

## # Code

```
line-fibonacci.h
int main()
{
    int n,a=0,b=1,c,i;
    clrscr();
    printf("Enter the range: ");
    scanf("%d", &n);
    printf("Fibonacci : a,b");
    for(i=1;i<=n;i++){
        c=a+b;
        a=b;
        b=c;
        printf("%d ",b);
    }
    printf("\nPress any key to continue . . .");
    getch();
}
```

## # Output

```
Enter the range: 10
0
1
1
2
3
5
8
13
21
34
55
89
Sukhmani Kaur CSE-1 00413202722
```

Expt. No.

1

Practical No - 1

Q write a program to print fibonacci series upto a range of numbers.

### • Description →

The Fibonacci series is a series in which each number is the sum of two preceding ones. Numbers that are part of Fibonacci series are known as Fibonacci numbers, commonly denoted by  $F_n$ . The sequence starts from 0 and 1, the first few values in the sequence are:-

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Thus, Fibonacci numbers may be defined by the recurrence relation:

$$F_0 = 0, \quad F_1 = 1$$

$$\text{And } F_n = F_{n-1} + F_{n-2}; \quad n \geq 1$$

Teacher's Signature :

Rajendra

# Code

File Edit Search Run Compile Debug Project

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

```
int main()
{
    int n, i, j;
    int sum=0;
    float avg;
    clrscr();
    printf("Enter the range of numbers : ");
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    {
        scanf("%d", &j);
        sum=sum+j;
    }
    avg=sum/n;
    printf("Sum = %d", sum);
    printf("Average = %f", avg);
    printf("Average = %f", avg);
    getch();
}
```

# Output

```
Enter the range of numbers : 3
11
22
33
Sum: 66
Average: 22.00
Sukhmani Kaur CSE-1 00411262722
```

Expt. No. .... 2 .....

### Practical No-2

Q write a program to find sum and average of the numbers given by user.

- Description

This program allows user to enter the numbers of which the sum and average are to be calculated.

Then with the help of the loop, the program will calculate the sum and average of the given numbers and provide the required output.

```

int main()
{
    int i,j,k,e,f,e2,g;
    int l,m,n,o,p,q,r,s,t,u,v,w,x,y,z;
    int a[10][10],b[10][10],c[10][10];
    clrscr();
    printf("Enter the number of rows and columns : ");
    scanf("%d %d", &r, &c);
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            a[i][j]=0;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            b[i][j]=0;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            c[i][j]=0;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            printf("Enter element (%d,%d) : ", i+1, j+1);
            scanf("%d", &a[i][j]);
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            printf("Enter element (%d,%d) : ", i+1, j+1);
            scanf("%d", &b[i][j]);
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            c[i][j]=a[i][j]+b[i][j];
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            printf("Element (%d,%d) = %d\n", i+1, j+1, c[i][j]);
}

```

```

for(i=0;i<10;i++)
    for(j=0;j<10;j++)
        print("%d\t",i+j);
    print("\n");
}

```

F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

File Edit Search Run

```
for(i=0; i<3; i++) {  
    for(k=0; k<3; k++) {  
        cout << i << " " << k << endl;  
    }  
}
```

```

    for(i=0;i<K;i++) {
        for(k=0;k<K;k++) {
            for(j=0;j<K;j++) {
                d += a[i][j]*b[j][k];
            }
        }
    }

```

```
print( );  
for(i=0; i<3; ++i){  
    for(j=0; j<3; ++j){  
        printf(" %c", i+j);  
    }  
    printf("\n");  
}  
  
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
```

- Description -  
Write a program to multiply two matrices ( $3 \times 3$ ) that are given/ input by the user.

Matrix = A rectangular array of  $m \times n$  numbers  
 $m$  is no. of rows and  $n$  is no. of columns

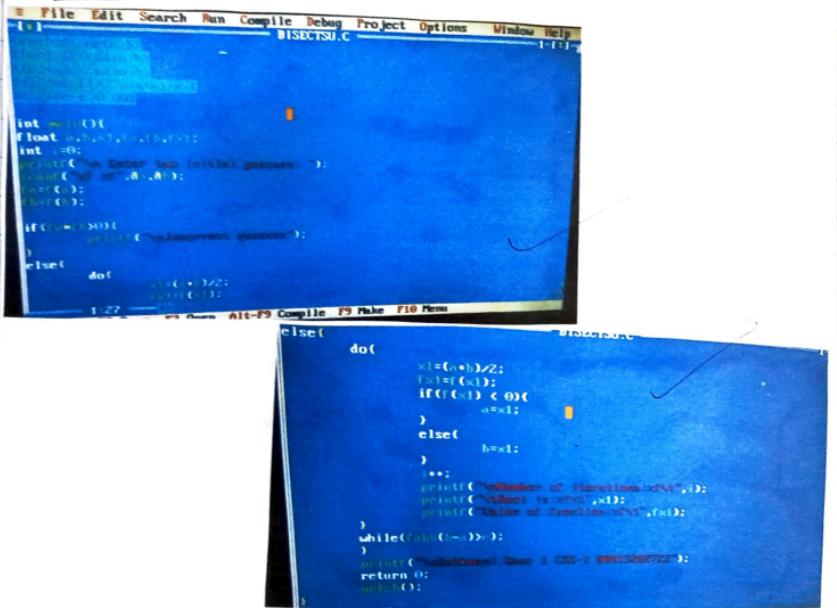
$$A_{ij} \times B_{jk} = C_{ik}$$

- The product of two matrices, A and B, is the sum of the products across some row of A with corresponding entries down some column of B.

## # Output

```
Enter first matrix:  
Enter element of row 1 column 1: 1  
Enter element of row 1 column 2: 2  
Enter element of row 1 column 3: 3  
Enter element of row 2 column 1: 4  
Enter element of row 2 column 2: 5  
Enter element of row 2 column 3: 6  
Enter element of row 3 column 1: 7  
Enter element of row 3 column 2: 8  
Enter element of row 3 column 3: 9  
Enter second matrix:  
enter element of row 1 column 1: 1  
enter element of row 1 column 2: 2  
enter element of row 1 column 3: 3  
enter element of row 2 column 1: 4  
enter element of row 2 column 2: 5  
enter element of row 2 column 3: 6  
enter element of row 3 column 1: 7  
enter element of row 3 column 2: 8  
enter element of row 3 column 3: 9  
display  
30 36 42  
66 81 96  
102 126 150  
Sukhmani Kaur CSE-1
```

# Code



```

#include <conio.h>
#include <iostream.h>
int main()
{
    float a,b,x;
    int i=0;
    cout<<"Enter two initial guesses: ";
    cin>>a>>b;
    if(a>b)
    {
        cout<<"Value of a must be less than b";
        return 0;
    }
    else
    {
        do
        {
            float x1=(a+b)/2;
            f(x1);
            if(f(x1)<0)
                a=x1;
            else
                b=x1;
            cout<<"Value of f(x1) = " << f(x1);
            cout<<"\n";
        } while(f(a)>0);
        cout<<"Root is between " << a << " & " << b;
        return 0;
    }
}

```

# Output

```

Enter two initial guesses: 1 2
Incorrect guesses
Sukhmani Kaur 1 CSE-1 00413202722
Enter two initial guesses:

```

Expt. No. 4

Practical No-4

- Aim - Write a program to solve a given function using Bisection method.

- Description →

The method is also called interval halving method. This method is used to find root of an equation in a given interval that is value of 'x' for which  $f(x)=0$ .

It is based on Intermediate Value Theorem which states that if  $f(x)$  is a continuous function and there are two real numbers such that  $f(a)*f(b) < 0$  then it is guaranteed to have atleast one root between them.

Steps -

- Let  $f(x)$  be a continuous function in  $[a,b]$ , then  $f(a)f(b) < 0$  then  $x_1 = \frac{a+b}{2}$ .
- If  $f(x_1) = 0$ ,  $x_1 \rightarrow$  root of  $f(x)$  otherwise root lies b/w  $a & x_1$  or  $x_1 & b$  as  $f(x_1) > 0$  or  $f(x_1) < 0$ .

```
Sukhmani Kaur 1 CSE-1 00413202722
Enter two initial guesses: 0 1

Number of iterations:-0.000000      Root is:0.500000      Value of function
n:-0.020574
Number of iterations:-2.388178452497233e+307      Root is:0.750000      Value of function:0.431639
Number of iterations:0.000000      Root is:0.625000      Value of function
n:0.210897
Number of iterations:0.000000      Root is:0.562500      Value of function
n:0.095883
Number of iterations:0.000000      Root is:0.531250      Value of function
n:0.037861
Number of iterations:-0.000000      Root is:0.515625      Value of function
n:0.008704
Number of iterations:0.000000      Root is:0.507812      Value of function
n:-0.005921
Number of iterations:-1.769368844537142e+167      Root is:0.511719      Value of function:0.001395
Number of iterations:0.000000      Root is:0.509766      Value of function
n:-0.002262
Number of iterations:-2.31450048931978878e+303      Root is:0.510742      Value of function:-0.000433
Sukhmani Kaur 1 CSE-1 00413202722
Enter two initial guesses:
```

# Code

```
[1] #include<cs51.h>
#include<math.h>
#include<stdio.h>
int main()
{
    float x0,x1,x2,f0,f1,f2,y0;
    int i=1,n;
    printf("Enter the function : ");
    scanf("%f %f %f %f %f %f", &x1, &x2, &f0, &f1, &f2, &y0);
    printf("Enter the tolerance value : ");
    scanf("%f", &n);
    if(f0*f1>0.0) {
        printf("Root does not exist");
        exit(0);
    }
    if(f1*f2>0.0) {
        printf("Root does not exist");
        exit(0);
    }
}
```

```
[1] f1=f(x1);
f2=f(x2);

if(f1*f2>0.0){
    printf("Root does not exist");
    exit(0);
}
else if(fabs(f1)<fabs(f2)){
    x0=x1;
}
else{
    x0=x2;
    do{
        f0=f(x0);
        g0=g(x0);
        x1=x0-(f0/g0);
        printf("Iteration number : %d", i);
        printf("Approximate Root : %f", x1);
        x0=x1;
        i=i+1;
    }
    while(fabs(x1-x0)>n);
}
```

Expt. No. 5

Date 11/10/23

Page No. 8

### Practical No-5

- Aim — Write a program to solve a given function using Newton Raphson Method.

- Description →

The Newton Raphson method uses below formula to find roots of the given equation.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- It is the best method to solve the non-linear equations.
  - The order of convergence is quadratic i.e. it makes the method comparatively faster.
1. Compute value of  $f(x)$  and derivative of  $f(x)$  (i.e.  $g(x)$ )
  2. Then put the values in the formula.

Teacher's Signature :

```
dot
    f0=(G0);
    y0=(G0);
    x1=x0-(f0/g0);
    printf("Intersection number %d\n", i);
    printf("Approximate Root of %d\n");
    z0=d;
    i=i+1;
    if(i>10)
        printf("Root converges");
        exit(0);
        printf("Failure of Function to converge\n");
    }
    while(fabs(f0/g0)>e);
    printf("Root %d Root is %f\n", x1);
}
return 0;
}

48:61
```

~~#Output~~

```
Enter intial guess: 0 1
Enter maximum iterations: 6
Iteration number:1      Approximate Root:0.453698
Iteration number:2      Approximate Root:0.510580
Iteration number:3      Approximate Root:0.510973
Hence Root is: 0.510973
Sukhmani Kaur CSE-1 00413202722
Enter intial guess: 1 2
Enter maximum iterations: 2
Wrong choice
Enter intial guess: 0 1
Enter maximum iterations: 2
Iteration number:1      Approximate Root:0.453698
Iteration number:2      Approximate Root:0.510580
Not convergent
Enter intial guess:
```

Program - 6

100

- Aim — Write a program to find the number of iterations and cost of solution of a given function by using Secant method.

Description -

The secant method is a root finding procedure in numerical analysis that uses a series of slopes of secant lines to better approximate a root of a function.

Steps -

Step 1 : Initialization  
x<sub>0</sub> and x<sub>1</sub> of  $\alpha$  are taken as initial guess

Step 2: Iteration  
In the case of  $n = 1, 2, 3, \dots$

$$x_{n+1} = x_n - f(x_n), \quad x_0 = x_{n-1}$$

$$f(x_n) - f(x_{n-1})$$

until a specific criterion of termination has been met (i.e. the desired accuracy of the answer or the maximum no. of iterations has been attained).

## Convergence

```

[ 8 ] ===== SECANT.SOU =====
scanf( " %f %f %f ", &x0, &x1 );
dot( );
f0=f(x0);
f1=f(x1);
x2=(x0*f1)-(x1*f0)/(f1-f0);
f2=f(x2);
x0=x1;
x1=x2;
i++;
printf( " Number of iterations = %d\n" , i );
printf( " Value of x2 = %f\n" , x2 );
printf( " Value of function f(x2) = %f\n" , f2 );
)
while(fabs(f2)>ε);
printf( " Value of function f(x2) = %f\n" , f2 );
return 0;
}

```

Sukhmani Kaur CSE-1 00413262722  
Enter value of x0 and x1: 0 1

Number of iterations:1 Root is:0.543044

Number of iterations:2 Root is:0.508093

Value of function is:-0.00595  
Value of function is:0.000923

Number of iterations:3 Root is:0.510986  
Sukhmani Kaur CSE-1 00413262722  
Enter value of x0 and x1: -

Number of iterations:2 Root is:0.509789

Number of iterations:1 Root is:0.543044

Value of function is:-0.00595  
Value of function is:0.000923

- Advantages
  - It converges quicker than a linear rate, making it more convergent than the bisection method.
  - It does not necessitate the usage of the function's derivative which is not available in number of application.
- Disadvantages
  - The secant method may not converge.
  - The computed iterates have no guaranteed error bounds.

Sukhmani Kaur CSE-1 00413262722  
Enter value of x0 and x1: 1 2

Number of iterations:1 Root is:0.211978

Value of function is:-0.577628  
Value of function is:0.214215

Number of iterations:2 Root is:0.622725

Value of function is:0.516968  
Value of function is:0.607296

Number of iterations:3 Root is:0.514926

Value of function is:0.000122  
Value of function is:-0.000122

Number of iterations:4 Root is:0.516968  
Sukhmani Kaur CSE-1 00413262722  
Enter value of x0 and x1: -

Program - 7

- Aim — Write a program to find integral of function using Trapezoidal Rule.

- Description —

In calculus, "Trapezoidal Rule" is one of the most important integration rules. The name trapezoidal is known when the area under the curve is evaluated, then the total area is divided into small trapezoids instead of rectangles. This rule is used for approximately the definite integrals where it uses the linear approximation of the function.

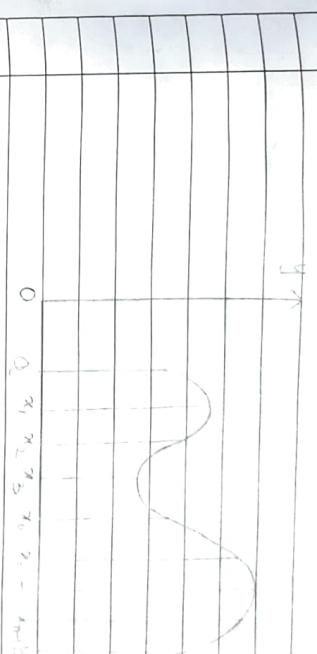
The trapezoidal rule is used mostly in the numerical analysis process. To evaluate the definite integrals, we can also use Riemann Sum, where we use small rectangles to evaluate the area under the curve.

Trapezoidal Rule is a rule that evaluates the area under the curves by dividing the total area into smaller trapezoids rather than using rectangles. This integration works by approximating the region under the graph of a function as a trapezoid and it calculates the area. This rule takes the average of the left and right sum. The trapezoidal rule does not give accurate value as Simpson's rule uses the quadratic approximation instead of linear approximation. Both Simpson's Rule and Trapezoidal Rule give the approximation value, but from Simpson's rule results in even more accurate approximation value of the integrals.

```
1:1 ━━━━ TRAPZIUD.C ━━━━ 1=[1:1]
int main()
{
    float a,b,f,a,fh,h,y,m, l = 0;
    int i,n;
    printf(" Enter the upper limit: ");
    scanf("%d", &n);
    printf(" Enter the lower limit: ");
    scanf("%d", &a);
    h=(b-a)/n;
    f=a;
    fh=f+b;
    m=f+a;
    for(i=1;i<=n-1;i++){
        l+=a+i*h;
        m+=f+2*i*h;
        f+=h;
    }
    l+=b*f;
    printf(" Definite Integration: %d", l);
}
```

```
1:1 ━━━━ TRAPZIUD.C ━━━━ 1=[1:1]
int main()
{
    float a,b,f,a,fh,h,y,m, l = 0;
    int i,n;
    printf(" Enter the upper limit: ");
    scanf("%d", &n);
    printf(" Enter the lower limit: ");
    scanf("%d", &a);
    h=(b-a)/n;
    f=a;
    fh=f+b;
    m=f+a;
    for(i=1;i<=n-1;i++){
        l+=a+i*h;
        m+=f+2*i*h;
        f+=h;
    }
    l+=b*f;
    printf(" Definite Integration: %d", l);
}
```

Enter lower limit integration: 1  
 Enter upper limit integration: 2  
 Enter number of sub-intervals: 7  
 Required value of integration is: 0.676863  
 Sukhmani Kaur CSE-1 60413262722



Let  $f(x)$  be a continuous function on the integral  $[a,b]$ . Now divide the intervals  $[a,b]$  into  $n$  equal subintervals with each of width  $\Delta x = \frac{b-a}{n}$ , such that  $a = x_0 < x_1 < x_2 < \dots < x_n = b$ .

Then the trapezoidal rule formula for area approximating the definite integrals →

$$\int_a^b f(x) dx = T_n = \frac{\Delta x}{2} [f(x_0) + 2\bar{f}(x_1) + f(x_2) + \dots + 2\bar{f}(x_{n-1})] + f(x_n)]$$

where  $x_i = a + i\Delta x$

If  $n \rightarrow \infty$ , RHS of the expression approaches the definite integral  $\int_a^b f(x) dx$ .

```
[1] 1.1 Line indicates directory
    The subdirectory
    file type is C code
    define f(x) (x*x - 10*x + 5)
    SIMPSUMU.C
    1-1:1]
```

```
int main()
{
    int i, n;
    float a, b, h, k, m, p = 0, q = 0;
    print("Enter the value of n: ");
    scanf("%d", &n);
    print("Enter the value of a: ");
    scanf("%f", &a);
    print("Enter the value of b: ");
    scanf("%f", &b);
    print("Enter the value of h: ");
    scanf("%f", &h);
    h = (b - a)/n;
    l = f(a) + f(b);
    for(i=1; i <= n-1; i++)
    {
        k = a + i * h;
        if((i%2 == 0))
            p = p + 4 * f(k);
        else
            p = p + 2 * f(k);
    }
}
```

```
[1] 1.1 SIMPSUMU.C
    1-1:1]
```

```
int main()
{
    int i, n;
    float a, b, h, k, m, p = 0, q = 0;
    print("Enter the value of n: ");
    scanf("%d", &n);
    print("Enter the value of a: ");
    scanf("%f", &a);
    print("Enter the value of b: ");
    scanf("%f", &b);
    print("Enter the value of h: ");
    scanf("%f", &h);
    h = (b - a)/n;
    l = f(a) + f(b);
    for(i=1; i <= n-1; i++)
    {
        k = a + i * h;
        if((i%2 == 0))
            p = p + 4 * f(k);
        else
            p = p + 2 * f(k);
    }
}
```

**Program-8**

- Aim** — Write a program to find integral of function using Simpson's 1/3 rule.
- Description** —

Simpson's rule is one of the numerical methods which is used to evaluate the definite integral. Usually, to find the definite integral, we use the fundamental theorem of calculus where we have to apply the antiderivative technique of integration. However, sometimes, it isn't easy to find the antiderivative of an integral line in scientific experiments, where the function has to be determined from the observed readings. Therefore, numerical methods are used to approximate the integral in such conditions. Other numerical methods used are trapezoidal rule, midpoint rule, left or right approximation using Riemann sums. Here we will use Simpson's 1/3 rule formula.

$$\int_a^b f(x) dx \approx S_n = \frac{\Delta x}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

where  $n$  is the even number  $\Delta x = (b-a)/n$  and  $x_i = a + i\Delta x$ . If we have  $f(x)=y$ , which is equally spaced between  $[a, b]$  and if  $a=x_0$ ,  $x_1=x_0+h$ ,  $x_2=x_0+2h \dots x_n=x_0+nh$ , where  $h$  is the difference between  $[a, b]$  and if  $a=x_0$  then we can say that  $y_0=f(x_0)$ ,  $y_1=f(x_1)$ ,  $y_2=f(x_2) \dots y_n=f(x_n)$  are the analogous values of  $y$  with each value of  $x$ .

# Output

Enter value of lower limit: 1  
Enter value of upper limit: 2  
Enter the number of sub-intervals: 6  
Required value of integration: 0.076311  
Sukhani Kaur CSE-1 00413202722

Expt. No. ....

Date .....  
Page No. 15

Formula

$$\int_a^b f(x) dx = \frac{h}{3} \left[ (y_0 + y_n) + 4(y_1 + y_3 + y_5 + \dots + y_{n-1}) + 2(y_2 + y_4 + y_6 + \dots + y_{n-2}) \right]$$

$$x_0 = a \quad x_1 = a+b \quad x_2 = b$$

Hence,

$$\frac{h}{3} \int_a^b f(x) dx \approx S_2 = \frac{h}{3} \left[ f(x_0) + 4f(x_1) + f(x_2) \right]$$

$$S_2 = \frac{h}{3} \left[ f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]$$

$h = \frac{b-a}{2}$ , This is the Simpson's 1/3 Rule for integration.

Program-9

- Aim — Write a program to find the integral of function using Simpson's 3/8 rule.

- Description —

Simpson's rule is one of the numerical methods which is used to evaluate the definite integral. Usually, to find the definite integral, we use the fundamental theorem of calculus where we have to apply the antiderivative techniques of integration. However, sometimes it isn't easy to find the antiderivative of an integral, like in scientific experiments, where the function has to be determined from the observed readings. Therefore, numerical methods are used to approximate the integral in such conditions. Other numerical methods used are trapezoidal rule, midpoint rule, left or right approximation using Riemann sums.

```

1.1 MATH.h
1.2 SIMPSON.C
1.3
1.4 #include <math.h>
1.5 #include <iostream.h>
1.6 #include <conio.h>
1.7
1.8 float f(float x)
1.9 {
2.0     return 3.14*x*x;
2.1 }
2.2
2.3 float simpson38(float a, float b, float n)
2.4 {
2.5     float h, sum = 0.0;
2.6     int i;
2.7
2.8     h = (b - a) / n;
2.9
3.0     for(i = 1; i <= n; i++)
3.1     {
3.2         if(i == 1 || i == n)
3.3             sum += 0.5 * f(a + i * h);
3.4         else if(i % 2 == 0)
3.5             sum += 2.0 * f(a + i * h);
3.6         else
3.7             sum += 3.0 * f(a + i * h);
3.8     }
3.9
3.10     return sum * h;
3.11 }
3.12
3.13
3.14
3.15
3.16
3.17
3.18
3.19
3.20
3.21
3.22
3.23
3.24
3.25
3.26
3.27
3.28
3.29
3.30
3.31
3.32
3.33
3.34
3.35
3.36
3.37
3.38
3.39
3.40
3.41
3.42
3.43
3.44
3.45
3.46
3.47
3.48
3.49
3.50
3.51
3.52
3.53
3.54
3.55
3.56
3.57
3.58
3.59
3.60
3.61
3.62
3.63
3.64
3.65
3.66
3.67
3.68
3.69
3.70
3.71
3.72
3.73
3.74
3.75
3.76
3.77
3.78
3.79
3.80
3.81
3.82
3.83
3.84
3.85
3.86
3.87
3.88
3.89
3.90
3.91
3.92
3.93
3.94
3.95
3.96
3.97
3.98
3.99
3.100
3.101
3.102
3.103
3.104
3.105
3.106
3.107
3.108
3.109
3.110
3.111
3.112
3.113
3.114
3.115
3.116
3.117
3.118
3.119
3.120
3.121
3.122
3.123
3.124
3.125
3.126
3.127
3.128
3.129
3.130
3.131
3.132
3.133
3.134
3.135
3.136
3.137
3.138
3.139
3.140
3.141
3.142
3.143
3.144
3.145
3.146
3.147
3.148
3.149
3.150
3.151
3.152
3.153
3.154
3.155
3.156
3.157
3.158
3.159
3.160
3.161
3.162
3.163
3.164
3.165
3.166
3.167
3.168
3.169
3.170
3.171
3.172
3.173
3.174
3.175
3.176
3.177
3.178
3.179
3.180
3.181
3.182
3.183
3.184
3.185
3.186
3.187
3.188
3.189
3.190
3.191
3.192
3.193
3.194
3.195
3.196
3.197
3.198
3.199
3.200
3.201
3.202
3.203
3.204
3.205
3.206
3.207
3.208
3.209
3.210
3.211
3.212
3.213
3.214
3.215
3.216
3.217
3.218
3.219
3.220
3.221
3.222
3.223
3.224
3.225
3.226
3.227
3.228
3.229
3.230
3.231
3.232
3.233
3.234
3.235
3.236
3.237
3.238
3.239
3.240
3.241
3.242
3.243
3.244
3.245
3.246
3.247
3.248
3.249
3.250
3.251
3.252
3.253
3.254
3.255
3.256
3.257
3.258
3.259
3.260
3.261
3.262
3.263
3.264
3.265
3.266
3.267
3.268
3.269
3.270
3.271
3.272
3.273
3.274
3.275
3.276
3.277
3.278
3.279
3.280
3.281
3.282
3.283
3.284
3.285
3.286
3.287
3.288
3.289
3.290
3.291
3.292
3.293
3.294
3.295
3.296
3.297
3.298
3.299
3.300
3.301
3.302
3.303
3.304
3.305
3.306
3.307
3.308
3.309
3.310
3.311
3.312
3.313
3.314
3.315
3.316
3.317
3.318
3.319
3.320
3.321
3.322
3.323
3.324
3.325
3.326
3.327
3.328
3.329
3.330
3.331
3.332
3.333
3.334
3.335
3.336
3.337
3.338
3.339
3.340
3.341
3.342
3.343
3.344
3.345
3.346
3.347
3.348
3.349
3.350
3.351
3.352
3.353
3.354
3.355
3.356
3.357
3.358
3.359
3.360
3.361
3.362
3.363
3.364
3.365
3.366
3.367
3.368
3.369
3.370
3.371
3.372
3.373
3.374
3.375
3.376
3.377
3.378
3.379
3.380
3.381
3.382
3.383
3.384
3.385
3.386
3.387
3.388
3.389
3.390
3.391
3.392
3.393
3.394
3.395
3.396
3.397
3.398
3.399
3.400
3.401
3.402
3.403
3.404
3.405
3.406
3.407
3.408
3.409
3.410
3.411
3.412
3.413
3.414
3.415
3.416
3.417
3.418
3.419
3.420
3.421
3.422
3.423
3.424
3.425
3.426
3.427
3.428
3.429
3.430
3.431
3.432
3.433
3.434
3.435
3.436
3.437
3.438
3.439
3.440
3.441
3.442
3.443
3.444
3.445
3.446
3.447
3.448
3.449
3.450
3.451
3.452
3.453
3.454
3.455
3.456
3.457
3.458
3.459
3.460
3.461
3.462
3.463
3.464
3.465
3.466
3.467
3.468
3.469
3.470
3.471
3.472
3.473
3.474
3.475
3.476
3.477
3.478
3.479
3.480
3.481
3.482
3.483
3.484
3.485
3.486
3.487
3.488
3.489
3.490
3.491
3.492
3.493
3.494
3.495
3.496
3.497
3.498
3.499
3.500
3.501
3.502
3.503
3.504
3.505
3.506
3.507
3.508
3.509
3.510
3.511
3.512
3.513
3.514
3.515
3.516
3.517
3.518
3.519
3.520
3.521
3.522
3.523
3.524
3.525
3.526
3.527
3.528
3.529
3.530
3.531
3.532
3.533
3.534
3.535
3.536
3.537
3.538
3.539
3.540
3.541
3.542
3.543
3.544
3.545
3.546
3.547
3.548
3.549
3.550
3.551
3.552
3.553
3.554
3.555
3.556
3.557
3.558
3.559
3.560
3.561
3.562
3.563
3.564
3.565
3.566
3.567
3.568
3.569
3.570
3.571
3.572
3.573
3.574
3.575
3.576
3.577
3.578
3.579
3.580
3.581
3.582
3.583
3.584
3.585
3.586
3.587
3.588
3.589
3.590
3.591
3.592
3.593
3.594
3.595
3.596
3.597
3.598
3.599
3.600
3.601
3.602
3.603
3.604
3.605
3.606
3.607
3.608
3.609
3.610
3.611
3.612
3.613
3.614
3.615
3.616
3.617
3.618
3.619
3.620
3.621
3.622
3.623
3.624
3.625
3.626
3.627
3.628
3.629
3.630
3.631
3.632
3.633
3.634
3.635
3.636
3.637
3.638
3.639
3.640
3.641
3.642
3.643
3.644
3.645
3.646
3.647
3.648
3.649
3.650
3.651
3.652
3.653
3.654
3.655
3.656
3.657
3.658
3.659
3.660
3.661
3.662
3.663
3.664
3.665
3.666
3.667
3.668
3.669
3.670
3.671
3.672
3.673
3.674
3.675
3.676
3.677
3.678
3.679
3.680
3.681
3.682
3.683
3.684
3.685
3.686
3.687
3.688
3.689
3.690
3.691
3.692
3.693
3.694
3.695
3.696
3.697
3.698
3.699
3.700
3.701
3.702
3.703
3.704
3.705
3.706
3.707
3.708
3.709
3.710
3.711
3.712
3.713
3.714
3.715
3.716
3.717
3.718
3.719
3.720
3.721
3.722
3.723
3.724
3.725
3.726
3.727
3.728
3.729
3.730
3.731
3.732
3.733
3.734
3.735
3.736
3.737
3.738
3.739
3.740
3.741
3.742
3.743
3.744
3.745
3.746
3.747
3.748
3.749
3.750
3.751
3.752
3.753
3.754
3.755
3.756
3.757
3.758
3.759
3.7510
3.7511
3.7512
3.7513
3.7514
3.7515
3.7516
3.7517
3.7518
3.7519
3.7520
3.7521
3.7522
3.7523
3.7524
3.7525
3.7526
3.7527
3.7528
3.7529
3.7530
3.7531
3.7532
3.7533
3.7534
3.7535
3.7536
3.7537
3.7538
3.7539
3.7540
3.7541
3.7542
3.7543
3.7544
3.7545
3.7546
3.7547
3.7548
3.7549
3.7550
3.7551
3.7552
3.7553
3.7554
3.7555
3.7556
3.7557
3.7558
3.7559
3.75510
3.75511
3.75512
3.75513
3.75514
3.75515
3.75516
3.75517
3.75518
3.75519
3.75520
3.75521
3.75522
3.75523
3.75524
3.75525
3.75526
3.75527
3.75528
3.75529
3.75530
3.75531
3.75532
3.75533
3.75534
3.75535
3.75536
3.75537
3.75538
3.75539
3.75540
3.75541
3.75542
3.75543
3.75544
3.75545
3.75546
3.75547
3.75548
3.75549
3.75550
3.75551
3.75552
3.75553
3.75554
3.75555
3.75556
3.75557
3.75558
3.75559
3.75560
3.75561
3.75562
3.75563
3.75564
3.75565
3.75566
3.75567
3.75568
3.75569
3.755610
3.755611
3.755612
3.755613
3.755614
3.755615
3.755616
3.755617
3.755618
3.755619
3.755620
3.755621
3.755622
3.755623
3.755624
3.755625
3.755626
3.755627
3.755628
3.755629
3.755630
3.755631
3.755632
3.755633
3.755634
3.755635
3.755636
3.755637
3.755638
3.755639
3.755640
3.755641
3.755642
3.755643
3.755644
3.755645
3.755646
3.755647
3.755648
3.755649
3.755650
3.755651
3.755652
3.755653
3.755654
3.755655
3.755656
3.755657
3.755658
3.755659
3.755660
3.755661
3.755662
3.755663
3.755664
3.755665
3.755666
3.755667
3.755668
3.755669
3.7556610
3.7556611
3.7556612
3.7556613
3.7556614
3.7556615
3.7556616
3.7556617
3.7556618
3.7556619
3.7556620
3.7556621
3.7556622
3.7556623
3.7556624
3.7556625
3.7556626
3.7556627
3.7556628
3.7556629
3.7556630
3.7556631
3.7556632
3.7556633
3.7556634
3.7556635
3.7556636
3.7556637
3.7556638
3.7556639
3.7556640
3.7556641
3.7556642
3.7556643
3.7556644
3.7556645
3.7556646
3.7556647
3.7556648
3.7556649
3.7556650
3.7556651
3.7556652
3.7556653
3.7556654
3.7556655
3.7556656
3.7556657
3.7556658
3.7556659
3.7556660
3.7556661
3.7556662
3.7556663
3.7556664
3.7556665
3.7556666
3.7556667
3.7556668
3.7556669
3.75566610
3.75566611
3.75566612
3.75566613
3.75566614
3.75566615
3.75566616
3.75566617
3.75566618
3.75566619
3.75566620
3.75566621
3.75566622
3.75566623
3.75566624
3.75566625
3.75566626
3.75566627
3.75566628
3.75566629
3.75566630
3.75566631
3.75566632
3.75566633
3.75566634
3.75566635
3.75566636
3.75566637
3.75566638
3.75566639
3.75566640
3.75566641
3.75566642
3.75566643
3.75566644
3.75566645
3.75566646
3.75566647
3.75566648
3.75566649
3.75566650
3.75566651
3.75566652
3.75566653
3.75566654
3.75566655
3.75566656
3.75566657
3.75566658
3.75566659
3.75566660
3.75566661
3.75566662
3.75566663
3.75566664
3.75566665
3.75566666
3.75566667
3.75566668
3.75566669
3.755666610
3.755666611
3.755666612
3.755666613
3.755666614
3.755666615
3.755666616
3.755666617
3.755666618
3.755666619
3.755666620
3.755666621
3.755666622
3.755666623
3.755666624
3.755666625
3.755666626
3.755666627
3.755666628
3.755666629
3.755666630
3.755666631
3.755666632
3.755666633
3.755666634
3.755666635
3.755666636
3.755666637
3.755666638
3.755666639
3.755666640
3.755666641
3.755666642
3.755666643
3.755666644
3.755666645
3.755666646
3.755666647
3.755666648
3.755666649
3.755666650
3.755666651
3.755666652
3.755666653
3.755666654
3.755666655
3.755666656
3.755666657
3.755666658
3.755666659
3.755666660
3.755666661
3.755666662
3.755666663
3.755666664
3.755666665
3.755666666
3.755666667
3.755666668
3.755666669
3.7556666610
3.7556666611
3.7556666612
3.7556666613
3.7556666614
3.7556666615
3.7556666616
3.7556666617
3.7556666618
3.7556666619
3.7556666620
3.7556666621
3.7556666622
3.7556666623
3.7556666624
3.7556666625
3.7556666626
3.7556666627
3.7556666628
3.7556666629
3.7556666630
3.7556666631
3.7556666632
3.7556666633
3.7556666634
3.7556666635
3.7556666636
3.7556666637
3.7556666638
3.7556666639
3.7556666640
3.7556666641
3.7556666642
3.7556666643
3.7556666644
3.7556666645
3.7556666646
3.7556666647
3.7556666648
3.7556666649
3.7556666650
3.7556666651
3.7556666652
3.7556666653
3.7556666654
3.7556666655
3.7556666656
3.7556666657
3.7556666658
3.7556666659
3.7556666660
3.7556666661
3.7556666662
3.7556666663
3.7556666664
3.7556666665
3.7556666666
3.7556666667
3.7556666668
3.7556666669
3.75566666610
3.75566666611
3.75566666612
3.75566666613
3.75566666614
3.75566666615
3.75566666616
3.75566666617
3.75566666618
3.75566666619
3.75566666620
3.75566666621
3.75566666622
3.75566666623
3.75566666624
3.75566666625
3.75566666626
3.75566666627
3.75566666628
3.75566666629
3.75566666630
3.75566666631
3.75566666632
3.75566666633
3.75566666634
3.75566666635
3.75566666636
3.75566666637
3.75566666638
3.75566666639
3.75566666640
3.75566666641
3.75566666642
3.75566666643
3.75566666644
3.75566666645
3.75566666646
3.75566666647
3.75566666648
3.75566666649
3.75566666650
3.75566666651
3.75566666652
3.75566666653
3.75566666654
3.75566666655
3.75566666656
3.75566666657
3.75566666658
3.75566666659
3.75566666660
3.75566666661
3.75566666662
3.75566666663
3.75566666664
3.75566666665
3.75566666666
3.75566666667
3.75566666668
3.75566666669
3.755666666610
3.755666666611
3.755666666612
3.755666666613
3.755666666614
3.755666666615
3.755666666616
3.755666666617
3.755666666618
3.755666666619
3.755666666620
3.755666666621
3.755666666622
3.755666666623
3.755666666624
3.755666666625
3.755666666626
3.755666666627
3.755666666628
3.755666666629
3.755666666630
3.755666666631
3.755666666632
3.755666666633
3.755666666634
3.755666666635
3.755666666636
3.755666666637
3.755666666638
3.755666666639
3.755666666640
3.755666666641
3.755666666642
3.755666666643
3.755666666644
3.755666666645
3.755666666646
3.755666666647
3.755666666648
3.755666666649
3.755666666650
3.755666666651
3.755666666652
3.755666666653
3.755666666654
3.755666666655
3.755666666656
3.755666666657
3.755666666658
3.755666666659
3.755666666660
3.755666666661
3.755666666662
3.755666666663
3.755666666664
3.755666666665
3.755666666666
3.755666666667
3.755666666668
3.755666666669
3.7556666666610
3.7556666666611
3.7556666666612
3.7556666666613
3.7556666666614
3.7556666666615
3.7556666666616
3.7556666666617
3.7556666666618
3.7556666666619
3.7556666666620
3.7556666666621
3.7556666666622
3.7556666666623
3.7556666666624
3.7556666666625
3.7556666666626
3.7556666666627
3.7556666666628
3.7556666666629
3.7556666666630
3.7556666666631
3.7556666666632
3.7556666666633
3.7556666666634
3.7556666666635
3.7556666666636
3.7556666666637
3.7556666666638
3.7556666666639
3.7556666666640
3.7556666666641
3.7556666666642
3.7556666666643
3.7556666666644
3.7556666666645
3.7556666666646
3.7556666666647
3.7556666666648
3.7556666666649
3.7556666666650
3.7556666666651
3.7556666666652
3.7556666666653
3.7556666666654
3.7556666666655
3.7556666666656
3.7556666666657
3.7556666666658
3.7556666666659

```



1<sup>st</sup> order Runge - kutta Method

$y_1 = y_0 + h f(x_0, y_0) = y_0 + hy_0$   $\because$  since  $y' = f(x, y)$   
 same as Euler's method.

2<sup>nd</sup> order Runge-kutta method

$$y_1 = y_0 + \frac{1}{2} (k_1 + k_2)$$

$$k_1 = hf(x_0, y_0) \quad k_2 = hf(x_0 + h, y_0 + k_1)$$

$$k_3 = hf(x_0 + h/2, y_0 + k_1/2); \quad k_4 = hf(x_0 + h, y_0 + k_3)$$

$$y_1 = y_0 + \frac{1}{6} (k_1 + 4k_2 + 3k_3 + 4k_4)/6$$

3<sup>rd</sup> order Runge - kutta Method

$$y_1 = y_0 + \left(\frac{1}{6}\right) (k_1 + 4k_2 + k_3)$$

$$\begin{aligned} \forall \quad k_1 &= hf(x_0, y_0) & k_2 &= hf[x_0 + (\frac{1}{2})h, y_0 + (\frac{1}{2})k_1] \\ k_3 &= hf[x_0 + h, y_0 + k_1] \text{ such that } k_1 = hf[x_0 + hy_0 + k_1] \end{aligned}$$

#Output

Enter the initial values of x and y: 0 1

Enter last value of x: 0.5

Enter step value of h: 0.1

X-value

Y-value

0.166666

1.000325

0.266666

1.002538

0.366666

1.008361

0.466666

1.019339

0.566666

1.036845

Sukhamani Kaur CSE-1

00413202722

#### 4<sup>th</sup> order

The Runge kutta method provides the approximate value of y for a given point x. Only the first order ODE's can be solved using the Runge kutta method.

$$y_1 = y_0 + \left(\frac{1}{6}\right) (k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = hf(x_0, y_0)$$

$$k_2 = hf\left(x_0 + \left(\frac{1}{2}h\right), y_0 + \left(\frac{1}{2}h\right)k_1\right)$$

$$k_3 = hf\left[x_0 + \left(\frac{1}{2}h\right)h, y_0 + \left(\frac{1}{2}h\right)k_2\right]$$

$$k_4 = hf\left[x_0 + h, y_0 + k_3\right]$$