2.2.2 Minimum and Maximum

Consider the standard problem formulation

\[
\min_{x \in D} f(x)
\]

where, \( D \subset \mathbb{R}^n \) denotes the feasible set. Any \( x \in D \) is said to be feasible point; conversely any \( x \in \mathbb{R}^n \setminus D = \{ x \in \mathbb{R}^n : x \notin D \} \) is said to be infeasible.

A point \( x^* \in D \) is said to be (global) minimum then \( f(x) \leq f(x^*) \ \forall x \in D \), i.e., a minimum is a feasible point whose objective function value is less than or equal to the objective function value of all other feasible points. It is said to be strict (global) minimum of \( f \) on \( D \) if

\[
f(x) > f(x^*) \ \forall x \in D, \ x \neq x^*.
\]

A point \( x^* \in D \) is said to be (global) maximum of \( f \) on \( D \) if \( f(x) \leq f(x^*) \ \forall x \in D \), i.e., a maximum is a feasible point whose objective function value is greater than or equal to the objective function value of all other feasible points. It is said to be strict (global) maximum of \( f \) on \( D \) if

\[
f(x) < f(x^*) \ \forall x \in D, \ x \neq x^*.
\]

The (global) maximum and (global) minimum is shown in Figure 2.2.

![Figure 2.2 A function with several local maxima and several local minima](image-url)
2.2.3 Concave and Convex Function

A single variable function, which when plotted, results in a curve always curving downwards or not curving at all, is called concave function. Figure 2.3 show a single variable concave function. The shape of this function is such that for any two points on the curve, in the feasible region $S$, the line joining the points is always below the function. There is always a unique global maximum of such a function, i.e., A function $f(x)$ is a concave function over a region $S$, if for any two points $a$ and $b$ in $S$,

$$f[\lambda a + (1 - \lambda)b] \geq \lambda f(a) + (1 - \lambda)f(b)$$

where, $0 \leq \lambda \leq 1$.

![Figure 2.3 Concave function](image)

A single variable function, which when plotted, results in a curve always curving upwards or not curving at all, is called convex function. Figure 2.4 show a single variable convex function. The shape of this function is such that for any two points on the curve, in the feasible region $S$, the line joining the points is always above the function. There is always a unique global minimum of such a function, i.e., A function $f(x)$ is a convex function over a region $S$, if for any two points $a$ and $b$ in $S$,

$$f[\lambda a + (1 - \lambda)b] \leq \lambda f(a) + (1 - \lambda)f(b)$$

where, $0 \leq \lambda \leq 1$. 
2.2.4 Convex Program

Let $C$ be a non empty convex set in $\mathbb{R}^n$ at let $f : C \to \mathbb{R}^n$ be a convex on $C$, then

$$\min_{x \in C} f(x)$$

is said to be a convex program or a convex optimization problem.

2.3 Analytic Optimization Methods

2.3.1 Dynamic programming

In optimization problems involving a large number of decision variables or the inequality constraints, it may not be possible to use the methods of calculus for obtaining a solution. Classical mathematics handles the problems in a way to find the optimal values for all the decision variables simultaneously which for large problems rapidly increases the computations that become difficult to handle even by the available computers. The obvious solution is to split up the original large problem into small sub-problems involving a few variables.

Dynamic programming is a widely used mathematical technique for solving problems that can be divided into stages and where the decision variables are required in each stage. It uses recursive equations to solve a large, complex problem, split into a series of interrelated decision stages (sub-problems) wherein the outcome of the decision at one stage affects the decisions at the remaining stages. The goal of dynamic programming is to find a combinations of decisions that optimize a certain
amount associated with a system. Dynamic programming technique was developed by Richard Bellman (1957). The Bellman’s optimality states that “given the current state of the system, an optimal policy for the remaining stages is independent of the policy adopted in the previous stages”. It is ideally suitable for the optimization of multistage decision problem. In most practical problems, decisions have to be made sequentially at different points in time, space and at different levels, say for a component or subsystem, and/or for a system.

As applied to dynamic programming, a multistage decision problem is one in which a number of single stage processes are connected in series so that the output of one stage is the input of the succeeding stage. This serial multistage decision problems arise in many types of practical problems as chemical process, design of experiments etc.

Consider a single stage decision problem, which is a component of the multistage decision problem. It is represented by a rectangular block. A decision process can be characterized by certain input parameters (S), certain decision variables (X) and certain output parameters (T) representing the outcome obtained as a result of making decision. the input parameters are called input state variables and the output parameters are called output state variables. Finally, there is a return or objective function R, which measures the effectiveness of the decision made and the output the results from the decisions.

The serial multistage decision problems are classified into three categories as follows,

- Initial value problem: If the value of the initial state variables is prescribed, the problem is called an initial value problem.

- Final value problem: If the value of the final state variable is prescribed, the problem is called a final value problem.

- Boundary value problem: If the values of both the input and output state variables are specified, the problem is called a boundary value problem.
The procedure followed in the analysis of dynamic programming problems is summarized as follows,

- Define the problem variables, objective function and constraints
- Define the stages of the problem. Find the state variables whose values constitute the state at each stage and the decision required at each stage.
- Identify the relationship by which the state at one stage can be expressed as a function of the state and decisions at the next stage.
- Develop a recursive relationship for the optimal return function which permits computation of the optimal policy at any stage.
- Decide whether forward or backward recursive approach is applied to solve the problem.
- Determine the optimal decision at each stage and then the overall optimal policy.

### 2.3.2 Lagrange’s Multiplier Method

The optimization problems having continuous objective function with equality or inequality type constraints are called constrained extremal problems. The solution of such problems, having differentiable objective function and equality type constraints can be obtained by a number of methods, but the most common is the Lagrange’s multiplier method (Gupta and Hira, 2008).

A general non-linear programming problem having \( n \) variables and \( m \) constraints can be expressed as

Maximize or Minimize \( Z = f(X) \), \( \text{for } X = (x_1, x_2, ..., x_n) \) \hspace{1cm} (2.1)

subject to

\[ g_i(X) = b_i \quad \text{for } i = 1, 2, ..., m \] \hspace{1cm} (2.2)

\[ X \geq 0 \] \hspace{1cm} (2.3)
The constraint is written as

\[ h^i(X) = g^i(X) - b_i, \quad \text{for } i = 1, 2, \ldots, m \]  

(2.4)

By introducing Lagrange’s multiplier’s \( \lambda = \lambda_1, \lambda_2, \ldots, \lambda_m \), the lagrangian function is constructed as

\[ L(X, \lambda) = f(X) - \sum_{i=1}^{m} \lambda_i h^i(X) \]  

(2.5)

Assuming that all the functions \( L, f \) and \( h^i \) are differentiable partially with respect to \( x_1, x_2, \ldots, x_n \) and \( \lambda_1, \lambda_2, \ldots, \lambda_m \). The necessary conditions for the objective function to be a maximum or a minimum are

\[ \frac{\partial L}{\partial x_j} = \frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} \lambda_i \frac{\partial h^i}{\partial x_j} = 0, \quad \text{for } j = 1, 2, \ldots, n \]  

(2.6)

and

\[ \frac{\partial L}{\partial \lambda_i} = 0 - h^i = 0, \quad \text{for } i = 1, 2, \ldots, m \]  

(2.7)

These \( m + n \) necessary conditions become the sufficient conditions for a maximum, if the objective function is concave and for a minimum, if the objective function is convex. When concavity or convexity is not known the necessary conditions provide an optimal solution to the problem.

### 2.3.3 Predictor Corrector Interior Point Method

A general non-linear programming problem having \( n \) variables and \( m \) constraints can be expressed as

Maximize or Minimize \( Z = f(X) \), \( \quad \text{for } X = (x_1, x_2, \ldots, x_n) \)  

(2.8)

subject to

\[ g^i(X) = 0, \quad \text{for } i = 1, 2, \ldots, m \]  

(2.9)

\[ h^i(X) \leq 0, \quad \text{for } i = 1, 2, \ldots, r \]  

(2.10)

By introducing slack variables, the inequality constraints can be written as equality constraints (Kothari and Dhillon, 2011). Therefore, the non-linear programming problem becomes

Maximize or Minimize \( Z = f(X) \)
subject to

\[ g^i(X) = 0, \quad \text{for } i = 1, 2, ..., m \]  
\[ h^i(X) - S_i = 0, \quad \text{for } i = 1, 2, ..., r \]  
\[ S_i \geq 0, \quad \text{for } i = 1, 2, ..., r \]  

To eliminate the above non-negativity constraints append the objective function with a logarithmic barrier term incorporating these constraints. The Lagrangian function is formulated as

\[ L(X, \lambda, \gamma, S) = f(X) - \mu \sum_{i=1}^{r} \log S_i - \sum_{i=1}^{m} \lambda_i g^i(X) - \sum_{i=1}^{r} \gamma_i (h^i(X) - S_i) \]  

where, the barrier parameter \( \mu > 0 \). The solutions of Lagrangian functions are defined by the Karush-Kuhn Tucker first order necessary conditions.

\[ \frac{\partial L}{\partial x_i} = 0, \quad \text{for } i = 1, 2, ..., n \]  
\[ \frac{\partial L}{\partial \lambda_i} = 0, \quad \text{for } i = 1, 2, ..., m \]  
\[ \frac{\partial L}{\partial \gamma_i} = 0, \quad \text{for } i = 1, 2, ..., r \]  
\[ \frac{\partial L}{\partial S_i} = 0, \quad \text{for } i = 1, 2, ..., r \]  

Such conditions are sufficient, if the problem is convex. The iteration of the algorithm consists in applying the Newton method to non-linear system of Karush-Kuhn-Tucker conditions.

By expanding the first order necessary conditions using Taylor’s expansion end these equations can be solved to obtain the incremental values of the variables \( x_i, \lambda_i, \gamma_i \) and \( S_i \) and the variables are updated with factor \( \delta \). i.e.,

\[ x_i = x_i + \delta \Delta x_i, \quad \text{for } i = 1, 2, ..., n \]  
\[ \lambda_i = \lambda_i + \delta \Delta \lambda_i, \quad \text{for } i = 1, 2, ..., m \]  
\[ \gamma_i = \gamma_i + \delta \Delta \gamma_i, \quad \text{for } i = 1, 2, ..., r \]  
\[ S_i = S_i + \delta \Delta S_i, \quad \text{for } i = 1, 2, ..., r \]
2.4 Numerical Optimization Methods

The advent of the digital computer has given a considerable impetus to the study of numerical methods for determining the maximum or minimum of a given function. As a result many algorithms have been proposed for solving the problem. This section is intended to be an introduction to the field of non-linear optimization and the ideas on which the algorithms are based. A very desirable property of an optimization algorithm is global convergence.

An algorithm is set to be globally convergent if, for any initial point \( x^0 \), if generates a sequence of points that converges to a point \( \bar{x} \) in the solution set. It is said to be locally convergent if there exists a \( \rho > 0 \) such that for any initial point \( x^0 \) such that \( \| x - x^0 \| < \rho \), if generates a sequence of points converging to \( \bar{x} \) in the solution set.

It should also be noted that for most algorithms, we set initial values for certain parameters such as the starting point and the initial step size as well as parameters for terminating the algorithm.

2.4.1 Newton’s Method

Newton’s method for a function of two variables is developed as follows

\[
f(x_1, x_2) = 0 \quad \text{and} \quad g(x_1, x_2) = 0 \tag{2.19}
\]

Take \( x_1^0 \) and \( x_2^0 \) be an initial approximate solution of above equation (Veerarajan and Ramachandran, 2010). The actual solution is given by \( (x_1^0 + h) \) and \( (x_2^0 + k) \) where, \( h \) and \( k \) is an incremental value of \( x_1 \) and \( x_2 \). Therefore

\[
f(x_1^0 + h, x_2^0 + k) = 0 \tag{2.20}
\]

\[
g(x_1^0 + h, x_2^0 + k) = 0 \tag{2.21}
\]

Expanding the equations (2.20) and (2.21) by Taylor’s series.

\[
f(x_1^0, x_2^0) + h \frac{\partial}{\partial x_1} [f(x_1^0, x_2^0)] + k \frac{\partial}{\partial x_2} [f(x_1^0, x_2^0)] = 0 \tag{2.22}
\]
\[ g(x_1^0, x_2^0) + h \frac{\partial}{\partial x_1} [g(x_1^0, x_2^0)] + k \frac{\partial}{\partial x_2} [g(x_1^0, x_2^0)] = 0 \]  
\[ (2.23) \]

Equations (2.22) and (2.23) can be written as

\[ f_0 + h(f_{x_1})_0 + k(f_{x_2})_0 = 0 \]  
\[ (2.24) \]
\[ g_0 + h(g_{x_1})_0 + k(g_{x_2})_0 = 0 \]  
\[ (2.25) \]

Solving the equations (2.24) and (2.25) for \( h \) and \( k \) using determinants i.e.,

\[ h = -\frac{D_{x_1}}{D} \quad \text{and} \quad k = -\frac{D_{x_2}}{D} \]

where,

\[ D = \begin{vmatrix} (f_{x_1})_0 & (f_{x_2})_0 \\ (g_{x_1})_0 & (g_{x_2})_0 \end{vmatrix} \]

\[ D_{x_1} = \begin{vmatrix} f_0 & (f_{x_2})_0 \\ g_0 & (g_{x_2})_0 \end{vmatrix} \]

\[ D_{x_2} = \begin{vmatrix} (f_{x_1})_0 & f_0 \\ (g_{x_1})_0 & g_0 \end{vmatrix} \]

By using the incremental values, we get the new values of \( x_1 \) and \( x_2 \) and then incremental values of \( h \) and \( k \) are calculated. The iteration is continued till the required accuracy is obtained.

**Algorithm**

The algorithm of Newton’s method for solving non-linear equations is given in the following Steps:

**Step 1:** Input: function, precision \( \varepsilon \) and initial approximate solution \( x_1^0, x_2^0 \).

**Step 2:** Calculate determinant values of \( D, D_{x_1} \) and \( D_{x_2} \).

**Step 3:** Find the incremental values \( h \) and \( k \) using determinants.

**Step 4:** Determine the better approximation for the solution as \( x_1 = x_1^0 + h \), \( x_2 = x_2^0 + k \).
Step 5: If $|x_1 - x_0| < \varepsilon$ and $|x_2 - x_0| < \varepsilon$, go to step 7.

Step 6: Assume $x_1$, $x_2$ as initial approximation, i.e., $x_0^1 = x_1$ and $x_0^2 = x_2$, go to step 2.

Step 7: Print $x_1$ and $x_2$, and stop.

### 2.4.2 Superlinear Bracketing Method

In superlinear bracketing method, it is assumed that the function $f(x)$ is continuous and has zero in the closed interval $[x_1, x_2]$ where, $f(x_1)f(x_2) < 0$. The next better approximate value is calculated by Muller’s method. In Muller’s method the second degree parabola $y = f(x)$ is assumed to pass through the three points. The first and second points are interval border points. The third point is calculated as mid point of the first two points and is denoted by $x_0$ where $x_0 = \frac{x_1 + x_2}{2}$. Therefore three points are $(x_1, f(x_1))$, $(x_0, f(x_0))$ and $(x_2, f(x_2))$. For the first iteration, the second degree polynomial passes through three points. The interpolating polynomial can be constructed as,

$$P(x) = A(x - x_0)^2 + B(x - x_0) + C$$

where, $A$, $B$ and $C$ are unknown parameters.

The polynomial equation passing through the three points, say $x_1$, $x_0$ and $x_2$ is given by

$$f(x_1) = A(x_1 - x_0)^2 + B(x_1 - x_0) + C$$

$$f(x_2) = A(x_2 - x_0)^2 + B(x_2 - x_0) + C$$

$$f(x_0) = C$$

The values of A and B can be obtained by solving equations (2.27), (2.28) and (2.29).

Subtracting equation (2.29) from the equation (2.27) gives

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = A(x_1 - x_0) + B$$

(2.30)
Subtracting equation (2.29) from the equation (2.28) gives
\[ \frac{f(x_2) - f(x_0)}{x_2 - x_0} = A(x_2 - x_0) + B \] (2.31)

Subtracting equation (2.31) from the equation (2.30) gives
\[ \frac{f(x_1) - f(x_0)}{x_1 - x_0} - \frac{f(x_2) - f(x_0)}{x_2 - x_0} = A(x_1 - x_0 - x_2 + x_0) \] (2.32)
therefore,
\[ A = \frac{f(x_1) - f(x_0)}{(x_1 - x_0)(x_1 - x_2)} - \frac{f(x_0) - f(x_2)}{(x_2 - x_0)(x_1 - x_2)} \] (2.33)

By substituting equation (2.33) in equation (2.30), we get
\[ \frac{f(x_1) - f(x - 0)}{x_1 - x_0} - \left[ \frac{f(x_1) - f(x_0)}{(x_1 - x_0)(x_1 - x_2)} + \frac{f(x_0) - f(x_2)}{(x_2 - x_0)(x_1 - x_2)} \right] (x_1 - x_0) = B \]
therefore,
\[ B = \frac{[f(x_0) - f(x_1)(x_2 - x_0))}{(x_1 - x_0)(x_1 - x_2)} - \frac{[f(x_0) - f(x_2)(x_1 - x_0)]}{(x_2 - x_0)(x_1 - x_2)} \] (2.34)

If \( x_P \) is root of the polynomial \( P(x) = 0 \) then
\[ A(x_P - x_0)^2 + B(x_P - x_0) + C = 0 \] (2.35)

From the equation (2.35),
\[ x_P - x_0 = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \] (2.36)

Multiplying both the numerator and denominator of equation (2.36) by \( (B \pm \sqrt{B^2 - 4AC}) \), we get
\[ x_P - x_0 = -\frac{2C}{B \pm \sqrt{B^2 - 4AC}} \]
therefore,
\[ x_P = x_0 - \frac{-2C}{B \pm \sqrt{B^2 - 4AC}} \] (2.37)

In equation (2.37) both roots are calculated and the root which is closer to \( x_0 \) is selected.
If \( f(x_1)f(x_0) < 0 \) then the root lies between \( x_1 \) and \( x_0 \).
If \( f(x_1)f(x_0) \geq 0 \) then the root lies between \( x_2 \) and \( x_0 \).

In next iteration, \( x_0 \) has a new value. Take \( x_0 = x_P \) and now equation (2.37) can be written as iterative form. Substitute \( x_i = x_0 \) for \( i \in \mathbb{N} \) as an approximate value of the root in the current iteration and \( x_{i+1} = x_P \). A sign before the square root in equation (2.37) and the sign of \( B \) must be equal.

\[
x_{i+1} = x_i - \frac{2f(x_i)}{B_i + \text{sign}(B_i)\sqrt{B_i^2 - 4A_if(x_i)}},
\]

(2.38)

where, \( A, B \) and \( C \) from equations (2.33), (2.34) and (2.29) are indexed by \( i \) in each iteration on the interval \([x_a, x_b]\) as follows

\[
A_i = \frac{f(x_1) - f(x_i)}{(x_1 - x_i)(x_1 - x_2)} + \frac{f(x_i) - f(x_2)}{(x_2 - x_i)(x_1 - x_2)}
\]

\[
B_i = \frac{f(x_i) - f(x_1)}{(x_1 - x_i)(x_2 - x_i)} + \frac{f(x_i) - f(x_2)}{(x_2 - x_i)(x_1 - x_2)}
\]

\[
C_i = f(x_i)
\]

(2.39)

In each iteration the interval points \( x_1, x_2 \) and \( x_0 \) are selected according to the following procedure

(i) If \( f(x_1)f(x_2) < 0 \), then \( x_2 = x_i \)

(ii) If \( f(x_1)f(x_2) \geq 0 \), then \( x_1 = x_i \)

and finally substitute \( x_0 = x_{i+1} \)

Suppose, if the minimal value of \( |x_{i+1} - x_i| \) corresponding to the roots lies outside of the interval \([x_1, x_2]\) then equation (2.36) can be written as iterative form.

\[
x_{i+1} = x_i - \frac{B_i + \text{sgn}(B_i)\sqrt{B_i^2 - 4A_iC_i}}{2A_i}
\]

(2.40)

The iterative process is continued until we get the required accuracy.

i.e., \( |x_{i+1} - x_i| \) is less than the specified value.

**Algorithm**

The algorithm of superlinear bracketing method for solving nonlinear equations (Suhadolnik, 2013) is given in the following Steps:

Step 1: Input: The inputs are the left side \( x_a \) and right side \( x_b \) of the interval, function \( f(x) \), precision \( \varepsilon \) and maximum number of iterations \( N_{\text{max}} \).
Step 2: Initial mid-point:  \( x_c = (x_a + x_b)/2 \),  \( n = 1 \),  \( x_{0c} = x_c \)

Step 3: Function values:  \( f_a = f(x_a); \)  \( f_b = f(x_b); \)  \( f_c = f(x_c) \).

Step 4: Initial condition check:  \( f(x_a)f(x_b) < 0 \) is fulfilled.

Step 5: Calculate  \( A, B \) and  \( C \) using equations (2.29), (2.33) and (2.34).

Step 6: Roots of the polynomial:  \( x_p = x_c - \dfrac{2C}{B + sgn(B) \sqrt{B^2 - 4AC}} \).

Step 7: if  \( x_p > x_b \) or  \( x_p > x_a \) then  \( x_p = x_c - \dfrac{B + sgn(B) \sqrt{B^2 - 4AC}}{2A} \).

Step 8: Set  \( f_p = f(x_p) \).

Step 9: New endpoints of the interval: If  \( f_a f_c < 0 \) then  \( x_b = x_c \),  \( f_b = f_c \) else  \( x_a = x_c \),  \( f_a = f_c \).

Step 10: Set  \( x_c = x_p \),  \( f_c = f_p \).

Step 11: if  \( n > N_{max} \) then stop.

Step 12: if  \( n > 1 \) and  \( |x_{0c} - x_c| < \epsilon \) then print  \( x_c \),  \( f_c \) and stop.

Step 13: Iteration  \( n = n + 1 \),  \( x_{0c} = x_c \).

Step 14: Go to step 5.