ECE183: MATHEMATICS FOR ROBOTICS

CA3 - SIMULATION BASED ASSIGNMENT

1. NEWTON'S FORWARD INTERPOLATION METHOD
2. LAGRANGE'S INTERPOLATION METHOD
3. NEWTON-RAPHSON METHOD
4. SECANT METHOD
5. NEWTON'S DIVIDED DIFFERENCE METHOD
6. NEWTON'S BACKWARD INTERPOLATION METHOD

PYTHON LIBRARIES UTILIZED

In [4]:
```
import numpy as np
import math
```

Question 1

Design a model and find $\theta(0.5)$ using newton's forward interpolation method for the following data:

t	0	1	2	3	4
θ	1	7	23	55	109

Newton's Forward Difference Formula & Notes

$$f(x) = y_i + \frac{p\Delta^1 y_i}{1!} + \frac{p(p-1)\Delta^2 y_i}{2!} + \frac{p(p-1)(p-2)\Delta^3 y_i}{3!} + \frac{p(p-1)(p-2)(p-3)\Delta^4 y_i}{4!} + ... + \frac{p(p-1)(p-2)(p-3)...(p-(n-1))\Delta^n y_i}{n!}$$

Where, $p = \frac{x - x_i}{h}$ here ***x*** - is the value/point to find and ***h*** - is the step value from the **x** points. And ***i*** is the initial value.

Note: Δ - **Forward Difference Operator (Capital Delta)**

In [7]:
```
# Newton's Forward Difference Method

# calculating p mentioned in the formula
def pcal(p, n):
    temp = p
    for i in range(1, n):
        temp = temp * (p - i);
    return temp

# calculating factorial of given number n
def fact(n):
    f = 1
    for i in range(2, n + 1):
        f *= i
    return f

# Reading number of unknowns
n = int(input('Enter number of data points: '))

# Making numpy array of n & n x n size and initializing
# to zero for storing x and y value along with differences of y
x = np.zeros((n))
y = np.zeros((n,n))

# Reading data points
print('Enter values for x and y: ')
for i in range(n):
    x[i] = float(input('x['+str(i)+']='))
    y[i][0] = float(input('y['+str(i)+']='))

# Generating forward difference table
```

```
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i] = y[j+1][i-1] - y[j][i-1]


print("\n***** FORWARD DIFFERENCE TABLE *****\n");

for i in range(0,n):
    print('%0.3f' %(x[i]), end='')
    for j in range(0, n-i):
        print('\t\t%0.3f' %(y[i][j]), end='')
    print()

# Value to interpolate at
value = float(input('\nEnter the interpolation point: '))

# Initializing p and sum
sum = y[0][0]
p = (value - x[0]) / (x[1] - x[0])

for i in range(1,n):
    sum = sum + (pcal(p, i) * y[0][i]) / fact(i)

print("\nInterpolated value at", value, "is", round(sum, 6));
```

```
Enter number of data points: 5
Enter values for x and y:
x[0]=0
y[0]=1
x[1]=1
y[1]=7
x[2]=2
y[2]=23
x[3]=3
y[3]=55
x[4]=4
y[4]=109

***** FORWARD DIFFERENCE TABLE *****

0.000           1.000           6.000           10.000          6.000           0.000
1.000           7.000           16.000          16.000          6.000
2.000           23.000          32.000          22.000
3.000           55.000          54.000
4.000           109.000

Enter the interpolation point: 0.5

Interpolated value at 0.5 is 3.125
```

Question 2

Design model and find $x(5)$ using Lagrange interpolation method for the following data:

t	2	4	9	10
x	4	56	711	980

Lagrange's Interpolation Formula & Notes

$$f(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} * f(x_0) + \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} * f(x_1) + \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} * f(x_2) + \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} * f(x_3)$$

Where, **x** - is the value to find in the numerical interpolation points.

In [3]:
```
## Lagrange's Interpolation Method

# Reading number of values
n = int(input('Enter number of data points: '))

# Making numpy array of n & n x n size and initializing
# to zero for storing x and y value along with differences of y
x = np.zeros((n))
```

```
y = np.zeros((n))


# Reading data points
print('Enter values for x and y: ')
for i in range(n):
    x[i] = float(input('x['+str(i)+']='))
    y[i] = float(input('y['+str(i)+']='))

print("\n***** LAGRANGE'S INTERPOLATION TABLE *****\n");

for i in range(0,n):
    print('%0.3f' %(x[i]), end='')
    print('\t\t%0.3f' %(y[i]), end='')
    print()

# Reading interpolation point
value = float(input('\nEnter the interpolation point: '))

# Set interpolated value initially to zero
sum = 0

# Implementing Lagrange Interpolation
for i in range(n):
    p = 1
    for j in range(n):
        if i != j:
            p = p * (value - x[j])/(x[i] - x[j])
    sum = sum + p * y[i]

# Displaying output : we can change the decimal precision by modifying %.3f value to %.6f and so on
print('\nInterpolated value at %.3f is %.3f' % (value, sum))
```

```
Enter number of data points: 4
Enter values for x and y:
x[0]=2
y[0]=4
x[1]=4
y[1]=56
x[2]=9
y[2]=711
x[3]=10
y[3]=980

***** LAGRANGE'S INTERPOLATION TABLE *****

2.000           4.000
4.000           56.000
9.000           711.000
10.000          980.000

Enter the interpolation point: 5

Interpolated value at 5.000 is 115.000
```

Question 3

Find root of equation $x^3 + 2x^2 + x - 1$ Using Newton Raphson method.

In [1]:

```
# Newton-Raphson Method

# Defining Function
def f(x):
    return x**3 + 2*x**2 + x - 1

# Defining derivative of function
def g(x):
    return 3*x**2 + 4*x + 1

# Implementing Newton Raphson Method

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

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

```
        print("Iteration '%d', x%d = %0.6f and f(x%d) = %0.6f" % (step, step, x_iterate, step, f(x_iterate)))
        x0 = x_iterate
        step = step + 1

        if step > N:
            flag = 0
            break

        condition = abs(f(x_iterate)) > e

    if flag==1:
        print('\nRequired root is: %0.8f' % x_iterate)
    else:
        print('\nNot Convergent.')

# Input Section
e = 0.000001 # Default Error rate
x0 = float(input('Enter a Guess: '))
N = int(input('Maximum Step Size: '))
# e = float(input('Tolerable Error: '))

# Starting Newton Raphson Method
newtonRaphson(x0,e,N)
```

```
Enter a Guess: 1
Maximum Step Size: 10

***** NEWTON RAPHSON METHOD CALCULATIONS *****

Iteration '1', x1 = 0.625000 and f(x1) = 0.650391
Iteration '2', x2 = 0.485786 and f(x2) = 0.072402
Iteration '3', x3 = 0.465956 and f(x3) = 0.001352
Iteration '4', x4 = 0.465571 and f(x4) = 0.000001

Required root is: 0.46557137
```

Question 4

Find root of equation $x^3 - x - 1$ Using Secant method.

In [2]:

```
# Secant Method

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

# Implementing Secant Method

def secant(x0,x1,e,N):
    print('\n***** SECANT METHOD CALCULATIONS *****\n')
    step = 1
    condition = True
    while condition:
        if f(x0) == f(x1):
            print('Divide by zero error!')
            break

        x_iterate = x0 - (x1-x0)*f(x0)/( f(x1) - f(x0) )
        print("Iteration '%d', x%d = %0.6f and f(x%d) = %0.6f" % (step, step+1, x_iterate, step+1, f(x_iterate)))
        x0 = x1
        x1 = x_iterate
        step = step + 1

        if step > N:
            print('Not Convergent!')
            break

        condition = abs(f(x_iterate)) > e
    print('\nRequired root is: %0.8f' % x_iterate)


# Input Section
e = 0.000001 # Default Error rate
x0 = float(input('Enter 1st Guess: '))
x1 = float(input('Enter 2nd Guess: '))
N = int(input('Maximum Step Size: '))
# e = float(input('Tolerable Error: '))

# Starting Secant Method
secant(x0,x1,e,N)
```

```
Enter 1st Guess: 2
Enter 2nd Guess: 3
Maximum Step Size: 10

***** SECANT METHOD CALCULATIONS *****

Iteration '1', x2 = 1.722222 and f(x2) = 2.385974
Iteration '2', x3 = 1.574326 and f(x3) = 1.327642
Iteration '3', x4 = 1.388794 and f(x4) = 0.289842
Iteration '4', x5 = 1.336978 and f(x5) = 0.052884
Iteration '5', x6 = 1.325414 and f(x6) = 0.002969
Iteration '6', x7 = 1.324726 and f(x7) = 0.000034
Iteration '7', x8 = 1.324718 and f(x8) = 0.000000

Required root is: 1.32471796
```

Question 5

Design model and find $y(2.7)$ using Newton's divided difference interpolation method for the following data:

t	2	2.5	3
y	0.69315	0.91629	1.09861

Newton's Divided Difference Method & Notes

$$f(x) = [\, f(x_0) + (x - x_0)f(x_1, x_2) + (x - x_0)(x - x_1)f(x_0, x_1, x_2) + (x - x_0)(x - x_1)(x - x_2)f(x_0, x_1, x_2,\ x_4) + \{(x - x_0)(x - x_1)(x - x_2) * 0\ \}]$$

Where, **x** - is the value to find in the numerical interpolation points.

In [5]:
```
# Newton's Divided Difference Method

# Function to find the productduct term
def productTerm(i, value, x):
    product = 1;
    for j in range(i):
        product = product * (value - x[j]);
    return product;

# Function for calculating divided difference table
def createDDTable(x, y, n):
    for i in range(1, n):
        for j in range(n - i):
            y[j][i] = ((y[j][i - 1] - y[j + 1][i - 1]) / (x[j] - x[i + j]));
    return y;

# Function for calulcate Newton's divided difference using formula
def calculateValues(value, x, y, n):
    sum = y[0][0];
    for i in range(1, n):
        sum = sum + (productTerm(i, value, x) * y[0][i]);
    return sum;

# Function for displaying divided difference table
def showDDTable(y, n):
    for i in range(0,n):
        print('%0.3f' %(x[i]), end='')
        for j in range(0, n-i):
            print('\t\t%0.3f' %(y[i][j]), end='')
        print()

# Reading number of values for calculation
n = int(input('Enter number of data points: '))

# Making numpy array of n & n x n size and initializing to zero
# for storing x and y value along with differences of y
x = np.zeros((n))
y = np.zeros((n,n))

# Reading data points
print('Enter values for x and y: ')
for i in range(n):
```

```
    x[i] = float(input( 'x['+str(i)+']='))
    y[i][0] = float(input( 'y['+str(i)+']='))

# calculating divided difference table
y=createDDTable(x, y, n);

# displaying divided difference table
print("\n***** NETOWN's DIVIDED DIFFERENCE TABLE *****\n");
showDDTable(y, n);

# value to be interpolated
value = float(input('\nEnter the interpolation point: '))

# printing the value
print("\nInterpolated value at", value, "is", round(calculateValues(value, x, y, n), 6))
```

```
Enter number of data points: 3
Enter values for x and y:
x[0]=2
y[0]=0.69315
x[1]=2.5
y[1]=0.91629
x[2]=3
y[2]=1.09861

***** NETOWN's DIVIDED DIFFERENCE TABLE *****

2.000           0.693           0.446           -0.082
2.500           0.916           0.365
3.000           1.099

Enter the interpolation point: 2.7

Interpolated value at 2.7 is 0.994116
```

Question 6

Design a model and find $\theta(3.7)$ using newton's backward interpolation method for the following data:

t	0	1	2	3	4
θ	1	7	23	55	109

Newton's Backward Difference Formula & Notes

$$f(x) = y_n + \frac{p\nabla^1 y_n}{1!} + \frac{p(p+1)\nabla^2 y_n}{2!} + \frac{p(p+1)(p+2)\nabla^3 y_n}{3!} + \frac{p(p+1)(p+2)(p+3)\nabla^4 y_n}{4!} + ... + \frac{p(p+1)(p+2)(p+3)...(p+(n-1))\nabla^n y_n}{n!}$$

Where, $p = \frac{x - x_n}{h}$ here ***x*** - is the value/point to find and ***h*** - is the step value from the **x** points and ***n*** is the final value.

Note: ∇ - **Backward Difference Operator (Inverse of Capital Delta)**

In [6]:
```
# Newton's Backward Difference Method

# calculating p mentioned in the formula
def pcal(p, n):
    temp = p
    for i in range(1, n):
        temp = temp * (p + i);
    return temp

# calculating factorial of given number n
def fact(n):
    f = 1
    for i in range(2, n + 1):
        f *= i
    return f

# Reading number of unknowns
n = int(input('Enter number of data points: '))

# Making numpy array of n & n x n size and initializing to zero
# for storing x and y value along with differences of y
x = np.zeros((n))
```

```
y = np.zeros((n,n))


# Reading data points
print('Enter values for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+']='))
    y[i][0] = float(input( 'y['+str(i)+']='))

# Generating backward difference table
for i in range(1,n):
    for j in range(n-1,i-1,-1):
        y[j][i] = y[j][i-1] - y[j-1][i-1]


print('\n***** BACKWARD DIFFERENCE TABLE *****\n');

for i in range(0,n):
    print('%0.3f' %(x[i]), end='')
    for j in range(0,i+1):
        print('\t\t%0.3f' %(y[i][j]), end='')
    print()

# Value to interpolate at
value = float(input('\nEnter the interpolation point: '))

# Initializing p and sum
sum = y[n-1][0]

p = (value - x[n-1]) / (x[1] - x[0])

for i in range(1, n):
    sum = sum + (pcal(p, i) * y[n-1][i]) / fact(i)

print("\nInterpolated value at", value, "is", round(sum, 6));
```

```
Enter number of data points: 5
Enter values for x and y:
x[0]=0
y[0]=1
x[1]=1
y[1]=7
x[2]=2
y[2]=23
x[3]=3
y[3]=55
x[4]=4
y[4]=109

***** BACKWARD DIFFERENCE TABLE *****

0.000           1.000
1.000           7.000           6.000
2.000           23.000          16.000          10.000
3.000           55.000          32.000          16.000          6.000
4.000           109.000         54.000          22.000          6.000           0.000

Enter the interpolation point: 3.7

Interpolated value at 3.7 is 90.133
```

THANK YOU!!!