# Week 4 Activities

● Graded

**Student**

Harper Chen

**Total Points**

**98 / 100 pts**

**Question 1**

## Code Numerical Point Derivative

**20** / 20 pts

✔ **– 0 pts** Correct

**– 0 pts** Excellent!

**Question 2**

## Compare your code to a Derivative Calculator on the web

**20** / 20 pts

✔ **– 0 pts** Correct

**– 5 pts** Show some results?

**– 0 pts** When I search "numerical derivative calculator" quite a few come up.

**– 0 pts** Good analysis

**Question 3**

## Sketch some derivative with #2 and #3

**18** / 20 pts

**– 1 pt** #1 horizontal  line

**– 1 pt** #2 horizontal line at y=0

**– 1 pt** #3 matches or closely follows y, does not cross or meet 0.  Slope is dramatically changing so it's not a straight line.

**– 1 pt** #4 crosses x axis in negative range ( -3 -ish)

**– 1 pt** #5 does not reach 0.

**– 1 pt** #6, bowl shaped. Crosses x-axis -5 and 5 (ish)

✔ **– 2 pts** #7,bowl shaped, does not meet or cross x axis. y does not have zero slope at any point.  (remember slope can be negative and doesn't have to be decreasing)

**– 1 pt** #8, same shape shifted horizontally.

**– 1 pt** #9 broken line segments

**– 0 pts** Correct

**– 10 pts** Sketching #3?

**Question 4**

## Make a Numerical Function Calculator

**20** / 20 pts

✔ **– 0 pts** Correct

**– 0 pts** Excellent!

**Question 5**

## Compare to the algebraic solutions

**20** / 20 pts

✔ **– 0 pts** Correct

✔ **– 0 pts** Okay. So I realized my instructions were not quite perfectly clear. The intension was that you graph the actual function curve of the derivative (given - except for the one with the typo), and then compare THAT smooth function to your numerically coded curve - not just points. So that's on me for not being clear. YOu may want to try it that way too.

Questions assigned to the following page: 1 and 2

# Week4

September 25, 2024

#1

[9]:
```
import sympy as sp
import math

def Numerical_Point_Derivative():
    x = sp.symbols('x')

    f_input = input("Enter an algebraic function x: ")
    f = sp.sympify(f_input)

    a = float(input("Enter the value of x (a) to find the slope: "))

    h = 1e-5  # Small value for numerical differentiation

    # Use Q2 = (f(a + h) - f(a)) / h for numerical differentiation
    # Chosen Q2 because it's a straightforward forward difference approximation,
    # providing a quick estimate of the derivative at the point 'a' without␣
 ↪needing the value at 'a - h'.
    f_a_plus_h = f.subs(x, a + h)
    f_a = f.subs(x, a)
    slope = (f_a_plus_h - f_a) / h

    return float(slope)

print("Slope of the function at the point: ", Numerical_Point_Derivative())
```

```
Enter an algebraic function x:  sin(7*x)
Enter the value of x (a) to find the slope:  3
```

```
Slope of the function at the point:  -3.834309799122248
```

[ ]:

[ ]:

#2

I compared my numerical derivative implementation with two online tools: Wolfram Alpha's Nu-

Questions assigned to the following page: 2 and 3

merical Derivative Calculator and SolveMyMath's Numerical Derivative Calculator. The results from both tools were quite similar to my forward difference method, but the precision and accuracy of the online calculators were notably higher.

Wolfram Alpha likely uses a more advanced method, potentially a central difference approach, which is generally more accurate than a simple forward difference method like mine. It may also dynamically adjust the step size $h$, making it more suitable for handling a wide variety of functions. The tool from SolveMyMath provides a basic numerical derivative but doesn't specify the exact method it uses, though it's possible that it's using a simple fixed $h$, like my implementation.

From what I observed, my approach with a fixed small $h$ was effective, but online calculators leverage more complex algorithms to fine-tune the accuracy. While it's hard to tell the exact value of $h$ they use, Wolfram's powerful engine suggests it's optimized for accuracy based on the function being evaluated. The tools offer a great learning comparison, and I could see how further refining the step size or using a more advanced method would improve my results in specific cases.

[ ]:

[ ]:

#3

*Question* 2 1. **For** $s(t)$: I expect this to be the simplest periodic wave, likely with a regular frequency. It's a standard sine or cosine wave with consistent oscillations.

2. **For** $s'(t)$: The derivative of $s(t)$ should resemble the original sine wave but with some shifts or changes in amplitude. It will still be periodic but might look steeper or shifted.
3. **For** $c(t) = s(2t)$: Since $c(t)$ is $s(t)$ but compressed along the time axis, I expect this function to have a higher frequency, meaning it will oscillate faster than $s(t)$.
4. **For** $c'(t)$: The derivative of $c(t)$ should have an even higher frequency compared to $s'(t)$ because it's the derivative of the compressed function. It should look like rapid oscillations, representing a steeper rate of change.

- **Graph a**: This shows a relatively smooth sine-like wave with a standard frequency, so I labeled this as $s(t)$.
- **Graph b**: The oscillations here are faster, suggesting that it's a compressed version of $s(t)$. This fits the description of $c(t) = s(2t)$.
- **Graph c**: This wave looks similar to $s(t)$ but seems to represent the rate of change, so it likely corresponds to $s'(t)$.
- **Graph d**: The most rapid oscillations appear here, indicating a much higher frequency function. This matches $c'(t)$, the derivative of $c(t)$.

My answers are:

- **a:** $s(t)$
- **b:** $c(t) = s(2t)$
- **c:** $s'(t)$
- **d:** $c'(t)$

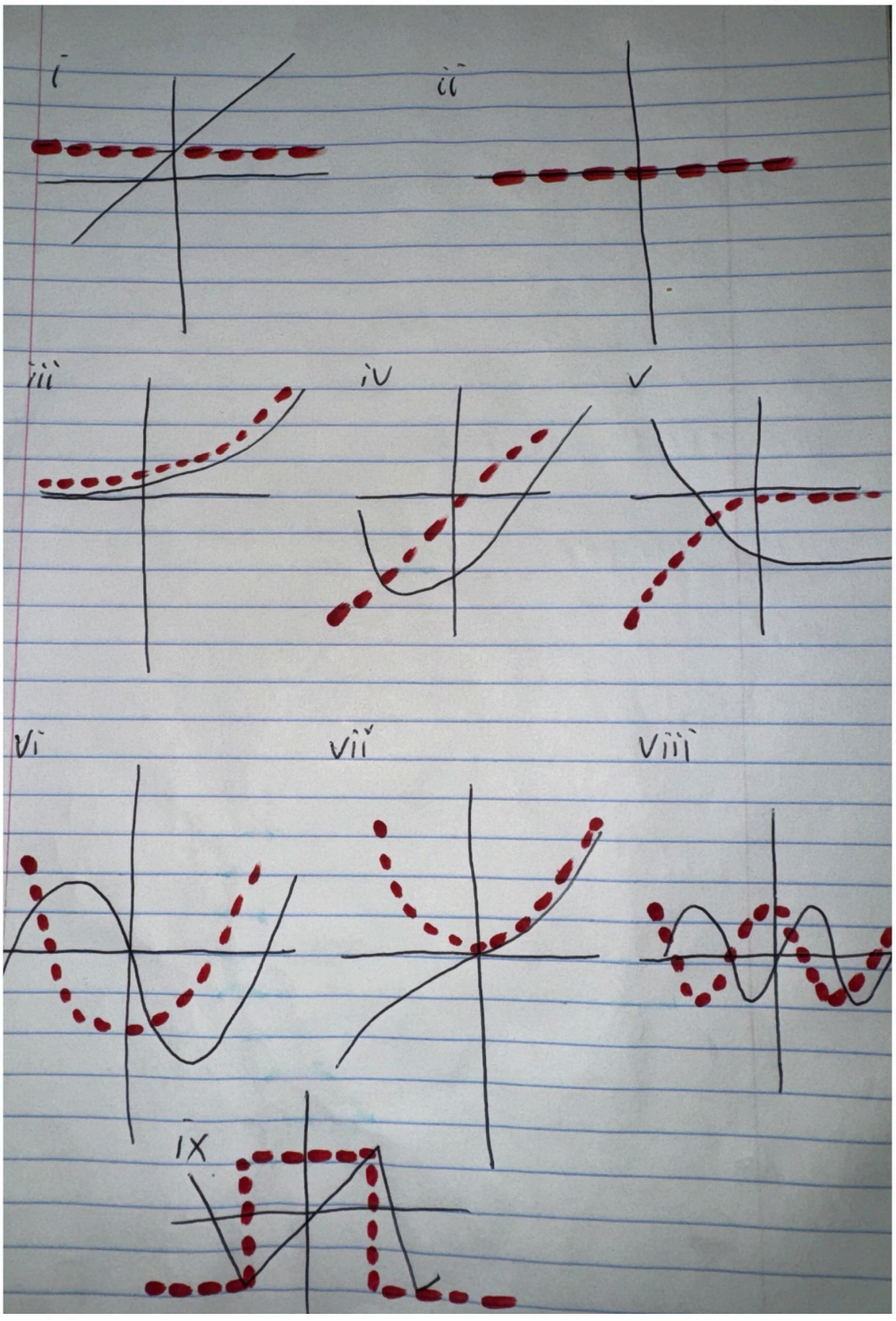
Question assigned to the following page: 3

*Question* 5

[2]:
```
from IPython.display import Image
Image(filename='Q5.jpg')
```

[2]:

Question assigned to the following page: 3


i
ii
iii
iv
v
vi
vii
viii
ix

Question assigned to the following page: 4

[ ]:

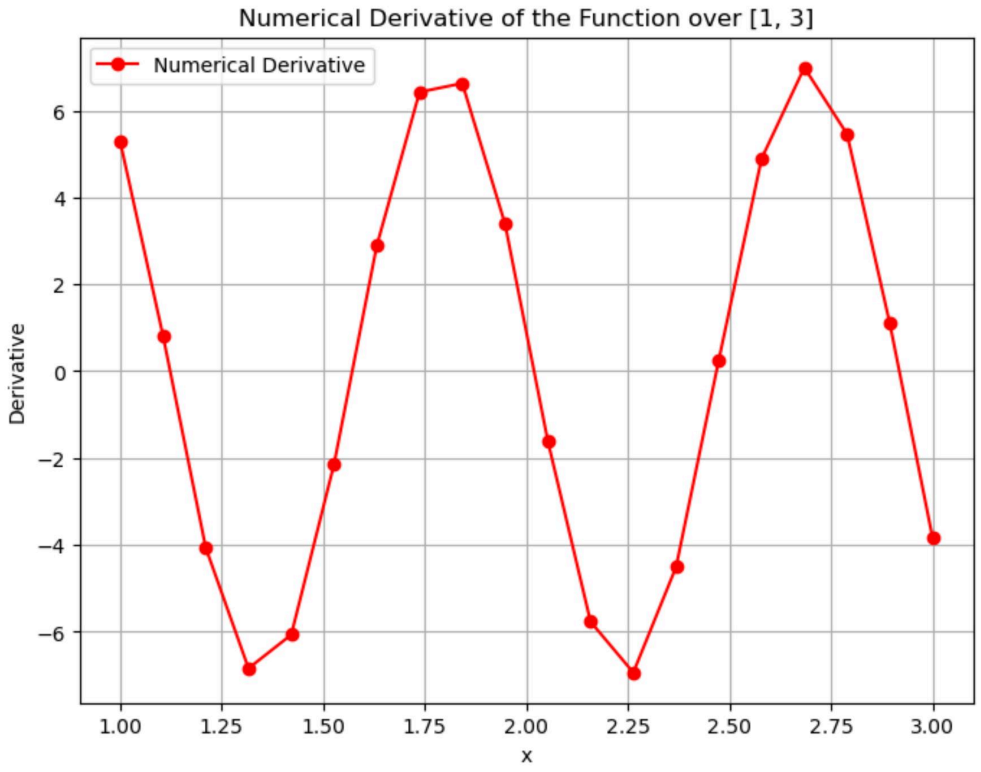
[ ]:

#4

[19]:
```
def Numerical_Function_Derivative():
    x = sp.symbols('x')

    f_input = input("Enter an algebraic function in terms of x: ")
    f = sp.sympify(f_input)
    h = 1e-5

    Xvals = np.linspace(1, 3, 20)
    DeriVals = []

    for a in Xvals:
        f_a_plus_h = f.subs(x, a + h)
        f_a = f.subs(x, a)
        slope = (f_a_plus_h - f_a) / h
        DeriVals.append(float(slope))


    plt.figure(figsize=(8, 6))
    plt.plot(Xvals, DeriVals, marker='o', label='Numerical Derivative',␣
 ↪color='red')
    plt.title("Numerical Derivative of the Function over [1, 3]")
    plt.xlabel("x")
    plt.ylabel("Derivative")
    plt.grid(True)
    plt.legend()
    plt.show()

Numerical_Function_Derivative()
```

```
Enter an algebraic function in terms of x:  sin(7*x)
```

Questions assigned to the following page: 4 and 5



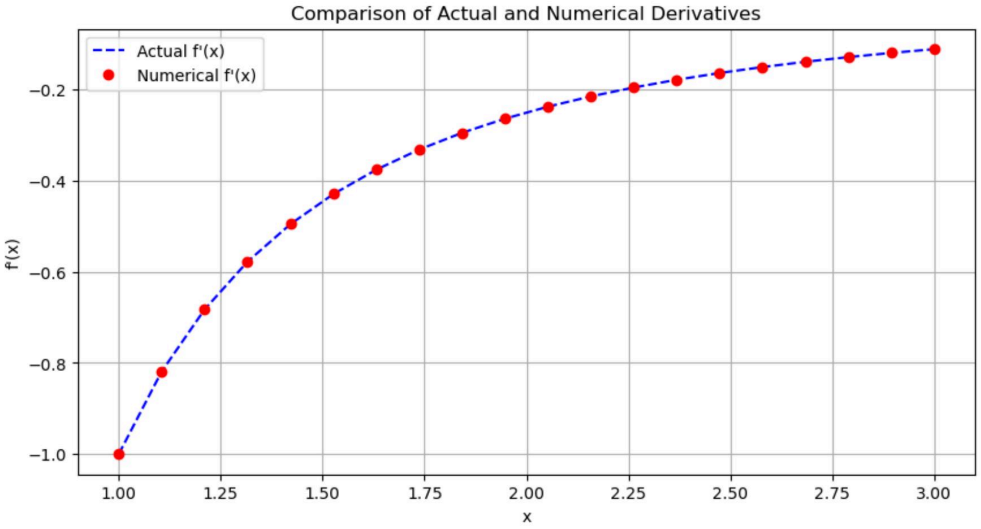

[ ]:

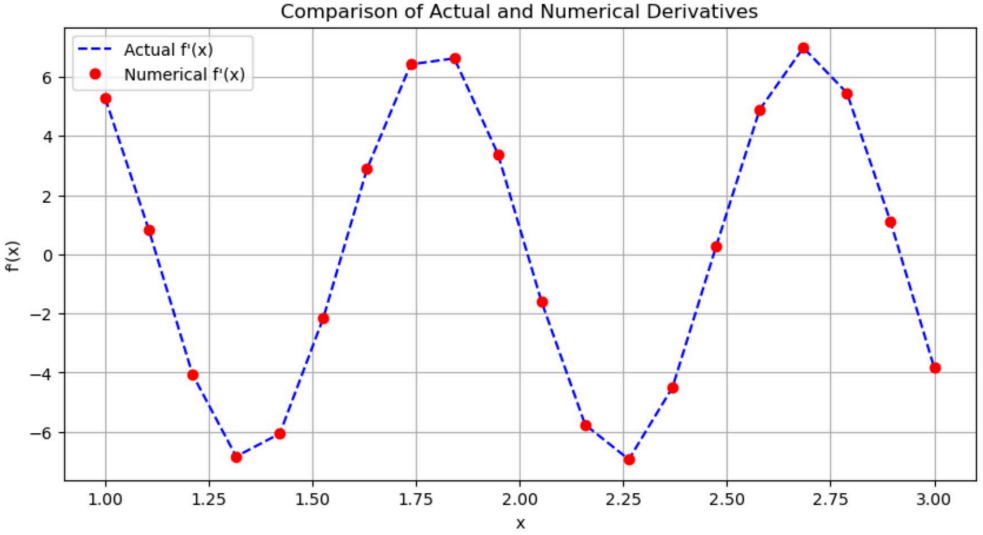
[21]:
```
def Numerical_Point_Derivative(f, a, h=1e-5):
    return (f(a + h) - f(a)) / h

# Functions and their derivatives
def f1(x):
    return 1/x

def df1(x):
    return -1/x**2

def f2(x):
    return np.sin(7*x)

def df2(x):
    return 7*np.cos(7*x)

def f3(x):
```

Question assigned to the following page: 5

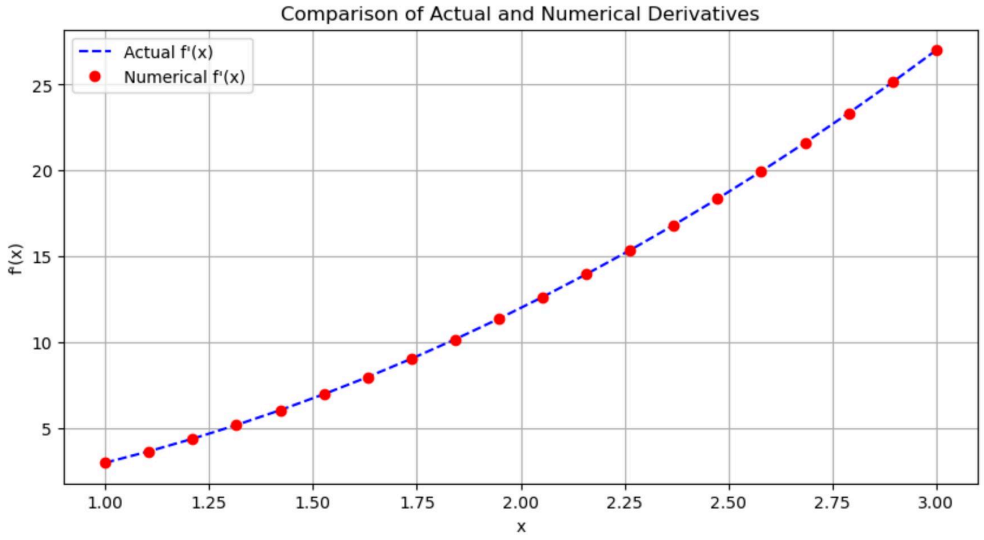
```
    return x**3

def df3(x):
    return 3*x**2

def f4(x):
    return 2**x

def df4(x):
    return np.log(2) * 2**x

def plot_function_and_derivative(f, df, x_start, x_end, num_points):
    x_values = np.linspace(x_start, x_end, num_points)
    actual_derivative_values = [df(x) for x in x_values]
    numerical_derivative_values = [Numerical_Point_Derivative(f, x) for x in␣
 ↪x_values]

    plt.figure(figsize=(10, 5))
    plt.plot(x_values, actual_derivative_values, label="Actual f'(x)",␣
 ↪linestyle='--', color='blue')
    plt.plot(x_values, numerical_derivative_values, label="Numerical f'(x)",␣
 ↪marker='o', linestyle='', color='red')
    plt.xlabel('x')
    plt.ylabel("f'(x)")
    plt.title('Comparison of Actual and Numerical Derivatives')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_function_and_derivative(f1, df1, 1, 3, 20)
plot_function_and_derivative(f2, df2, 1, 3, 20)
plot_function_and_derivative(f3, df3, 1, 3, 20)
plot_function_and_derivative(f4, df4, 1, 3, 20)
```

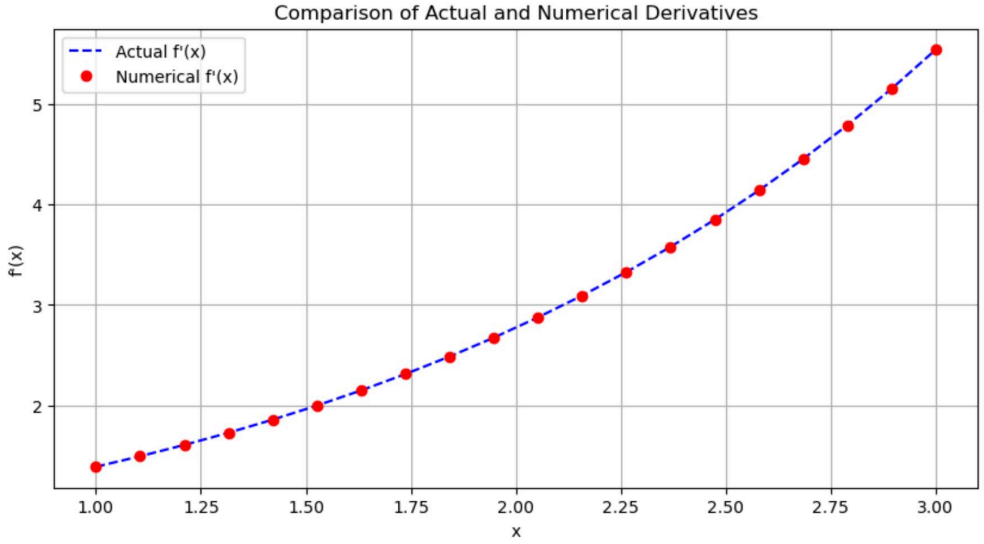
Question assigned to the following page: 5


Comparison of Actual and Numerical Derivatives
Actual f'(x)
Numerical f'(x)
f'(x)
x
−0.2
−0.4
−0.6
−0.8
−1.0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


Comparison of Actual and Numerical Derivatives
Actual f'(x)
Numerical f'(x)
f'(x)
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

Question assigned to the following page: 5




When I graph the actual derivatives $f'(x)$ on the interval $(1, 3)$ and compare them to the numerical results, they look similar overall. For functions like $f(x) = \frac{1}{x}$ and $f(x) = x^3$, 20 samples seem to be enough to get an accurate shape of the derivative, as both functions are smooth and don't change too quickly.

However, for $f(x) = \sin(7x)$, the derivative $f'(x) = 7\cos(7x)$ oscillates very quickly. With only 20 points, the numerical method misses some of the finer details of the derivative. This makes

Question assigned to the following page: 5

graphing this function more challenging.

For $f(x) = 2^x$, the exponential function, the numerical derivative $f'(x) = \ln(2) \cdot 2^x$ matches well with the actual derivative using 20 points, since the curve grows smoothly.

In summary, 20 samples are enough for most functions, but for quickly changing functions like $\sin(7x)$, more points are needed to capture the details accurately.