Week4

September 24, 2024

1 Week 4 Activities:

09-25-24

1.1 1. Your goal is to create a function or code snippet you can use to find the slope at a point of any function numerically. Return to last week's activity, #3, page 131 and make the following modifications:

- Rename this function or routine "Numerical_Point_Derivative"
- Choose a small h to use. "h = …"
- Add a line to enter an algebraic function, like x^2, or sin(x), or 3x^4 - 2x.
- Add a line "a = …" to enter the x value where you want to find the slope.
- Choose one of the 2 methods to calculate the slope of a function at a single point.
- Add a code comment to tell me why you chose the formula you did.
- Return the slope/rate of change/derivative of the function at point a.
- Test it out with function/point examples from #10 page 133.

Work from Last Week:

[3]:
```
from tabulate import tabulate
def Q1(f, a, h):
    xp = f(a+h)
    xm = f(a-h)
    return (xp-xm)/(2*h)

def Q2(f, a, h):
    xp = f(a+h)
    xa = f(a)
    return (xp-xa)/h

def f3a(x):
    return x**3

a = 1
tab = [['h', 'Q1', 'Q2']]
g_tab = []
for k in range(0, 8):
    h = 0.5**k
    q1 = Q1(f3a, a, h)
```

```
    q2 = Q2(f3a, a, h)
    tab.append([h, q1, q2])
    g_tab.append([h, q1, q2])

print(tabulate(tab))
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[3], line 1
----> 1 from tabulate import tabulate
      2 def Q1(f, a, h):
      3     xp = f(a+h)

ModuleNotFoundError: No module named 'tabulate'
```

Single Point Derivative: I am using the Q1 method shown above to calculate the derivative at the point based on the exercises we did last week, which showed that the Q1 method gave a more accurate value of the slope at a lower x-interval h.

[4]:
```
def Numerical_Point_Derivative(f, h, a):
    xp = f(a+h)
    xm = f(a-h)
    return (xp-xm)/(2*h) #Using q1 method
```

[5]:
```
# from #3, p 131: f(x) = x^3, a = 1, h not given so I am using 0.005

def f1(x):
    return x**3

h = 0.005
a = 1
xp = Numerical_Point_Derivative(f1, h, a)

print(f'the slope of f(x) at {a} is {xp:.4f}')
```

```
the slope of f(x) at 1 is 3.0000
```

Examples from #10

- a) f (x) = 1/x at x = 2.
- b) f (x) = sin(7x) at x = 3.
- c) f (x) = x^3 at x = 200.
- d) f (x) = 2^x at x = 5.

[6]:
```
import numpy as np
def f1a(x):
    return 1/x
def f1b(x):
    return np.sin(7*x)
def f1c(x):
    return x**3
def f1d(x):
    return 2**x

print(f'The slope of Equation a at x = 2 is {Numerical_Point_Derivative(f1a, h,␣
↪2):.4f}')
print(f'The slope of Equation b at x = 3 is {Numerical_Point_Derivative(f1b, h,␣
↪3):.4f}')
print(f'The slope of Equation c at x = 200 is {Numerical_Point_Derivative(f1c,␣
↪h, 200):.4f}')
print(f'The slope of Equation d at x = 5 is {Numerical_Point_Derivative(f1d, h,␣
↪5):.4f}')
```

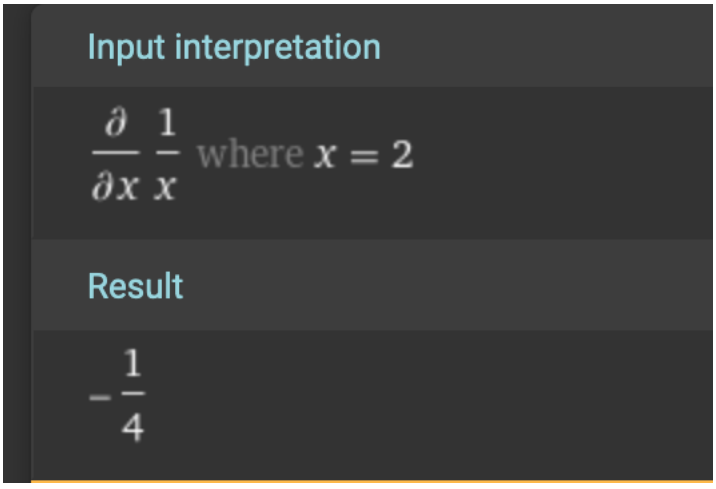
```
The slope of Equation a at x = 2 is -0.2500
The slope of Equation b at x = 3 is -3.8333
The slope of Equation c at x = 200 is 120000.0000
The slope of Equation d at x = 5 is 22.1808
```

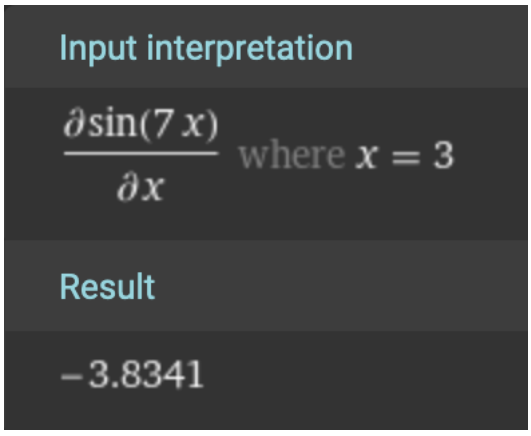
1.2 2. Numerical Derivative Calculator

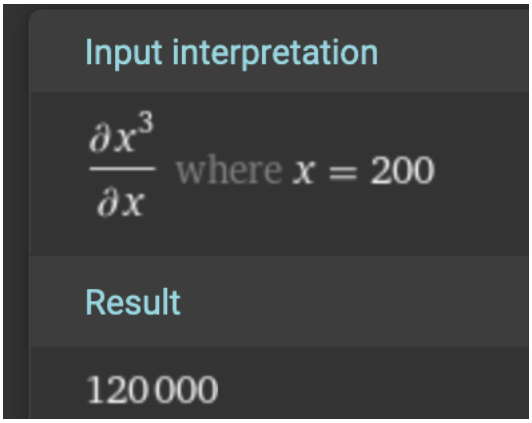
Search the internet for a "Numerical Derivative calculator." Compare your results with the tool you found. How did you do? *Extra Optional Opportunity – can you figure out which method and what h the tool you found is using? I am not sure you can because of today's computing power, but it might be interesting to try.*

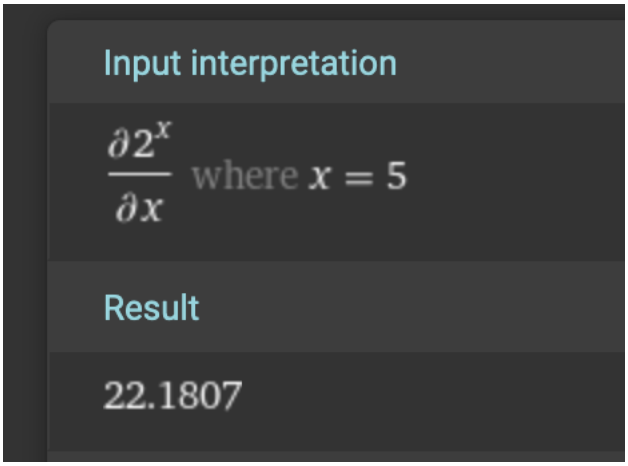
I found that Wolfram Alpha had a numerical derivative widget, which I used :https://www.wolframalpha.com/widgets/view.jsp?id=a278064e754d61cbecc14f913b8d5295

I used the tool to find these solutions:









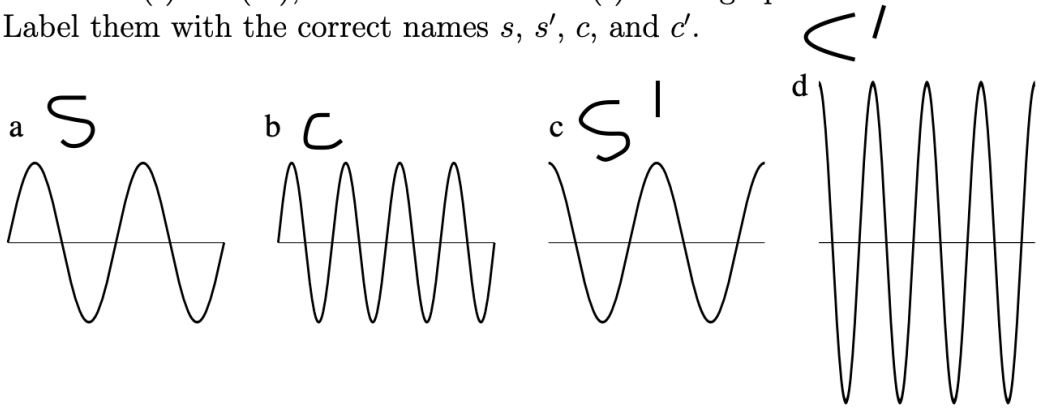
Which are almost exactly the same as the solutions I found above using an h = 0.005, with the exception of equation d which is off by the last decimal place.

I am not sure what h value the Wolfram alpha calculator uses, (I don't actually know if it does use this method), but if it does then it is definitely less than 0.005.

1.3 3. Read section 3.5 and do #2 and #5 (pages 151, 152)

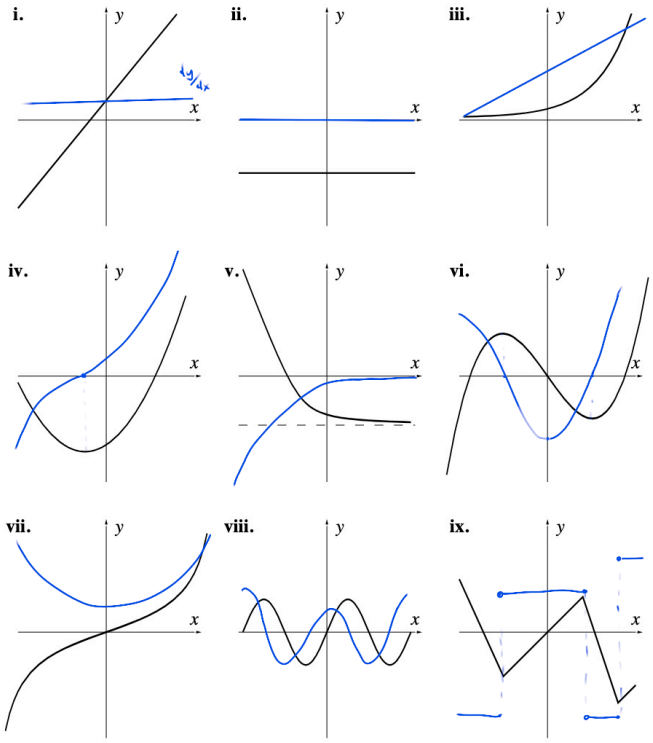
1.3.1 2:

2. Here are the graphs of four related functions: s, its derivative s', another function $c(t) = s(2t)$, and *its* derivative $c'(t)$. The graphs are out of order. Label them with the correct names s, s', c, and c'.



1.3.2 5:

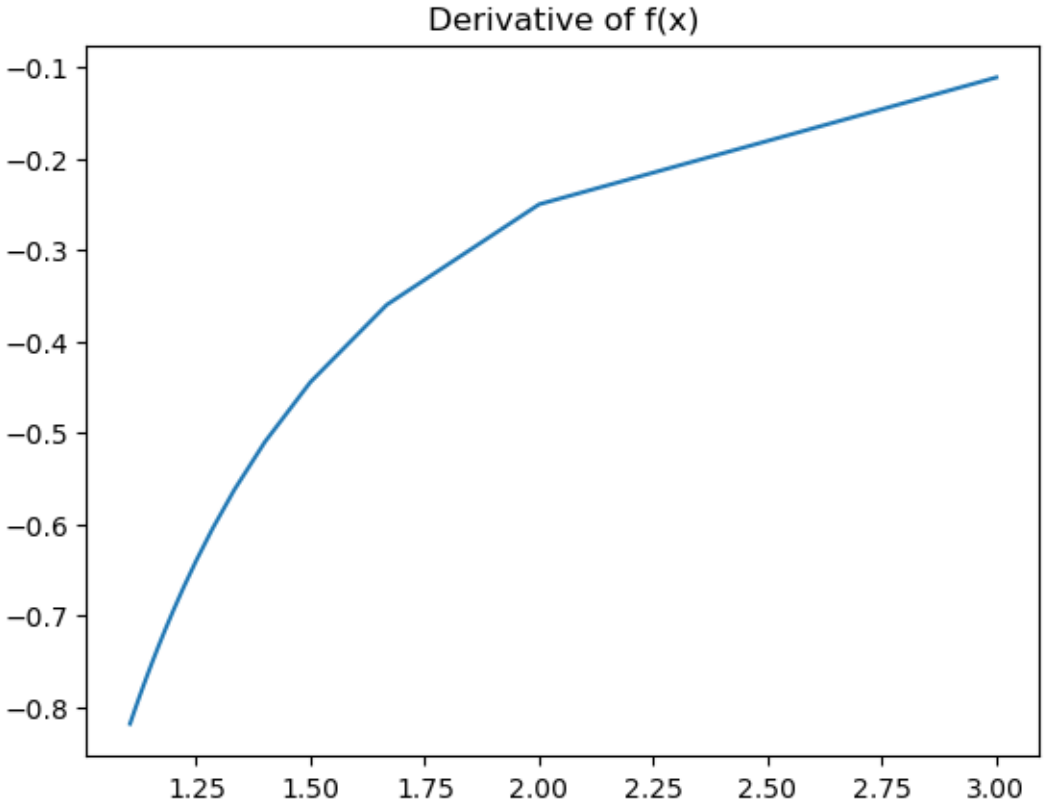
5. For each of the functions graphed below, sketch the graph of its derivative.

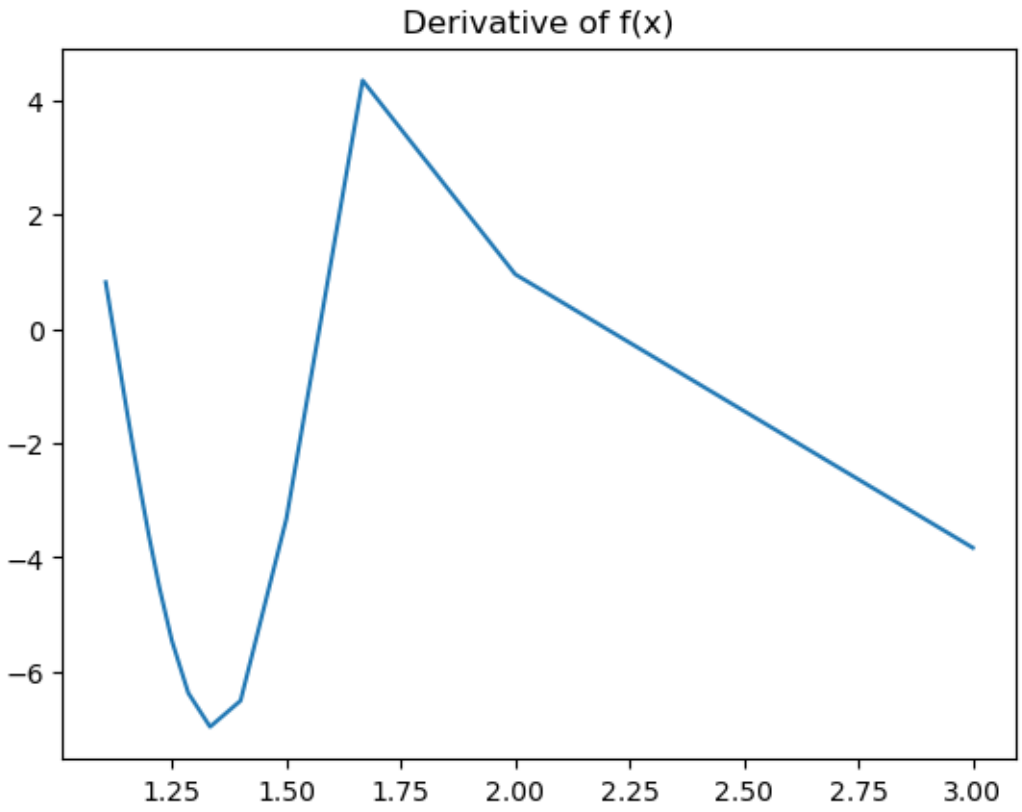


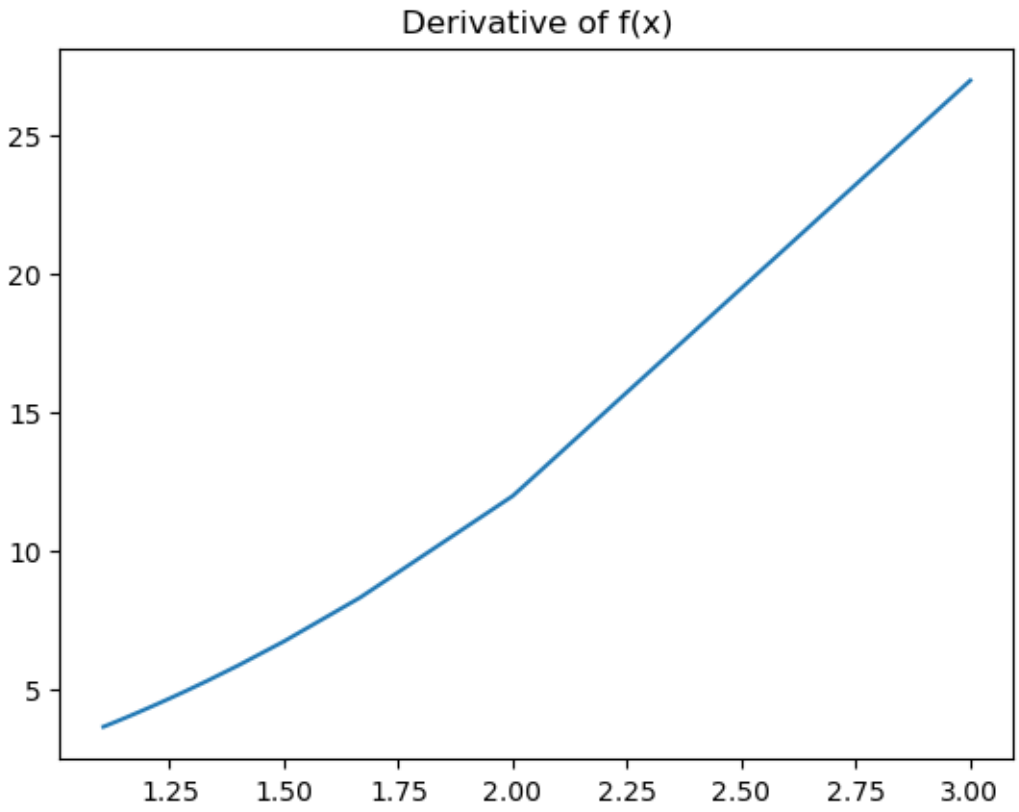
1.4 4. Copy and Modify your code in #1 to calculate the derivative at multiple points (at 20 points) over an x- interval from 1 to 3 and graph the results. You will now get an approximation to the graph of the derivative function. Call this new code "Numerical_Function_Derivative" Use the same examples from #10 page 133.

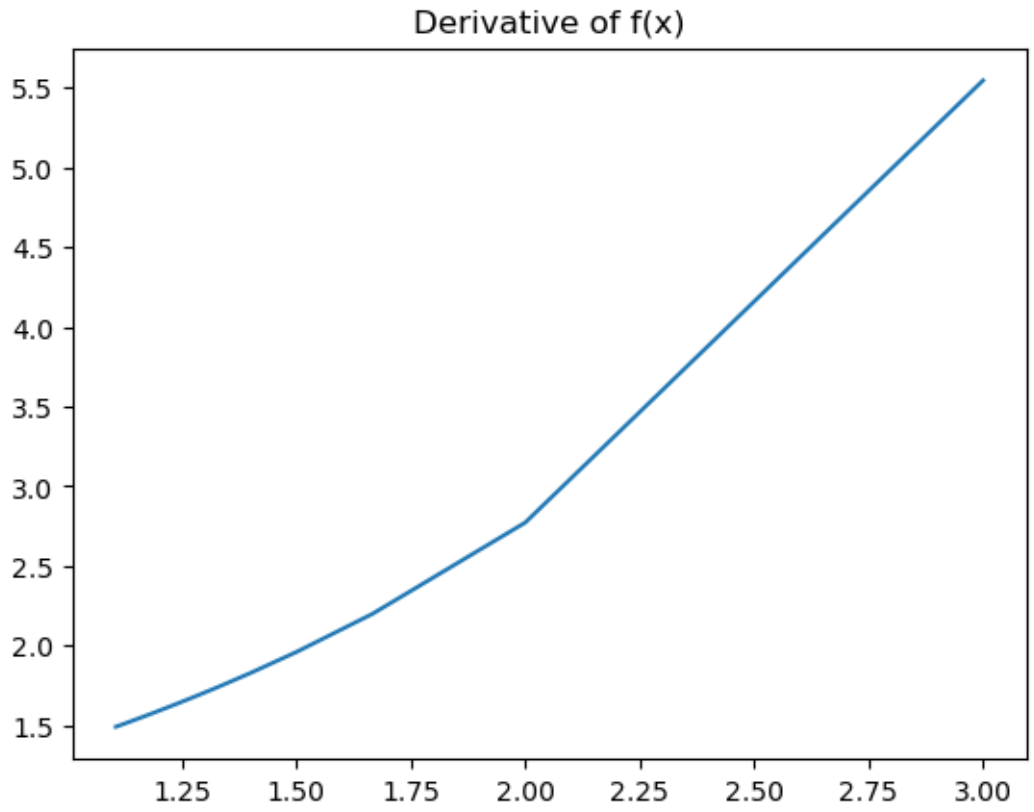
[7]:
```
import matplotlib.pyplot as plt

def Numerical_Function_Derivative(f, h):
    X = []
    fx = []
    for x in range(1, 20):
        i = 1+(2/x)
        X.append(i)
        fx.append(Numerical_Point_Derivative(f, h, i))

    fig, ax = plt.subplots()
    ax.plot(X, fx)
    ax.set_title("Derivative of f(x)")
    plt.show()
```

[8]:
```
Numerical_Function_Derivative(f1a, 0.005)
Numerical_Function_Derivative(f1b, 0.005)
Numerical_Function_Derivative(f1c, 0.005)
Numerical_Function_Derivative(f1d, 0.005)
```




Derivative of f(x)
4
2
0
−2
−4
−6
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


Derivative of f(x)
25
20
15
10
5
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

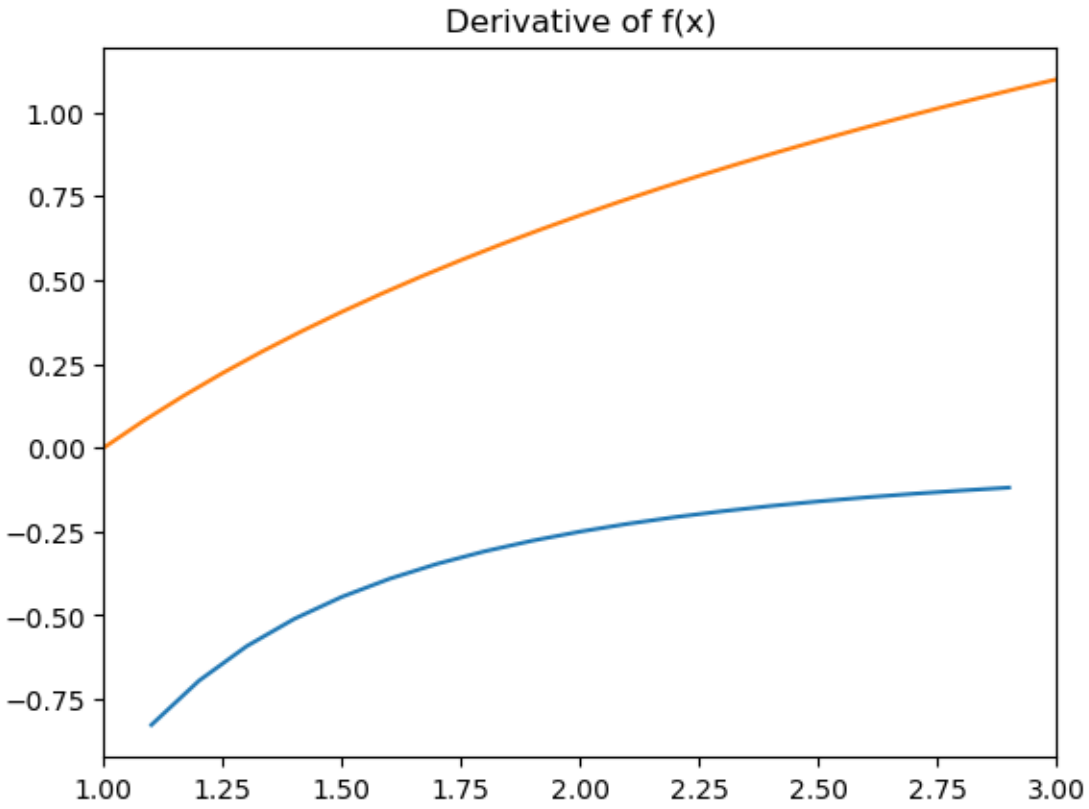


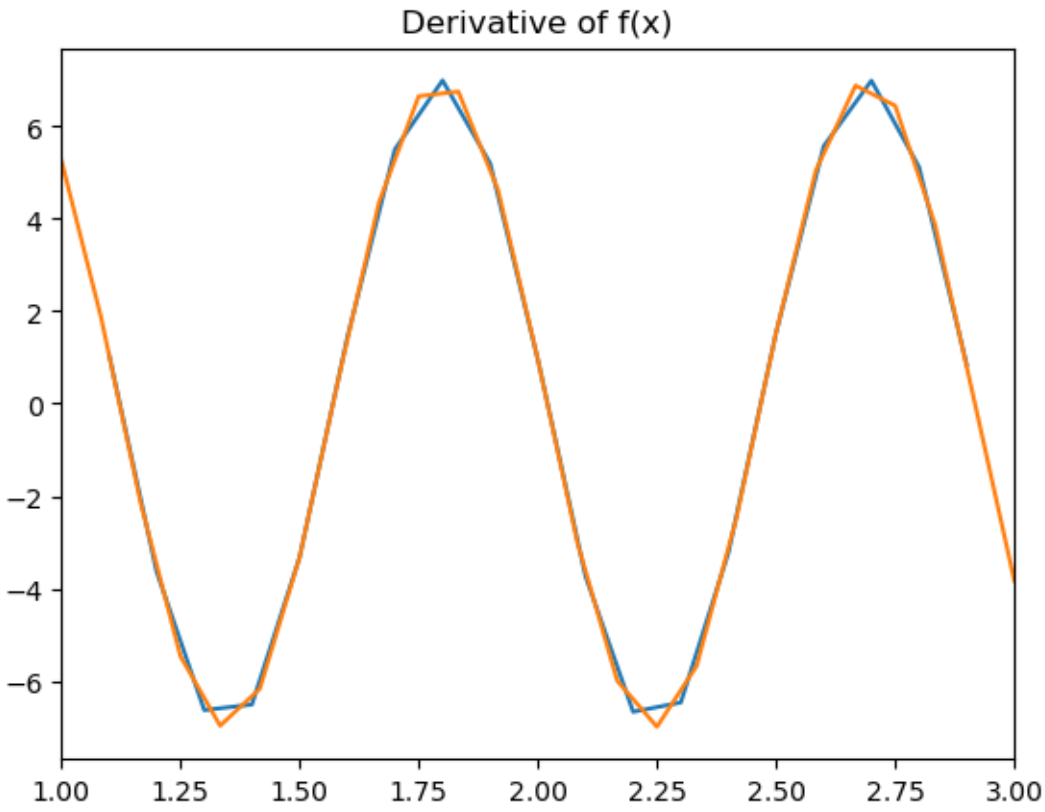
1.5 5. Graph these f'(x) functions on the interval (1,3), how do they compare to your numerical results? Are 20 samples enough to graph the derivative on this interval? (you can codethese or just use Demos.com and see if you got the same shape. What's tricky about b)?)

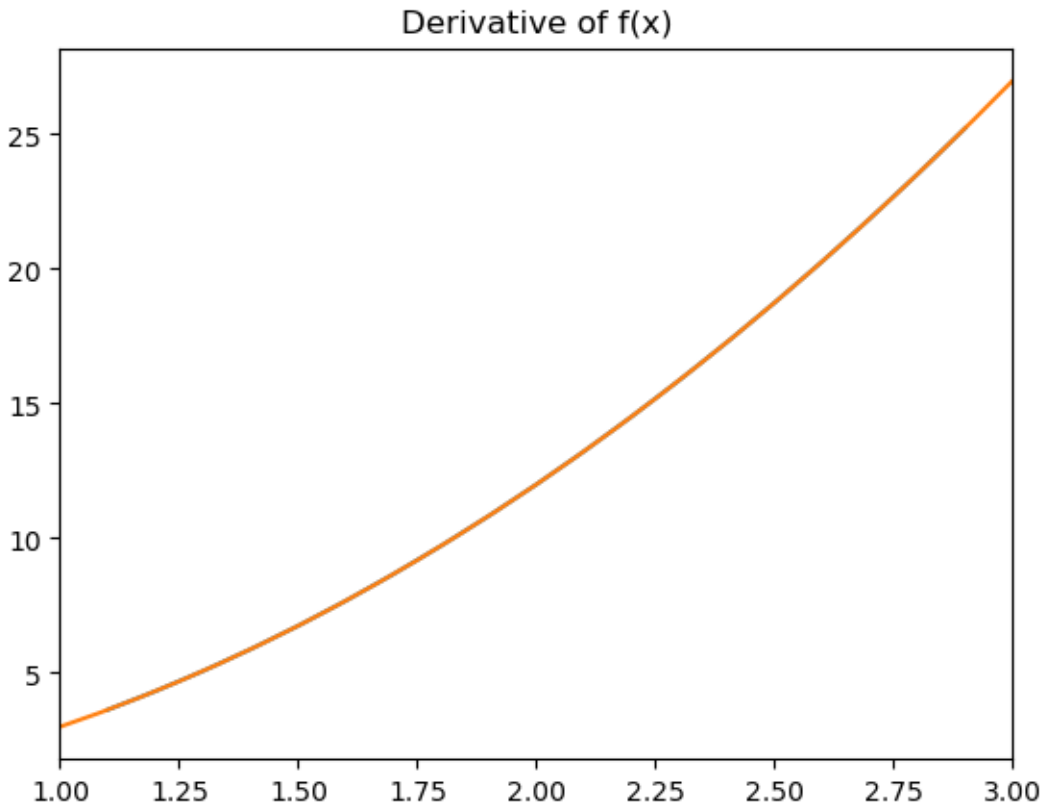
[13]:
```
def Numerical_Function_Derivative_cont(f, fp, h):
    X = []
    fx = []
    xcont = np.linspace(1, 3, 25)
    y = list(map(fp, list(xcont)))
    for x in range(1, 20):
        i = 1+(2 *x/20)
        X.append(i)
        fx.append(Numerical_Point_Derivative(f, h, i))

    fig, ax = plt.subplots()
    ax.plot(X, fx)
    ax.plot(xcont, y)
    ax.set_title("Derivative of f(x)")
    ax.set_xlim(1, 3)
```

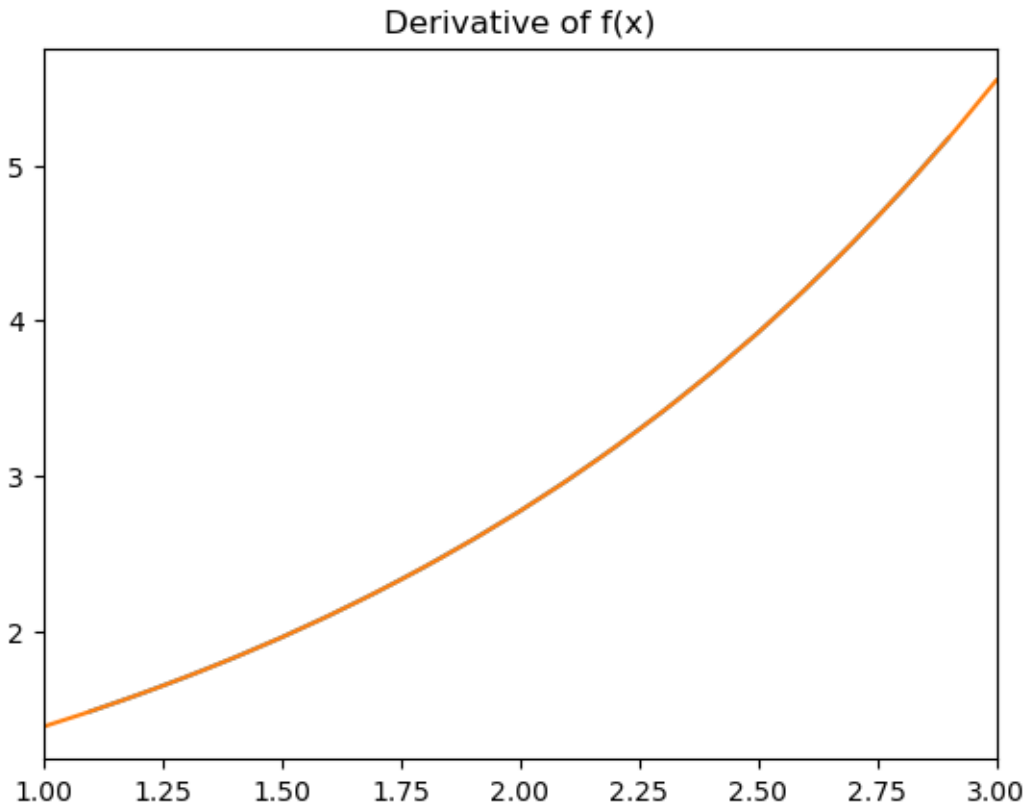
```
    plt.show()
```

[17]:
```
def fcont1(x):
    return np.log(x)
def fcont2(x):
    return 7*np.cos(7*x)
def fcont3(x):
    return (3*(x**2))
def fcont4(x):
    return (np.log(2) * 2**x)

Numerical_Function_Derivative_cont(f1a,fcont1, 0.005)
Numerical_Function_Derivative_cont(f1b,fcont2, 0.005)
Numerical_Function_Derivative_cont(f1c,fcont3, 0.005)
Numerical_Function_Derivative_cont(f1d, fcont4, 0.005)
```




Derivative of f(x)
6
4
2
0
−2
−4
−6
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


Derivative of f(x)
25
20
15
10
5
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00



Yes, the Graphs of the derivative functions line up well with the estimations used by evaluating the derivative at different x-values. Finding the derivative for part b is tricky because you have to use the chain rule to solve the derivative

[ ]: