Week 3 - Exploring the Derivative

November 21, 2024

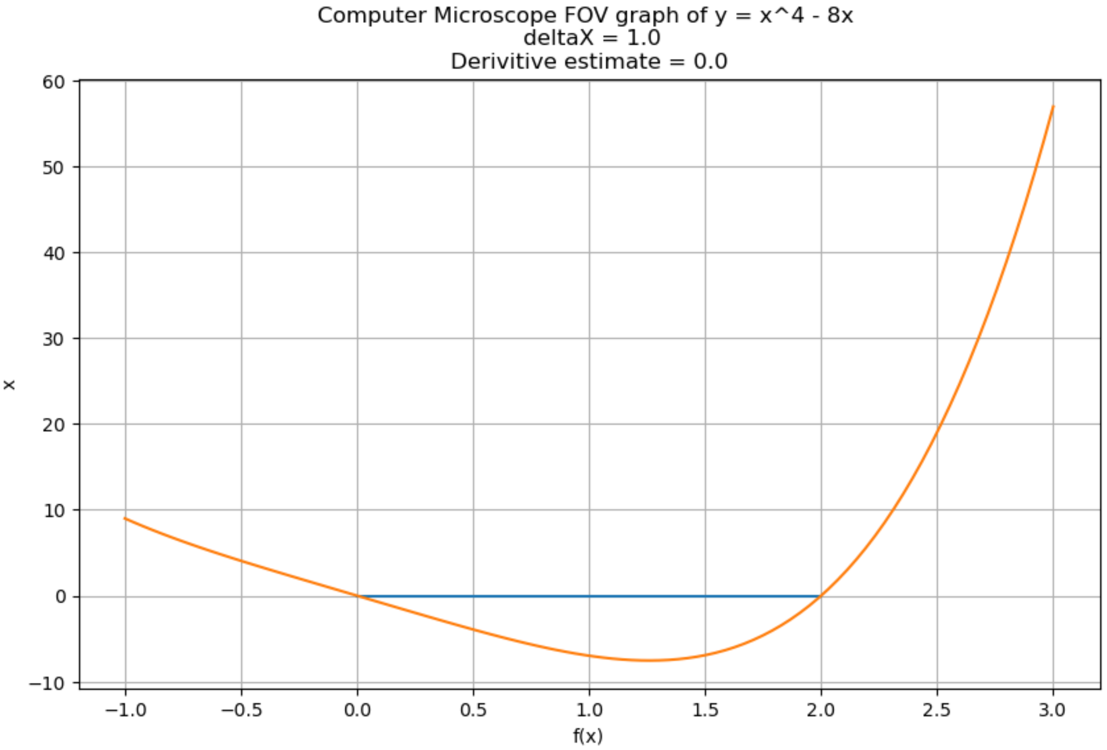
[1]:

```
import math
```

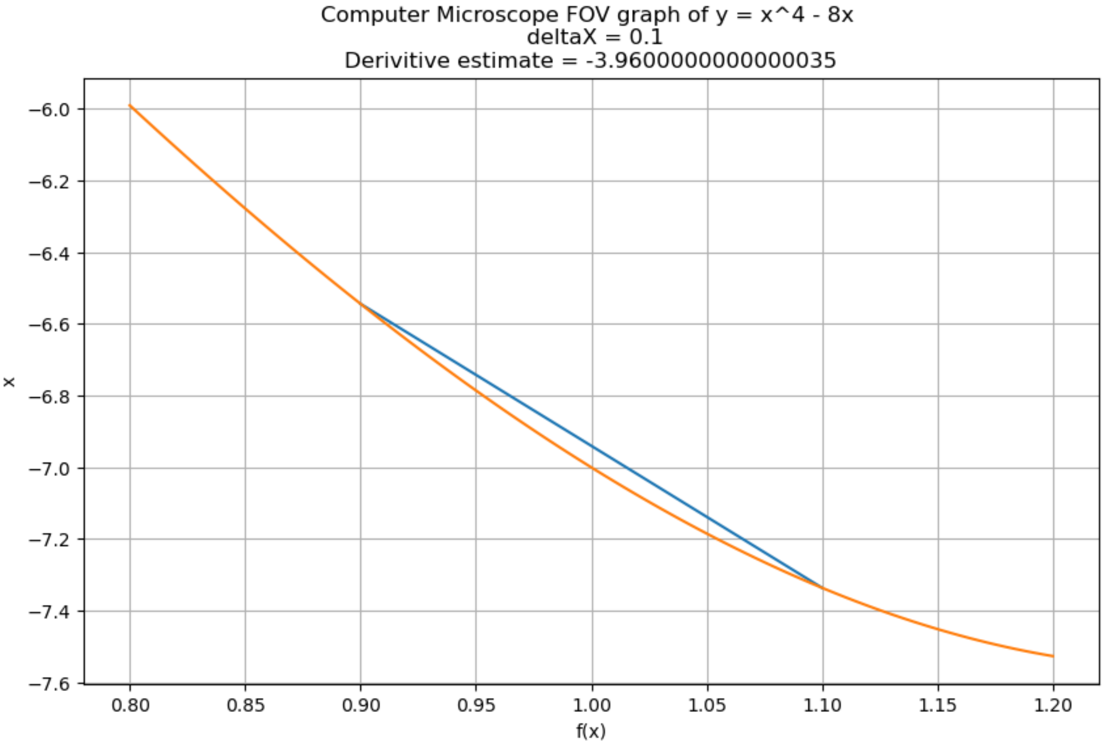
1 #2A Page 115

[2]:

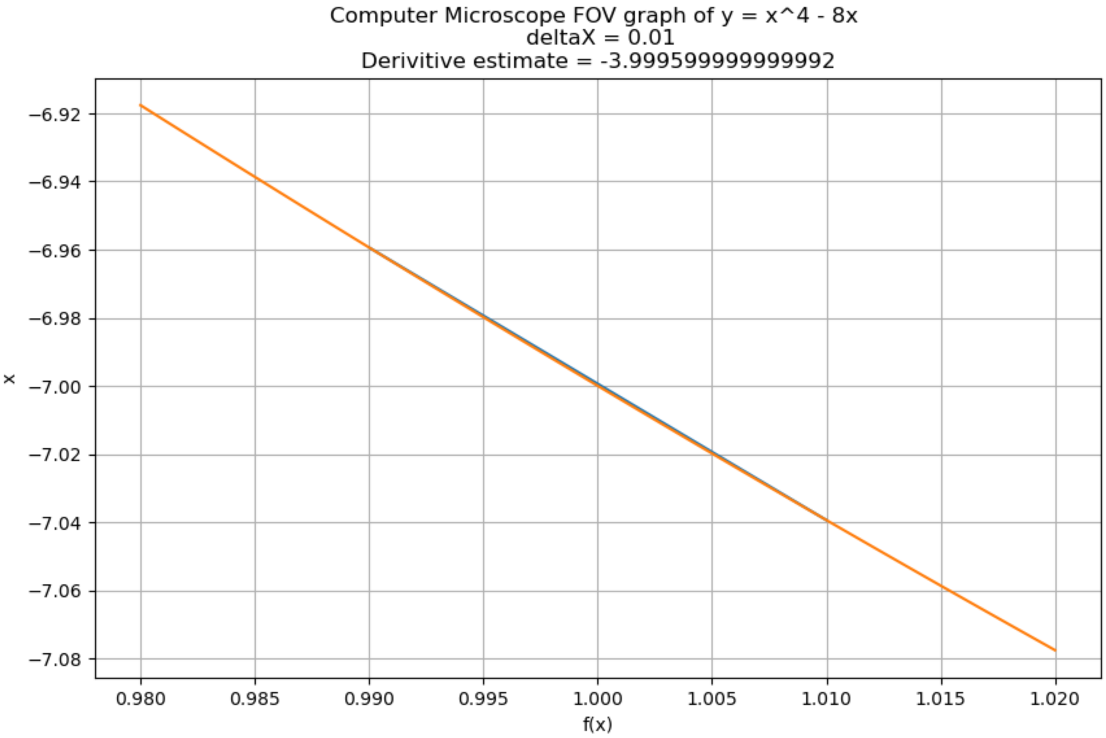
```
import numpy as np
import matplotlib.pyplot as plt

a=1 # define the point we want the value around.
h_vals = [1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001] #window sizes

def f(x):
    #calculating y for each x.
    return x ** 4 - 8 * x



def fprime(a, h):
    return (f(a+h)- f(a-h))/(2*h)

for h in h_vals:
    y = [f(a+h), f(a-h)]
    x = [(a+h),(a-h)]
    xrange = np.linspace((a+2*h),(a-2*h),100)
    #print ('fprime estimate FOV: ', h,' of x =', fprime(a, h))
    plt.figure(figsize=(10, 6))
    plt.plot(x,y)
    plt.plot(xrange,f(xrange))
    plt.xlabel('f(x)')
    plt.ylabel('x')
    plt.title('Computer Microscope FOV graph of y = x^4 - 8x \n deltaX = ' +␣
 ↪str(h) + "\nDerivitive estimate = " +str(fprime(a, h)))
    #plt.legend()
    plt.grid(True)
plt.show()
```

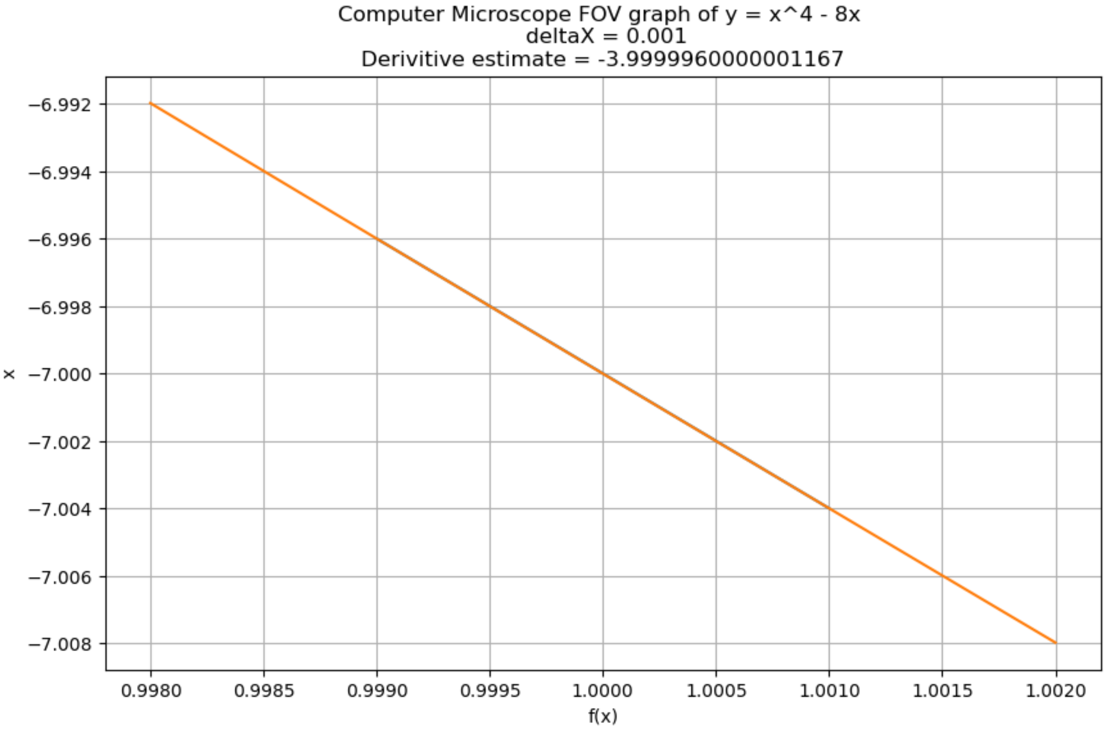


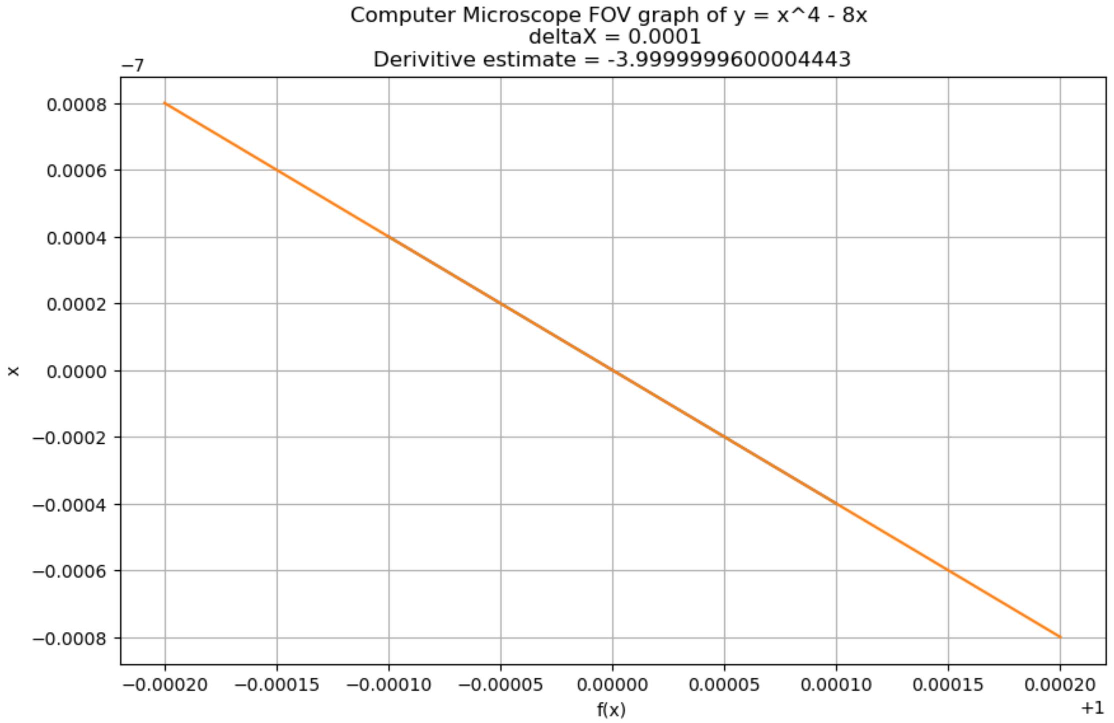
Computer Microscope FOV graph of y = x^4 - 8x
deltaX = 1.0
Derivitive estimate = 0.0

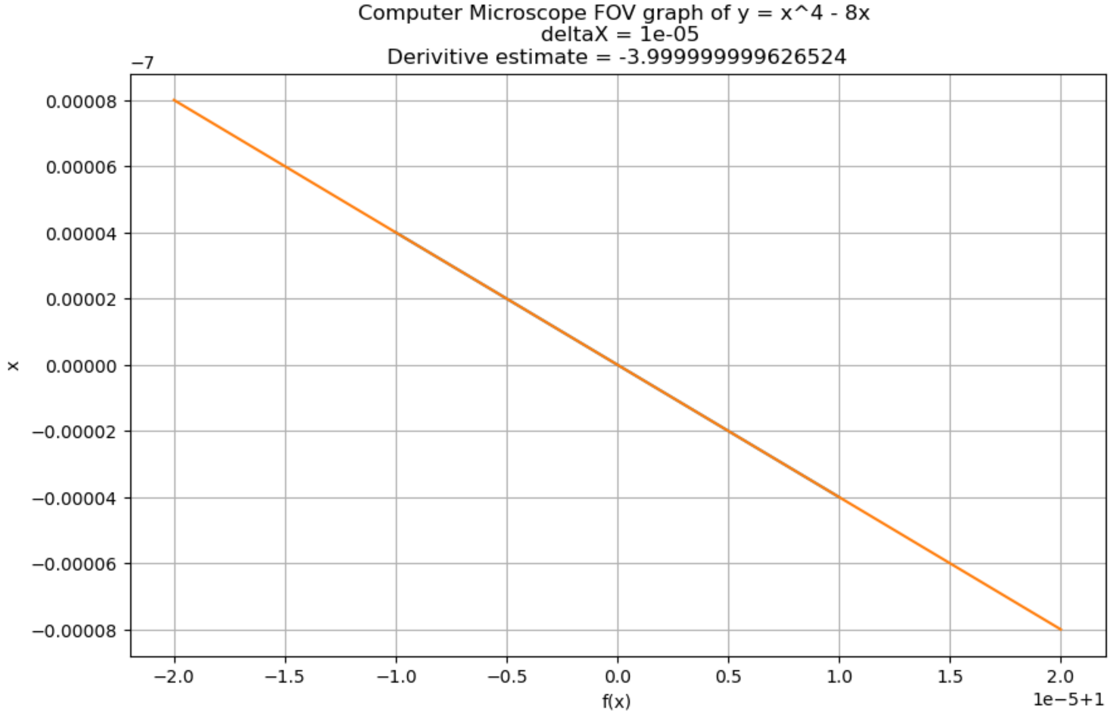


Computer Microscope FOV graph of y = x^4 - 8x
deltaX = 0.1
Derivitive estimate = -3.9600000000000035


Computer Microscope FOV graph of y = x^4 - 8x
deltaX = 0.01
Derivitive estimate = -3.999599999999992
x
−6.92
−6.94
−6.96
−6.98
−7.00
−7.02
−7.04
−7.06
−7.08
0.980
0.985
0.990
0.995
1.000
1.005
1.010
1.015
1.020
f(x)


Computer Microscope FOV graph of y = x^4 - 8x
deltaX = 0.001
Derivitive estimate = -3.9999960000001167
x
−6.992
−6.994
−6.996
−6.998
−7.000
−7.002
−7.004
−7.006
−7.008
0.9980
0.9985
0.9990
0.9995
1.0000
1.0005
1.0010
1.0015
1.0020
f(x)


Computer Microscope FOV graph of y = x^4 - 8x
deltaX = 0.0001
Derivitive estimate = -3.9999999600004443
−7
0.0008
0.0006
0.0004
0.0002
0.0000
−0.0002
−0.0004
−0.0006
−0.0008
x
−0.00020
−0.00015
−0.00010
−0.00005
0.00000
0.00005
0.00010
0.00015
0.00020
+1
f(x)


Computer Microscope FOV graph of y = x^4 - 8x
deltaX = 1e-05
Derivitive estimate = -3.999999999626524
−7
0.00008
0.00006
0.00004
0.00002
0.00000
−0.00002
−0.00004
−0.00006
−0.00008
x
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
1e−5+1
f(x)

2 pg 131 #2

[3]:

```
h_vals = [0.1, 0.01, 0.001, 0.0001, 0.00001]

x = 2

#1/x

print("a) Values for 1/x at x = 2")
for h in h_vals:
    q1 = (1/(x+h) - 2**(x-h))/(2*h)
    print(h, q1)
print()
#sin(7x)

x = 3

print("Values for sin(7x) at x=3")
for h in h_vals:
    q1 = (math.sin(7*(x+h)) - math.sin(7*(x-h)))/(2*h)
    print(h, q1)
print()
#x^3

x = 200

print("Values for x^3 at x = 200")
for h in h_vals:
    q1 = ((x+h)**3 - (x-h)**3)/(2*h)
    print(h, q1)
print()
#2^x
x = 5
print("Values for 2^x at x = 5")
for h in h_vals:
    q1 = ((1/2)**(x+h) - (1/2)**(x-h))/(2*h)
    print(h, q1)
```

```
a) Values for 1/x at x = 2
0.1 -16.279707449783764
0.01 -173.74287719685992
0.001 -1748.7391235121397
0.0001 -17498.738747433385
1e-05 -174998.73870981837

Values for sin(7x) at x=3
0.1 -3.528568772540893
```

```
0.01 -3.83097440301659
0.001 -3.8340735097894263
0.0001 -3.8341045084616665
1e-05 -3.8341048184897804

Values for x^3 at x = 200
0.1 120000.00999999233
0.01 120000.00009988435
0.001 120000.00000186265
0.0001 120000.00000465661
1e-05 120000.00006519257

Values for 2^x at x = 5
0.1 -0.021678198593669575
0.01 -0.0216610228432541
0.001 -0.02166085112701062
0.0001 -0.021660849409809585
1e-05 -0.02166084939175111
```

3 pg 131 #3 a:

[4]:

```python
# pg 131 #3 a:

a = 1
for k in range(9):
    h = 1/(2**k)
    q1 = ((a+h)**3 - (a-h)**3)/(2*h)
    q2 = ((a+h)**3 - a**3)/h
    print(k, h, q1, q2)
```

```
0 1.0 4.0 7.0
1 0.5 3.25 4.75
2 0.25 3.0625 3.8125
3 0.125 3.015625 3.390625
4 0.0625 3.00390625 3.19140625
5 0.03125 3.0009765625 3.0947265625
6 0.015625 3.000244140625 3.047119140625
7 0.0078125 3.00006103515625 3.02349853515625
8 0.00390625 3.0000152587890625 3.0117340087890625
```

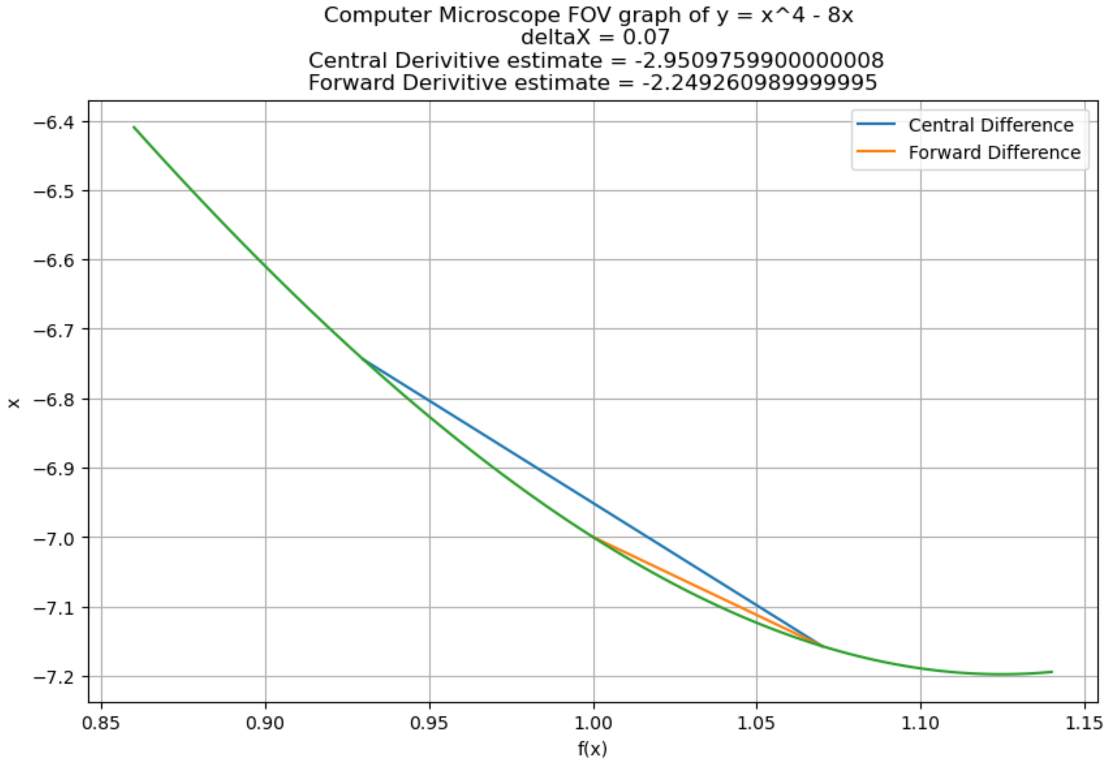
4 pg 131 #3 b:

Q1 stabilized to 4 decimal places while Q2 only stabilized to one decimal place after these iterations. # pg 131 #3 c: Q1 is a better estimator. It stabilized at k = 4 to a closer value than Q2 did at k = 8

5 pg 131 #3 d:

[5]:

```
a=1 # define the point we want the value around.
h_vals = [0.07] #window sizes

def f(x):
    #calculating y for each x.
    return x ** 5 - 8 * x



def fprime(a, h):
    return (f(a+h)- f(a-h))/(2*h)

def fprime1(a, h):
    return (f(a+h)- f(a))/(h)

for h in h_vals:
    y = [f(a+h), f(a-h)]
    x = [(a+h),(a-h)]
    x1 = [a+h, a]
    y1 = [f(a+h), f(a)]
    xrange = np.linspace((a+2*h),(a-2*h),100)
    #print ('fprime estimate FOV: ', h,' of x =', fprime(a, h))
    plt.figure(figsize=(10, 6))
    plt.plot(x,y,label="Central Difference")
    plt.plot(x1,y1,label="Forward Difference")
    plt.plot(xrange,f(xrange))
    plt.xlabel('f(x)')
    plt.ylabel('x')
    plt.title('Computer Microscope FOV graph of y = x^4 - 8x \n deltaX = ' +␣
 ↪str(h) + "\n Central Derivitive estimate = " +str(fprime(a, h))
              + "\nForward Derivitive estimate = " + str(fprime1(a,h)))
    plt.legend()
    plt.grid(True)
plt.show()
```



The above graph shows why the central difference is more likely to be accurate because by taking a value in front and after the point, the change in curve is averaged out to be closer to the tangential line at the point. If only going from the point forward, the slope will be biased towards the direction the function is curving even as the h value decreases. Visually we can see this by noting how close the Central Difference line appears to be tangential to the curve at x = 1 while the Forward difference originates from this point and therefore can't be exactly tangential unless the line is actually linear at that point and forward.

6 pg 131 #4 a:

[6]:

```python
a = 9
for k in range(9):
    h = 1/(2**k)
    q1 = (math.sqrt(a+h) - math.sqrt(a-h))/(2*h)
    q2 = (math.sqrt(a+h) - math.sqrt(a))/h
    print(k, h, q1, q2)
```

```
0 1.0 0.16692526771109462 0.16227766016837952
1 0.5 0.16673105406183764 0.16441400296897601
2 0.25 0.1666827471986032 0.16552506059643868
3 0.125 0.16667068578158784 0.16609194718914466
4 0.0625 0.16666767138179495 0.1663783151691831
```

```
5 0.03125 0.16666691784147503 0.16652224137045835
6 0.015625 0.1666667294601183 0.16659439142901533
7 0.0078125 0.16666668236501891 0.16663051337502566
8 0.00390625 0.16666667059126894 0.16664858609942712
```

pg 131 #4 b: In this table Q1 stabilized to 6 digits and Q2 stabilized to 3 digits.

pg 131 #4 c: Q1 is still the better estimator. In this case Q1 stabilizes in K = 2 to the same accuracy as the Q2 does at k = 8.

7 pg 132 #6

[7]:

```
h_vals = [0.1, 0.01, 0.001, 0.0001, 0.00001]

a = 0

#2^x

print("Values for 2^x")
for h in h_vals:
    q1 = (2**(a+h) - 2**(a-h))/(2*h)
    print(h, q1)
print()
#3^x

print("Values for 3^x")
for h in h_vals:
    q1 = (3**(a+h) - 3**(a-h))/(2*h)
    print(h, q1)
print()
#10^x

print("Values for 10^x")
for h in h_vals:
    q1 = (10**(a+h) - 10**(a-h))/(2*h)
    print(h, q1)
print()
#(1/2)^x

print("Values for (1/2)^x")
for h in h_vals:
    q1 = ((1/2)**(a+h) - (1/2)**(a-h))/(2*h)
    print(h, q1)
```

```
Values for 2^x
0.1 0.6937023549974286
0.01 0.6931527309841479
0.001 0.6931472360641178
```

```
0.0001 0.6931471811144618
1e-05 0.69314718055824

Values for 3^x
0.1 1.100823570965711
0.01 1.098634388284142
0.001 1.0986125096629218
0.0001 1.0986122908784868
1e-05 1.0986122886857963

Values for 10^x
0.1 2.322985885349429
0.01 2.3027885662471714
0.001 2.3025871276731724
0.0001 2.302585113340694
1e-05 2.302585093194587

Values for (1/2)^x
0.1 -0.6937023549974286
0.01 -0.6931527309841479
0.001 -0.6931472360641178
0.0001 -0.6931471811144618
1e-05 -0.69314718055824
```

2^x has accuracy to 8 decimal places. 3^x to 7, 10^x to 6, and (1/2)^x to 8. It makes sense that the larger the value of the base the more rapidly the derivative value is changing and the smaller the value of h required to get higher accuracy.