EEE 282: Numerical Techniques
Experiment 1: Interpolation

Introduction:

Forming a polynomial:

A polynomial, p(x) of degree n in MATLAB is stored as a row vector, **p**, of length n+1. The components represent the coefficients of the polynomial and are given in the descending order of the power of x, that is
p = [a_n a_{n-1} ……….. a_1 a_0]
is interpreted as
$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots\ldots\ldots + a_1 x + a_0$

In MATLAB the following commands are used to evaluate a polynomial:
polyval, **poly**, **roots**, **conv** etc.

Lab Task 1: Construct a polynomial such that C(x)= A(x)*B(x)
Where $A(x)= 3x^2+2x-4$ and $B(x)= 2x^3-2$ Also find the roots of A(x), B(x) and C(x).

Interpolation:

In the mathematical subfield of numerical analysis, **interpolation** is a method of constructing new data points from a discrete set of known data points.

In engineering and science one often has a number of data points, as obtained by sampling or some experiment, and tries to construct a function which closely fits those data points. This is called curve fitting. Interpolation is a specific case of curve fitting, in which the function must go exactly through the data points.

Definition:

Given a sequence of *n distinct* numbers x_k called **nodes** and for each x_k a second number y_k, we are looking for a function *f* so that

$$f(x_k) = y_k \ , \ k = 1, \ldots, n$$

A pair x_k,y_k is called a **data point** and *f* is called the **interpolant** for the data points.

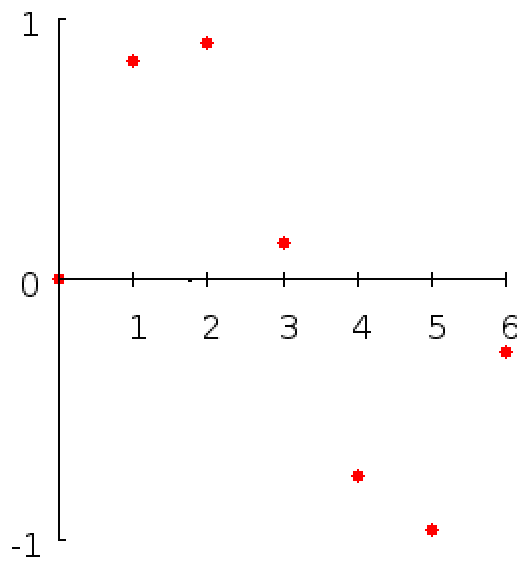
For example, suppose we have a table like this, which gives some values of an unknown function *f*. The data are given in the table:

Table 1

x	*f(x)*
0	0
1	0.8415
2	0.9093
3	0.1411
4	-0.7568
5	-0.9589
6	-0.2794

The plot can be shown as:

.

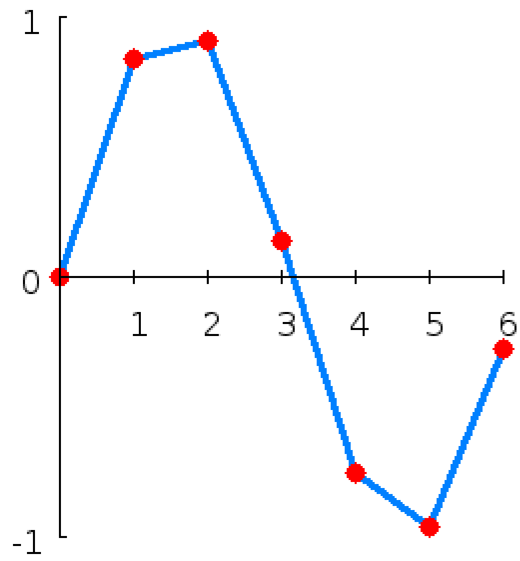


What value does the function have at, say, $x = 2.5$? Interpolation answers questions like this.

Types of interpolation:

A. Linear interpolation

One of the simplest methods is linear interpolation. Consider the above example of determining $f(2.5)$. We join the data points by linear interpolation and get the following plot:



Noe we can get $f(2.5)$. Since 2.5 is midway between 2 and 3, it is reasonable to take $f(2.5)$ midway between $f(2) = 0.9093$ and $f(3) = 0.1411$, which yields 0.5252.

Generally, linear interpolation takes two data points, say (x_a,y_a) and (x_b,y_b), and the interpolant is given by

$$f(x) = \frac{x - x_b}{x_a - x_b} y_a - \frac{x - x_a}{x_a - x_b} y_b$$

This formula can be interpreted as a weighted mean.

Linear interpolation is quick and easy, but it is not very precise.

Lab Task 2. Write a MATLAB code to implement Linear Interpolation and Plot the curve corresponding to table1 using Linear Interpolation.

B. Polynomial interpolation

Polynomial interpolation is a generalization of linear interpolation. Note that the linear interpolant is a linear function. We now replace this interpolant by a polynomial of higher degree.

Consider again the problem given above. The following sixth degree polynomial goes through all the seven points:

$$\boldsymbol{f(x) = -0.0001521x^6 - 0.003130x^5 + 0.07321x^4 - 0.3577x^3 + 0.2255x^2 + 0.9038x}$$

Substituting $x = 2.5$, we find that $f(2.5) = 0.5965$.

Generally, if we have n data points, there is exactly one polynomial of degree $n-1$ going through all the data points. The interpolation error is proportional to the distance between the data points to the power n.

However, polynomial interpolation also has some disadvantages. Calculating the interpolating polynomial is relatively very computationally expensive Furthermore, polynomial interpolation may not be so exact after all, especially at the end points.

a. Lagrange Polynomial:

The Lagrange interpolating polynomial is the polynomial $P(x)$ of degree $(n-1)$ that passes through the n points $(x_1, y_1 = f(x_1))$, $(x_2, y_2 = f(x_2))$, ..., $(x_n, y_n = f(x_n))$, and is given by

$$P(x) = \sum_{j=1}^{n} P_j(x),$$

where

$$P_j(x) = y_j \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}.$$

Written explicitly,

$$P(x) = \frac{(x-x_2)(x-x_3)\cdots(x-x_n)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_n)} y_1 + \frac{(x-x_1)(x-x_3)\cdots(x-x_n)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_n)} y_2 + \cdots + \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\cdots(x_n-x_{n-1})} y_n.$$

When constructing interpolating polynomials, there is a tradeoff between having a better fit and having a smooth well-behaved fitting function. The more data points that are used in the interpolation, the higher the degree of the resulting polynomial, and therefore the greater oscillation it will exhibit between the data points. Therefore, a high-degree interpolation may be a poor predictor of the function between points, although the accuracy at the data points will be "perfect."

For $n = 3$ points,

$$P(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} y_3$$

Note that the function $P(x)$ passes through the points (x_i, y_i), as can be seen for the case $n = 3$,

$$
\begin{aligned}
P(x_1) &= \frac{(x_1 - x_2)(x_1 - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x_1 - x_1)(x_1 - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x_1 - x_1)(x_1 - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 = y_1 \\
P(x_2) &= \frac{(x_2 - x_2)(x_2 - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x_2 - x_1)(x_2 - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x_2 - x_1)(x_2 - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 = y_2 \\
P(x_3) &= \frac{(x_3 - x_2)(x_3 - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x_3 - x_1)(x_3 - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x_3 - x_1)(x_3 - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 = y_3.
\end{aligned}
$$

Algorithm for the Lagrange Polynomial: To construct the Lagrange polynomial

$$P(x) = \sum_{k=0}^{n} y_k L_{n,k}(x)$$

of degree n, based on the n+1 points (x_k, y_k) for $k = 0, 1, \cdots, n$. The Lagrange coefficient polynomials for degree n are:

$$L_{n,k}(x) = \frac{(x - x_0) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}$$

for $k = 0, 1, \cdots, n$.

So, for a given x and a set of $(N+1)$ data pairs, (x_i, f_i), $i= 0, 1, \ldots\ldots N$:

Set SUM=0

DO FOR *i*=0 to *N*

Set *P*=1

DO FOR *j*=0 to *N*

IF *j~=i*

Set *P=P*(x-x(j))/(x(i)-x(j))*

End DO(*j*)

Lab Task 3. Construct Lagrange interpolating Polynomials for the data points given in table -1 and Plot the curve.

EEE 282: Numerical Techniques
Experiment 2: Regression (Curve Fitting)

Introduction:

Data is often given for discrete values along a continuum. However we may require estimates at points between the discrete values. Then we have to fit curves to such data to obtain intermediate estimates. In addition, we may require a simplified version of a complicated function. One way to do this is to compute values of the function at a number of discrete values along the range of interest. Then a simpler function may be derived to fit these values. Both of these applications are known as **curve fitting**.

There are two general approaches of curve fitting that are distinguished from each other on the basis of the amount of error associated with the data. First, where the data exhibits a significant degree of error, the strategy is to derive a single curve that represents the general trend of the data. Because any individual data may be incorrect, we make no effort to intersect every point. Rather, the curve is designed to follow the pattern of the points taken as a group. One approach of this nature is called ***least squares regression***.

Second, where the data is known to be very precise, the basic approach is to fit a curve that passes directly through each of the points. The estimation of values between well known discrete points from the fitted exact curve is called ***interpolation***.

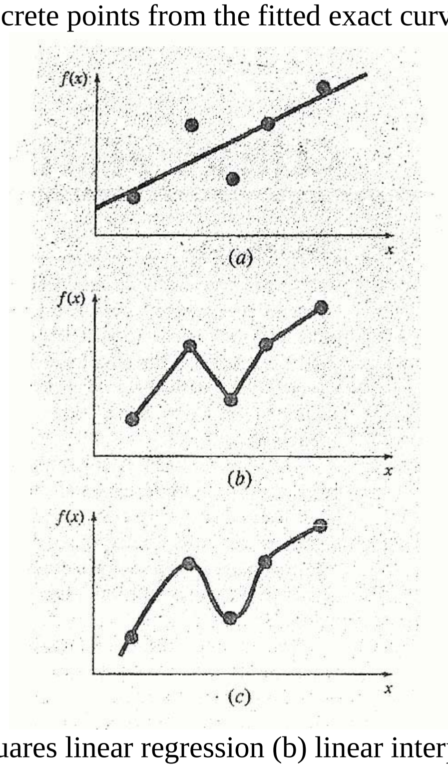


Figure 1: (a) Least squares linear regression (b) linear interpolation (c) curvilinear interpolation

Least squares Regression:

Where substantial error is associated with data, polynomial interpolation is inappropriate and may yield unsatisfactory results when used to predict intermediate values. A more appropriate strategy for such cases is to derive an approximating function that fits the shape or general trend of the data without necessarily matching the individual points. Now some criterion mush be devised to establish a basis for the fit. One way to do this is to derive a curve that minimizes the discrepancy between the data points and the curve. A technique for accomplishing this objective is called least squares regression, where the goal is to minimize the sum of the square errors between the data points and the curve. Now depending on whether we want to fit a straight line or other higher order polynomial, regression may be linear or polynomial. They are described below.

Linear regression:

The simplest example of least squares regression is fitting a straight line to a set of paired observations: (x1, y1), (x2, y2), , , ,(xn, yn). The mathematical expression for straight line is

$y_m=a_0+a_1x$

Where a_0 and a_1 are coefficients representing the intercept and slope and y_m is the model value. If y_0 is the observed value and e is error or residual between the model and observation then

$e=y_0-y_m=y_0 - a_0 - a_1x$

Now we need some criteria such that the error e is minimum and also we can arrive at a unique solution (for this case a unique straight line). One such strategy is to minimize the sum of the square errors. So sum of square errors

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_{i,observed} - y_{i,\mathrm{mod}el})^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots..1$$

To determine the values of a_0 and a_1, equation (1) is differentiated with respect to each coefficient.

$$\frac{\partial S_r}{\partial a_0} = -2\sum (y_i - a_0 - a_1 x_i)$$

$$\frac{\partial S_r}{\partial a_1} = -2\sum (y_i - a_0 - a_1 x_i) x_i$$

Setting these derivatives equal to zero will result in a minimum Sr. If this is done, the equations can be expressed as

$$0 = \sum y_i - \sum a_0 - \sum a_1 x_i$$

$$0 = \sum y_i x_i - \sum a_0 x_i - \sum a_1 x_i^2$$

Now realizing that $\sum a_0 = na_0$ **,** we can express the above equations as a set of two simultaneous linear equations with two unknowns a_0 and a_1.

$$na_0 + (\sum x_i)a_1 = \sum y_i$$

$$(\sum x_i)a_0 + (\sum x_i^2)a_1 = \sum x_i y_i$$

from where

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$$

$$a_0 = \overline{y} - a_1 \overline{x}$$

Where $\overline{y}$ and $\overline{x}$ are the means of y and x respectively

Lab Task 1:

Fit a straight line to the x and y values of table 1

Table 1:

x	y
1	0.5
2	2.5
3	2.0
4	4.0
5	3.5
6	6.0
7	5.5

Ans: a_0=0.071142857, a_1=0.8392857

Polynomial Regression:

In some cases, we have some engineering data that cannot be properly represented by a straight line. We can fit a polynomial to these data using polynomial regression.

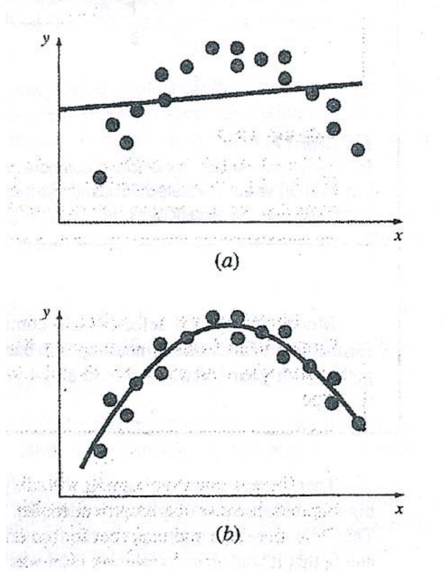


Figure 2: (a) Data that is ill-suited for linear least squares regression (b) indication that a parabola is preferable

The least squares procedure can be readily extended to fit the data to a higher order polynomial. For example, we want to fit a second order polynomial

$y_m = a_0 + a_1x + a_2x^2$

For this case the sum of the squares of residuals is

$$S_r = \sum_{i=1}^{n} (y_i - a_0 - a_1x_i - a_2x_i^2)^2 \;\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 2$$

Taking derivative of equation (2) with respect to unknown coefficients a_0, a_1 and a_2

$$\frac{\partial S_r}{\partial a_0} = -2\sum (y_i - a_0 - a_1x_i - a_2x_i^2)$$
$$\frac{\partial S_r}{\partial a_1} = -2\sum x_i(y_i - a_0 - a_1x_i - a_2x_i^2)$$
$$\frac{\partial S_r}{\partial a_2} = -2\sum x_i^2(y_i - a_0 - a_1x_i - a_2x_i^2)$$

These equations can be set equal to zero and rearranged to develop the following set of normal equations:

$$na_0 + (\sum x_i)a_1 + (\sum x_i^2)a_2 = \sum y_i$$
$$(\sum x_i)a_0 + (\sum x_i^2)a_1 + (\sum x_i^3)a_2 = \sum x_iy_i$$
$$(\sum x_i^2)a_0 + (\sum x_i^3)a_1 + (\sum x_i^4)a_2 = \sum x_i^2y_i$$

Now a_0, a_1 and a_2 can be calculated using matrix inversion.

Lab Task 2: Fit a second order polynomial to the data given in table 2

Table 2

x	y
0	2.1
1	7.7
2	13.6
3	27.2
4	40.9
5	61.1

Ans: a_0=2.47857, a_1=2.35929, a_2=1.86071

EEE 282: Numerical Techniques
Experiment 3: Simultaneous Linear Equation

Objective

Systems of linear algebraic equations occur often in diverse fields of science and engineering and is an important area of study. In this experiment we will be concerned with the different techniques of finding solution of a set of n linear algebraic equations in n unknowns.

Concept of linear equations and their solution

A set of linear algebraic equations looks like this:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots a_{1N}x_N = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \ldots a_{2N}x_N = b_2 \\ &\ldots \quad \ldots \quad \ldots \quad \ldots \\ &a_{M1}x_1 + a_{M2}x_2 + \ldots a_{MN}x_N = b_M \end{aligned} \tag{1}$$

Here the N unknowns x_j , $j = 1, 2, \ldots, N$ are related by M equations. The coefficients a_{ij} with $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$ are known numbers, as are the *right-hand side* quantities b_i, $i = 1, 2, \ldots, M$.

Existence of solution

If $N = M$ then there are as many equations as unknowns, and there is a good chance of solving for a unique solution set of x_j's. Analytically, there can fail to be a unique solution if one or more of the M equations is a linear combination of the others (This condition is called *row degeneracy*), or if all equations contain certain variables only in exactly the same linear combination(This is called *column degeneracy*). (For square matrices, a row degeneracy implies a column degeneracy, and vice versa.) A set of equations that is degenerate is called *singular*.

Numerically, at least two additional things can go wrong:

• While not exact linear combinations of each other, some of the equations may be so close to linearly dependent that round off errors in the machine renders them linearly dependent at some stage in the solution process. In this case your numerical procedure will fail, and it can tell you that it has failed.

• Accumulated round off errors in the solution process can swamp the true solution. This problem particularly emerges if N is too large. The numerical procedure does not fail algorithmically. However, it returns a set of x's that are wrong, as can be discovered by direct substitution back into the original equations. The closer a set of equations is to being singular, the more likely this is to happen.

Matrices

Equation (1) can be written in matrix form as

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{2}$$

Here the raised dot denotes matrix multiplication, **A** is the matrix of coefficients, **x** is the column vector of unknowns and **b** is the right-hand side written as a column vector,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ .. \\ x_N \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ .. \\ b_M \end{bmatrix}$$

Finding Solution

There are so many ways to solve this set of equations. Below are some important methods.

(1) Using the backslash and pseudo-inverse operator

In MATLAB, the easiest way to determine whether Ax = b has a solution, and to find such a solution when it does, is to use the backslash operator. Exactly what $\mathbf{A} \backslash \mathbf{b}$ returns is a bit complicated to describe, but if there is a solution to $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, then $\mathbf{A} \backslash \mathbf{b}$ returns one. Warnings: (1) $\mathbf{A} \backslash \mathbf{b}$ returns a result in many cases when there is no solution to $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. (2) $\mathbf{A} \backslash \mathbf{b}$ sometimes causes a warning to be issued, even when it returns a solution. This means that you can't just use the backslash operator: you have to check that what it returns is a solution. (In any case, it's just good common sense to check numerical computations as you do them.) In MATLAB this can be done as follows:

Using backslash operator:
```
x = A\b;
```

You can also use the pseudo-inverse operator:
```
x=pinv(A)*b; % it is also guaranteed to solve Ax = b, if Ax = b has a solution.
```

As with the backslash operator, you have to check the result.

(2) Using Gauss-Jordan Elimination and Pivoting

To illustrate the method let us consider three equations with three unknowns:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14} \qquad \text{(A)}$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = a_{24} \qquad \text{(B)}$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = a_{34} \qquad \text{(C)}$$

Here the quantities b_i, $i = 1, 2, \dots, M$'s are replaced by a_{iN+1}, where i=1,2, ….M for simplicity of understanding the algorithm.

The First Step is to eliminate the first term from Equations (B) and (C). (Dividing (A) by a_{11} and multiplying by a_{21} and subtracting from (B) eliminates x_1 from (B) as shown below)

$$(a_{21} - \frac{a_{11}}{a_{11}}a_{21})x_1 + (a_{22} - \frac{a_{12}}{a_{11}}a_{21})x_2 + (a_{23} - \frac{a_{13}}{a_{11}}a_{21})x_3 = (a_{24} - \frac{a_{14}}{a_{11}}a_{21})$$

Let, $\frac{a_{21}}{a_{11}} = k_2$, then

$$(a_{21} - k_2 a_{11})x_1 + (a_{22} - k_2 a_{12})x_2 + (a_{23} - k_2 a_{13})x_3 = (a_{24} - k_2 a_{14})$$

Similarly multiplying equation (A) by $\frac{a_{31}}{a_{11}} = k_3$ and subtracting from (C), we get

$$(a_{31} - k_3 a_{11})x_1 + (a_{32} - k_3 a_{12})x_2 + (a_{33} - k_3 a_{13})x_3 = (a_{34} - k_3 a_{14})$$

Observe that $(a_{21} - k_2 a_{11})$ and $(a_{31} - k_3 a_{11})$ are both zero.

In the steps above it is assumed that a_{11} is not zero. This case will be considered later in this experiment.

The above elimination procedure is called triangularization.

<u>Algorithm for triangularizing n equations in n unknowns:</u>

```
for i = 1 to n and j = 1 to (n+1) in steps of 1 do read a_ij endfor
for k = 1 to (n-1) in steps of 1 do
    for i = (k+1) to n in steps of 1 do
    u <- a_ik / a_kk
    for j = k to (n+1) in steps of 1 do
        a_ij <- a_ij - u a_kj endfor
    endfor
endfor
```

The reduced equations are:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14}$$
$$a_{22}x_2 + a_{23}x_3 = a_{24}$$
$$a_{32}x_2 + a_{33}x_3 = a_{34}$$

The next step is to eliminate a_{32} from the third equation. This is done by multiplying second equation by $u = a_{32} / a_{22}$ and subtracting the resulting equation from the third. So, same algorithm can be used.

Finally the equations will take the form:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14}$$
$$a_{22}x_2 + a_{23}x_3 = a_{24}$$
$$a_{33}x_3 = a_{34}$$

The above set of equations are said to be in triangular (Upper) form.

From the above upper triangular form of equations, the values of unknowns can be obtained by back substitution as follows:

$$x_3 = a_{34} / a_{33}$$
$$x_2 = (a_{24} - a_{23}x_3) / a_{22}$$
$$x_2 = (a_{14} - a_{12}x_2 - a_{13}x_3) / a_{11}$$

Algorithmically, the back substitution for n unknowns is shown below:

```
x_n <- a_n(n+1) / a_nn
for i = (n-1) to 1 in step of -1 do
sum <- 0
    for j = (i+1) to n in steps of 1 do
    sum <- sum + a_ij x_j endfor
x_i <- (a_i(n+1) - sum) / a_ii
endfor
```

Lab Task 1. Given the simultaneous equations shown below, solve for x_1, x_2, x_3, using Gaussian Elimination.

$$\boxed{\begin{aligned} 2x_1 + 3x_2 + 5x_3 &= 23 \\ 3x_1 + 4x_2 + x_3 &= 14 \\ 6x_1 + 7x_2 + 2x_3 &= 26 \end{aligned}}$$

Pivoting

In the triangularization algorithm we have used,

$$u \leftarrow a_{ik} / a_{kk}$$

Here it is assumed that a_{kk} is not zero. If it happens to be zero or nearly zero, the algorithm will lead to no results or meaningless results. If any of the a_{kk} is small it would be necessary to reorder the equations. It is noted that the value of a_{kk} would be modified during the elimination process and there is no way of predicting their values at the start of the procedure.

The elements a_{kk} are called pivot elements. In the elimination procedure the pivot should not be zero or a small number. In fact for maximum precision the pivot element should be the largest in absolute value of all the elements below it in its column, i.e. a_{kk} should be picked up as the maximum of all a_{mk} where, $m \geq k$

So, during the Gauss elimination, a_{mk} elements should be searched and the equation with the maximum value of a_{mk} should be interchanged with the current position. For example if during elimination we have the following situation:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 4 \\ 0.3x_2 + 4x_3 &= 5 \\ -8x_2 + 3x_3 &= 6 \end{aligned}$$

As $|-8| > 0.3$, 2nd and 3rd equations should be interchanged to yield:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 4 \\ -8x_2 + 3x_3 &= 6 \\ 0.3x_2 + 4x_3 &= 5 \end{aligned}$$

It should be noted that interchange of equations does not affect the solution.

The algorithm for picking the largest element as the pivot and interchanging the equations is called pivotal condensation.

<u>Algorithm for pivotal condensation</u>

```
1    max ← |a_kk|
2    p ← k
3    for m = (k+1) to n in steps of 1 do
4        if (|a_mk| > max) then
5        max ← |a_mk|
6        p ← m
7        endif
     endfor
8    if (p ~= k)
9        for q = k to (n+1) in steps of 1 do
```

```
            temp ← a_kq
            a_kq ← a_pq
            a_pq ← temp
        endfor
    endif
```

Lab Task 2. Modify the MATLAB program of Naive Gaussian Elimination to include pivotal condensation and singularity check.

Lab Task 3. Try to solve the following systems of equations (i) Gaussian elimination, (ii) Gaussian elimination with pivoting

(A)
$$\begin{aligned} 2x_1 + 4x_2 - 6x_3 &= -4 \\ x_1 + 5x_2 + 3x_3 &= 10 \\ x_1 + 3x_2 + 2x_3 &= 5 \end{aligned}$$

(B)
$$\begin{aligned} x_1 + x_2 + 6x_3 &= 7 \\ -x_1 + 2x_2 + 9x_3 &= 2 \\ x_1 - 2x_2 + 3x_3 &= 10 \end{aligned}$$

(C)
$$\begin{aligned} 4x_1 + 8x_2 + 4x_3 &= 8 \\ x_1 + 5x_2 + 4x_3 - 3x_4 &= -4 \\ x_1 + 4x_2 + 7x_3 + 2x_4 &= 10 \\ x_1 + 3x_2 - 2x_4 &= -4 \end{aligned}$$

EEE 282: Numerical Techniques
Experiment 4: Numerical Differentiation

Introduction:

We are familiar with the analytical method of finding the derivative of a function when the functional relation between the dependent variable y and the independent variable x is known. However, in practice, most often functions are defined only by tabulated data, or the values of y for specified values of x can be found experimentally. Also in some cases, it is not possible to find the derivative of a function by analytical method. In such cases, the analytical process of differentiation breaks down and some numerical process have to be invented. The process of calculating the derivatives of a function by means of a set of given values of that function is called numerical differentiation. This process consists in replacing a complicated or an unknown function by an interpolation polynomial and then differentiating this polynomial as many times as desired.

Forward Difference Formula:

All numerical differentiation are done by expansion of Taylor series

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2} + \frac{f'''(x)h^3}{6} + \ldots\ldots\ldots\ldots\ldots..(1)$$

From (1)

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \ldots\ldots\ldots\ldots\ldots..(2)$$

Where, $O(h)$ is the truncation error, which consists of terms containing h and higher order terms of h.

Lab Task 1:

Write the MATLAB code for numerical differentiation using Forward Difference Method.
Given f(x) = e^x, find $f'(1)$ using h=10^{-1}, 10^{-2},,,, upto 10^{-6}. Find out the error in each case by comparing the calculated value with exact value.

Central Difference Formula (of order O (h^2)):

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2} + \frac{f'''(c_1)h^3}{6} \ldots\ldots\ldots (3)$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)h^2}{2} - \frac{f'''(c_2)h^3}{6} \ldots\ldots\ldots (4)$$

Using (3) and (4)

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \ldots\ldots\ldots (5)$$

Where, $O(h^2)$ is the truncation error, which consists of terms containing h^2 and higher order terms of h.

Lab Task 2:

Write the MATLAB code for numerical differentiation using Central Difference Method.
Given f(x) = sin (cos (1/x)) find $f'(1/\sqrt{2})$ using h=10^{-1}, 10^{-2},,,, up to 10^{-6}. Find out the error in each case by comparing the calculated value with exact value.

Central Difference Formula (of order O (h^4)):

Using Taylor series expansion it can be shown that

$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + O(h^4) \ldots\ldots\ldots (6)$$

Here the truncation error reduces to h^4

Richardson's Extrapolation:

We have seen that

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

Which can be written as

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} + Ch^2$$

Or, $f'(x) \approx D_0(h) + Ch^2$ ……………………………….(7)

If step size is converted to 2h

$$f'(x) \approx D_0(2h) + 4Ch^2 \text{ ………………………….……(8)}$$

Using (7) and (8)

$$f'(x) \approx \frac{4D_0(h) - D_0(2h)}{3} = \frac{-f_2 + 8f_1 - 8f_{-1} + f_{-2}}{12h} \text{ ……………….(9)}$$

Equation (9) is same as equation (6)

The method of obtaining a formula for $f'(x)$ of higher order from a formula of lower order is called extrapolation. The general formula for Richardson's extrapolation is

$$f'(x) = D_k(h) + O(h^{2k+2}) = \frac{4^k D_{k-1}(h) - D_{k-1}(2h)}{4^k - 1} + O(h^{2k+2}) \text{ ……(10)}$$

EEE 282: Numerical Techniques
Experiment 5: Numerical Integration

Introduction

There are two cases in which engineers and scientists may require the help of numerical integration technique. (1) Where experimental data is obtained whose integral may be required and (2) where a closed form formula for integrating a function using calculus is difficult or so complicated as to be almost useless. For example the integral

$$\Phi(t) = \int_0^x \frac{t^3}{e^t - 1}\, dt.$$

Since there is no analytic expression for $\Phi(x)$, numerical integration technique must be used to obtain approximate values of $\Phi(x)$.

Formulae for numerical integration called quadrature are based on fitting a polynomial through a specified set of points (experimental data or function values of the complicated function) and integrating (finding the area under the fitted polynomial) this approximating function. Any one of the interpolation polynomials studied earlier may be used.

Some of the Techniques for Numerical Integration

Trapezoidal Rule

Assume that the values of a function $f(x)$ are given at x_1, x_1+h, x_1+2h ……x_1+nh and it is required to find the integral of $f(x)$ between x_1 and x_1+nh. The simplest technique to use would be to fit **straight lines** through $f(x_1)$, $f(x_1+h)$ ……and to determine the **area** under this approximating function as shown in Fig 7.1.

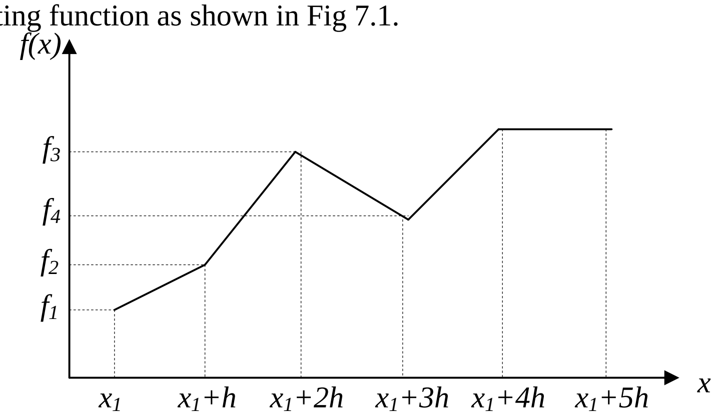


Fig. 7.1 Illustrating trapezoidal rule

For the first two points we can write:

$$\int_{x_1}^{x_1+h} f(x)dx = \frac{h}{2}(f_1 + f_2)$$

This is called first-degree Newton-Cotes formula.

From the above figure it is evident that the result of integration between x_1 *and* x_1+nh is nothing but the sum of areas of some trapezoids. In equation form this can be written as:

$$\int_{x_1}^{x_1+nh} f(x)dx = \sum_{i=1}^{n} \frac{(f_i + f_{i+1})}{2} h$$

The above integration formula is known as *Composite Trapezoidal rule*.
The composite trapezoidal rule can explicitly be written as:

$$\int_{x_1}^{x_1+nh} f(x)dx = \frac{h}{2}(f_1 + 2f_2 + 2f_3 + ...2f_n + f_{n+1})$$

Simpson's 1/3 Rule

This is based on approximating the function *f(x)* by fitting **quadratics** through sets of **three** points. For only three points it can be written as:

$$\int_{x_1}^{x_1+2h} f(x)dx = \frac{h}{3}(f_1 + 4f_2 + f_3)$$

This is called second-degree Newton-Cotes formula.
It is evident that the result of integration between x_1 *and* x_1+nh can be written as

$$\int_{x_1}^{x_1+nh} f(x)dx = \sum_{i=1,3,5,\ldots,n-1} \frac{h}{3}(f_i + 4f_{i+1} + f_{i+2})$$

$$= \frac{h}{3}(f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + 4f_6 + \ \ldots\ 4f_n + f_{n+1})$$

In using the above formula it is implied that *f* is known at an **odd number of points** (***n*+1 is odd,** where n is the no. of subintervals).

Simpson's 3/8 Rule

This is based on approximating the function *f(x)* by fitting **cubic** interpolating polynomial through sets of **four** points. For only four points it can be written as:

$$\int_{x_1}^{x_1+3h} f(x)dx = \frac{3h}{8}(f_1 + 3f_2 + 3f_3 + f_4)$$

This is called third-degree Newton-Cotes formula.
It is evident that the result of integration between x_1 *and* x_1+nh can be written as

$$\int_{x_1}^{x_1+nh} f(x)dx = \sum_{i=1,4,7,\ldots,n-2} \frac{h}{3}(f_i + 3f_{i+1} + 3f_{i+2} + f_{i+3})$$

$$= \frac{3h}{8}(f_1 + 3f_2 + 3f_3 + 2f_4 + 3f_5 + 3f_6 + 2f_7 + \ \ldots\ + 2f_{n-2} + 3f_{n-1} + 3f_n + f_{n+1})$$

In using the above formula it is implied that *f* is known at (*n+1*) points where **n is divisible by 3**.
An algorithm for integrating a **tabulated function** using composite trapezoidal rule:
Remarks: $f_1, f_2, \ldots\ldots\ldots, f_{n+1}$ are the tabulated values at $x_1, x_{1+h}, \ldots\ldots\ldots x_{1+nh}$ (*n+1* points)

1 Read h
2 *for* $i = 1$ to $n+1$ Read f_i *endfor*
3 $sum \leftarrow (f_1 + f_{n+1})/2$
4 *for* $j = 2$ to n *do*
5 $\quad sum \leftarrow sum + f_j$
endfor
6 $integral \leftarrow h\ .\ sum$
7 *write* $integral$
stop

Ex. 1. Integrate the function tabulated in Table 7.1 over the interval from x=1.6 to x=3.8 using composite trapezoidal rule with (a) h=0.2, (b) h=0.4 and (c) h=0.6

Table 7.1

X	*f(x)*	*X*	*f(x)*
1.6	4.953	2.8	16.445
1.8	6.050	3.0	20.086
2.0	7.389	3.2	24.533
2.2	9.025	3.4	29.964
2.4	11.023	3.6	36.598
2.6	13.468	3.8	44.701

The data in Table 7.1 are for $f(x) = e^x$. Find the true value of the integral and compare this with those found in (a), (b) and (c).

Ex. 2. (a) Integrate the function tabulated in Table 7.1 over the interval from x=1.6 to x=3.6 using Simpson's composite 1/3 rule.

(b) Integrate the function tabulated in Table 7.1 over the interval from x=1.6 to x=3.4 using Simpson's composite 3/8 rule.

An algorithm for integrating a **known function** using composite trapezoidal rule:

If *f(x)* is given as a closed form function such as $f(x) = e^{-x}\cos x$ and we are asked to integrate it from x_1 *to* x_2, we should decide first what *h* should be. Depending on the value of *h* we will have to evaluate the value of *f(x)* inside the program for $x=x_1+nh$ where n=0,1, 2,….n and $n = (x_2 - x_1)/h$.

1 $h = (x_2 - x_1)/n$

2 $x \leftarrow x_1$

3 $sum \leftarrow f(x)$

4 $for\ i = 2$ to $n\ do$

5 $\quad x \leftarrow x + h$

6 $\quad sum \leftarrow sum + 2f(x)$

$endfor$

7 $x \leftarrow x_2$

8 $sum \leftarrow sum + f(x)$

9 $\text{int}\,egral \leftarrow \frac{h}{2}\,.\,sum$

10 $write\ \text{int}\,egral$

$stop$

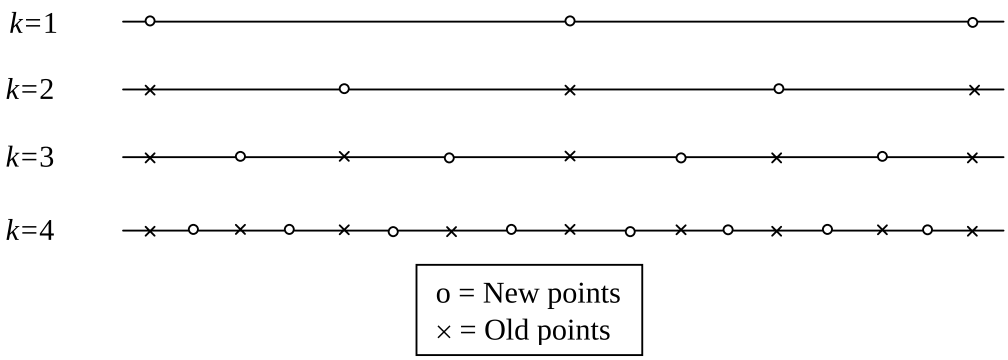
Ex. 3. (a) Find (approximately) each integral given below using the composite trapezoidal rule with $n = 12$.

$(i)\ \int_{-1}^{1} (1+x^2)^{-1}dx$ $\qquad$ $(ii)\ \int_{0}^{4} x^2 e^{-x} dx$

Adaptive Integration

When *f(x)* is a known function we can choose the value for *h* arbitrarily. The problem is that we do not know a priori what value to choose for *h* to attain a desired accuracy (for example, for an arbitrary *h* sharp picks of the function might be missed). To overcome this problem, we

can start with two subintervals, $h = h_1 = (x_2 - x_1)/2$ and apply either trapezoidal or Simpson's 1/3 rule. Then we let $h_2 = h_1/2$ and apply the formula again, now with four subintervals and the results are compared. If the new value is sufficiently close, the process is terminated. If the 2nd result is not close enough to the first, *h* is halved again and the procedure is repeated. This is continued until the last result is close enough to its predecessor. This form of numerical integration is termed as adaptive integration.
The no. of computations can be reduced because when *h* is halved, all of the old points at which the function was evaluated appear in the new computation and thus repeating evaluation can be avoided. This is illustrated below.



An algorithm for adaptive integration of a **known function** using trapezoidal rule:

Read x_1, x_2, e Remark: The allowed error in integral is *e*

$h \leftarrow x_2 - x_1$

$S_1 \leftarrow (f(x_1) + f(x_2))/2$

$I_1 \leftarrow h\ .\ S_1$

$i \leftarrow 1$

Repeat

    $x \leftarrow x_1 + h/2$

    for $j = 1$ to i *do*

        $S_1 \leftarrow S_1 + f(x)$

        $x \leftarrow x + h$

    endfor

    $i \leftarrow 2i$

    $h \leftarrow h/2$

    $I_0 \leftarrow I_1$

    $I_1 \leftarrow h\ .\ S_1$

until $|I_1 - I_0| \le e\ .\ |I_1|$

write $I_1,\ h,\ i$

stop

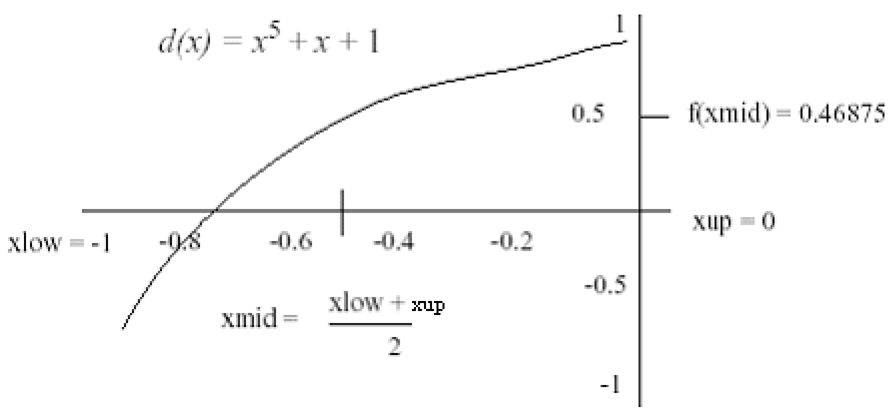
Ex. 4. Evaluate the integral of xe^{-2x^2} between *x*=0 and *x*=2 using a tolerance value sufficiently small as to get an answer within 0.1% of the true answer, 0.249916.

Ex. 5. Evaluate the integral of $\sin^2(16x)$ between $x = 0$ and $x = \pi/2$. Why the result is erroneous? How can this be solved? (The correct result is $\pi/4$)

EEE 282: Numerical Techniques
Experiment 6: Solution to Non-Linear Equation

Bisection method:

The Bisection method is one of the simplest procedures for finding root of a function in a given interval.
The procedure is straightforward. The approximate location of the root is first determined by finding two values that bracket the root (a root is bracketed or enclosed if the function changes sign at the endpoints). Based on these a third value is calculated which is closer to the root than the original two value. A check is made to see if the new value is a root. Otherwise a new pair of bracket is generated from the three values, and the procedure is repeated.



Consider a function $d(x)$ and let there be two values of x, x_{low} and x_{up} ($x_{up} >$ x_{low}), bracketing a root of $d(x)$.

Steps:

1. The first step is to use the brackets x_{low} and x_{up} to generate a third value that is closer to the root. This new point is calculated as the mid-point between x_{low} and, namely $x_{mid} = \frac{x_{low} + x_{up}}{2}$. The method therefore gets its name from this bisecting of two values. It is also known as interval halving method.
2. Test whether x_{mid} is a root of $d(x)$ by evaluating the function at x_{mid}.
3. If x_{mid} is not a root,

   a. If $d(x_{low})$ and $d(x_{mid})$ have opposite signs i.e. $d(x_{low}) \cdot d(x_{mid})$ <0, root is in left half of interval.

b. If $d(x_{low})$ and $d(x_{mid})$ have same signs i.e. $d(x_{low}) \,.\, d(x_{mid}) > 0$, root is in right half of interval.

4. Continue subdividing until interval width has been reduced to a size ≤ ε where ε = selected *x* tolerance.

Algorithm: Bisection Method

```
Input xLower, xUpper, xTol
yLower = f(xLower) (* invokes fcn definition *)
xMid = (xLower + xUpper)/2.0
yMid = f(xMid)
iters = 0 (* count number of iterations *)
While ( (xUpper - xLower)/2.0 > xTol )
iters = iters + 1
if( yLower * yMid > 0.0) Then xLower = xMid
Else xUpper = xMid
Endofif
xMid = (xLower + xUpper)/2.0
yMid = f(xMid)
Endofwhile
Return xMid, yMid, iters (* xMid = approx to root *)
```

Exercise 1. Find the real root of the equation $d(x) = x^5 + x + 1$ using Bisection Method. x_{low} = -1, x_{up} =0 and ε = selected *x* tolerance $=10^{-4}$.

Note: For a given *x* tolerance (epsilon), we can calculate the number of iterations directly. The number of divisions of the original interval is the smallest value of *n* that satisfies: $\frac{x_{up} - x_{low}}{2^n} \langle \varepsilon$ or $2^n \rangle \frac{x_{up} - x_{low}}{\varepsilon}$

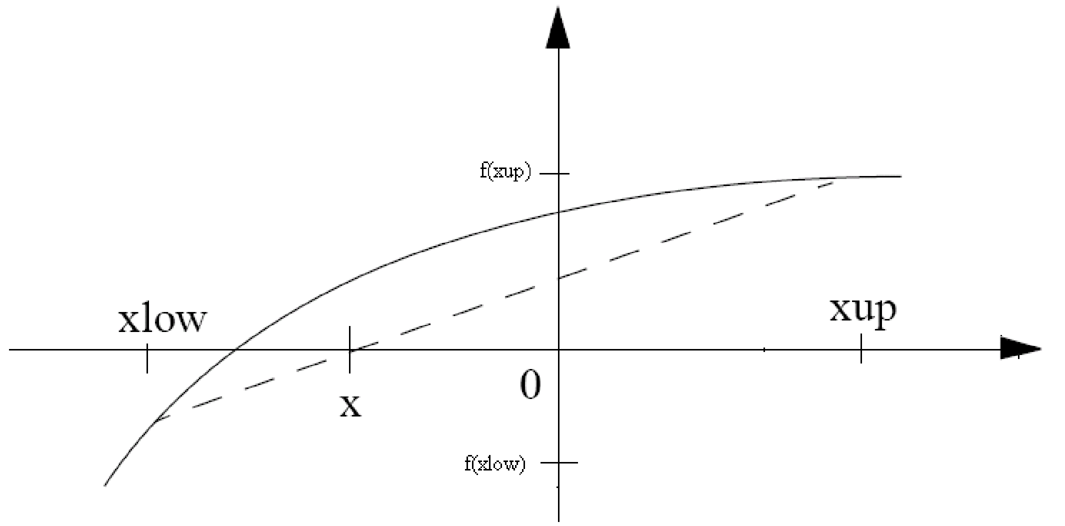
Thus $n \rangle \log 2\left(\frac{x_{up} - x_{low}}{\varepsilon}\right)$.

In our previous example, x_{low} = -1, x_{up} =0 and ε = selected *x* tolerance $=10^{-4}$.

So we have $n = 14$.

False-Position Method (Regula Falsi)

A shortcoming of the bisection method is that, in dividing the interval from x_{low} to x_{up} into equal halves, no account is taken of the magnitude of $f(x_{low})$ and $f(x_{up})$. For example, if $f(x_{low})$ is much closer to zero than $f(x_{up})$, it is likely that the root is closer to x_{low} than to x_{up}. An alternative method that exploits this graphical insight is to join $f(x_{low})$ and $f(x_{up})$ by a straight line. The intersection of this line with the x axis represents an improved estimate of the root. The fact that the replacement of the curve by a straight line gives the false position of the root is the origin of the name, method of false position, or in Latin, Regula Falsi. It is also called the Linear Interpolation Method.



Using similar triangles, the intersection of the straight line with the x axis can be estimated as

$$\frac{f(x_{low})}{x - x_{low}} = \frac{f(x_{up})}{x - x_{up}}$$

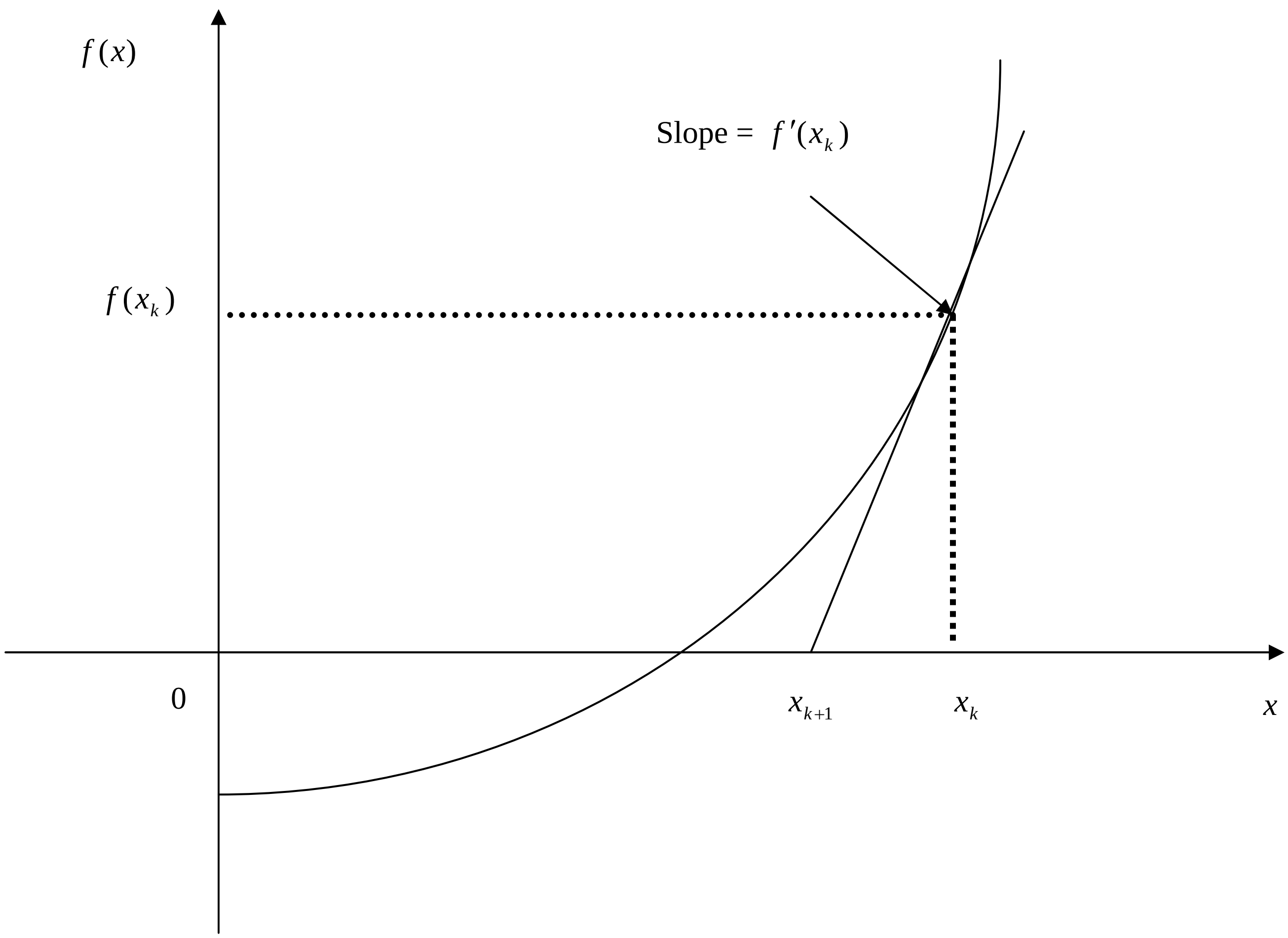
That is $x = x_{up} - \dfrac{f(x_{up})(x_{low} - x_{up})}{f(x_{low}) - f(x_{up})}$

This is the False Position formulae. The value of x then replaces whichever of the two initial guesses, x_{low} or x_{up}, yields a function value with the same sign as $f(x)$. In this way, the values of x_{low} and x_{up} always bracket the true root. The process is repeated until the root is estimated adequately.

Newton Raphson Method:

If $f(x)$, $f'(x)$ and $f''(x)$ are continuous near a root x, then this extra information regarding the nature of $f(x)$ can be used to develop algorithms that will produce sequences $\{x_k\}$ that converge faster to x than either the bisection or false position method. The Newton-Raphson (or simply Newton's) method is one of the most useful and best-known algorithms that relies on the continuity of $f'(x)$ and $f''(x)$.

The attempt is to locate root by repeatedly approximating $f(x)$ with a linear function at each step. If the initial guess at the root is x_k, a tangent can be extended from the point $[x_k, f(x_k)]$. The point where this tangent crosses the x axis usually represents an improved estimate of the root.



The Newton-Raphson method can be derived on the basis of this geometrical interpretation. As in the figure, the first derivative at x is equivalent to the slope:

$$f'(x_k) = \frac{f(x_k) - 0}{x_k - x_{k+1}}$$

which can be rearranged to yield

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

which is called the Newton Raphson formulae.

So the Newton-Raphson Algorithm actually consists of the following steps:

1. Start with an initial guess x_0 and an x-tolerance ε.
2. Calculate $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ $k = 0,1,2,\dots$

Algorithm - Newton's Method

```
Input x0, xTol
iters = 1
dx = -f(x0)/fDeriv(x0) (* fcns f and fDeriv *)
root = x0 + dx
While (Abs(dx) > xTol)
dx = -f(root)/fDeriv(root)
root = root + dx
iters = iters + 1
End of while
Return root, iters
```

Exercise 2. Use the Newton Raphson method to estimate the root of $f(x) = e^{-x} - 1$, employing an initial guess of $x_0 = 0$. The tolerance is = 10^{-8}.