## Complete Algorithm for Bisection Method

**1. start**

**2. Define function f(x)**

**3. Choose initial guesses x0 and x1 such that f(x0)f(x1) < 0**

**4. Choose pre-specified tolerable error e.**

**5. Calculate new approximated root as x2 = (x0 + x1)/2**

**6. Calculate f(x0)f(x2)**
- **a. if f(x0)f(x2) < 0 then x0 = x0 and x1 = x2**
- **b. if f(x0)f(x2) > 0 then x0 = x2 and x1 = x1**
- **c. if f(x0)f(x2) = 0 then goto (8)**

**7. if |f(x2)|>e then goto (5) otherwise goto (8)**

**8. Display x2 as root.**

**9. Stop**

## Program in C

```
/* Header Files */

#include<stdio.h>

#include<conio.h>

#include<math.h>

/*

 Defining equation to be solved.

 Change this equation to solve another problem.

*/
```

```
#define f(x) cos(x) - x * exp(x)

void main()

{

	float x0, x1, x2, f0, f1, f2, e;

	int step = 1;

	clrscr();

	/* Inputs */

	up:

	printf("\nEnter two initial guesses:\n");

	scanf("%f%f", &x0, &x1);

	printf("Enter tolerable error:\n");

	scanf("%f", &e);

	/* Calculating Functional Value */

	f0 = f(x0);

	f1 = f(x1);

	/* Checking whether given guesses brackets the root or not. */

	if( f0 * f1 > 0.0)

	{

		printf("Incorrect Initial Guesses.\n");
```

```
        goto up;
    }
/* Implementing Bisection Method */
    printf("\nStep\t\tx0\t\tx1\t\tx2\t\tf(x2)\n");
    do
    {
        x2 = (x0 + x1)/2;
        f2 = f(x2);

        printf("%d\t\t%f\t%f\t%f\t%f\n",step, x0, x1, x2, f2);

        if( f0 * f2 < 0)
        {
            x1 = x2;
            f1 = f2;
        }
        else
        {
            x0 = x2;
```

```
                    f0 = f2;
            }
            step = step + 1;
        }while(fabs(f2)>e);
        printf("\nRoot is: %f", x2);
        getch();
}
```

Output: Bisection Method Using C

Output: Bisection Method Using C

```
Enter two initial guesses:
0
1
Enter tolerable error:
0.0001

Step            x0              x1              x2              f(x2)
1               0.000000        1.000000        0.500000        0.053222
2               0.500000        1.000000        0.750000        -0.856061
3               0.500000        0.750000        0.625000        -0.356691
4               0.500000        0.625000        0.562500        -0.141294
5               0.500000        0.562500        0.531250        -0.041512
6               0.500000        0.531250        0.515625        0.006475
7               0.515625        0.531250        0.523438        -0.017362
8               0.515625        0.523438        0.519531        -0.005404
9               0.515625        0.519531        0.517578        0.000545
10              0.517578        0.519531        0.518555        -0.002427
11              0.517578        0.518555        0.518066        -0.000940
12              0.517578        0.518066        0.517822        -0.000197
13              0.517578        0.517822        0.517700        0.000174
14              0.517700        0.517822        0.517761        -0.000012

Root is: 0.517761
```

# Newton Raphson Method Algorithm

**Newton Raphson Method** is open method and starts with one initial guess for finding real root of non-linear equations.

In Newton Raphson method if *x0* is initial guess then next approximated root *x1* is obtained by following formula:

```
x1 = x0 - f(x0) / g(x0)
```

And then process is repeated i.e. we use *x1* to find *x2* and so on until we find the root within desired accuracy.

# Complete Algorithm for Newton Raphson Method

```
1. Start

2. Define function as f(x)

3. Define first derivative of f(x) as g(x)

4. Input initial guess (x0), tolerable error (e)
   and maximum iteration (N)

5. Initialize iteration counter i = 1

6. If g(x0) = 0 then print "Mathematical Error"
   and goto (12) otherwise goto (7)

7. Calcualte x1 = x0 - f(x0) / g(x0)

8. Increment iteration counter i = i + 1

9. If i >= N then print "Not Convergent"
   and goto (12) otherwise goto (10)

10. If |f(x1)| > e then set x0 = x1
    and goto (6) otherwise goto (11)

11. Print root as x1

12. Stop
```

**C program**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
```

```
/* Defining equation to be solved.
   Change this equation to solve another problem. */
#define    f(x)    3*x - cos(x) -1

/* Defining derivative of g(x).
   As you change f(x), change this function also. */
#define   g(x)   3 + sin(x)

void main()
{
            float x0, x1, f0, f1, g0, e;
            int step = 1, N;
            clrscr();
    /* Inputs */
            printf("\nEnter initial guess:\n");
            scanf("%f", &x0);
            printf("Enter tolerable error:\n");
            scanf("%f", &e);
            printf("Enter maximum iteration:\n");
            scanf("%d", &N);
            /* Implementing Newton Raphson Method */
            printf("\nStep\t\tx0\t\tf(x0)\t\tx1\t\tf(x1)\n");
            do
            {
                        g0 = g(x0);
                        f0 = f(x0);
                        if(g0 == 0.0)
                        {
                                    printf("Mathematical Error.");
                                    exit(0);
                        }


                        x1 = x0 - f0/g0;


                        printf("%d\t\t%f\t%f\t%f\t%f\n",step,x0,f0,x1,f1);
                        x0 = x1;

                        step = step+1;

                        if(step > N)
```

```
		{
			printf("Not Convergent.");
			exit(0);
		}

		f1 = f(x1);

	}while(fabs(f1)>e);

	printf("\nRoot is: %f", x1);
	getch();
}
```

Output:

```
Enter initial guess:
1
Enter tolerable error:
0.00001
Enter maximum iteration:
5

Step        x0          f(x0)       x1          f(x1)
1           1.000000    1.459698    0.620016    0.046179
2           0.620016    0.046179    0.607121    0.000068
3           0.607121    0.000068    0.607102    -0.000000

Root is: 0.607102
```

# Fixed Point Iteration Method Algorithm

Fixed point iteration method is open and simple method for finding real root of non-linear equation by successive approximation. It requires only one initial guess to start. Since it is open method its convergence is not guaranteed. This method is also known as **Iterative Method**

To find the root of nonlinear equation $f(x)=0$ by fixed point iteration method, we write given equation $f(x)=0$ in the form of $x = g(x)$.

If *x0* is initial guess then next approximated root in this method is obtaine by:

```
x1 = g(x1)
```

And similarly, next to next approximated root is obtained by using value of *x1* i.e.

```
x2 = g(x2)
```

And the process is repeated until we get root within desired accuracy.

**Note:** While expressing *f(x)=0* to *x = g(x)* we can have many different forms. For convergence, following criteraia must be satisfied.

```
|g'(x)|< 1
```

# Complete Algorithm for Fixed Point Iteration Method

```
1. Start

2. Define function f(x)

3. Define function g(x) which is obtained
   from f(x)=0 such that x = g(x) and |g'(x) < 1|

4. Choose intial guess x0, Tolerable Error e
   and Maximum Iteration N

5. Initialize iteration counter: step = 1

6. Calculate x1 = g(x0)

7. Increment iteration counter: step = step + 1

8. If step > N then print "Not Convergent"
   and goto (12) otherwise goto (10)

9. Set x0 = x1 for next iteration

10. If |f(x1)| > e then goto step (6) otherwise goto step (11)

11. Display x1 as root.

12. Stop
```

**C program**

/* Header Files */

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

/* Define function f(x) which
  is to be solved */
#define   f(x)   cos(x)-3*x+1
/* Write f(x) as x = g(x) and
  define g(x) here */
#define   g(x)   (1+cos(x))/3

int main()
{
	int step=1, N;
	float x0, x1, e;
	clrscr();
	/* Inputs */
	printf("Enter initial guess: ");
	scanf("%f", &x0);
	printf("Enter tolerable error: ");
	scanf("%f", &e);
	printf("Enter maximum iteration: ");
	scanf("%d", &N);
	/* Implementing Fixed Point Iteration */
	printf("\nStep\tx0\t\tf(x0)\t\tx1\t\tf(x1)\n");
	do
	{
		x1 = g(x0);
		printf("%d\t%f\t%f\t%f\t%f\n",step, x0, f(x0), x1, f(x1));

		step = step + 1;

		if(step>N)
		{
			printf("Not Convergent.");
			exit(0);
		}

		x0 = x1;

	}while( fabs(f(x1)) > e);
```

```
	printf("\nRoot is %f", x1);

	getch();
	return(0);
}
```

Output:

```
Enter initial guess: 1
Enter tolerable error: 0.000001
Enter maximum iteration: 10

Step    x0              f(x0)           x1              f(x1)
1       1.000000        -1.459698       0.513434        0.330761
2       0.513434        0.330761        0.623688        -0.059333
3       0.623688        -0.059333       0.603910        0.011391
4       0.603910        0.011391        0.607707        -0.002162
5       0.607707        -0.002162       0.606986        0.000411
6       0.606986        0.000411        0.607124        -0.000078
7       0.607124        -0.000078       0.607098        0.000015
8       0.607098        0.000015        0.607102        -0.000003
9       0.607102        -0.000003       0.607102        0.000001

Root is 0.607102
```

# Gauss Elimination Method Algorithm

In linear algebra, **Gauss Elimination Method** is a procedure for solving systems of linear equation. It is also known as **Row Reduction Technique**. In this method, the problem of systems of linear equation having n unknown variables, matrix having rows n and columns n+1 is formed. This matrix is also known as **Augmented Matrix**. After forming **n x n+1** matrix, matrix is transformed to **upper trainagular matrix by row operations**. Finally result is obtained by **Back Substitution**.

## Algorithm for Gauss Elimination Method

```
1. Start

2. Read Number of Unknowns: n
```

```
3. Read Augmented Matrix (A) of n by n+1 Size

4. Transform Augmented Matrix (A)
   to Upper Trainagular Matrix by Row Operations.

5. Obtain Solution by Back Substitution.

6. Display Result.

7. Stop
```

## C program

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>

#define   SIZE   10

int main()
{
	float a[SIZE][SIZE], x[SIZE], ratio;
	int i,j,k,n;

	clrscr();

	/* Inputs */
	/* 1. Reading number of unknowns */
	printf("Enter number of unknowns: ");
	scanf("%d", &n);
	/* 2. Reading Augmented Matrix */
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n+1;j++)
		{
			printf("a[%d][%d] = ",i,j);
			scanf("%f", &a[i][j]);
		}
	}
	/* Applying Gauss Elimination */
	for(i=1;i<=n-1;i++)
	{
		if(a[i][i] == 0.0)
		{
```

```
                    printf("Mathematical Error!");
                    exit(0);
            }
            for(j=i+1;j<=n;j++)
            {
                    ratio = a[j][i]/a[i][i];

                    for(k=1;k<=n+1;k++)
                    {
                              a[j][k] = a[j][k] - ratio*a[i][k];
                    }
            }
    }
    /* Obtaining Solution by Back Subsitution */
    x[n] = a[n][n+1]/a[n][n];

    for(i=n-1;i>=1;i--)
    {
            x[i] = a[i][n+1];
            for(j=i+1;j<=n;j++)
            {
                        x[i] = x[i] - a[i][j]*x[j];
            }
            x[i] = x[i]/a[i][i];
    }
    /* Displaying Solution */
    printf("\nSolution:\n");
    for(i=1;i<=n;i++)
    {
           printf("x[%d] = %0.3f\n",i, x[i]);
    }
    getch();
    return(0);
}
```

**Output:**

```
Enter number of unknowns: 3
Enter coefficients of Augmented Matrix:
a[1][1] = 1
a[1][2] = 1
a[1][3] = 1
a[1][4] = 9
a[2][1] = 2
a[2][2] = -3
a[2][3] = 4
a[2][4] = 13
a[3][1] = 3
a[3][2] = 4
a[3][3] = 5
a[3][4] = 40

Solution:
x[1] = 1.000
x[2] = 3.000
x[3] = 5.000
```

# Gauss Jordan Method Algorithm

In linear algebra, **Gauss Jordan Method** is a procedure for solving systems of linear equation. It is also known as **Row Reduction Technique**. In this method, the problem of systems of linear equation having n unknown variables, matrix having rows n and columns n+1 is formed. This matrix is also known as **Augmented Matrix**. After forming **n x n+1** matrix, matrix is transformed to **diagonal matrix by row operations**. Finally result is obtained by making all diagonal element to 1 i.e. identity matrix.

## Algorithm for Gauss Jordan Method

```
1. Start

2. Read Number of Unknowns: n

3. Read Augmented Matrix (A) of n by n+1 Size

4. Transform Augmented Matrix (A)
   to Diagonal Matrix by Row Operations.

5. Obtain Solution by Making All Diagonal Elements to 1.
```

```
6. Display Result.

7. Stop
```

## C Program

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>

#define SIZE 10

int main()
{
	float a[SIZE][SIZE], x[SIZE], ratio;
	int i,j,k,n;
	clrscr();
	/* Inputs */
	/* 1. Reading number of unknowns */
	printf("Enter number of unknowns: ");
	scanf("%d", &n);
	/* 2. Reading Augmented Matrix */
	printf("Enter coefficients of Augmented Matrix:\n");
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n+1;j++)
		{
			printf("a[%d][%d] = ",i,j);
			scanf("%f", &a[i][j]);
		}
	}
	/* Applying Gauss Jordan Elimination */
	for(i=1;i<=n;i++)
	{
		if(a[i][i] == 0.0)
		{
			printf("Mathematical Error!");
			exit(0);
		}
		for(j=1;j<=n;j++)
		{
			if(i!=j)
			{
```

```
                        ratio = a[j][i]/a[i][i];
                        for(k=1;k<=n+1;k++)
                        {
                            a[j][k] = a[j][k] - ratio*a[i][k];
                        }
                    }
                }
        }
        /* Obtaining Solution */
        for(i=1;i<=n;i++)
        {
            x[i] = a[i][n+1]/a[i][i];
        }
        /* Displaying Solution */
        printf("\nSolution:\n");
        for(i=1;i<=n;i++)
        {
            printf("x[%d] = %0.3f\n",i, x[i]);
        }
        getch();
        return(0);
}
```

**Output:**

```
Enter number of unknowns: 3
Enter coefficients of Augmented Matrix:
a[1][1] = 3
a[1][2] = 1
a[1][3] = 2
a[1][4] = 3
a[2][1] = 2
a[2][2] = -3
a[2][3] = -1
a[2][4] = -3
a[3][1] = 1
a[3][2] = 2
a[3][3] = 1
a[3][4] = 4

Solution:
x[1] = 1.000
x[2] = 2.000
x[3] = -1.000
```

# Matrix Inverse Using Gauss Jordan Method Algorithm

In linear algebra, **Gauss Jordan Method** is a procedure for solving systems of linear equation using **Row Reduction Technique**. In this method, the problem of systems of linear equation having n unknown variables, matrix having rows n and columns n+1 is formed. This matrix is also known as **Augmented Matrix**. After forming **n x n+1** matrix, matrix is transformed to **diagonal matrix by row operations**.

Gauss Jordan method can also be applied for finding inverse of a matrix by similar row operations.

## Algorithm for Finding Inverse of Matrix Gauss Jordan Method

```
1. Start

2. Read Order of Matrix (n).

3. Read Matrix (A) of Order (n).

4. Augment and Identity Matrix of Order n to Matrix A.

5. Apply Gauss Jordan Elimination on Augmented Matrix (A).

6. Perform Row Operations to Convert the Principal Diagonal to 1.

7. Display the Inverse Matrix.

8. Stop.
```

## C program

```c
#include<stdio.h>
#include<math.h>

#define SIZE 10

int main()
{
		float a[SIZE][SIZE], x[SIZE], ratio, temp;
		int i,j,k,n;

		/* Inputs */
		/* 1. Reading order of matrix */
		printf("Enter order of matrix: ");
		scanf("%d", &n);
		/* 2. Reading Matrix */
		printf("Enter coefficients of Matrix:\n");
		for(i=1;i<=n;i++)
		{
			for(j=1;j<=n;j++)
			{
				printf("a[%d][%d] = ",i,j);
				scanf("%f", &a[i][j]);
			}
		}
		/* Augmenting Identity Matrix of Order n */
		for(i=1;i<=n;i++)
		{
```

```
			for(j=1;j<=n;j++)
			{
				if(i==j)
				{
					a[i][j+n] = 1;
				}
				else
				{
					a[i][j+n] = 0;
				}
			}
		}
		/* Applying Gauss Jordan Elimination */
		for(i=1;i<=n;i++)
		{
			if(a[i][i] == 0.0)
			{
				printf("Mathematical Error!");
				exit(0);
			}
			for(j=1;j<=n;j++)
			{
				if(i!=j)
				{
					ratio = a[j][i]/a[i][i];
					for(k=1;k<=2*n;k++)
					{
						a[j][k] = a[j][k] - ratio*a[i][k];
					}
				}
			}
		}
		/* Row Operation to Make Principal Diagonal to 1 */
		for(i=1;i<=n;i++)
		{
			temp = a[i][i];
			for(j=1;j<=2*n;j++)
			{
				a[i][j] = a[i][j]/temp;
			}
		}
		/* Displaying Inverse Matrix */
```

```
		printf("\nInverse Matrix is:\n");
		for(i=1;i<=n;i++)
		{
			for(j=n+1;j<=2*n;j++)
			{
				printf("%0.3f\t",a[i][j]);
			}
			printf("\n");
		}

		return(0);
}
```

Output

```
Enter order of matrix: 3
Enter coefficients of Matrix:
a[1][1] = 1
a[1][2] = 1
a[1][3] = 3
a[2][1] = 1
a[2][2] = 3
a[2][3] = -3
a[3][1] = -2
a[3][2] = -4
a[3][3] = -4
Inverse Matrix is:
 3.000   1.000   1.500
-1.250  -0.250  -0.750
-0.250  -0.250  -0.250
```

**Power Method Using C Programming Language**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

#define SIZE 10

int main()
{
	float a[SIZE][SIZE], x[SIZE],x_new[SIZE];
	float temp, lambda_new, lambda_old, error;
	int i,j,n, step=1;
```

```
clrscr();
/* Inputs */
printf("Enter Order of Matrix: ");
scanf("%d", &n);
printf("Enter Tolerable Error: ");
scanf("%f", &error);
/* Reading Matrix */
printf("Enter Coefficient of Matrix:\n");
for(i=1;i<=n;i++)
{
	for(j=1;j<=n;j++)
	{
		printf("a[%d][%d]=",i,j);
		scanf("%f", &a[i][j]);
	}
}
/* Reading Intial Guess Vector */
printf("Enter Initial Guess Vector:\n");
for(i=1;i<=n;i++)
{
	printf("x[%d]=",i);
	scanf("%f", &x[i]);
}
/* Initializing Lambda_Old */
lambda_old = 1;
/* Multiplication */
up:
for(i=1;i<=n;i++)
{
	temp = 0.0;
	for(j=1;j<=n;j++)
	{
		temp = temp + a[i][j]*x[j];
	}
	x_new[i] = temp;
}
/* Replacing */
for(i=1;i<=n;i++)
{
	x[i] = x_new[i];
}
/* Finding Largest */
```

```
            lambda_new = fabs(x[1]);
            for(i=2;i<=n;i++)
            {
                        if(fabs(x[i])>lambda_new)
                        {
                                    lambda_new = fabs(x[i]);
                        }
            }
            /* Normalization */
            for(i=1;i<=n;i++)
            {
                        x[i] = x[i]/lambda_new;
            }
            /* Display */
            printf("\n\nSTEP-%d:\n", step);
            printf("Eigen Value = %f\n", lambda_new);
            printf("Eigen Vector:\n");
            for(i=1;i<=n;i++)
            {
                        printf("%f\t", x[i]);
            }
            /* Checking Accuracy */
            if(fabs(lambda_new-lambda_old)>error)
            {
                        lambda_old=lambda_new;
                        step++;
                        goto up;
            }
            getch();
            return(0);
}
```

**Ouput: Power Method Using C Programming**

```
Enter Order of Matrix: 2 ↲
Enter Tolerable Error: 0.001 ↲
Enter Coefficient of Matrix:
a[1][1]=5 ↲
a[1][2]=4 ↲
a[2][1]=1 ↲
a[2][2]=2 ↲
Enter Initial Guess Vector:
x[1]=1 ↲
x[2]=1 ↲
```

STEP-1:
Eigen Value = 9.000000
Eigen Vector:
1.000000    0.333333

STEP-2:
Eigen Value = 6.333333
Eigen Vector:
1.000000    0.263158

STEP-3:
Eigen Value = 6.052631
Eigen Vector:
1.000000    0.252174

STEP-4:
Eigen Value = 6.008696
Eigen Vector:
1.000000    0.250362

STEP-5:
Eigen Value = 6.001447
Eigen Vector:
1.000000    0.250060

STEP-6:
Eigen Value = 6.000241
Eigen Vector:
1.000000    0.250010

STEP-7:
Eigen Value = 6.000040
Eigen Vector:
1.000000    0.250002

# Numerical Integration Using Trapezoidal Method Algorithm

In numerical analysis, Trapezoidal method is a technique for evaluating definite integral. This method is also known as **Trapezoidal rule** or **Trapezium rule.**

This method is based on **Newton's Cote Quadrature Formula** and Trapezoidal rule is obtained when we put value of n=1 in this formula.

In this article, we are going to develop an algorithm for Trapezoidal method.

## Trapezoidal Method Algorithm

```
1. Start

2. Define function f(x)

3. Read lower limit of integration, upper limit of
   integration and number of sub interval

4. Calcultae: step size = (upper limit - lower limit)/number of sub
interval

5. Set: integration value = f(lower limit) + f(upper limit)

6. Set: i = 1

7. If i >= number of sub interval then goto step 11

8. Calculate: k = lower limit + i * h

9. Calculate: Integration value = Integration Value + 2* f(k)

10. Increment i by 1 i.e. i = i+1 and go to step 7

11. Calculate: Integration value = Integration value * step size/2

12. Display Integration value as required answer

13. Stop
```

**C program**

```
#include<stdio.h>
#include<math.h>
```

```c
/* Define function here */
#define f(x) 1/(1+pow(x,2))

int main()
{
 float lower, upper, integration=0.0, stepSize, k;
 int i, subInterval;

 /* Input */
 printf("Enter lower limit of integration: ");
 scanf("%f", &lower);
 printf("Enter upper limit of integration: ");
 scanf("%f", &upper);
 printf("Enter number of sub intervals: ");
 scanf("%d", &subInterval);

 /* Calculation */

 /* Finding step size */
 stepSize = (upper - lower)/subInterval;

 /* Finding Integration Value */
 integration = f(lower) + f(upper);

 for(i=1; i<= subInterval-1; i++)
 {
  k = lower + i*stepSize;
  integration = integration + 2 * (f(k));
 }
 integration = integration * stepSize/2;
 printf("\nRequired value of integration is: %.3f", integration);

 return 0;
}
```

# Output

```
Enter lower limit of integration: 0
Enter upper limit of integration: 6
Enter number of sub intervals: 6

Required value of integration is: 1.411
```

# C Program for Euler's Method

```
#include<stdio.h>

/* defining ordinary differential equation to be solved */
/* In this example we are solving dy/dx = x + y */
#define f(x,y) x+y

int main()
{
 float x0, y0, xn, h, yn, slope;
 int i, n;

 printf("Enter Initial Condition\n");
 printf("x0 = ");
 scanf("%f", &x0);
 printf("y0 = ");
 scanf("%f", &y0);
 printf("Enter calculation point xn = ");
 scanf("%f", &xn);
 printf("Enter number of steps: ");
 scanf("%d", &n);

 /* Calculating step size (h) */
 h = (xn-x0)/n;

 /* Euler's Method */
 printf("\nx0\ty0\tslope\tyn\n");
 printf("------------------------------\n");
```

```
 for(i=0; i < n; i++)
 {
  slope = f(x0, y0);
  yn = y0 + h * slope;
  printf("%.4f\t%.4f\t%0.4f\t%.4f\n",x0,y0,slope,yn);
  y0 = yn;
  x0 = x0+h;
 }

 /* Displaying result */
 printf("\nValue of y at x = %0.2f is %0.3f",xn, yn);

 return 0;
}
```

## Output

```
Enter Initial Condition
x0 = 0
y0 = 1
Enter calculation point xn = 1
Enter number of steps: 10

x0      y0      slope   yn
------------------------------
0.0000  1.0000  1.0000  1.1000
0.1000  1.1000  1.2000  1.2200
0.2000  1.2200  1.4200  1.3620
0.3000  1.3620  1.6620  1.5282
0.4000  1.5282  1.9282  1.7210
0.5000  1.7210  2.2210  1.9431
0.6000  1.9431  2.5431  2.1974
0.7000  2.1974  2.8974  2.4872
0.8000  2.4872  3.2872  2.8159
0.9000  2.8159  3.7159  3.1875

Value of y at x = 1.00 is 3.187
```

# Ordinary Differential Equation Using Fourth Order Runge Kutta (RK) Method Using C

This program is implementation of **Runge Kutta Fourth Order method for solving ordinary differential equation** using C programming language with output.

Output of this is program is solution for **dy/dx = ($y^2$ - $x^2$)/($y^2$+$x^2$)** with initial condition **y = 1 for x = 0 i.e. y(0) = 1 and we are trying to evaluate this differential equation at y = 0.4 in two steps i.e. n = 2.** ( Here y = 0.4 i.e. y(0.4) = ? is our calculation point)

## C Program for RK-4 Method

```
#include<stdio.h>

/* Defining ordinary differential equation to be solved */
#define f(x,y) (y*y-x*x)/(y*y+x*x)

int main()
{
 float x0, y0, xn, h, yn, k1, k2, k3, k4, k;
 int i, n;

 printf("Enter Initial Condition\n");
 printf("x0 = ");
 scanf("%f", &x0);
 printf("y0 = ");
 scanf("%f", &y0);
 printf("Enter calculation point xn = ");
 scanf("%f", &xn);
 printf("Enter number of steps: ");
 scanf("%d", &n);

 /* Calculating step size (h) */
 h = (xn-x0)/n;
```

```
 /* Runge Kutta Method */
 printf("\nx0\ty0\tyn\n");
 for(i=0; i < n; i++)
 {
  k1 = h * (f(x0, y0));
  k2 = h * (f((x0+h/2), (y0+k1/2)));
  k3 = h * (f((x0+h/2), (y0+k2/2)));
  k4 = h * (f((x0+h), (y0+k3)));
  k = (k1+2*k2+2*k3+k4)/6;
  yn = y0 + k;
  printf("%0.4f\t%0.4f\t%0.4f\n",x0,y0,yn);
  x0 = x0+h;
  y0 = yn;
 }

 /* Displaying result */
 printf("\nValue of y at x = %0.2f is %0.3f",xn, yn);

 return 0;
}
```

# Output

```
Enter Initial Condition
x0 = 0
y0 = 1
Enter calculation point xn = 0.4
Enter number of steps: 2

x0      y0      yn
0.0000  1.0000  1.1960
0.2000  1.1960  1.3753

Value of y at x = 0.40 is 1.375
```

## Lagrange Interpolation

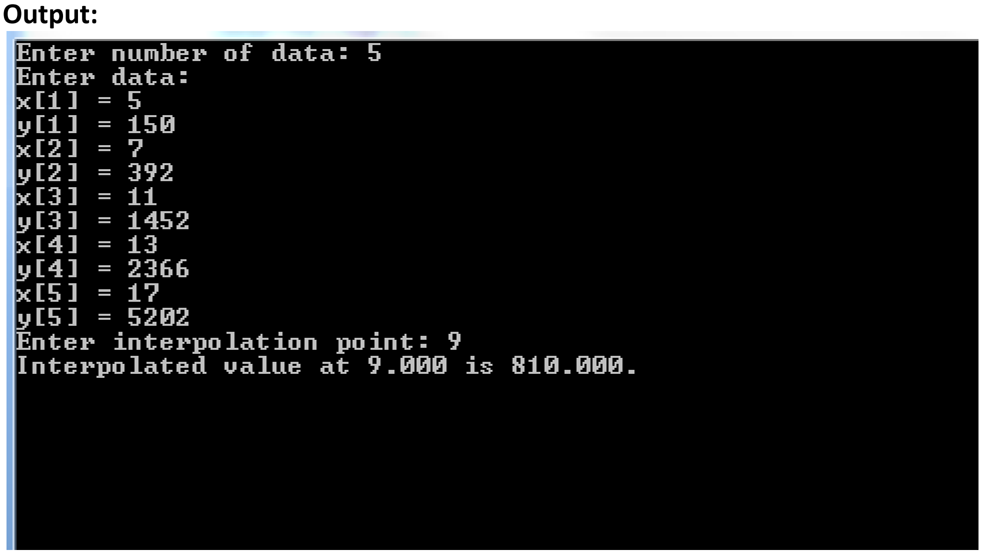
#include<stdio.h>

#include<conio.h>

```
void main()

{

float x[100],y[100],xp,yp=0,p;

int i,j,n;

clrscr();

/* Input Section */

printf("Enter number of data: ");

scanf("%d", &n);

printf("Enter data:\n");

for(i=1;i<=n;i++)

{

printf("x[%d] = ",i);

scanf("%f", &x[i]);

printf("y[%d] = ",i);

scanf("%f", &y[i]);

}

printf("Enter interpolation point: ");

scanf("%f", &xp);

/* Implementing Lagrange Interpolation */

for(i=1;i<=n;i++)

{

p=1;

for(j=1;j<=n;j++)

{
```

```
if(i!=j)

{

p =p* (xp -x[j])/(x[i] -x[j]);

}

}

yp =yp +p *y[i];

}

printf("Interpolated value at %.3f is %.3f.",xp,yp);

getch();

}
```

**Output:**



```
Enter number of data: 5
Enter data:
x[1] = 5
y[1] = 150
x[2] = 7
y[2] = 392
x[3] = 11
y[3] = 1452
x[4] = 13
y[4] = 2366
x[5] = 17
y[5] = 5202
Enter interpolation point: 9
Interpolated value at 9.000 is 810.000.
```