Lab 8 Report

Amlan Nayak MS18197

25th March 2021

1 Get a table of x and $sin(x)$ in range [0.10 , 0.50] in steps of 0.05 radians. Find the forward difference, backward difference and divided difference table

1.1 Algorithm & Discussion

The divided difference table works in the way shown below. You can successive differences in each column and dividing by the corresponding x to form a new column. Here the divided difference symbol is:

$$f[x_0, x_1, x_2...., x_n] = \sum_{i=0, i\neq j}^{n} \frac{f(x_i)}{\prod_{j=0,i\neq j}^{n}(x_i - x_j)}$$

x	$f(x)$	1st diff	2nd diff	3rd diff
x_0	f_0			
		$f[x_0, x_1]$		
x_1	f_1		$f[x_0, x_1, x_2]$	
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$
x_2	f_2		$f[x_1, x_2, x_3]$	
		$f[x_2, x_3]$		
x_3	f_3			

The forward difference table works in the way shown below. You take successive differences in each column to form a new column.

x	$f(x)$	Δf	$\Delta^2 f$	$\Delta^3 f$
x_0	f_0			
		$\Delta f_0 = f_1 - f_0$		
x_1	f_1		$\Delta^2 f_0 = \Delta f_1 - \Delta f_0$	
		$\Delta f_1 = f_2 - f_1$		$\Delta^3 f_0 = \Delta^2 f_1 - \Delta^2 f_0$
x_2	f_2		$\Delta^2 f_1 = \Delta f_2 - \Delta f_1$	
		$\Delta f_2 = f_2 - f_1$		
x_3	f_3			

The backward difference table works in a similar way as the forward difference table as shown below. You take successive differences in each column to form a new column.

x	$f(x)$	∇f	$\nabla^2 f$	$\nabla^3 f$
x_0	f_0			
		$\nabla f_1 = f_1 - f_0$		
x_1	f_1		$\nabla^2 f_2 = \nabla f_1 - \nabla f_0$	
		$\nabla f_2 = f_2 - f_1$		$\nabla^3 f_3 = \nabla^2 f_3 - \nabla^2 f_2$
x_2	f_2		$\nabla^2 f_3 = \nabla f_2 - \nabla f_1$	
		$\nabla f_3 = f_2 - f_1$		
x_3	f_3			

1.2 Code & Output

```python
import numpy as np
from sympy import *
x = symbols('x')
X = list(np.round(np.linspace(0.1,0.50, num=9),decimals=5))
F = [np.sin(i) for i in X]

Y = F
print("Forward Difference table:")
print("x=",X)
print("F(x)=",F)
for k in range(len(X)-1):
  Y = [np.round(Y[i+1]-Y[i],decimals=5) for i in range(len(Y)-1)]
  print("Difference number "+str(k+1)+" :", Y)

Y = F
print("\nBackward Difference table:")
print("x=",X)
print("F(x)=",F)
for k in range(len(X)-1):
  Y = [np.round((Y[i+1]-Y[i]),decimals=5) for i in range(len(Y)-1)]
  print("Difference number "+str(k+1)+" :", Y)

Y = F
print("\nDivided Difference table:")
print("x=",X)
print("F(x)=",F)
for k in range(len(X)-1):
  Y = [np.round((Y[i+1]-Y[i])/(X[i+k+1]-X[i]),decimals=5) for i in range(len(Y)-1)]
  print("Difference number "+str(k+1)+" :", Y)
```

Figure 1: Code for displaying all the difference tables vertically

```
Forward Difference table:
x= [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
F(x)= [0.09983341664682815, 0.14943813247359922, 0.19866933079506122, 0.24740395925452294
, 0.29552020666133955, 0.34289780745545134, 0.3894183423086505, 0.43496553411123023, 0.47
9425538604203]
Difference number 1 : [0.0496, 0.04923, 0.04873, 0.04812, 0.04738, 0.04652, 0.04555, 0.04
446]
Difference number 2 : [-0.00037, -0.0005, -0.00061, -0.00074, -0.00086, -0.00097, -0.0010
9]
Difference number 3 : [-0.00013, -0.00011, -0.00013, -0.00012, -0.00011, -0.00012]
Difference number 4 : [2e-05, -2e-05, 1e-05, 1e-05, -1e-05]
Difference number 5 : [-4e-05, 3e-05, 0.0, -2e-05]
Difference number 6 : [7e-05, -3e-05, -2e-05]
Difference number 7 : [-0.0001, 1e-05]
Difference number 8 : [0.00011]

Backward Difference table:
x= [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
F(x)= [0.09983341664682815, 0.14943813247359922, 0.19866933079506122, 0.24740395925452294
, 0.29552020666133955, 0.34289780745545134, 0.3894183423086505, 0.43496553411123023, 0.47
9425538604203]
Difference number 1 : [0.0496, 0.04923, 0.04873, 0.04812, 0.04738, 0.04652, 0.04555, 0.04
446]
Difference number 2 : [-0.00037, -0.0005, -0.00061, -0.00074, -0.00086, -0.00097, -0.0010
9]
Difference number 3 : [-0.00013, -0.00011, -0.00013, -0.00012, -0.00011, -0.00012]
Difference number 4 : [2e-05, -2e-05, 1e-05, 1e-05, -1e-05]
Difference number 5 : [-4e-05, 3e-05, 0.0, -2e-05]
Difference number 6 : [7e-05, -3e-05, -2e-05]
Difference number 7 : [-0.0001, 1e-05]
Difference number 8 : [0.00011]
```

Figure 2: Output of Forward and Backward Difference Tables

```
Divided Difference table:
x= [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
F(x)= [0.09983341664682815, 0.14943813247359922, 0.19866933079506122, 0.24740395925452294
, 0.29552020666133955, 0.34289780745545134, 0.3894183423086505, 0.43496553411123023, 0.47
9425538604203]
Difference number 1 : [0.99209, 0.98462, 0.97469, 0.96232, 0.94755, 0.93041, 0.91094, 0.8
892]
Difference number 2 : [-0.0747, -0.0993, -0.1237, -0.1477, -0.1714, -0.1947, -0.2174]
Difference number 3 : [-0.164, -0.16267, -0.16, -0.158, -0.15533, -0.15133]
Difference number 4 : [0.00665, 0.01335, 0.01, 0.01335, 0.02]
Difference number 5 : [0.0268, -0.0134, 0.0134, 0.0266]
Difference number 6 : [-0.134, 0.08933, 0.044]
Difference number 7 : [0.63809, -0.12951]
Difference number 8 : [-1.919]
```

Figure 3: Output Divided Difference Table

2 Interpolate the values at $sin(0.13)$, $sin(0.23)$, $sin(0.39)$ and $sin(0.47)$ using appropriate forward, backward and divided difference

2.1 Algorithm & Discussion

We can determine the interpolating polynomial for each of the tables mentioned in first problem. For divided difference table, the polynomial is given by:

$$f(x) = f(x_0) + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + ...+$$
$$(x - x_0)(x - x_1).......(x - x_n)f[x_0, x_1, ..., x_n]$$

For forward difference table, the polynomial is given by:

$$f(x) = f(x_0) + (x - x_0)\frac{\Delta f_0}{1!h} + (x - x_0)(x - x_1)\frac{\Delta^2 f_0}{2!h^2} + ...+$$
$$(x - x_0)(x - x_1)........(x - x_n)\frac{\Delta^n f_0}{n!h^n}$$

For backward difference table, the polynomial is given by:

$$f(x) = f(x_n) + (x - x_n)\frac{\nabla f(x_n)}{1!h} + (x - x_n)(x - x_{n-1})\frac{\nabla^2 f(x_n)}{2!h^2} + ...+$$

$$(x - x_n)(x - x_{n-1})........(x - x_1)\frac{\nabla^n f(x_n)}{n!h^n}$$

We had interpolate the values at $sin(0.13)$, $sin(0.23)$, $sin(0.39)$ and $sin(0.47)$ using appropriate forward, backward and divided difference.

First we interpolated using the divided difference polynomial for all points. We then choose $x = 0.13$ to be put in the forward difference polynomial as it is near the top of the data points.

Similarly, for $x = 0.47$, we chose the backward difference polynomial as it is near the end of the data points.

For all methods, the error percentage with the actual sin function was compared and displayed. I used sympy module in python to deal with the complicated algebraic expressions.

2.2 Code and Output

```python
import numpy as np
from sympy import *
x = symbols('x')
X = list(np.round(np.linspace(0.1,0.50, num=9),decimals=4))
F = [np.round(np.sin(i),decimals=4) for i in X]
exp= F[0]
Y=F
for k in range(len(X)-1):
  Y = [np.round((Y[i+1]-Y[i])/(X[i+k+1]-X[i]),decimals=4) for i in range(len(Y)-1)]
  sum=1
  for j in range(k+1):
    sum= (x-X[j])*sum
  exp = exp + sum*Y[0]
print("\nDivided Difference Interpolation Polynomial")
for i in [0.13,0.23,0.39,0.47]:
  print("\nThe value of Interpolation Polynomial at "+str(i)+":" ,exp.subs(x,i))
  print("The value of sine function at "+str(i)+":",np.sin(i))
  print("Error % between interpolation and actual value:",
        100*abs((exp.subs(x,i)-np.sin(i))/np.sin(i)))


exp= F[0]
Y=F
h=0.05
for k in range(len(X)-1):
  Y = [np.round(Y[i+1]-Y[i],decimals=4) for i in range(len(Y)-1)]
  sum=1
  for j in range(k+1):
    sum= ((x-X[j])*sum)
  exp = exp + sum*Y[0]/((factorial(j+1))*(h**(j+1)))
print("\nForward Difference Polynomial")
print("\nThe value of Interpolation Polynomial at 0.13: ",exp.subs(x,0.13))
print("The value of sine function at 0.13: ",np.sin(0.13))
print("Error % between interpolation and actual value:",
      100*abs((exp.subs(x,0.13)-np.sin(0.13))/np.sin(0.13)))
```

Figure 4: Code for finding the polynomial part1

```python

X=X[::-1]
F=F[::-1]
exp= F[0]
Y=F
h=0.05
for k in range(len(X)-1):
    Y = [np.round(-Y[i+1]+Y[i],decimals=4) for i in range(len(Y)-1)]
    sum=1
    for i in range(0,k+1):
        sum= (x-X[i])*sum
    exp = exp + (sum*Y[0])/ (factorial(i+1)*(h**(i+1)))
print("\nBackward Difference Polynomial")
print("\nThe value of Interpolation Polynomial at 0.47: ",exp.subs(x,0.47))
print("The value of sine function at 0.47: ",np.sin(0.47))
print("Error % between interpolation and actual value:",
            100*abs((exp.subs(x,0.47)-np.sin(0.47))/np.sin(0.47)))
```

Figure 5: Code for finding the polynomial part2

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab8$ pytho

Divided Difference Interpolation Polynomial

The value of Interpolation Polynomial at 0.13: 0.129562337165029
The value of sine function at 0.13: 0.12963414261969486
Error % between interpolation and actual value: 0.0553908509093296

The value of Interpolation Polynomial at 0.23: 0.227993217430688
The value of sine function at 0.23: 0.2279775235351884
Error % between interpolation and actual value: 0.00688396612799413

The value of Interpolation Polynomial at 0.39: 0.380175988707883
The value of sine function at 0.39: 0.3801884151231614
Error % between interpolation and actual value: 0.00326848867136825

The value of Interpolation Polynomial at 0.47: 0.452980366276261
The value of sine function at 0.47: 0.4528862853790683
Error % between interpolation and actual value: 0.0207736246889495

Forward Difference Polynomial

The value of Interpolation Polynomial at 0.13:  0.129562326528000
The value of sine function at 0.13:  0.12963414261969486
Error % between interpolation and actual value: 0.0553990563315823

Backward Difference Polynomial

The value of Interpolation Polynomial at 0.47:  0.452980266496000
The value of sine function at 0.47:  0.4528862853790683
Error % between interpolation and actual value: 0.0207515926107122
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab8$
```

Figure 6: Outputs for divided difference, forward difference and backward difference with the error percentages

3 Use cubic spline to estimate $f(2.5)$ from following table:

x	1	2	3	4	5
$f(x)$	30	15	32	18	25

3.1 Algorithm & Discussion

Here we use the formula derived in class for the cubic spline method to estimate $f(2.5)$.

$$f(x) = \frac{(x_{i+1}-x)^3}{6h} + \frac{(x-x_i)^3}{6h} + \frac{(x_{i+1}-x)}{h}(f_i - \frac{h^2}{6}M_i) + \frac{(x-x_i)}{h}(f_{i+1} - \frac{h^2}{6}M_{i+1})$$

Here h refers to the gap between successive points and $x_i and x_{i+1}$ refers to the points between which we have to interpolate to find the function using the cubic spline. Here $f_i and f_{i+1}$ refers to functional value at those those.

Here we get the value of M_i and M_{i+1} by solving linear equations given by:

$$M_{i-1} + 4M_i + M_{i+1} = \frac{6}{h^2}(f_{i-1} - 2f_i + f_{i+1})$$
$$i = 1, 2.....n-1, M_0 = 0, M_n = 0$$

In the given question, the point lies between 2 and 3. Hence we need to determine M_1 and M_2. We can then determine the interpolated polynomial between the two points and find the functional value at 2.5. I used sympy module in python to deal with the complicated algebraic expressions.

3.2 Code and Output

```
from sympy import *
X = [1,2,3,4,5]
F = [30,15,32,18,25]
h= X[1]-X[0]
m1,m2,m3,x = symbols('m1,m2,m3,x')
eq1= 4*m1 + m2 - 6*(F[0] - 2*F[1] + F[2])
eq2= m1 + 4*m2 + m3 - 6*(F[1] - 2*F[2] + F[3])
eq3= 4*m2 + m3 - 6*(F[2] - 2*F[3] + F[4])
M= list(solve([eq1,eq2,eq3],(m1,m2,m3)).values())
function = (((X[2] - x)**3)*M[0])/6  + (((x - X[1])**3)*M[1])/6 +
          (X[2] - x)*(F[1] - M[0]/6) + (x - X[1])*(F[2] - M[1]/6 )
print("x:",X)
print("f(x):",F)
print("The function through cubic spline method in interval (2,3) is "
        ,simplify(function))
print("The function's value at x=2.5 is",function.subs(x,2.5))
```

Figure 7: Code for cubic spline method

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab8$ python3 MS18197_8_code3.py
x: [1, 2, 3, 4, 5]
f(x): [30, 15, 32, 18, 25]
The function through cubic spline method in interval (2,3) is  292*x**3 - 1908*x**2 + 4009*x - 2707
The function's value at x=2.5 is -47.0000000000000
```

Figure 8: output for $f(2.5)$ by cubic spline method

4 Use Lagrange's technique to get $f(4.3)$ and also estimate x when $f(x) = 12$

x	1.2	2.1	2.8	4.1	4.9	6.2
$f(x)$	4.2	6.8	9.8	13.4	15.5	19.6

4.1 Algorithm & Discussion

For interpolating with Lagrange's method, we use the following relations derived in class:

$$w(x) = (x - x_0)(x - x_1)......(x - x_n)$$

Differentiating with respect to x and substituting x_i:

$$w^{'}(x) = (x_i - x_0)(x_i - x_1)......(x_i - x_{i-1})(x_i - x_{i+1})......(x_i - x_n)$$

$$l_i(x) = \frac{w(x)}{(x - x_i)w'(x)}$$

The Lagrange interpolating polynomial $P_n(x)$ is then determined by:

$$P_n(x) = l_0(x)f(x_0) + l_1(x)f(x_1) + ... + l_n(x)f(x_n)$$

After determining $P_n(x)$, we just substitute $x = 4.3$ to find the value of $P_n(4.3)$. As for determining the values of x for which $P_n(x) = 12$, I used the the code of Bairstow method from Lab4 to determine complex and real roots of the expression $P_n(x) - 12 = 0$. I have not put images of Bairstow code as it is redundant but I have included the outputs of the method. I used sympy module in python to deal with the complicated algebraic expressions.

4.2 Code & Output

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab8$ python3 MS18
197_8_code4.py
x: [1.2, 2.1, 2.8, 4.1, 4.9, 6.2]
f(x): [4.2, 6.8, 9.8, 13.4, 15.5, 19.6]
The langrange interpolation polynomial is:
-0.0511323110463686*x**5 + 0.990685523023259*x**4 - 7.22782440221614*x**3 +
 24.3805959155069*x**2 - 34.0164139915377*x + 20.4742672902267
Value of interpolation polynomial at x=4.3 is 13.8651221978976
Roots at f(x)=12 are:
[3.5019, 0.31316, 4.1032 + 1.9275*I, 4.1032 - 1.9275*I, 7.3536]
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab8$
```

Figure 9: Output for the Lagrange's technique. We determined the polynomial, found $P_n(4.3)$ and found the values of x for which $P_n(x) = 12$

```python
from sympy import *
x = symbols('x')
X = [1.2,2.1,2.8,4.1,4.9,6.2]
F = [4.2,6.8,9.8,13.4,15.5,19.6]
pol=0
for i in range(len(X)):
    a=1
    b=1
    for j in range(0,i):
        a = a*((x-X[j])/(X[i]-X[j]))
    for k in range(i+1,len(X)):
        b = b*((x-X[k])/(X[i]-X[k]))
    pol = pol + F[i]*a*b
print("x:",X)
print("f(x):",F)
print("The langrange interpolation polynomial is:")
print(simplify(pol))
print("Value of interpolation polynomial at x=4.3 is",pol.subs(x,4.3))

#Here we use the bairstow method from Lab4
roots=[]
coef = Poly(pol-12,x).all_coeffs()
if len(coef)%2==0:
```

Figure 10: Code for for the Lagrange's technique. I have not put pictures of the Bairstow code but it is in the python file.