Matoshri Education Society's

Matoshri College of Engineering and Research Centre, Nashik

Electrical Engineering Department

Second Year

Numerical Method & Computer Programming

PRACTICAL BOOK

Year 2014 – 2015

Name of Student: _____

Roll No: _____ **Exam No:** _____

Electrical Engineering Department,
Matoshri College of Engineering and Research Centre, Nashik

CERTIFICATE

This is to certify that Mr/Ms. _____

Seat No._____ Roll No._____ from **Second Year Electrical Engineering of Academic Year 2014 - 15** has successfully completed his / her Practical work on **Numerical Method & Computer Programming** at Matoshri College of Engineering and Research Centre, Nashik in the partial fulfillment of the Bachelors Degree in Engineering.

(Prof. S. S. Hadpe)	(Prof. S. S. Khule)	(Dr. G. K. Kharate)
Practical In-charge	**Head of Department**	**Principal**

Matoshri Education Society's

Matoshri College of Engineering & Research Center, Nashik

Electrical Engineering Department

INDEX

Sub: Numerical Method & Computer Programming **Class:** S.E.Electrical

Experiment No: 01 **Date: -**

Aim: Solution of a transcendental equation using Bisection.

Title: Use bisection method to determine the root of equation $f(x) = X^3 - 5X - 7$.

Theory:

The root of $f(x) = 0$ has been bracketed in the interval (a, b). Bisection method can be used to close in on it. The Bisection method accomplishes this by successfully halving the interval until it becomes sufficiently small. Once a and b has been bracketed, Bisection method will always close in on it.

We assume that $f(x)$ is a function that is real-valued and that x is a real variable. Suppose that $f(x)$ is continuous on an interval $a \le x \le b$ and that $f(a)\, f(b) < 0$. When this is the case, $f(x)$ will have opposite signs at the end points of the interval (a, b).

If the function $f(x)$ satisfies $f(a_0)\, f(b_0) < 0$, then the equation $f(x) = 0$ has at least one real root or an odd number of real roots in the interval (a_0, b_0). If $m1 = (a_0 + b_0)/2$ is the midpoint of this interval, then the root will lie either in the interval (a_0, m_1) or in the interval (m_1, b_0) provided that $f(m_1) \neq 0$. If $f(m_1) = 0$, then m_1 is the required root. Repeating this procedure a number of times, we obtain the bisection method.

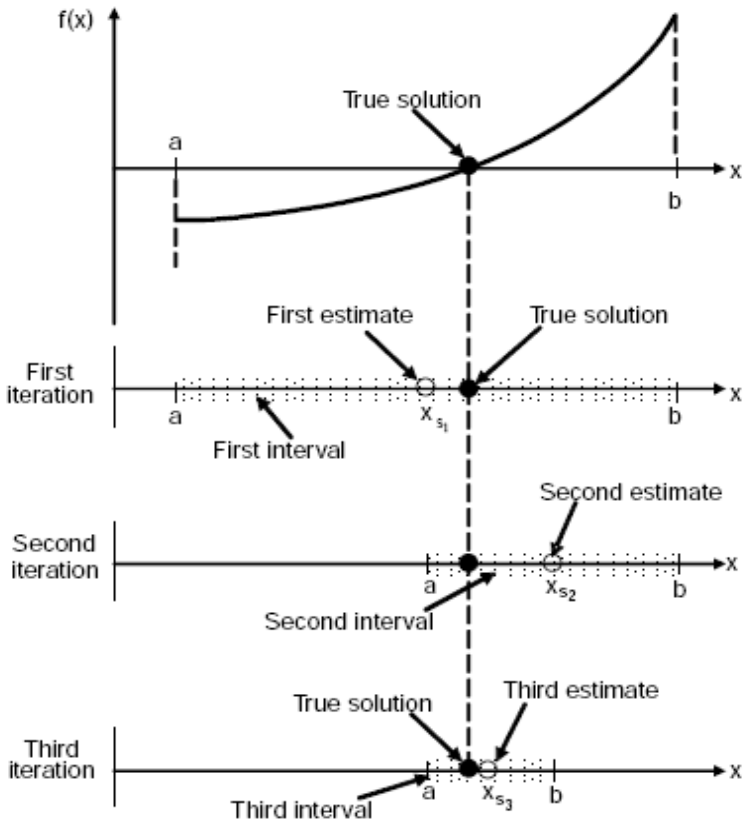


Fig. 2: Bisection method

The method of finding a solution with the Bisection method is illustrated in Fig. 2. It starts by finding point a_0 and b_0 that define an interval where a solution exists. The midpoint of the interval m_1 is then taken as the first estimate for the numerical solution. The true solution is either in the portion between point a_0 and m_1, or in the portion between points m_1 and b. If the solution obtained is not accurate enough, a new interval that contains the true solution is defined. The new interval selected is the half of the original interval that contains the true solution, and its midpoint is taken as the new (second) estimate of the numerical solution. The procedure is repeated until the numerical solution is accurate enough according to a certain criterion that is selected.

Procedure for the Bisection Method

1. Compute the first estimate $a_0 + b_0$ of the numerical solution m_1 by
2. Determine whether the true solution is between a and $m_1$or between m_1 and b by checking the sign of the product $f(a)\, f(m_1)$ has following conditions.

   If $f(a)\, f(m_1) < 0$, the true solution is between a and m_1.
   If $f(a)\, f(m_1) > 0$, the true solution is between m_1 and b.
   If $b - c \leq$ error, then accept c as the root and stop. is the error tolerance, $\in > 0$.
3. Choose the subinterval that contains the true solution (a to m_1) or (m_1 to b) as the new interval (a_1, b_1), and go back to step 1.

Steps 1 through 3 are repeated until a specified tolerance or error bound is attained.

Advantages of Bisection method

- The method is guaranteed to Accuracy. The method always to an answer, provided a root was bracketed in the interval (a, b) to start with. In addition, the error bound, is guaranteed to decrease by one-half with each iteration.

Disadvantage of Bisection method

- The method may fail when the function is tangent to the axis and does not cross the x-axis at $f(x) = 0$.
- The disadvantage of the Bisection method is that it generally more slowly than most other methods. For functions $f(x)$ that have a continuous derivative, other methods are usually faster.
- Bisection method requires large number of iteration.
- Bisection method requires two initial guess.
- initial guess a and b *of* $f(x)$ must be bound as $f(a)\, f(b) < 0$

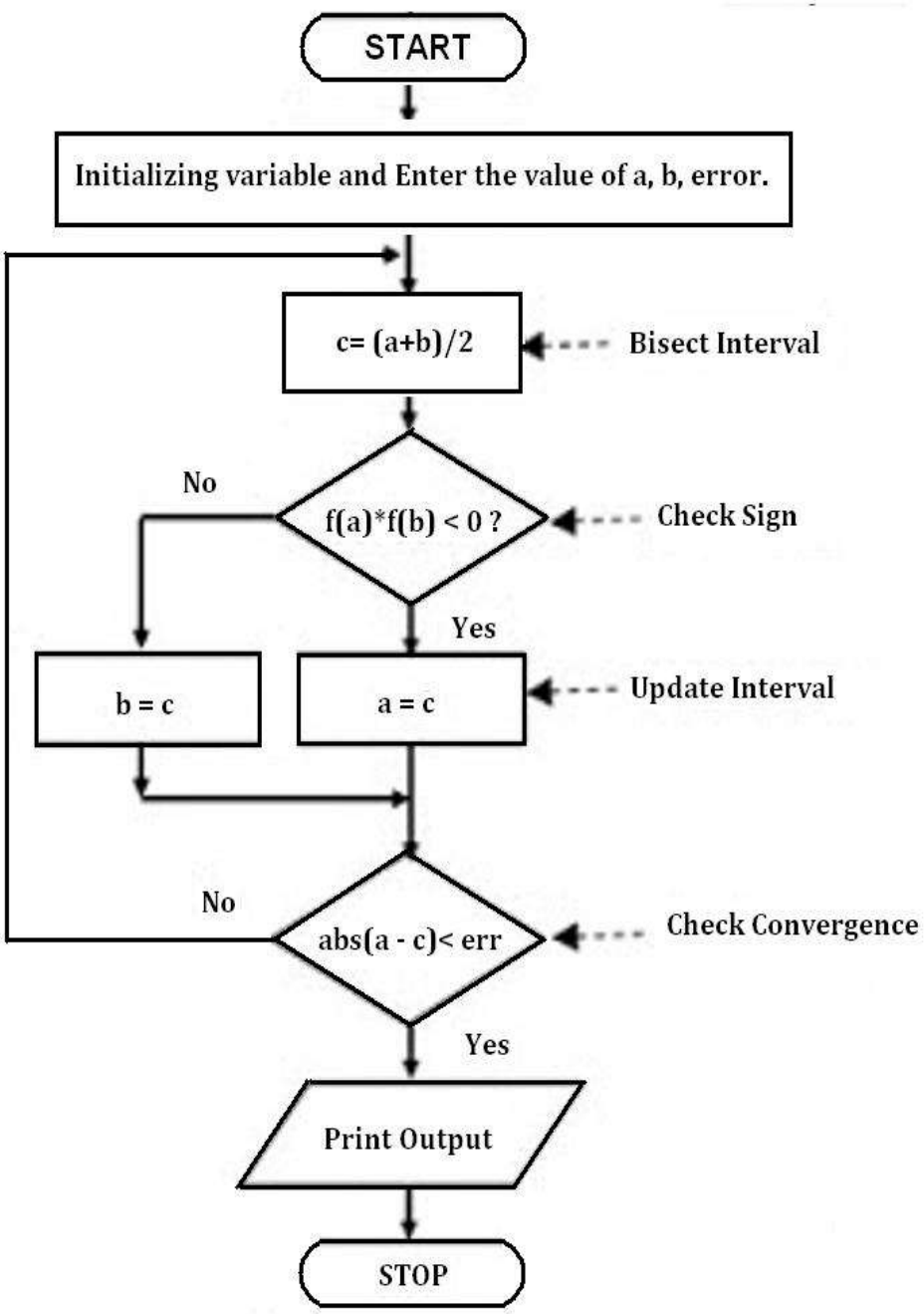
Algorithm for the Bisection Method:

Given a continuous function **f(x)**

1. Find points **a** and **b** such that **a < b** and **f(a) * f(b) < 0**.
2. Take the interval **[a, b]** and find its midpoint **c**.
3. If **f(c) = 0** then $\mathbf{x_1}$ is an exact root, else if **f(c) * f(b) < 0** then let **a = c,** else if **f(a) * f(c) < 0** then let **b = c**.
4. Repeat steps 2 & 3 until **f(c) = 0** or **|f(c)| <= degree of accuracy**.

Calculation:

Practical In-charge Sign



Flow Chart 1: Bisection method

Experiment No: 02 **Date: -**

Aim: Solution of a polynomial equation using Birge - Vieta method

Title: Use Birge-Vieta method to determine the root of equation of a given function

$$f(x) = X^4 - 11X^3 + 8X^2 - 5X + 20.$$

Theory:

Using this method, We can find a real root of a polynomial equation f(x) = 0. We use the fact that if r is a real root of the equation, then (x-r) is a factor of the polynomial f(x). Let

$$f(x) = a_0 x^n + a_1 x^{n-1} + \ldots\ldots + a_n$$

When f(x) is divided by the expression (x-r), the quotient q(x) will be a polynomial of degree (n-1),

$$q(x) = b_0 x^{n-1} + b_1 x^{n-2} + \ldots\ldots + b_{n-1}$$

and remainder R will depends on r, Since

$$f(x) = (x-r)\, q(x) + R$$

We will start with an initial approximation r_0 to r and use Newton's Raphson method to improve the value of r such that f(x) = 0, then

$$r_1 = r_0 - \frac{f'(x)}{f(x)}$$

In Birge- Vieta method, we will not compute *f (x)* and *f '(x)* but compute them by using synthetic division as explained below,

$$f(x) = (x - r)\, q(x) + R$$

Equating, $a_0 x^n + a_1 x^{n-1} + \ldots\ldots + a_n = (x - r_0)(b_0 x^{n-1} + b_1 x^{n-2} + \ldots + b_{n-1})$

$$a_0 x^n + a_1 x^{n-1} + \ldots\ldots + a_n = b_0 x^n + (b_1 - r_0 b_0) x^{n-1} + \ldots.$$

Equating like terms on both side of above equation, we have

$a_0 = b_0$	$b_0 = a_0$
$a_1 = b_1 - r_0 b_0$	$b_1 = a_1 - r_0 b_0$
$a_2 = b_2 - r_0 b_1$	$b_2 = a_2 - r_0 b_1$
... .	... .

$$a_n = b_n - r_0 b_{n-1} \qquad b_n = a_n - r_0 b_{n-1}$$

If we define $b_n = R = f(r_0)$, *the* relation in above equation can be represented by,

$$b_i = a_i - r_0 b_{i-1} \quad \text{where} \quad i = 0, 1, 2, \ldots, n$$

Differentiating both side with respect to r_0, we have

$$\frac{\partial b_i}{\partial r_0} = b_{i-1} + r_0 \frac{\partial b_{i-1}}{\partial r_0}$$

If $$\frac{\partial b_i}{\partial r_0} = c_{i-1} = b_{i-1} + r_0 \frac{\partial b_{i-1}}{\partial r_0}$$

$$c_{i-1} = b_{i-1} + r_0 c_{i-2}$$

or $$c_i = b_i + r_0 c_{i-1}, \quad i = 0, 1, 2, \ldots.(n-1)$$

Hence we can say, $$f'(r_0) = \frac{dR}{dr_0} = \frac{db_n}{dr_0} = c_{n-1}$$

Therefore $$r_1 = r_0 - \frac{b_n}{c_{n-1}}$$

r_0	a_0	a_1	a_2	------	a_{n-1}	a_n
		$r_0 b_0$	$r_0 b_1$	------	$r_0 b_{n-2}$	$r_0 b_{n-1}$
r_0	b_0	b_1	b_2	-------	b_{n-1}	b_n
		$r_0 c_0$	$r_0 c_1$	------	$r_0 c_{n-2}$	
	C_0	C_1	C_2	------	C_{n-1}	

In general, solution by using Birge Vieta method is,

$$r_{n+1} = r_n - \frac{b_n}{c_{n-1}}$$

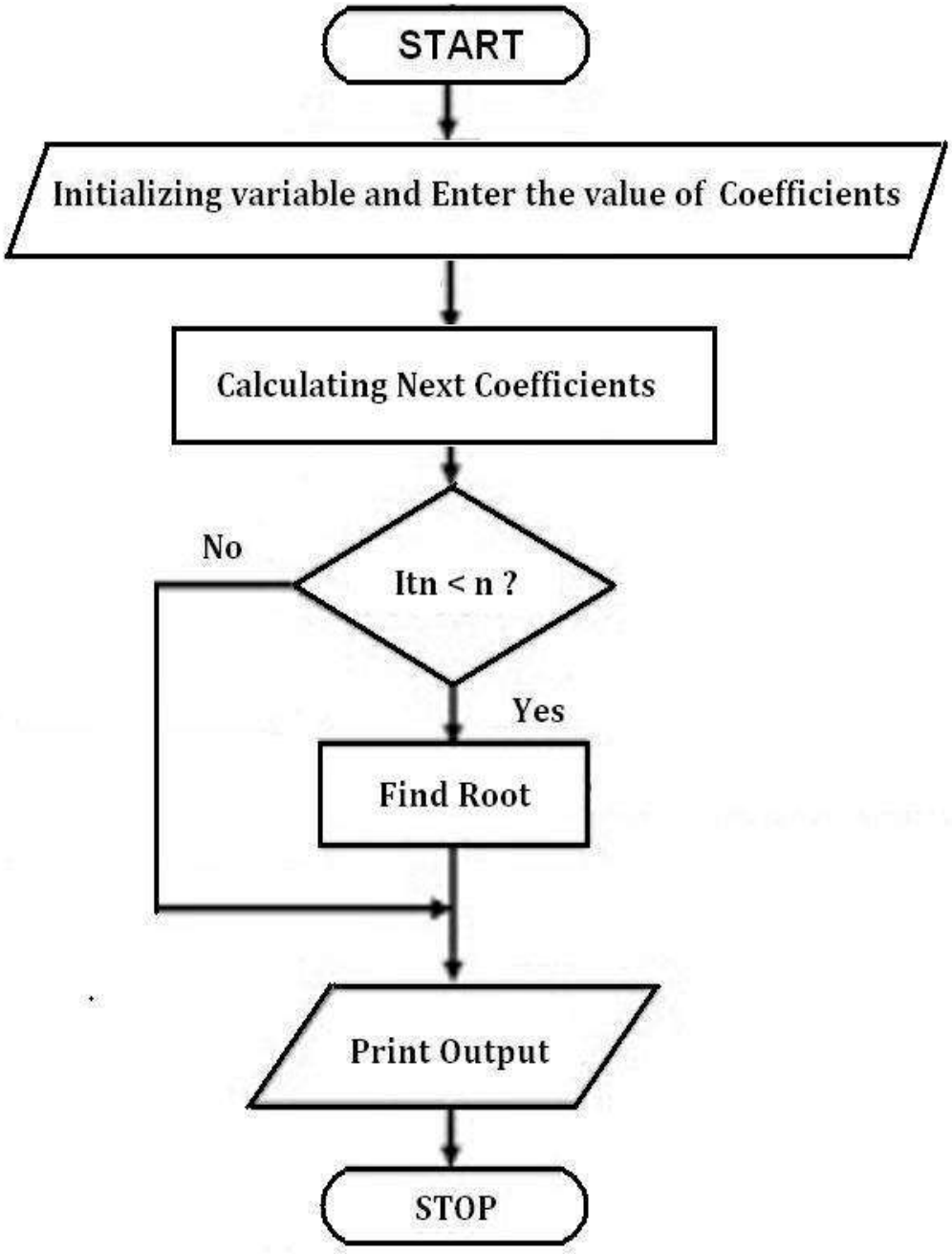
Algorithm for the Birge-Vieta Method:

Given a continuous polynomial equation **f(x)**

1. Arrange the coefficient of given polynomial equation.
2. Perform the synthetic division with initial value $\mathbf{r_0}$.
3. Find next approximate root by $r_{n+1} = r_n - \frac{b_n}{c_{n-1}}$
4. Repeat step 2 & step 3 r having minimum error.

Calculation:

Practical In-charge Sign



Flow Chart 2: Birge-Vieta method

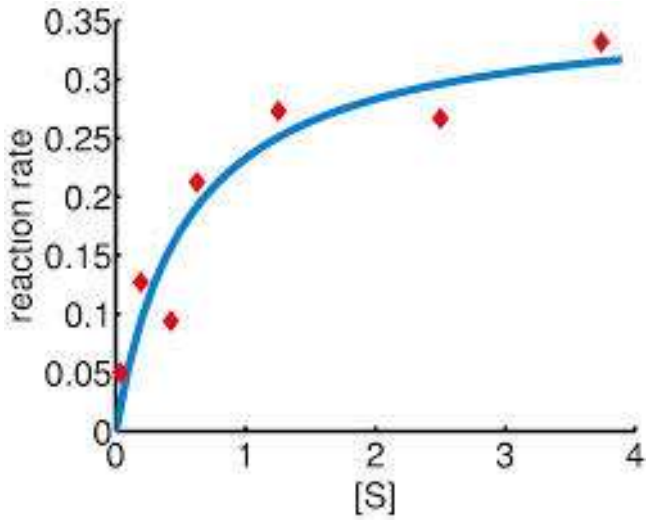
Experiment No: 03 **Date: -**

Title: Second order curve fitting using least square approximation coefficient

Title: Use Second order curve fitting using least square approximation coefficient of given distributed data.

X	0	10	20	30	40	50
Y	53.5	59.5	65.2	70.6	75.5	80.2

Theory:



The least-squares parabola uses a second degree curve tomorrow

$$y = a + bx + cx^2 .$$

to approximate the given set of data, (x_1, y_1) , (x_2, y_2) , ..., (x_n, y_n) , where $n \geq 3$. The best fitting curve $f(x)$ has the least square error, i.e.,

$$\Pi = \sum_{i=1}^{n} [y_i - f(x_i)]^2 = \sum_{i=1}^{n} \left[y_i - (a + bx_i + cx_i^2) \right]^2 = \min.$$

Please note that a , b , and c are unknown coefficients while all x_i and y_i are given. To obtain the least square error, the unknown coefficients a , b , and c must yield zero first derivatives.

$$\begin{cases} \dfrac{\partial \Pi}{\partial a} = 2\sum_{i=1}^{n}\left[y_i - (a + bx_i + cx_i{}^2)\right] = 0 \\ \dfrac{\partial \Pi}{\partial b} = 2\sum_{i=1}^{n} x_i\left[y_i - (a + bx_i + cx_i{}^2)\right] = 0 \\ \dfrac{\partial \Pi}{\partial c} = 2\sum_{i=1}^{n} x_i{}^2\left[y_i - (a + bx_i + cx_i{}^2)\right] = 0 \end{cases}$$

Expanding the above equations, we have

$$\begin{cases} \sum_{i=1}^{n} y_i = a\sum_{i=1}^{n} 1 + b\sum_{i=1}^{n} x_i + c\sum_{i=1}^{n} x_i{}^2 \\ \sum_{i=1}^{n} x_i y_i = a\sum_{i=1}^{n} x_i + b\sum_{i=1}^{n} x_i{}^2 + c\sum_{i=1}^{n} x_i{}^3 \\ \sum_{i=1}^{n} x_i{}^2 y_i = a\sum_{i=1}^{n} x_i{}^2 + b\sum_{i=1}^{n} x_i{}^3 + c\sum_{i=1}^{n} x_i{}^4 \end{cases}$$

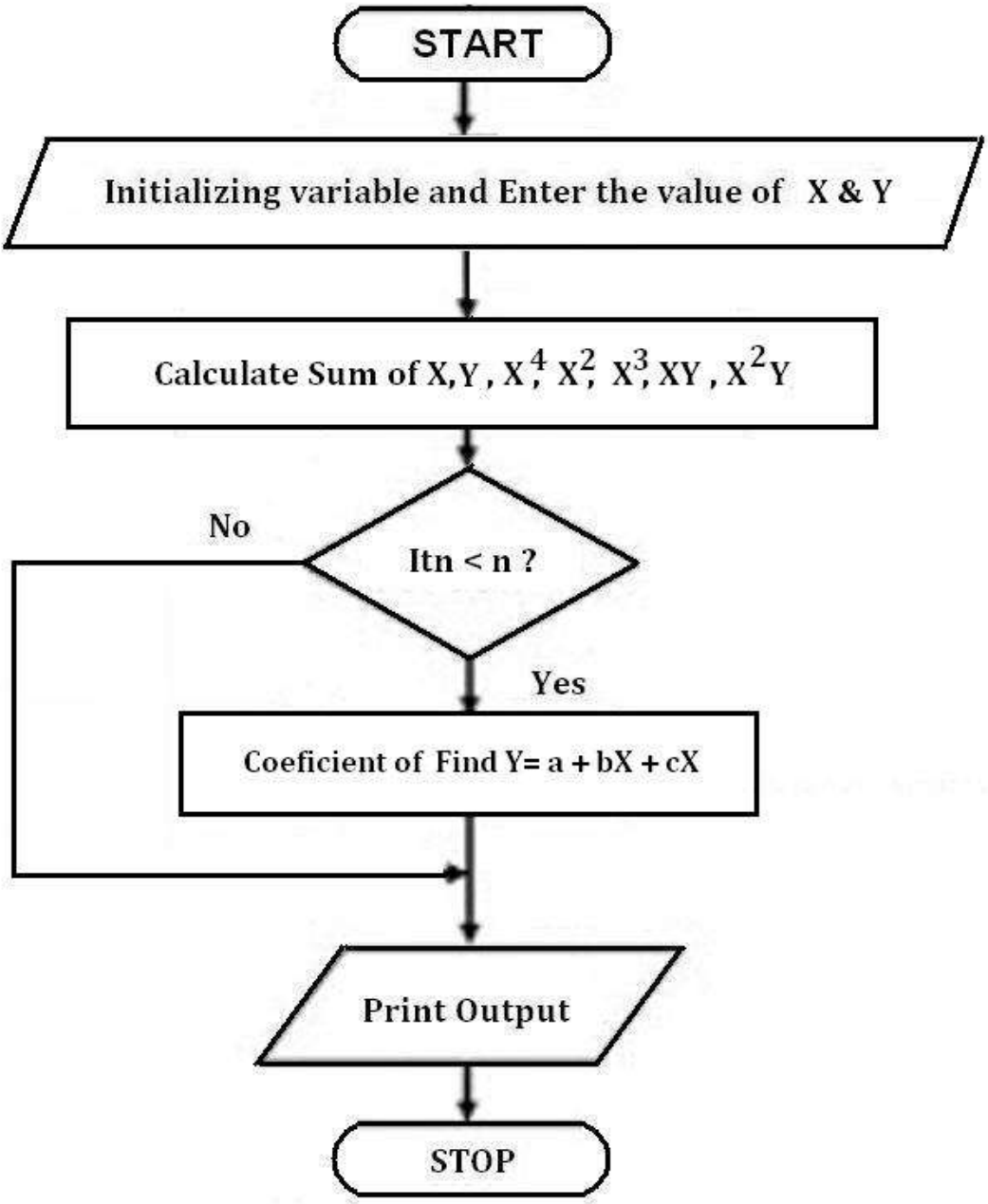
The unknown coefficients a, b, and c can hence be obtained by solving the above linear equations.

Algorithm for the Second order curve fitting :
From the given a distributed data.

1. Enter the values of x corresponds to y.
2. Calculate the values of $\Sigma x, \Sigma x^2, \Sigma x^3, \Sigma x^4, \Sigma xy, \Sigma x^2 y$ from the given distributed data.
3. Calculate the value of coefficient a, b, c by least square equation
4. Print the result as $y = a + bx + cx^2$.

Calculation:

Practical In-charge Sign



Flow Chart 3: curve fitting

Experiment No: 04 **Date: -**

Aim: Program for interpolation using Newton's forward or backward interpolation

Title: Use Newton's Forward Interpolation formula calculate y at x=6.25 form given distributed data.

X	**4**	**6**	**8**	**10**
Y	1	3	8	16

Theory:

Let $y = f(x)$, which takes the values $y_0, y_1, y_2, \ldots, y_n$ that is the set of $(n + 1)$ functional values $y_0, y_1, y_2, \ldots, y_n$ are given corresponding to the set of $(n + 1)$ equally spaced values of the independent variable,

$x_i = x_0 + ih$, $i = 0, 1, 2, \ldots, n$ where h is the spacing. Let $p(x)$ be a polynomial of the *nth* degree in x taking the same values as y corresponding to $x = x_0, x_1, \ldots, x_n$. Then, $p(x)$ represents the continuous function $y = f(x)$ such that $f(xi) = p(xi)$ for $i = 0, 1, 2, \ldots, n$ and at all other points $f(x) = p(x) + R(x)$ where $R(x)$ is called

the *error term* (remainder term) of the interpolation formula. Let

$$f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2)$$

$$+ a_n (x - x_0)(x - x_1)(x - x_2)\cdots(x - x_{n-1}) + \ldots\ldots\ldots\ldots$$

The constants $a_0, a_1, a_2, \ldots, a_n$ can be find as follows:

To find a_0 , Substituting $x = x_0$, we get, $a_0 = y_0$

To find a_1 , Substituting $x = x_1$, we get,

$$y_1 = a_0 + a_1(x_1 - x_0) \quad \text{or} \quad y_1 = y_0 + a_1(x_1 - x_0)$$

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y_0}{h}$$

To find a_2 , Substituting $x = x_2$, we get

$$y_2 = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) = y_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1)$$

$$(y_2 - y_0) - \frac{y_1 - y_0}{x_1 - x_0}(x_2 - x_0) = a_2(x_2 - x_0)(x_2 - x_1)$$

$$(y_2 - y_0) - \frac{y_1 - y_0}{h} 2h = a_2 2h\, h$$

$(y_2 - y_0) - 2(y_1 - y_0) = a_2\, 2!\, h^2$

$(y_2 - y_0 - 2y_1 + 2y_0) = a_2\, 2!\, h^2$

$(y_2 - 2y_1 + y_0) = a_2\, 2!\, h^2$

$$a_2 = \frac{\Delta^2 y_0}{2! h^2}$$

Similarly,

$$a_3 = \frac{\Delta^3 y_0}{2! h^3}$$

In general,

$$a_n = \frac{\Delta^n y_0}{2! h^n}$$

$$f(x) = a_0 + \frac{\Delta y_0}{1! h}(x - x_0) + \frac{\Delta^2 y_0}{2! h^2}(x - x_0)(x - x_1) + \frac{\Delta^3 y_0}{2! h^3}(x - x_0)(x - x_1)(x - x_2) + \ldots\ldots$$

$$+ \frac{\Delta^n y_0}{2! h^n}(x - x_0)(x - x_1)(x - x_2)\cdots(x - x_{n-1}) \qquad \text{---------- (2)}$$

Let $x = x_0 + uh$ or $(x - x_0) = uh$

And $(x - x_1) = (x - x_0) - (x_1 - x_0)$

$= uh - h = (u - 1)\, h$

$(x - x_2) = (x - x_1) - (x_2 - x_1)$

$= (u - 1)\, h - h = (u - 2)h$, etc.

Equation (2) becomes,

$$f(x) = a_0 + u\,\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \frac{u(u-1)(u-2)}{3!}\Delta^3 y_0 + \ldots + \frac{u(u-1)(u-2)\ldots(u-(n-1))}{2!}\Delta^n y_0$$

Above equation is Newton's Forward Interpolation formula.

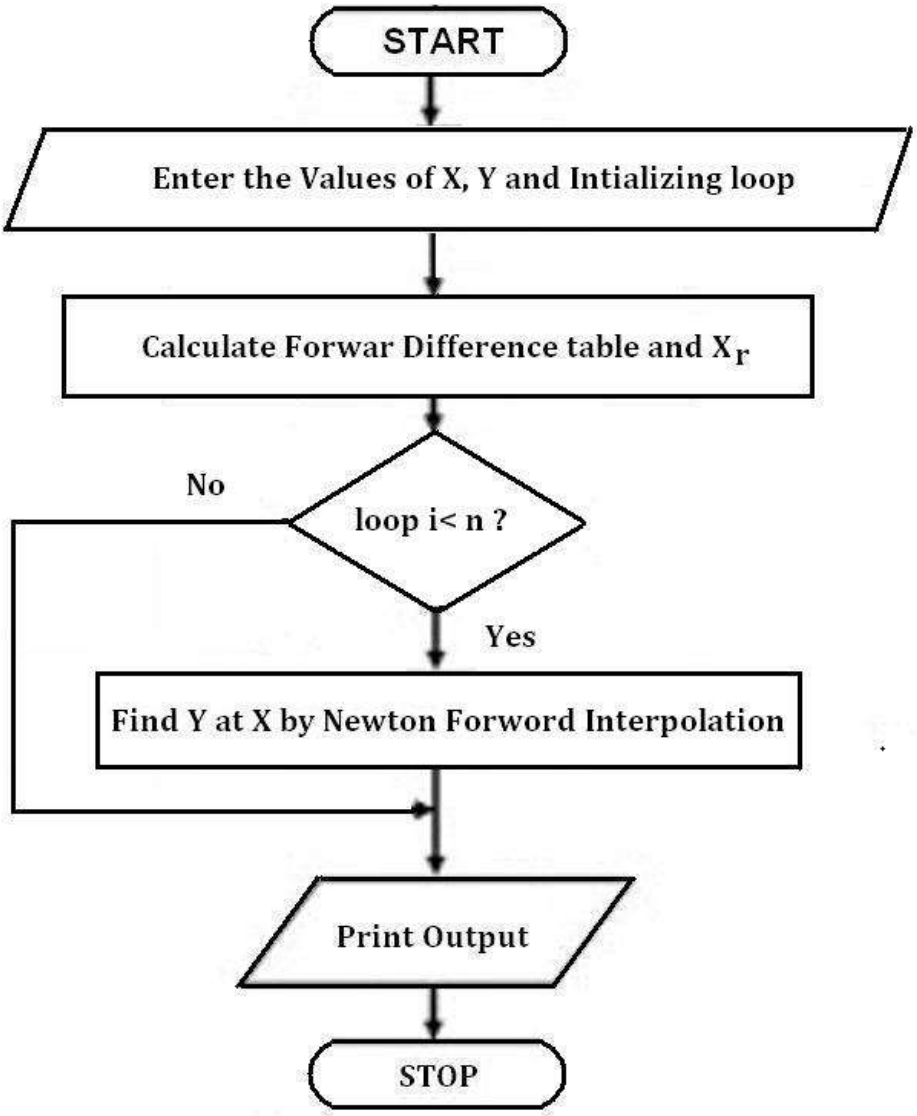
Algorithm for the Newton's Forward Interpolation Method:

From the given a distributed data.

1. Enter the values of x corresponds to y.
2. Calculate the values of Newton's forward difference table from the given distributed data.
3. Calculate the value of y at a given value of x.
4. Print the result.

Calculation:

Practical In-charge Sign



Flow Chart 4: Newton's forward or backward interpolation Method

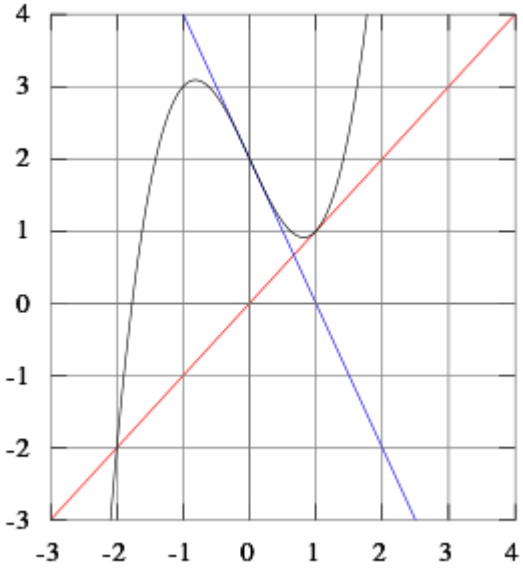
Experiment No: 05 **Date: -**

Aim: Newton's Raphson's method for two variables

Title: Find the root of equation using N R Method for two variables. $f(x, y) = X^2 + XY - 10$, $g(x, y) = Y + 3XY^2 - 57$. Solve upto two iteration only take initial value $X_0 = 1.5$ and $Y_0 = 3.5$.

Theory:

Let them be $f(x,y) = 0$ and $g(x,y) = 0$. Now (x_0, y_0) be an initial approximate solution of the equation.



Let (x_0+h, y_0+k) be the actual solution. Using Taylor's series ,

$$f(x_{i+1}) = f(x_i) + h f'(x_i) + \frac{1}{2!} h^2 f''(x_i) + \ldots + \frac{1}{n!} h^n f^n(x_i)$$

For a two function of two variables and omitting h^2 and higher order

$$f(x_{i+1}, y_{i+1}) = f(x_i, y_i) + h f'(x_i) + k f'(y_i)$$

$$g(x_{i+1}, y_{i+1}) = g(x_i, y_i) + h g'(x_i) + k g'(y_i)$$

Where,

$$f x = f'(x_i) = \frac{\partial f(x, y)}{\partial x} \Big|_{y = \text{const.}} \qquad f y = f'(y_i) = \frac{\partial f(x, y)}{\partial y} \Big|_{x = \text{const.}}$$

$$g x = g'(x_i) = \frac{\partial g(x, y)}{\partial x} \Big|_{y = \text{const.}} \qquad g x = g'(y_i) = \frac{\partial g(x, y)}{\partial y} \Big|_{x = \text{const.}}$$

$f(x_{i+1}, y_{i+1}) = f\,x\,y + h\,f\,x + k\,f\,y$ .....(1)

$g(x_{i+1}, y_{i+1}) = g\,x\,y + h\,g\,x + k\,g\,y$ .....(2)

Then for finding root of equation,

$f(x_{i+1}, y_{i+1}) = 0$ and $g(x_{i+1}, y_{i+1}) = 0$

Hence eq. (1) and (2) become,

$f\,x\,y + h\,f\,x + k\,f\,y = 0$ or $h\,f\,x + k\,f\,y = -\,f\,x\,y$

$g\,x\,y + h\,g\,x + k\,g\,y = 0$ *or* $h\,g\,x + k\,g\,y = -\,g\,x\,y$

$$D = \begin{vmatrix} fx & fy \\ gx & gy \end{vmatrix} \qquad Dx = \begin{vmatrix} fxy & fy \\ gxy & gy \end{vmatrix} \qquad Dy = \begin{vmatrix} fx & fxy \\ gx & gxy \end{vmatrix}$$

Hence, Formula for Newton's Raphson's method for two variables is,

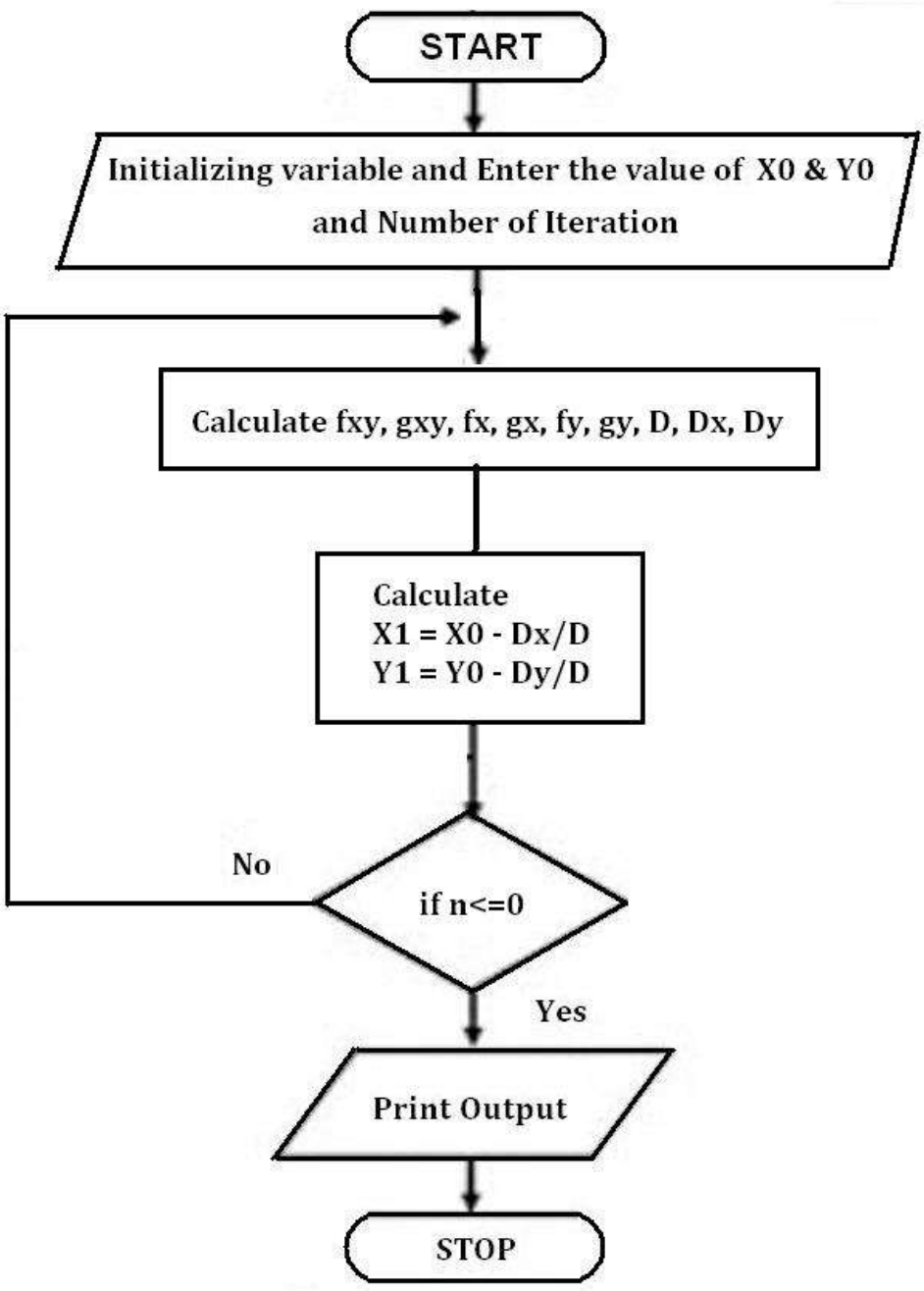
$$x_{n+1} = x_n - \frac{Dx}{D} \qquad \text{And} \qquad y_{n+1} = y_n - \frac{Dy}{D}$$

Algorithm for the Newton's Raphson method for two variables:

1. Enter the initial values of x and y.
2. Calculate the values of $fxy, gxy, fx, gx, fy, gy, D, Dx, Dy.$
3. Calculate the value of x_{n+1} and y_{n+1}.
4. Repeat step 2 and step 3 up to required iteration.
5. Print the result.

Calculation:

Practical In-charge Sign



Flow Chart 5: Newton's Raphson method for two variables

Experiment No: 06 **Date: -**

Aim: Simpson's 1/3 Rule for Integration:

Title: Evaluate $\int_0^1 \frac{dx}{1+x^2}$ by Simpson's 1/3rd rule dividing the intervals into 6 equal parts.

Theory:

Trapezoidal rule approximate the graph between two adjacent points on the graph by the straight line joining them. This against Simpson's Rule makes use of the parabola joining the three adjacent points to approximate the graph. A parabola can be drawn through any three non collinear points. The Simpson formula is also arrived at by dividing the given interval [a,b] into 'n' sub intervals of equal length $\Delta x = b-an$. But, **n** in this case is an even integer. Then, on each consecutive pair of intervals, we approximate the curve y = f(x), by parabolas as shown in the diagram here. A general parabola passes through the consecutive points P_i, P_{i+1} and P_{i+2}.The definite integral is then approximated to the sum of the areas under all such parabolas

A quick approximation for definite integrals can be got by taking 2 partitions for a small interval $[x_0,x_1]$.

Trapezoidal formula:

Since **f(x)** is approximated with a first degree polynomial in each sub interval **(x_{i+1}, x_i)**, the integration of **f(x)** between**x_{i+1}**and **x_i** is knothing but the area of the Trapezoid bounded by **x** axis, **$f(x_{i-1})$**, **$f(x_i)$** and the straight line joining the points **$(x_{i-1}, f(x_{i-1}))$** and **$(x_i, f(x_i))$**.

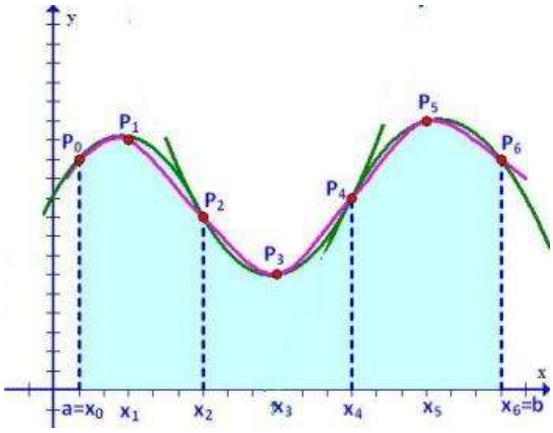


Fig:

Consider the integral

$$\int_{x_0}^{x_n} f(x) = \int_{x_0}^{x_1} p_1(x)dx = \int_{x_0}^{x_1} (y_0 + r\Delta y_0)dx$$

By Newton's forward difference formula,

$$r = \frac{x - x_0}{x_1 - x_0} = \frac{x - x_0}{h}$$

Since $\Delta y_0 = y_1 - y_0$ first forward difference of Δy_0

Consider $$\int_{x_0}^{x_n} f(x) = \int_{x_0}^{x_1} p_1(x)dx$$

$$I_1 = h\int_0^1 p_1(x)dr$$

$$I_1 = h(y_o + \frac{1}{2}\Delta y_0)$$

$$I_1 = \frac{h}{2}(y_o + y_1)$$

Then

$$\int_{x_0}^{x_n} f(x)dx = I = I_1 + I_2 + \ldots\ldots\ldots + I_n$$

$$I = \frac{h}{2}(y_o + y_1) + \frac{h}{2}(y_1 + y_2) + \ldots\ldots\ldots + \frac{h}{2}(y_{n-1} + y_n)$$

$$I = \frac{h}{2}(y_o + y_1) + \frac{h}{2}(y_1 + y_2) + \ldots\ldots\ldots + \frac{h}{2}(y_{n-1} + y_n)$$

$$I = \frac{h}{2}\left[(y_o + y_n) + 2(y_1 + y_2 + y_{n-1})\right]$$

This is called the Trapezoidal formula.

Simpsons 1/3 Rule:

Consider the integral

$$\int_{x_0}^{x_n} f(x) = \int_{x_0}^{x_2} p_2(x)dx$$

$$I_1 = \int_0^2 p_2(x)dx$$

$$I_1 = \int_0^2 \left[y_0 + r\Delta y_0 + \frac{r(r-1)}{2!}\Delta^2 y_0 \right] dx$$

$$I_1 = h\int_0^2 \left[y_0 + r\Delta y_0 + \frac{r(r-1)}{2!}\Delta^2 y_0 \right] dr$$

$$I_1 = 2h\left[y_0 + 4\Delta y_0 + \frac{1}{6}\Delta^2 y_0 \right] = \frac{h}{3}\left[y_0 + 4y_1 + y_2\right]$$

Since $\Delta y_0 = y_1 - y_0$ and $\Delta^2 y_0 = y_1 - 2y_1 - y_0$ second forward difference

Therefore, $$\int_{x_0}^{x_n} f(x) = \int_{x_0}^{x_2} p_2(x)dx$$

$$I_1 = \frac{h}{3}\left[y_0 + 4y_1 + y_2\right]$$

Now, $$\int_{x_0}^{x_2} f(x)dx = I = I_1 + I_2 + \ldots\ldots\ldots + I_n$$

$$I = \frac{h}{3}\left[y_0 + 4y_1 + y_2\right] + \frac{h}{3}\left[y_0 + 4y_1 + y_2\right] + \ldots\ldots\ldots + \frac{h}{3}\left[y_{n-2} + 4y_{n-1} + y_n\right]$$

$$I = \frac{h}{3}\left[(y_0 + y_n) + 4(y_1 + y_2 + \ldots + y_{n-1}) + 2(y_2 + y_4 + \ldots\ldots y_{n-2})\right]$$

This is called the Simpsons 1/3 Rule

Simpsons 3/8 Rule:

Consider the integral

$$\int_{x_0}^{x_n} f(x) = \int_{x_0}^{x_3} p_3(x)dx$$

$$I_1 = \int_0^3 p_2(x)dx$$

$$I_1 = \int_0^3 \left[y_0 + r\Delta y_0 + \frac{r(r-1)}{2!}\Delta^2 y_0 + \frac{r(r-1)(r-2)}{3!}\Delta^3 y_0 \right] dx$$

$$I_1 = h\int_0^3 \left[y_0 + r\Delta y_0 + \frac{r(r-1)}{2!}\Delta^2 y_0 + \frac{r(r-1)(r-2)}{3!}\Delta^3 y_0 \right] dr$$

$$I_1 = 3h\left[y_0 + \frac{3}{2}\Delta y_0 + \frac{3}{2}\Delta^2 y_0 + \frac{1}{8}\Delta^3 y_0 \right]$$

Since $\Delta y_0 = y_1 - y_0$ and $\Delta^2 y_0 = y_1 - 2y_1 - y_0$ second forward difference

Therefore, $$\int_{x_0}^{x_n} f(x) = \int_{x_0}^{x_3} p_3(x)dx$$

$$I_1 = \frac{3h}{8}\left[y_0 + 3y_1 + 3y_2 + y_3\right]$$

Now,

$$\int_{x_0}^{x_3} f(x)dx = I = I_1 + I_2 + \ldots\ldots\ldots + I_n$$

$$I_1 = \frac{3h}{8}\left[(y_0 + y_n) + 3(y_1 + y_2 + y_4 + y_5 + y\ldots. + y_{n-2} + y_{n-1}) + 2(y_3 + y_6 + \ldots + y_{n-3})\right]$$

This is called the Simpsons 3/8 Rule. The no. of data points needed for this rule are **3n+1** for any **n > 0**.

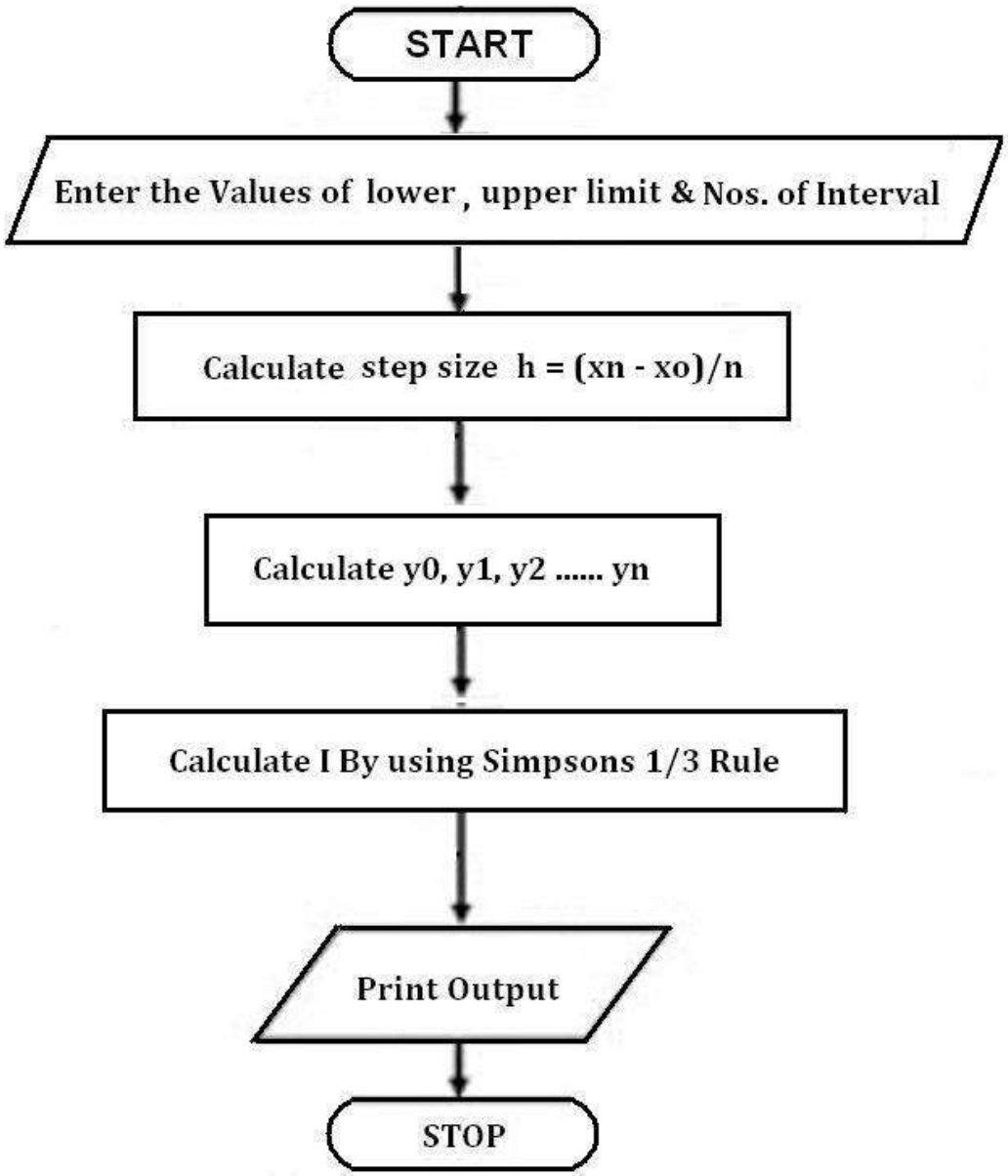
Observe here that the order of the error in both Simpson's one third rule and three eighth rule is four (O(**h**^{4})). Moreover the coefficient in one third rule **(-1/90)** is less then the corresponding three eighth rule **(-3/80)** hence **Simpson's 1/3** rule performs better than the **Simpson's 3/8** rule. In fact this phenomina is true for all even-order Newton-cotes formulae. Hence Newton-cotes even order formulae are more useful than odd order formulae.

Algorithm for the Simpson's 1/3 Rule for Integration:

1. Enter the initial values upper limit and lower limit.
2. Calculate the values of x corresponds to y according to value step size(h).
3. Calculate the sum of odd term value of y_{odd} and sum of even term value of y_{even}.
4. Print the result.

Calculation:

Practical In-charge Sign



Flow Chart 6: Simpson's 1/3 Rule for Integration method

Experiment No: 07 **Date: -**

Aim: Solution of second order ODE using 4th order RK method.

Title: solve the second order ODE $\frac{d^2y}{dx^2} = x.\frac{dy}{dx} - y$ by using 4th order RK method. When y(0)=1, $\frac{dy}{dx} = z = 0$ having $h = 0.2$. Calculate y (0.4).

Theory:

The formula for the Euler method is,

$$y_{n+1} = y_n + h * y(x_n, y_n)$$

Which advances a solution from x_n to $x_{n+1} = x_n + h$. The formula is unsymmetrical:

It advances the solution through an interval *h*, but uses derivative information only at the beginning of that interval. That means (and you can verify by expansion in power series) that the step's error is only one power of *h* smaller than the correction. There are several reasons that Euler's method is not recommended for practical use, among them, the method is not very accurate when compared to other methods run at the equivalent step size, and neither is it very stable. Consider, however, the Euler method use of a step like to take a "trial" step to the midpoint of the interval. Then use the value of both *x* and *y* at that midpoint to compute the "real" step across the whole interval.

Second order Runge-Kutta Method:

The first four terms Taylor series:

$$y_{n+1} = y_n + \frac{h}{1!}y(x_n, y_n) + \frac{h^2}{2!}y'(x_n, y_n) + \frac{h^3}{3!}y''(x_n, y_n) + \frac{h^4}{4!}y'''(x_n, y_n)$$

The fourth order Runge-Kutta method is one of the standard method to solve differential equations. Before we give the algorithm of the fourth order Runge-Kutta method we will derive the second order Runge Kutta method.

We start with the original differential equation and integrate it formally.

$$\frac{dy}{dx} = f(x_n, y_n) = y(x) = \int_0^n f(x_n, y_n).dx$$

Omitting the term from h^2 of Taylor's series, we get,

$$y_{n+1} = y_n + y(x_n, y_n)h$$

$$y_{n+1} = y_{n+1} + h\int_{n}^{n+1} f(x,y).dx$$

We essentially changed the task at hand from performing a differentiation to an integration. To do this we expand f(t) in a second order Taylor series around the midpoint of the integration subinterval.

$$f(x,y) = f(x_{n+1/2}, y_{n+1/2}) + (x - x_{n+1/2})\frac{dy}{dx} \qquad \text{..... (1)}$$

Yet since the integral of $(X - X_{n-1/2})$ vanishes when evaluated about the midpoint, we automatically get improved precision using only the first term in (1).

$$f(x,y) = f(x_{n+1/2}, y_{n+1/2})$$

Therefore, $$y_{n+1} = y_n + h * f(x_{n+1/2}, y_{n+1/2})$$

$$= y_n + h * f(x_n + \frac{h}{2}, y_{n+1/2})$$

This algorithm cannot be applied immediately since it requires a knowledge of $y_{n+1/2}$ which is not in the scheme of things. We thus approximate $y_{n+1/2}$ with Euler's algorithm.

$$y_{n+1/2} = y_n + \frac{dy}{dx}\Delta x = y_n + \frac{dy}{dx}\left(\frac{h}{2}\right)$$

$$= y_n + \frac{h}{2} * f(x_n, y_n)$$

Hence the second order Runge-Kutta

$$y_{n+1} = y_n + \frac{h}{2} * (k_1 + k_2) \qquad \text{---------- (2)}$$

Where, $k_1 = f(x_n, y_n)$

$$k_2 = f(x_{n+1}, y_n + k_1)$$

The second order Runge-Kutta equation (2) requires the known derivative function y(x) at the endpoints and midpoint of the interval, and the unknown function y at the previous point. Since we start with initial conditions, the algorithm is self starting. Note too that it is applicable with a general function y(x) (for example nonlinear), and simple to program.

Fourth order Runge-Kutta Method:

A method of numerically integrating ordinary differential equations by using a trial step at the midpoint of an interval to cancel out lower-order error terms. The fourth-order formula is

$$y_{n+1} = y_n + \frac{h}{6} * (k_1 + 2k_2 + 2k_3 + k_4) \qquad \text{---------- (3)}$$

Where, $k_1 = f(x_n, y_n)$ Euler (start-point) slope

$k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$ Midpoint slope

$k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$ Better midpoint slope

$k_4 = f(x_n + h, y_n + hk_3)$ Full step (endpoint) slope

Solution of Second Order ODE by fourth order Runge-Kutta Method:

Consider the Second Order ODE,

$$\frac{d^2y}{dx^2} = f(x, y, \frac{dy}{dx}) \quad \text{Or} \quad \frac{d^2y}{dx^2} = f(x, y, z) \quad \text{Where, } z = \frac{dy}{dx}$$

We begin by writing as two first order ODE is,

$$\frac{dz}{dx} = g(x, y, z) \qquad \text{---------- (i)}$$

$$\frac{dy}{dx} = f(x, y, z) \qquad \text{---------- (ii)}$$

Next, we apply Runge-Kutta formula to each of the above two equation as follows,

Where, $k_1 = f(x_n, y_n, z_n)$ Euler (start-point) slope

$m_1 = f(x_n, y_n, z_n)$

$$k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1, z_n + \frac{h}{2}m_1)$$ Midpoint slope

$$m_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1, z_n + \frac{h}{2}m_1)$$

$$k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2, z_n + \frac{h}{2}m_2)$$ Better midpoint slope

$$m_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2, z_n + \frac{h}{2}m_2)$$

$$k_4 = f(x_n + h, y_n + hk_3, z_n + hm_3)$$ Full step (endpoint) slope

$$m_4 = f(x_n + h, y_n + hk_3, z_n + hm_3)$$

Above value of k_1, k_2, k_3, k_4 is calculated by $f(x, y, z)$ and m_1, m_2, m_3, m_4 is calculated by $g(x, y, z)$.

$$y_{n+1} = y_n + \frac{h}{6} * (k_1 + 2k_2 + 2k_3 + k_4)$$ ---------- (3)

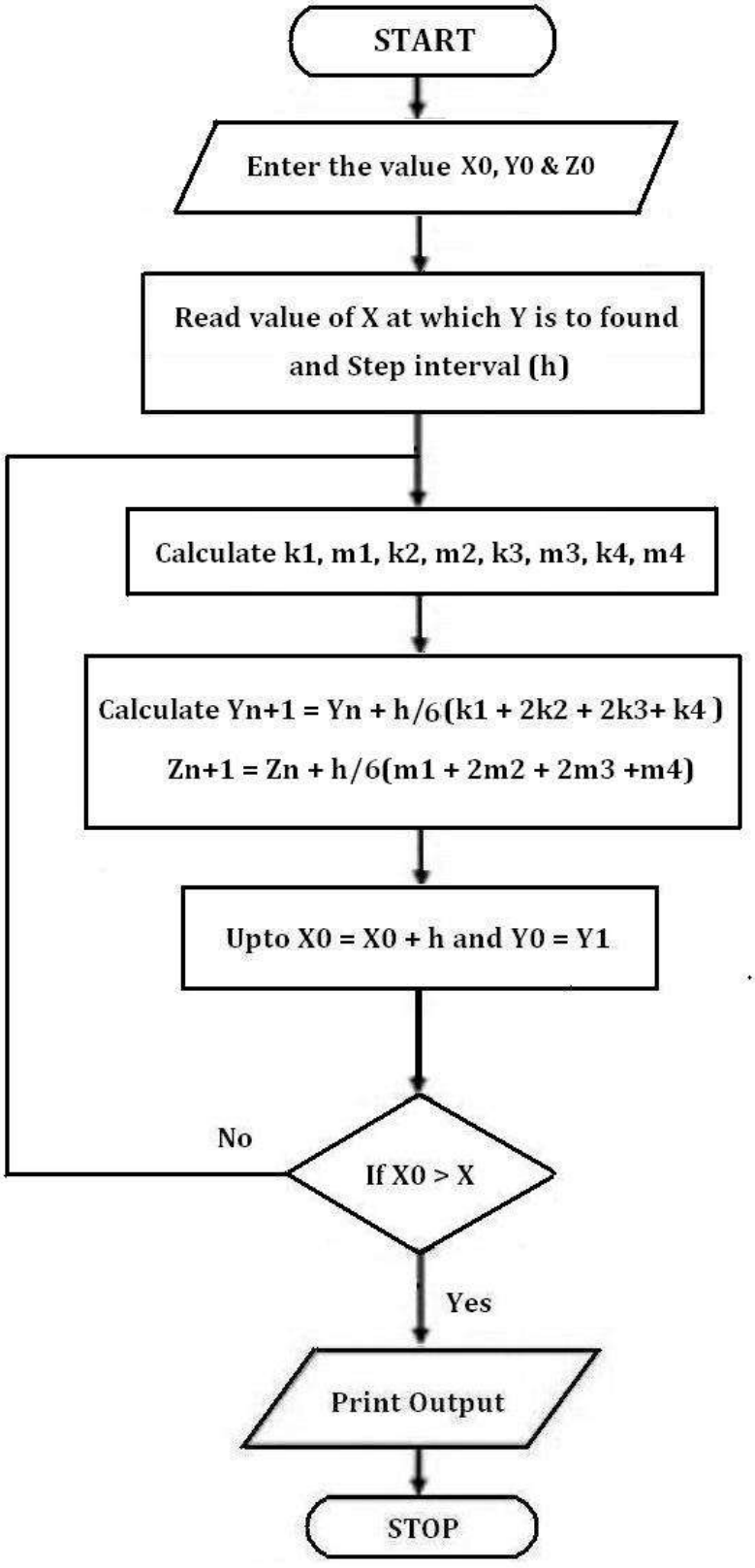
$$z_{n+1} = z_n + \frac{h}{6} * (m_1 + 2m_2 + 2m_3 + m_4)$$ ---------- (3)

Algorithm for the second order ODE using 4th order RK method:

1. Enter the initial values x and y.
2. Calculate the values k_1, k_2, k_3, k_4 and m_1, m_2, m_3, m_4 according to value step size (h).
3. Define the last point of the x to find value of y.
4. Repeat step number 2 up to the last point of x.
5. Print the result.

Calculation:

Practical In-charge Sign



Flow Chart 7: Solution of second order ODE using 4th order RK method

Experiment No: 08 **Date: -**

Aim: Solution of simultaneous equation using Gauss Seidel or Jacobi method.

Title: Find the solution to the following system of equations using the Gauss-Seidel method.

$$20x_1 + x_2 - 2x_3 = 17$$
$$3x_1 + 20x_2 - x_3 = -18$$
$$2x_1 + 3x_2 + 20x_3 = 25$$

Theory:

In certain cases, such as when a system of equations is large, iterative methods of solving equations are more advantageous. Elimination methods, such as Gaussian elimination, are prone to large round-off errors for a large set of equations. Iterative methods, such as the Gauss-Seidel method, give the user control of the round-off error. Also, if the physics of the problem are well known, initial guesses needed in iterative methods can be made more judiciously leading to faster convergence.

What is the algorithm for the Gauss-Seidel method? Given a general set of n equations and n unknowns, we have,

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = c_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = c_2$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = c_n$$

If the diagonal elements are non-zero, each equation is rewritten for the corresponding unknown, that is, the first equation is rewritten with x_1 on the left hand side, the second equation is rewritten with x_2 on the left hand side and so on as follows

$$x_2 = \frac{c_2 - a_{21}x_1 - a_{23}x_3 \ldots\ldots - a_{2n}x_n}{a_{22}}$$
$$\vdots$$
$$\vdots$$
$$x_{n-1} = \frac{c_{n-1} - a_{n-1,1}x_1 - a_{n-1,2}x_2 \ldots\ldots - a_{n-1,n-2}x_{n-2} - a_{n-1,n}x_n}{a_{n-1,n-1}}$$
$$x_n = \frac{c_n - a_{n1}x_1 - a_{n2}x_2 - \ldots\ldots - a_{n,n-1}x_{n-1}}{a_{nn}}$$

These equations can be rewritten in a summation form as

$$x_1 = \frac{c_1 - \sum_{\substack{j=1 \\ j\neq 1}}^{n} a_{1j} x_j}{a_{11}}$$

$$x_2 = \frac{c_2 - \sum_{\substack{j=1 \\ j\neq 2}}^{n} a_{2j} x_j}{a_{22}}$$

.
.
.

$$x_n = \frac{c_n - \sum_{\substack{j=1 \\ j\neq n}}^{n} a_{nj} x_j}{a_{nn}}$$

Hence for any row i,

$$x_i = \frac{c_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} x_j}{a_{ii}}, i = 1,2,\ldots,n.$$

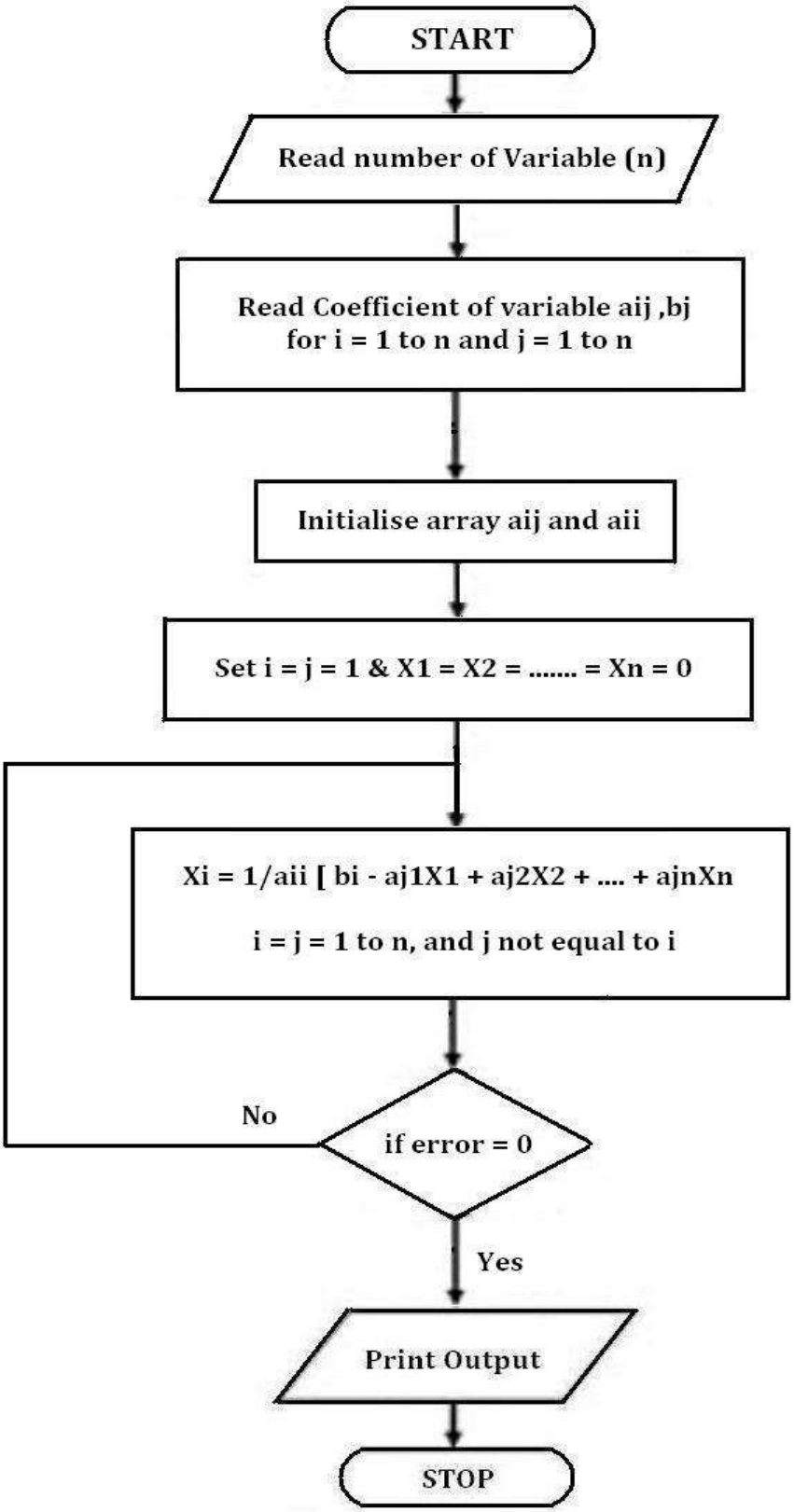
Now to find x_i's, one assumes an initial guess for the x_i's and then uses the rewritten equations to calculate the new estimates. Remember, one always uses the most recent estimates to calculate the next estimates, x_i. At the end of each iteration, one calculates the absolute relative approximate error for each x_i as

$$\left|\in_a\right|_i = \left|\frac{x_i^{\text{new}} - x_i^{\text{old}}}{x_i^{\text{new}}}\right| \times 100$$

where x_i^{new} is the recently obtained value of x_i, and x_i^{old} is the previous value of x_i. When the absolute relative approximate error for each x_i is less than the pre-specified tolerance, the iterations are stopped.

Calculation:

Practical In-charge Sign



Flow Chart 8: Solution of simultaneous equation using Jacobi method.

Experiment No: 09 **Date: -**

Aim: To find Eigen values and vector using Jacobi method.

Title: Determine the largest Eigen value and corresponding Eigen vector of the following matrix.

$$\begin{bmatrix} 1 & 6 & 1 \\ 1 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$ By Jacobi method.

Theory:

If the action of a matrix on a (nonzero) vector changes its magnitude but not its direction, then the vector is called an eigenvector of that matrix. A vector which is "flipped" to point in the opposite direction is also considered an eigenvector. Each eigenvector is, in effect, multiplied by a scalar, called the eigenvalue corresponding to that eigenvector. The eigenspace corresponding to one eigenvalue of a given matrix is the set of all eigenvectors of the matrix with that eigenvalue.

Many kinds of mathematical objects can be treated as vectors: ordered pairs, functions, harmonic modes, quantum states, and frequencies are examples. In these cases, the concept of *direction* loses its ordinary meaning, and is given an abstract definition. Even so, if this abstract *direction* is unchanged by a given linear transformation, the prefix "eigen" is used, as in *eigenfunction*, *eigenmode*, *eigenstate*, and *eigenfrequency*.

When a transformation is represented by a square matrix *A*, the eigenvalue equation can be expressed as $(A x - \lambda I x) = 0$.
This can be rearranged to

$$(A - \lambda I)\, x = 0.$$

If there exists an inverse,

$$(A - \lambda I)^{-1} = 0.$$

Then both sides can be left multiplied by the inverse to obtain the trivial solution: **x** = **0**.
Thus we require there to be no inverse by assuming from linear algebra that the determinant equals zero:

$$\text{Det}\,(A - \lambda I) = 0.$$

The determinant requirement is called the *characteristic equation* (less often, secular equation) of *A*, and the left-hand side is called the *characteristic polynomial*. When expanded, this gives a polynomial equation for λ. The eigenvector **x** or its components are not present in the characteristic equation.

In vector calculus, the **Jacobian matrix** is the matrix of all first-order partial derivatives of a vector-valued function. Suppose $F : \mathbf{R}^n \to \mathbf{R}^m$ is a function from Euclidean *n*-space to Euclidean *m*-space. Such a function is given by *m* real-valued component functions, $y_1(x_1,...,x_n)$, ..., $y_m(x_1,...,x_n)$. The partial derivatives of all these functions (if they exist) can be organized in an *m*-by-*n* matrix, the Jacobian matrix *J* of *F*, as follows:

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}.$$

This matrix is also denoted by $J_F(x_1, \ldots, x_n)$ and $\frac{\partial(y_1, \ldots, y_m)}{\partial(x_1, \ldots, x_n)}$. The *i* th row (*i* = 1, ..., *m*) of this matrix is the gradient of the i^{th} component function y_i: (∇y_i).

The **Jacobian determinant** (often simply called the **Jacobian**) is the determinant of the Jacobian matrix.

The Jacobian of a function describes the orientation of a tangent plane to the function at a given point. In this way, the Jacobian generalizes the gradient of a scalar valued function of multiple variables which itself generalizes the derivative of a scalar-valued function of a scalar. Likewise, the Jacobian can also be thought of as describing the amount of "stretching" that a transformation imposes. For example, if $(x_2,y_2) = f(x_1,y_1)$ is used to transform an image, the Jacobian of *f*, $J(x_1,y_1)$ describes how much the image in the neighborhood of (x_1,y_1) is stretched in the *x*, *y*, and *xy* directions.

If a function is differentiable at a point, its derivative is given in coordinates by the Jacobian, but a function doesn't need to be differentiable for the Jacobian to be defined, since only the partial derivatives are required to exist.

The importance of the Jacobian lies in the fact that it represents the best linear approximation to a differentiable function near a given point. In this sense, the Jacobian is the derivative of a multivariate function. For a function of *n* variables, *n* > 1, the derivative of a numerical function must be matrix-valued, or a partial derivative.

If **p** is a point in $\mathbf{R}^n$ and *F* is differentiable at **p**, then its derivative is given by $J_F(\mathbf{p})$. In this case, the linear map described by $J_F(\mathbf{p})$ is the best linear approximation of *F* near the point **p**, in the sense that

$$F(\mathbf{x}) = F(\mathbf{p}) + J_F(\mathbf{p})(\mathbf{x} - \mathbf{p}) + o(\|\mathbf{x} - \mathbf{p}\|^2)$$

for **x** close to **p** and where *o* is the little o-notation (for $x \to p$, not $x \to \infty$) and $\|\mathbf{x} - \mathbf{p}\|$ is the distance between **x** and **p**.

In a sense, both gradient and Jacobian are "first derivatives", the former of a scalar function of several variables and the latter of a vector function of several variables. Jacobian of the gradient has a special name: the Hessian matrix which in a sense is the "second derivative" of the scalar function of several variables in question. (More generally, gradient is a special version of Jacobian; it is the Jacobian of a scalar function of several variables.)

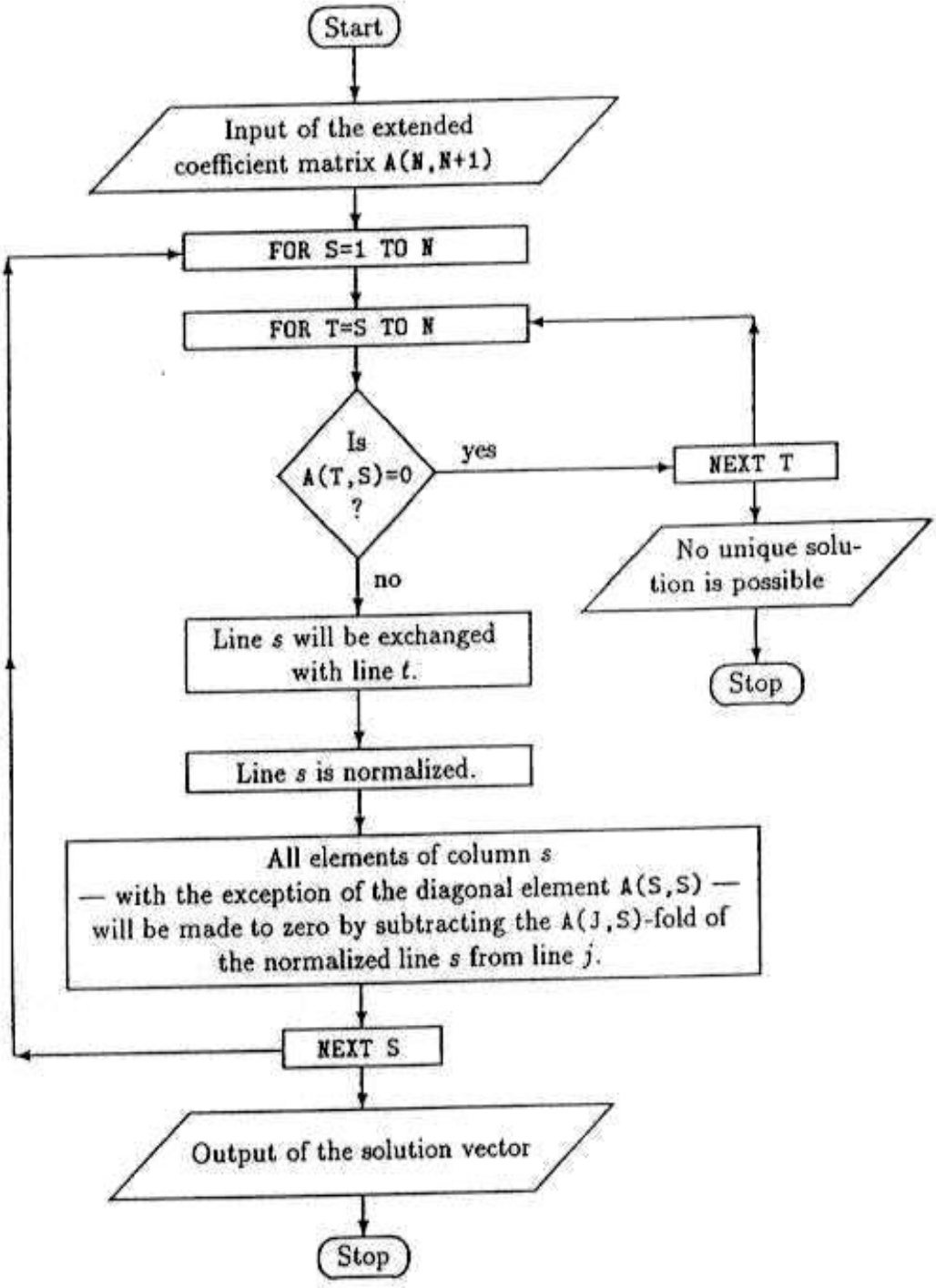
According to the inverse function theorem, the matrix inverse of the Jacobian matrix of a function is the Jacobian matrix of the *inverse* function. That is, for some function $F : \mathbf{R}^n \to \mathbf{R}^n$ and a point p in $\mathbf{R}^n$,

$$J_{F^{-1}}(F(p)) = [J_F(p)]^{-1}.$$

It follows that the (scalar) inverse of the Jacobian determinant of a transformation is the Jacobian determinant of the inverse transformation.

Calculation:

Practical In-charge Sign



Flow Chart 9: To find Eigen values and vector using Jacobi method.