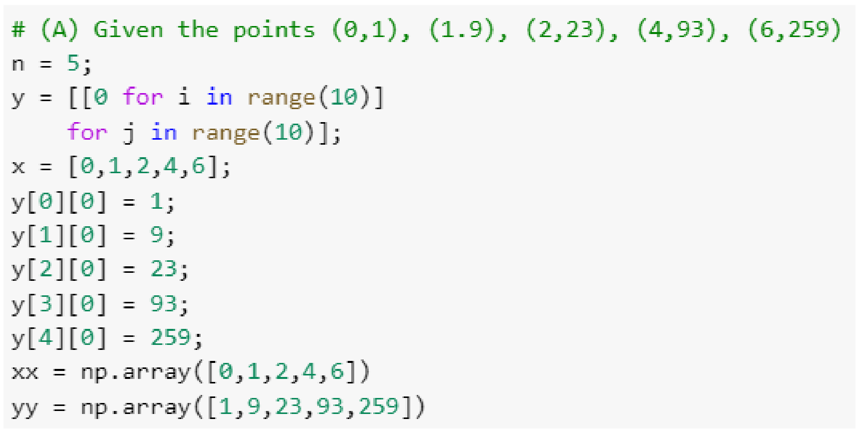
Ocean Lu
CS 3010.01
Professor Raheja
12/03/19

Project 3: Report

Please view the source code link:
https://colab.research.google.com/drive/1W4izIJbjjEY05pQO29momhW_j-gCOCeo

(A) Store the given points: (0,1), (1.9), (2,23), (4,93), (6,259):

a. Source code:



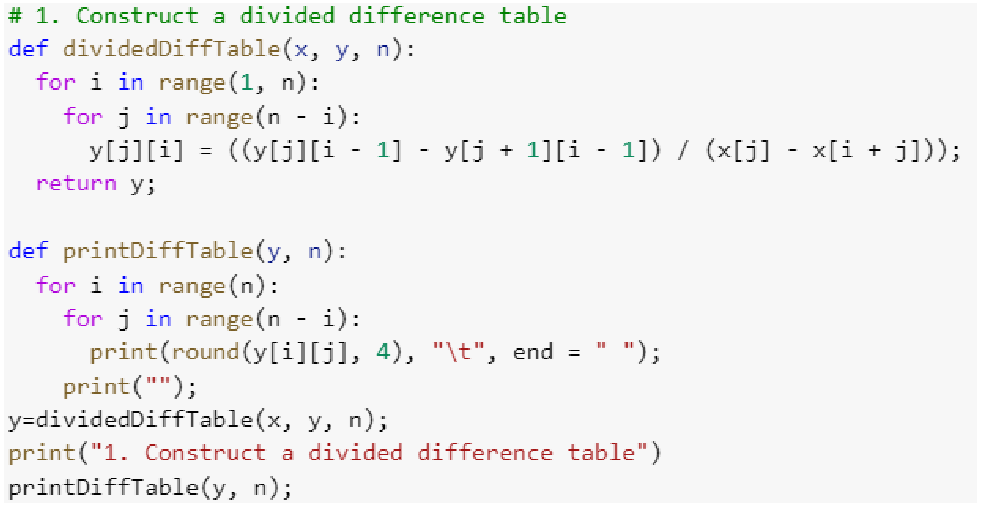
```
# (A) Given the points (0,1), (1.9), (2,23), (4,93), (6,259)
n = 5;
y = [[0 for i in range(10)]
    for j in range(10)];
x = [0,1,2,4,6];
y[0][0] = 1;
y[1][0] = 9;
y[2][0] = 23;
y[3][0] = 93;
y[4][0] = 259;
xx = np.array([0,1,2,4,6])
yy = np.array([1,9,23,93,259])
```

b. n is the number of inputs given, y[][] is used for the divided difference table where y[][0] is used as input, and xx is the x coordinates in space, while yy is f(x)

I had to construct a program to:

1. Construct a divided difference table

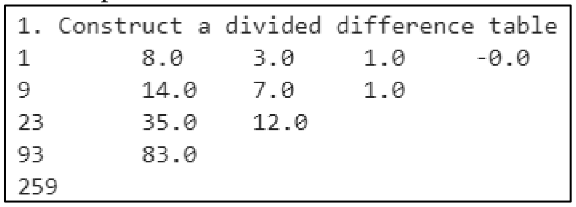
a. Source code:



```
# 1. Construct a divided difference table
def dividedDiffTable(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;

def printDiffTable(y, n):
  for i in range(n):
    for j in range(n - i):
      print(round(y[i][j], 4), "\t", end = " ");
    print("");
y=dividedDiffTable(x, y, n);
print("1. Construct a divided difference table")
printDiffTable(y, n);
```

b. Function dividedDiffTable is used to calculating the divided difference table, and function printDiffTable is used to display the divided difference table. I first calculate the divided difference table, stored in the variable y, and print it by calling the void function printDiffTable.

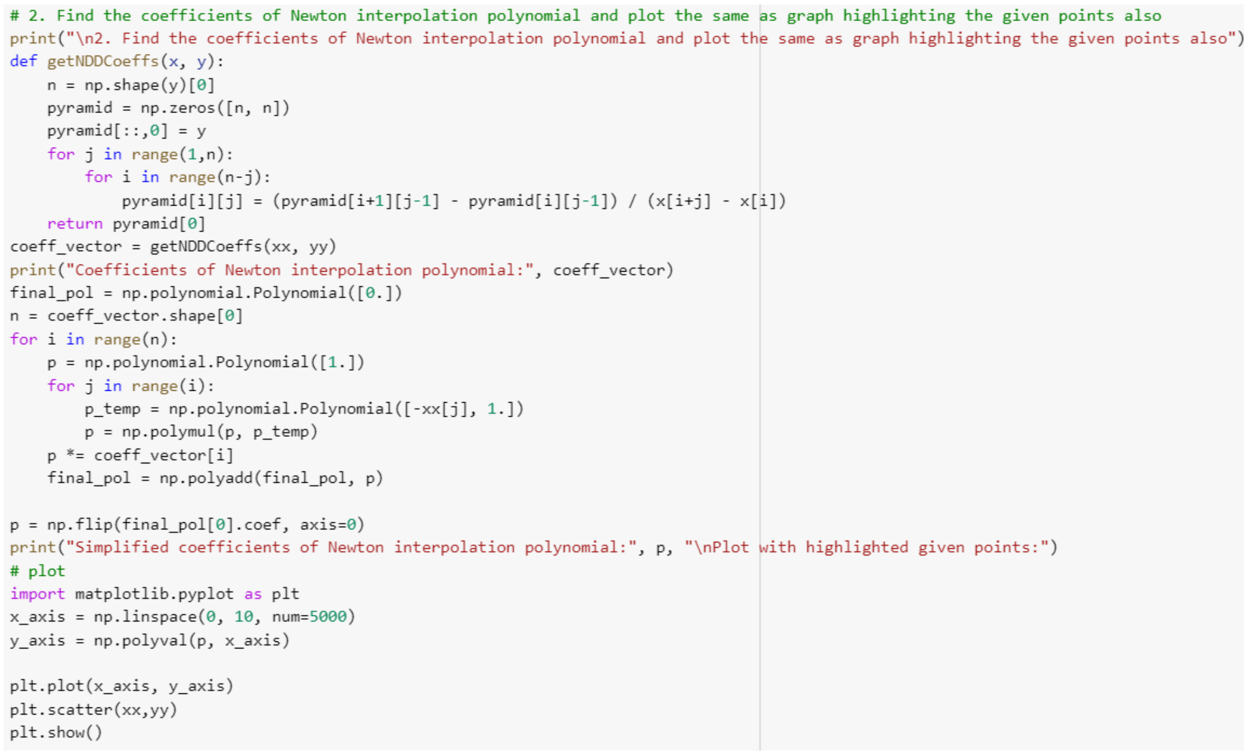
c. The output contains a divided difference table:



```
1. Construct a divided difference table
1         8.0     3.0     1.0     -0.0
9         14.0    7.0     1.0
23        35.0    12.0
93        83.0
259
```

2. Find the coefficients of Newton interpolation polynomial and plot the same as graph highlighting the given points also

a. Source code:

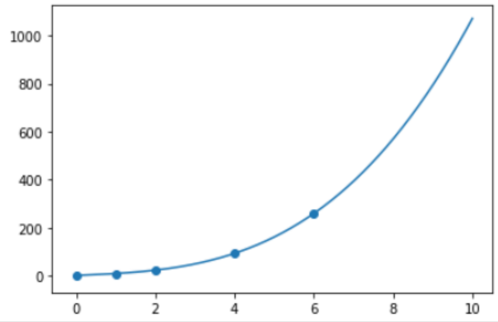


```
# 2. Find the coefficients of Newton interpolation polynomial and plot the same as graph highlighting the given points also
print("\n2. Find the coefficients of Newton interpolation polynomial and plot the same as graph highlighting the given points also")
def getNDDCoeffs(x, y):
    n = np.shape(y)[0]
    pyramid = np.zeros([n, n])
    pyramid[::,0] = y
    for j in range(1,n):
        for i in range(n-j):
            pyramid[i][j] = (pyramid[i+1][j-1] - pyramid[i][j-1]) / (x[i+j] - x[i])
    return pyramid[0]
coeff_vector = getNDDCoeffs(xx, yy)
print("Coefficients of Newton interpolation polynomial:", coeff_vector)
final_pol = np.polynomial.Polynomial([0.])
n = coeff_vector.shape[0]
for i in range(n):
    p = np.polynomial.Polynomial([1.])
    for j in range(i):
        p_temp = np.polynomial.Polynomial([-xx[j], 1.])
        p = np.polymul(p, p_temp)
    p *= coeff_vector[i]
    final_pol = np.polyadd(final_pol, p)

p = np.flip(final_pol[0].coef, axis=0)
print("Simplified coefficients of Newton interpolation polynomial:", p, "\nPlot with highlighted given points:")
# plot
import matplotlib.pyplot as plt
x_axis = np.linspace(0, 10, num=5000)
y_axis = np.polyval(p, x_axis)

plt.plot(x_axis, y_axis)
plt.scatter(xx,yy)
plt.show()
```

b. Function getNDDCoeffs creates a Newton Divided Difference pyramid and extracts the coefficients. The variable pyramid creates a square matrix to hold the pyramid and the first column is y. In the nested for loop, we create the pyramid by updating other columns. Once we've completed our pyramid, we return the first row, before simplifying, as it is our target polynomial. Because we want to find the resulting polynomial, we take in our target polynomial. n is the number of coefficients, and we create a dummy polynomial to store it into p. We know that each vector has a degree of i, and the terms are dependent on 'x' values, so we multiply the dummy. Then we apply our coefficient and add to the target polynomial. We receive the resulting polynomial once this nested for loop finishes and can now display both data. For plotting, we evaluate the polynomial at X axis and plot the data.

c. The output contained a polynomial in nested form and the graph:

```
2. Find the coefficients of Newton interpolation polynomial and plot the same as graph highlighting the given points also
Coefficients of Newton interpolation polynomial: [1. 8. 3. 1. 0.]
Simplified coefficients of Newton interpolation polynomial: [1. 0. 7. 1.]
Plot with highlighted given points:
```



3. Find the value of the function at X = 4.2

a. Source code:

```python
# 3. Find the value of the function at X = 4.2
def proterm(i, value, x):
  pro = 1;
  for j in range(i):
    pro = pro * (value - x[j]);
  return pro;

def applyFormula(value, x, y, n):
  sum = y[0][0];
  for i in range(1, n):
    sum = sum + (proterm(i, value, x) * y[0][i]);
  return sum;
print("\n3. Find the value of the function at X = 4.2")
value = 4.2;
print("X at", value, "is", applyFormula(value, x, y, n))
```

b. Function preterm is used to find the product term, and function applyFormula is used to applying Newton’s divided difference formula. The variable value is used to store the value to be interpolated, which is 4.2 in this case. We then print out the value received from calling it into the function.

c. The output contained the value of the function at X = 4.2

```
3. Find the value of the function at X = 4.2
X at 4.2 is 104.48800000000003
```