NUMERICAL METHODS

1.BISECTION METHOD

```
# Defining Function
def f(x):
    return x**3-2*x-5


# Implementing Bisection Method
def bisection(x0,x1,e):
    step = 1
    print('\n\n*** BISECTION METHOD IMPLEMENTATION ***')
    condition = True
    while condition:
        x2 = (x0 + x1)/2
        print('Iteration-%d, x2 = %0.6f and f(x2) = %0.6f' % (step, x2,
f(x2)))

        if f(x0) * f(x2) < 0:
            x1 = x2
        else:
            x0 = x2

        step = step + 1
        condition = abs(f(x2)) > e

    print('\nRequired Root is : %0.8f' % x2)


# Input Section
x0 = input('First Guess: ')
x1 = input('Second Guess: ')
e = input('Tolerable Error: ')

# Converting input to float
x0 = float(x0)
x1 = float(x1)
e = float(e)

#Note: You can combine above two section like this
# x0 = float(input('First Guess: '))
# x1 = float(input('Second Guess: '))
# e = float(input('Tolerable Error: '))


# Checking Correctness of initial guess values and bisecting
if f(x0) * f(x1) > 0.0:
    print('Given guess values do not bracket the root.')
    print('Try Again with different guess values.')
else:
    bisection(x0,x1,e)
```

```
First Guess: 1
Second Guess: 3
Tolerable Error: 0.00001


*** BISECTION METHOD IMPLEMENTATION ***
Iteration-1, x2 = 2.000000 and f(x2) = -1.000000
Iteration-2, x2 = 2.500000 and f(x2) = 5.625000
Iteration-3, x2 = 2.250000 and f(x2) = 1.890625
Iteration-4, x2 = 2.125000 and f(x2) = 0.345703
Iteration-5, x2 = 2.062500 and f(x2) = -0.351318
Iteration-6, x2 = 2.093750 and f(x2) = -0.008942
Iteration-7, x2 = 2.109375 and f(x2) = 0.166836
Iteration-8, x2 = 2.101562 and f(x2) = 0.078562
Iteration-9, x2 = 2.097656 and f(x2) = 0.034714
Iteration-10, x2 = 2.095703 and f(x2) = 0.012862
Iteration-11, x2 = 2.094727 and f(x2) = 0.001954
Iteration-12, x2 = 2.094238 and f(x2) = -0.003495
Iteration-13, x2 = 2.094482 and f(x2) = -0.000771
Iteration-14, x2 = 2.094604 and f(x2) = 0.000592
Iteration-15, x2 = 2.094543 and f(x2) = -0.000090
Iteration-16, x2 = 2.094574 and f(x2) = 0.000251
Iteration-17, x2 = 2.094559 and f(x2) = 0.000081
Iteration-18, x2 = 2.094551 and f(x2) = -0.000004

Required Root is : 2.09455109
```

2.REGULA FALSI METHOD

```
# Defining Function
def f(x):
    return x**3-5*x+1


# Implementing False Position Method
def falsePosition(x0,x1,e):
    step = 1
    print('\n\n*** FALSE POSITION METHOD IMPLEMENTATION ***')
    condition = True
    while condition:
        x2 = x0 - (x1-x0) * f(x0)/( f(x1) - f(x0) )
        print('Iteration-%d, x2 = %0.6f and f(x2) = %0.6f' % (step, x2,
f(x2)))
```

```
        if f(x0) * f(x2) < 0:
            x1 = x2
        else:
            x0 = x2

        step = step + 1
        condition = abs(f(x2)) > e

    print('\nRequired root is: %0.8f' % x2)

  # Input Section
x0 = input('First Guess: ')
x1 = input('Second Guess: ')
e = input('Tolerable Error: ')
# Converting input to float
x0 = float(x0)
x1 = float(x1)
e = float(e)

# Checking Correctness of initial guess values and false positioning
if f(x0) * f(x1) > 0.0:
    print('Given guess values do not bracket the root.')
    print('Try Again with different guess values.')
else:
    falsePosition(x0,x1,e)
```

```
First Guess: 0
Second Guess: 1
Tolerable Error: 0.00001


*** FALSE POSITION METHOD IMPLEMENTATION ***
Iteration-1, x2 = 0.250000 and f(x2) = -0.234375
Iteration-2, x2 = 0.202532 and f(x2) = -0.004351
Iteration-3, x2 = 0.201654 and f(x2) = -0.000072
Iteration-4, x2 = 0.201640 and f(x2) = -0.000001

Required root is: 0.20163992
```

3.NEWTON RAPHSON METHOD

```
# Defining Function
def f(x):
    return x**2 -4*x -7
```

```
# Defining derivative of function
def g(x):
    return 2*x-4


# Implementing Newton Raphson Method
def newtonRaphson(x0,e,N):
    print('\n\n*** NEWTON RAPHSON METHOD IMPLEMENTATION ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        if g(x0) == 0.0:
            print('Divide by zero error!')
            break

        x1 = x0 - f(x0)/g(x0)
        print('Iteration-%d, x1 = %0.6f and f(x1) = %0.6f' % (step, x1,
f(x1)))
        x0 = x1
        step = step + 1

        if step > N:
            flag = 0
            break

        condition = abs(f(x1)) > e

    if flag==1:
        print('\nRequired root is: %0.8f' % x1)
    else:
        print('\nNot Convergent.')
x0 = input('Enter Guess: ')
e = input('Tolerable Error: ')
N = input('Maximum Step: ')
x0 = float(x0)
e = float(e)
N = int(N)
newtonRaphson(x0,e,N)
```

```
Enter Guess: 5
Tolerable Error: 0.00001
Maximum Step: 100


*** NEWTON RAPHSON METHOD IMPLEMENTATION ***
Iteration-1, x1 = 5.333333 and f(x1) = 0.111111
Iteration-2, x1 = 5.316667 and f(x1) = 0.000278
Iteration-3, x1 = 5.316625 and f(x1) = 0.000000
```

```
Required root is: 5.31662479
```

LINEAR SYSTEM OF ALGEBRIC EQUATIONS.

Direcr Method

1.GAUSS ELIMINATION METHOD

```python
# Importing NumPy Library
import numpy as np
import sys


n = int(input('Enter number of unknowns: '))
a = np.zeros((n,n+1))
x = np.zeros(n)
print('Enter Augmented Matrix Coefficients:')
for i in range(n):
    for j in range(n+1):
        a[i][j] = float(input( 'a['+str(i)+']['+ str(j)+']='))
for i in range(n):
    if a[i][i] == 0.0:
        sys.exit('Divide by zero detected!')

    for j in range(i+1, n):
        ratio = a[j][i]/a[i][i]

        for k in range(n+1):
            a[j][k] = a[j][k] - ratio * a[i][k]
x[n-1] = a[n-1][n]/a[n-1][n-1]

for i in range(n-2,-1,-1):
    x[i] = a[i][n]

    for j in range(i+1,n):
        x[i] = x[i] - a[i][j]*x[j]

    x[i] = x[i]/a[i][i]


print('\nRequired solution is: ')
for i in range(n):
    print('X%d = %0.2f' %(i,x[i]), end = '\t')
```

```
Enter number of unknowns: 3
Enter Augmented Matrix Coefficients:
```

```
a[0][0]=0
a[0][1]=2
a[0][2]=5
a[0][3]=7
a[1][0]=1
a[1][1]=-2
a[1][2]=2
a[1][3]=2
a[2][0]=3
a[2][1]=8
a[2][2]=7
a[2][3]=6
An exception has occurred, use %tb to see the full traceback.
```

2.GAUSS JORDAN METHOD

```
# Importing NumPy Library
import numpy as np
import sys


n = int(input('Enter number of unknowns: '))
a = np.zeros((n,n+1))
x = np.zeros(n)
print('Enter Augmented Matrix Coefficients:')
for i in range(n):
    for j in range(n+1):
        a[i][j] = float(input( 'a['+str(i)+']['+ str(j)+']='))
for i in range(n):
    if a[i][i] == 0.0:
        sys.exit('Divide by zero detected!')

    for j in range(n):
        if i != j:
            ratio = a[j][i]/a[i][i]

            for k in range(n+1):
                a[j][k] = a[j][k] - ratio * a[i][k]

for i in range(n):
    x[i] = a[i][n]/a[i][i]
print('\nRequired solution is: ')
for i in range(n):
    print('X%d = %0.2f' %(i,x[i]), end = '\t')
```

```
Enter number of unknowns: 3
Enter Augmented Matrix Coefficients:
a[0][0]=1
```

```
a[0][1]=1
a[0][2]=1
a[0][3]=4
a[1][0]=3
a[1][1]=-1
a[1][2]=3
a[1][3]=5
a[2][0]=3
a[2][1]=3
a[2][2]=2
a[2][3]=5

Required solution is:
X0 = -4.75     X1 = 1.75      X2 = 7.00
```

ITERATIVE METHODS

1.GAUSS JOCOBI ITERATIVE METHOD

```
 # Importing NumPy Library
 import numpy as np
 import sys
```

```
Enter number of unknowns: 3
Enter Augmented Matrix Coefficients:
a[0][0]=4
a[0][1]=1
a[0][2]=1
a[0][3]=1
a[1][0]=5
a[1][1]=2
a[1][2]=1
a[1][3]=2
a[2][0]=3
a[2][1]=2
a[2][2]=-6
a[2][3]=-4

Required solution is:
X0 = -0.32     X1 = 1.32      X2 = 0.95
```

2.GAUSS-SEIDEL ITERATIVE METHOD OR METHOD OF SUCCESSIVE DISPLACEMENT

```
Enter tolerable error: 0.00001

Count   x       y       z

1       0.8500  -1.0275 1.0109

2       1.0025  -0.9998 0.9998

3       1.0000  -1.0000 1.0000

4       1.0000  -1.0000 1.0000


Solution: x=1.000, y=-1.000 and z = 1.000
```

TRAPEZOIDAL RULE

```python
# Trapezoidal Method

# Define function to integrate
def f(x):
    return 1/(5+3*x)

# Implementing trapezoidal method
def trapezoidal(x0,xn,n):
    # calculating step size
    h = (xn - x0) / n

    # Finding sum
    integration = f(x0) + f(xn)

    for i in range(1,n):
        k = x0 + i*h
        integration = integration + 2 * f(k)

    # Finding final integration value
    integration = integration * h/2

    return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
```

```
result = trapezoidal(lower_limit, upper_limit, sub_interval)
print("Integration result by Trapezoidal method is: %0.6f" % (result) )
```

```
Enter lower limit of integration: 1
Enter upper limit of integration: 2
Enter number of sub intervals: 20
Integration result by Trapezoidal method is: 0.106156
```

SIMPSONDS 1/3 RULE

```
# Simpson's 1/3 Rule

# Define function to integrate
def f(x):
    return 1/(1+x)

# Implementing Simpson's 1/3
def simpson13(x0,xn,n):
    # calculating step size
    h = (xn - x0) / n

    # Finding sum
    integration = f(x0) + f(xn)

    for i in range(1,n):
        k = x0 + i*h

        if i%2 == 0:
            integration = integration + 2 * f(k)
        else:
            integration = integration + 4 * f(k)

    # Finding final integration value
    integration = integration * h/3

    return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = simpson13(lower_limit, upper_limit, sub_interval)
print("Integration result by Simpson's 1/3 method is: %0.6f" % (result) )
```

```
Enter lower limit of integration: 0
Enter upper limit of integration: 1
Enter number of sub intervals: 6
```

```
Integration result by Simpson's 1/3 method is: 0.693170
```

SIMPSONDS 3/8 RULE

```
# Simpson's 3/8 Rule

# Define function to integrate
def f(x):
    return 1/(1 + x**2)

# Implementing Simpson's 3/8
def simpson38(x0,xn,n):
    # calculating step size
    h = (xn - x0) / n

    # Finding sum
    integration = f(x0) + f(xn)

    for i in range(1,n):
        k = x0 + i*h

        if i%2 == 0:
            integration = integration + 2 * f(k)
        else:
            integration = integration + 3 * f(k)

    # Finding final integration value
    integration = integration * 3 * h / 8

    return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = simpson38(lower_limit, upper_limit, sub_interval)
print("Integration result by Simpson's 3/8 method is: %0.6f" % (result) )
```

```
Enter lower limit of integration: 0
Enter upper limit of integration: 1
Enter number of sub intervals: 6
Integration result by Simpson's 3/8 method is: 0.735877
```