Week 4 - Numeric Differentiation

November 21, 2024

0.1 1

[1]:
```python
import math
import matplotlib.pyplot as plt

h = 0.00001
def a_f(x):
    return (1/x) #making different functions for each idea from Jasmine
def b_f(x):
    return (math.sin(7*x))
def c_f(x):
    return (x**3)
def d_f(x):
    return (2**x)

def Numerical_Point_Derivative(a, f):
    return ((f(a+h) - f(a-h))/(2*h)) # Chose central difference because last␣
 ↪week proved it's more accurate.

print("a) Values for 1/x at x = 2: ", Numerical_Point_Derivative(2,a_f))

print("b) Values for sin(7x) at x = 3: ", Numerical_Point_Derivative(3, b_f))

print("c) Values for x^3 at x = 200: ", Numerical_Point_Derivative(200, c_f))

print("d) Values for 2^x at x = 5: ", Numerical_Point_Derivative(5, d_f))
```

```
a) Values for 1/x at x = 2:  -0.2500000000099645
b) Values for sin(7x) at x = 3:  -3.8341048184897804
c) Values for x^3 at x = 200:  120000.00006519257
d) Values for 2^x at x = 5:  22.180709777153137
```

0.2 2

I used Wolfram Apha as a Numerical Derivative calculator to compare my work. Here are the answers it gave me:

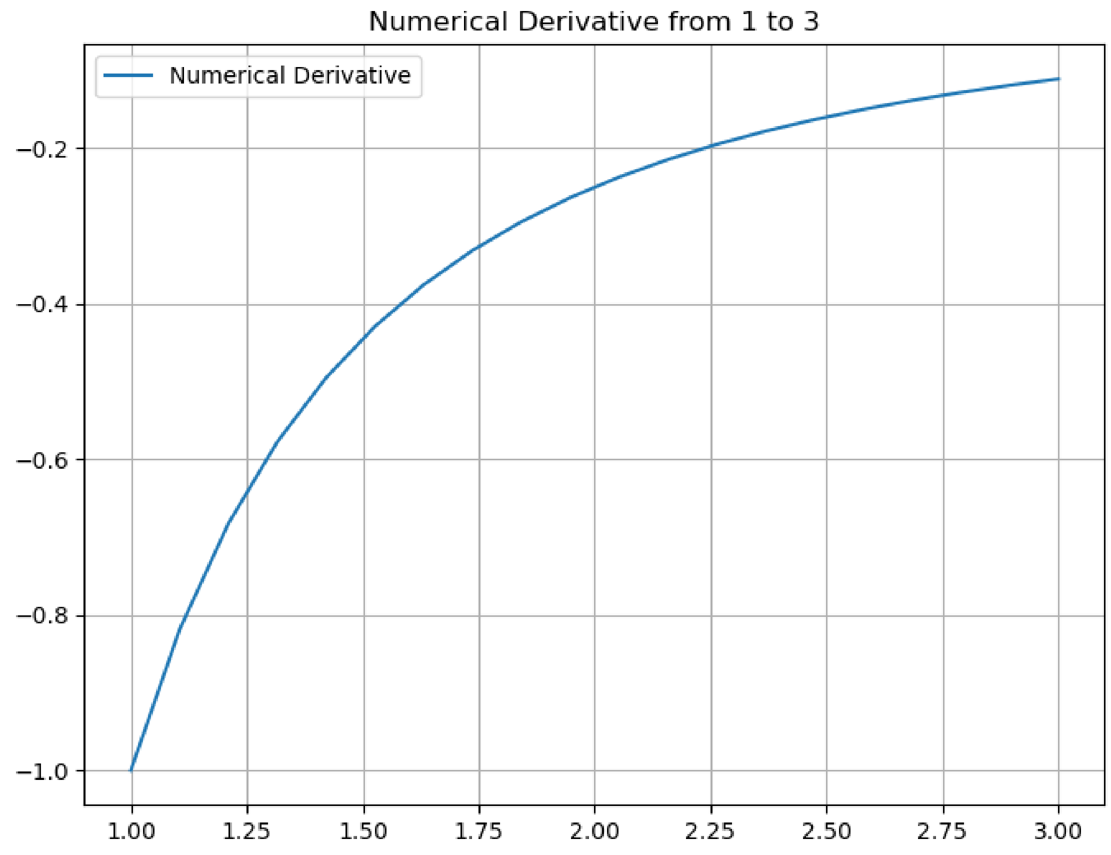
a) -0.25
b) -3.8341

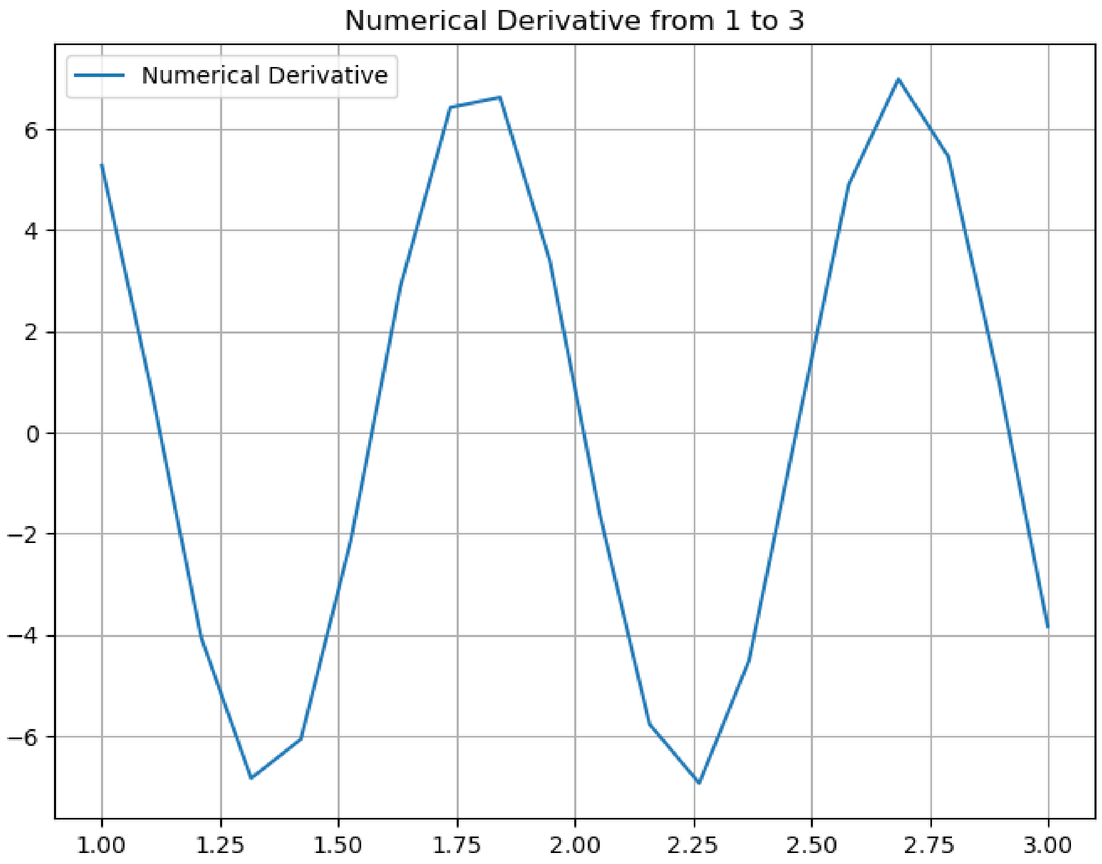
c) 120000
d) 22.1807

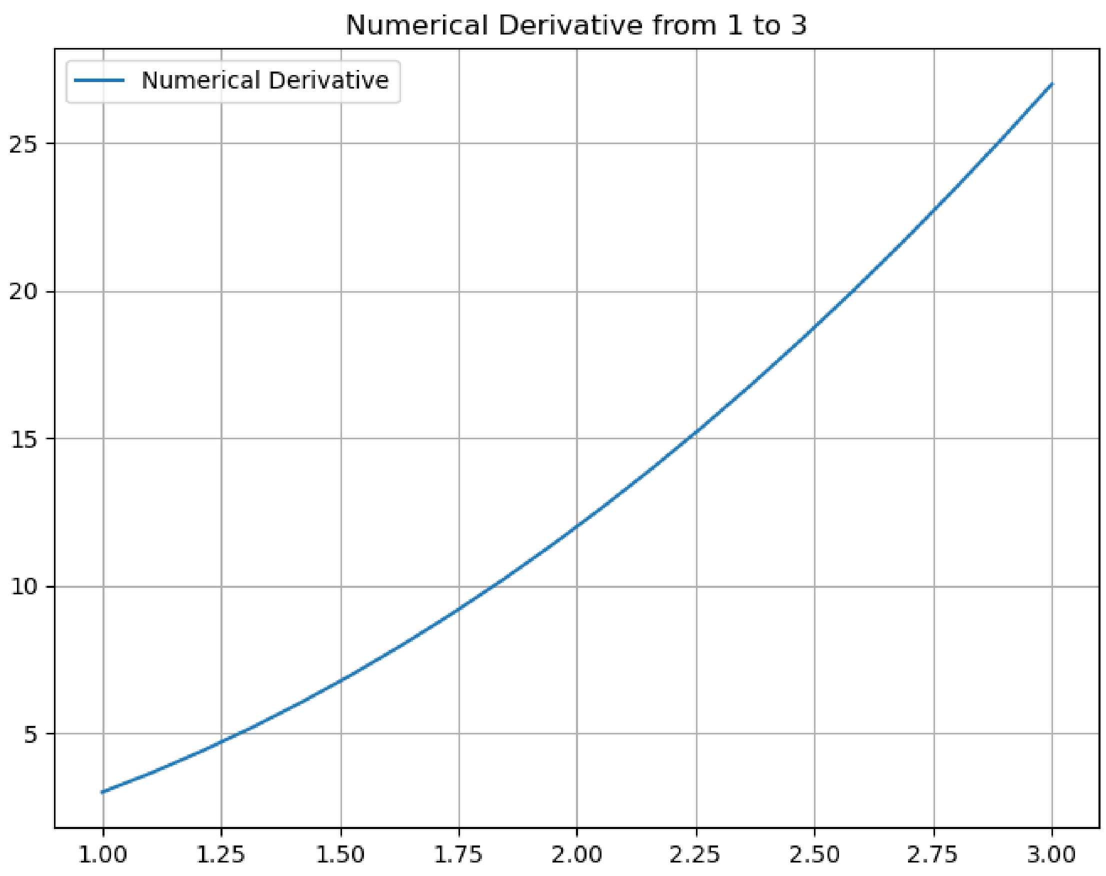
I found them to be fairly accurate but was not able to tell which method or h they were using mostly because most solutions were only provided to four decimal places. It seems likely that most calculators would use the secant method as it refines fastest from the tests we did in previous homework problems.

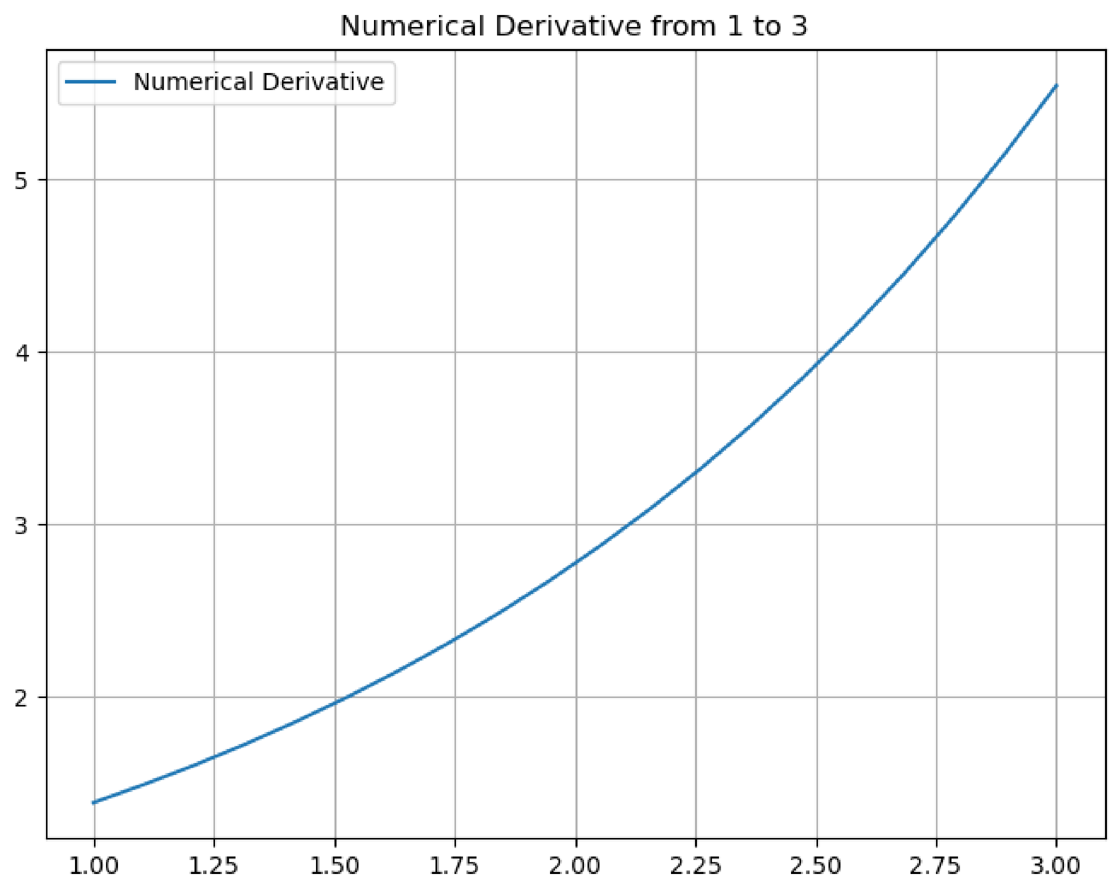
0.3 4

[2]:
```python
import math
import numpy as np
import matplotlib.pyplot as plt

h = 0.00001
def a_f(x):
    return (1/x) #making different functions for each idea from Jasmine
def a_f1(x):
    return (-1/x**2) #actual derivative for part 5

def b_f(x):
    return (math.sin(7*x))
def b_f1(x):
    return (7*np.cos(7*x))

def c_f(x):
    return (x**3)
def c_f1(x):
    return (3*x**2)

def d_f(x):
    return (2**x)
def d_f1(x):
    return (np.log(2) * 2**x)

def Numerical_Point_Derivative(a, f): #from earlier
    return ((f(a+h) - f(a-h))/(2*h)) #

def Numerical_Function_Derivative(x1,x2,num_points, f):
    xvals = np.linspace(x1,x2,num_points)
    num_derivative_vals = [Numerical_Point_Derivative(x, f) for x in xvals]
    return (xvals, num_derivative_vals)

def plot_figure(x1,x2,num_points,f):
    (xvals,yvals) = Numerical_Function_Derivative(x1,x2,num_points, f)
    plt.figure(figsize=(8,6))
    plt.plot(xvals, yvals, label = 'Numerical Derivative')
    plt.title("Numerical Derivative from " + str(x1) + " to " + str(x2))
    plt.grid(True)
    plt.legend()
    plt.show
x1 = 1
x2 = 3
num_points = 20
funcs = [a_f, b_f, c_f, d_f]
for f in funcs:
    plot_figure(x1,x2,num_points, f)
```




Numerical Derivative from 1 to 3
Numerical Derivative
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

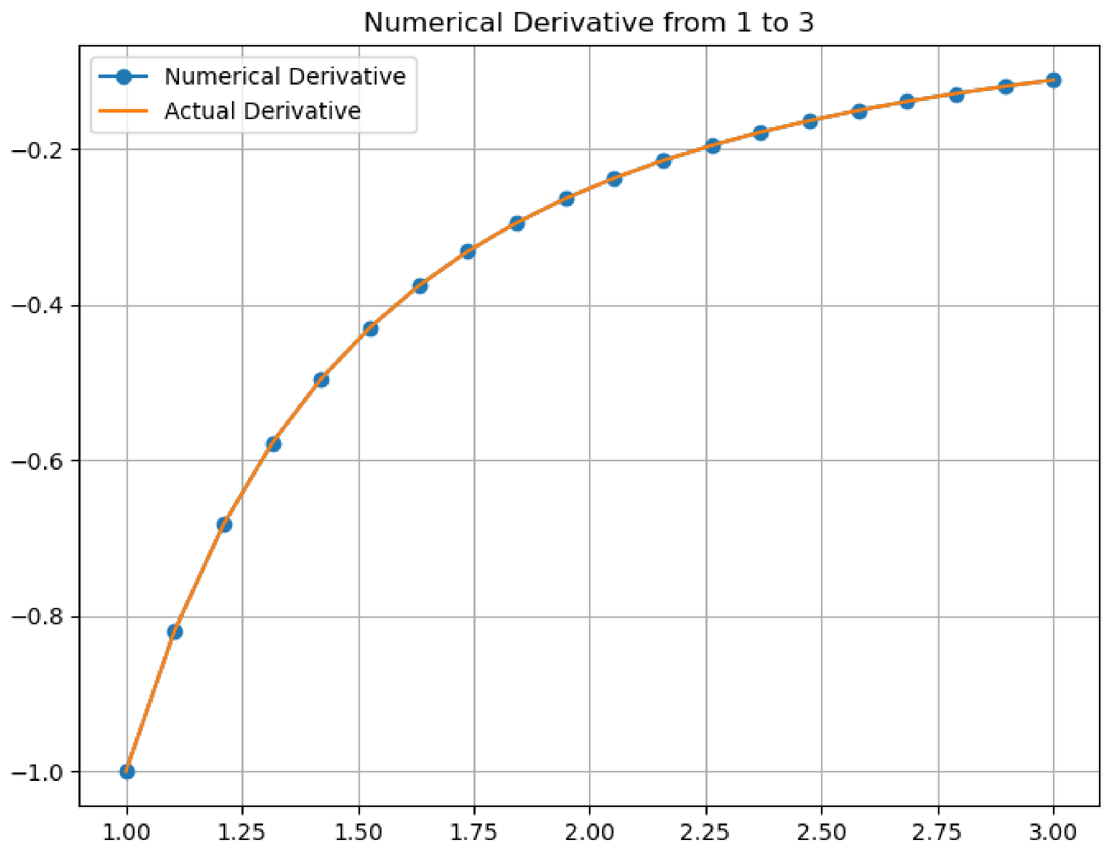

Numerical Derivative from 1 to 3
Numerical Derivative
25
20
15
10
5
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

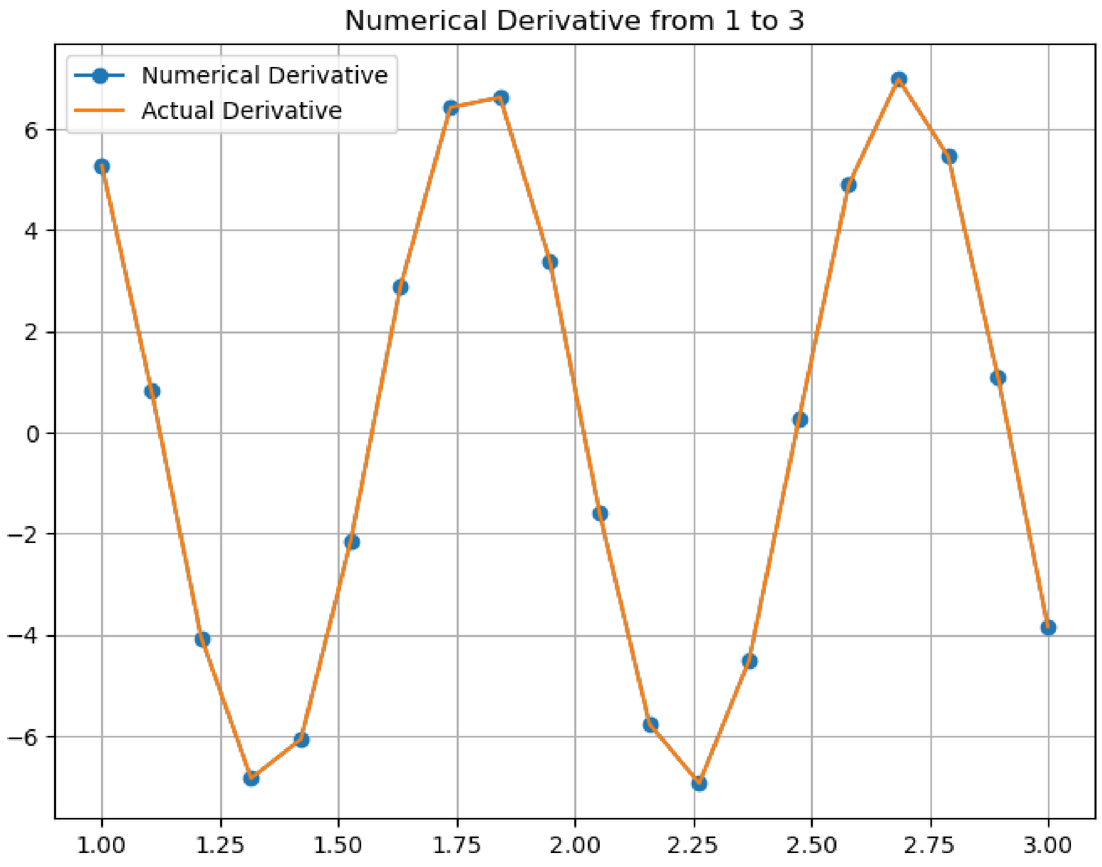


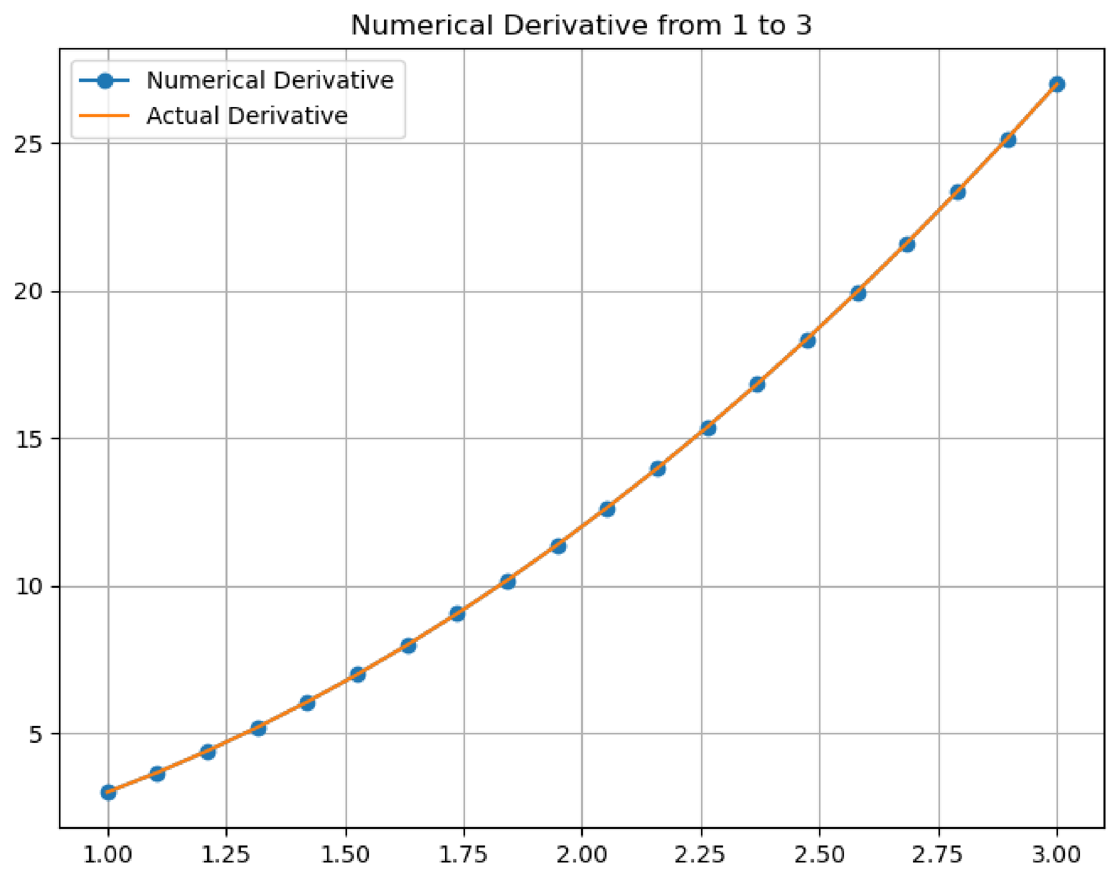
0.4 5

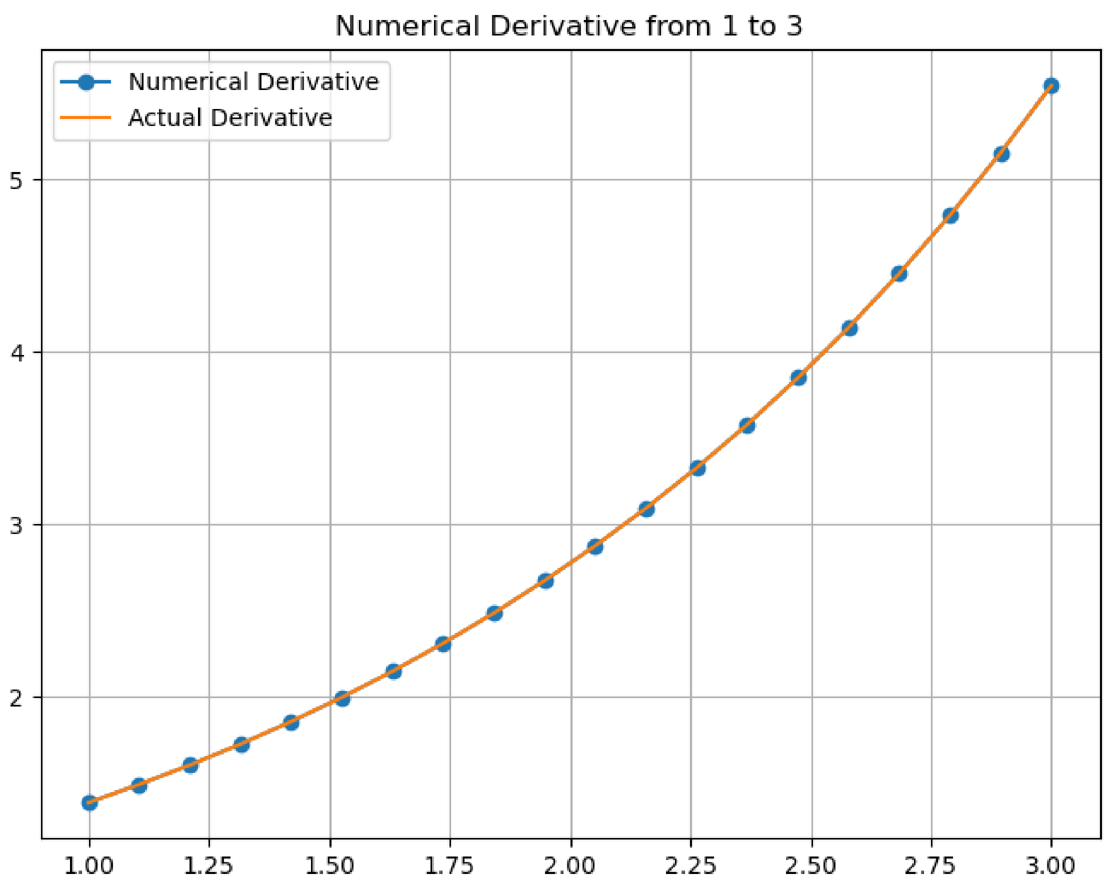
[3]:
```
def plot_figure(x1,x2,num_points,f, f1):
    (xvals,yvals) = Numerical_Function_Derivative(x1,x2,num_points, f)
    actual_derivatives = [f1(x) for x in xvals]

    plt.figure(figsize=(8,6))
    plt.plot(xvals, yvals, label = 'Numerical Derivative', marker='o')
    plt.plot(xvals, actual_derivatives, label = 'Actual Derivative')
    plt.title("Numerical Derivative from " + str(x1) + " to " + str(x2))
    plt.grid(True)
    plt.legend()
    plt.show
x1 = 1
x2 = 3
num_points = 20
funcs = [(a_f,a_f1), (b_f, b_f1), (c_f, c_f1), (d_f, d_f1)] #redo funcs to␣
 ↪include the actual derivatives
for (f,f1) in funcs:
```

```
plot_figure(x1,x2,num_points, f, f1)
```




Numerical Derivative from 1 to 3
Numerical Derivative
Actual Derivative
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

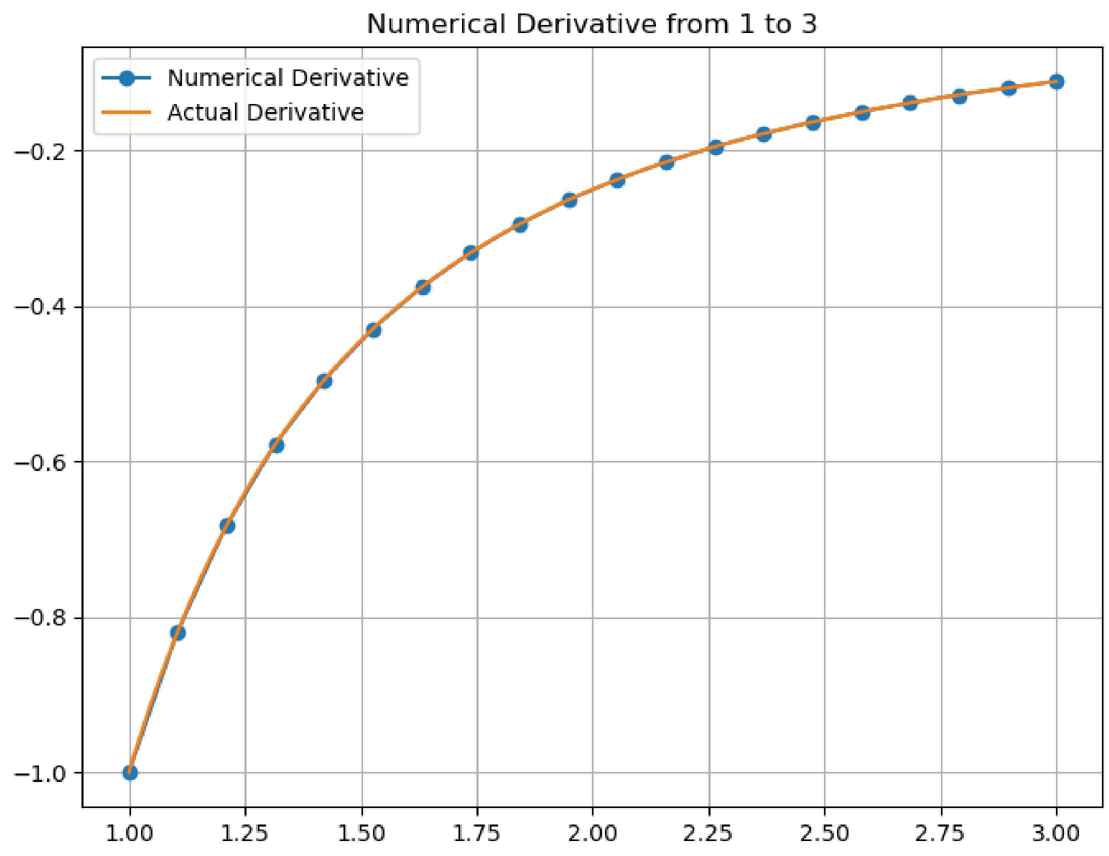

Numerical Derivative from 1 to 3
Numerical Derivative
Actual Derivative
25
20
15
10
5
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

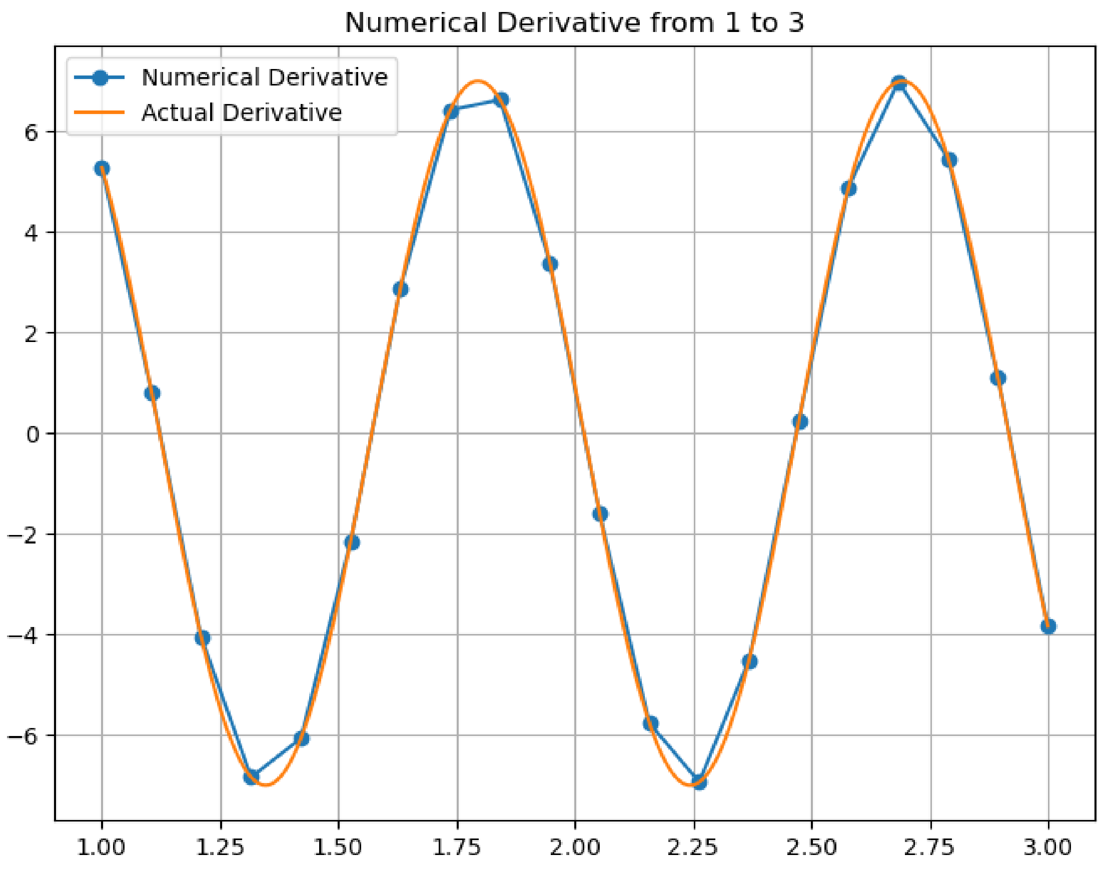


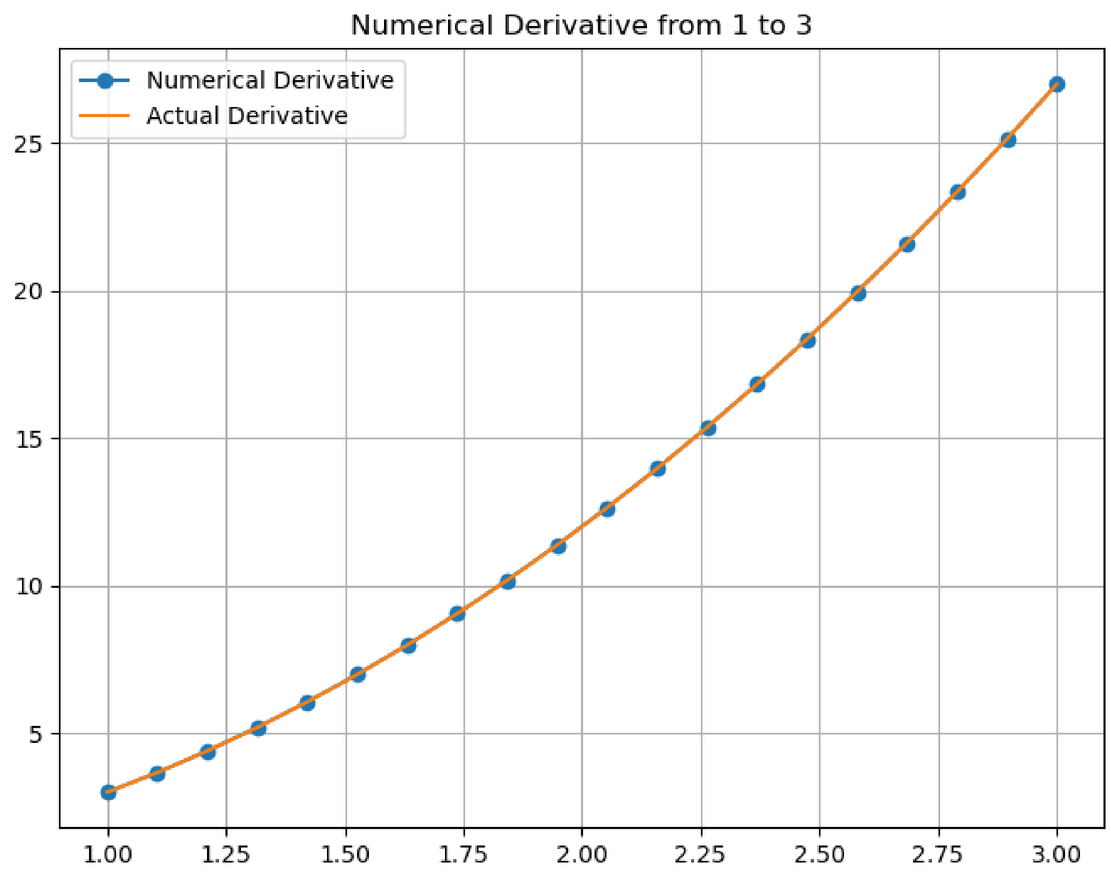
It seemed like 20 points was enough to get a close derivative expect for sin(7x). For fun I ran it again to with a higher interval for the actual functions which showed the derivative was accurate at that point but not nearly as smooth.

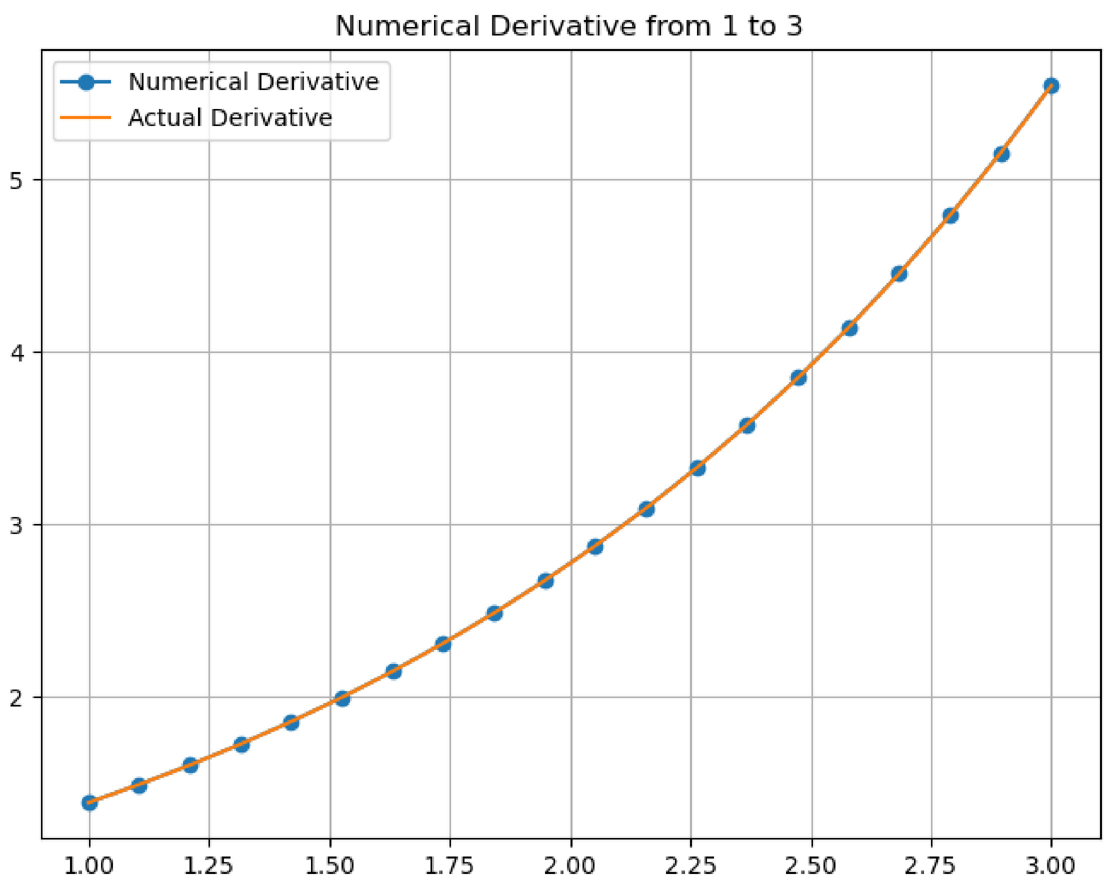
```
[4]: def plot_figure(x1,x2,num_points,f, f1):
         (xvals,yvals) = Numerical_Function_Derivative(x1,x2,num_points, f)
         higher_xvals = np.linspace(x1,x2,1000)
         actual_derivatives = [f1(x) for x in higher_xvals]

         plt.figure(figsize=(8,6))
         plt.plot(xvals, yvals, label = 'Numerical Derivative', marker='o')
         plt.plot(higher_xvals, actual_derivatives, label = 'Actual Derivative')
         plt.title("Numerical Derivative from " + str(x1) + " to " + str(x2))
         plt.grid(True)
         plt.legend()
         plt.show
     x1 = 1
     x2 = 3
     num_points = 20
```

```
funcs = [(a_f,a_f1), (b_f, b_f1), (c_f, c_f1), (d_f, d_f1)] #redo funcs to␣
↪include the actual derivatives
for (f,f1) in funcs:
    plot_figure(x1,x2,num_points, f, f1)
```




Numerical Derivative from 1 to 3
Numerical Derivative
Actual Derivative
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

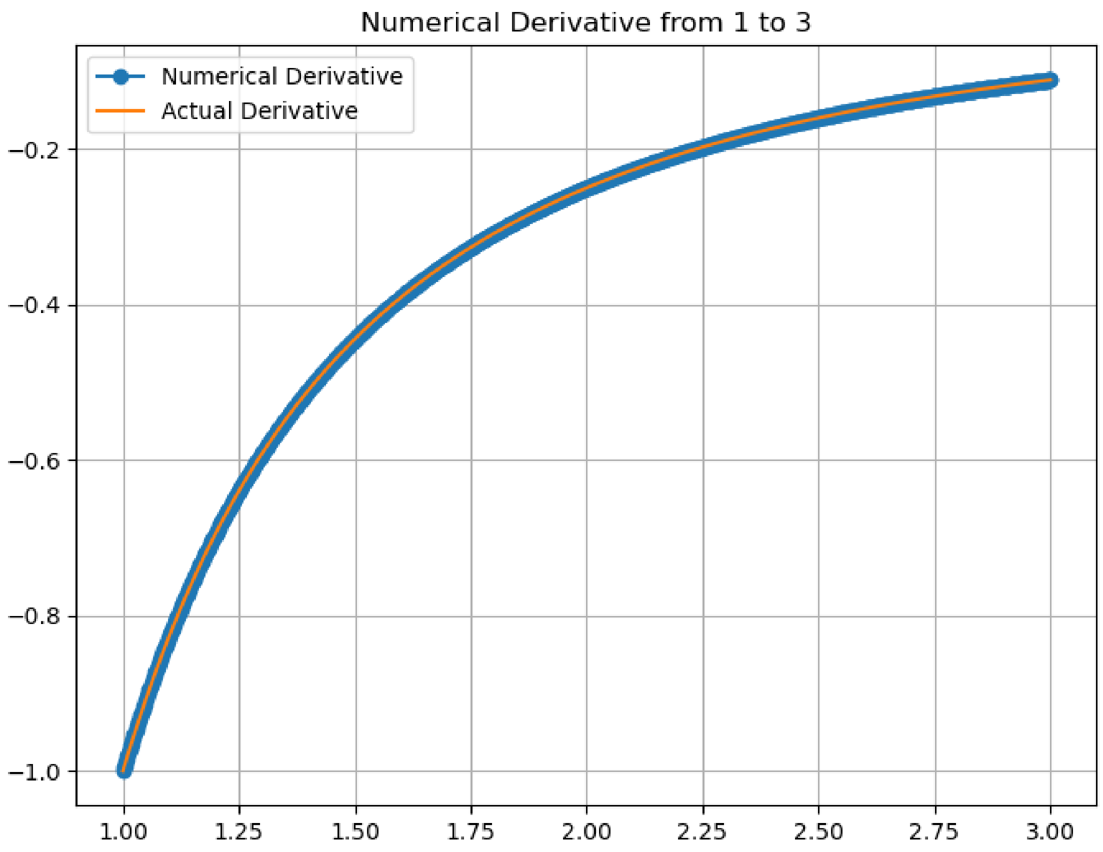

Numerical Derivative from 1 to 3
Numerical Derivative
Actual Derivative
25
20
15
10
5
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

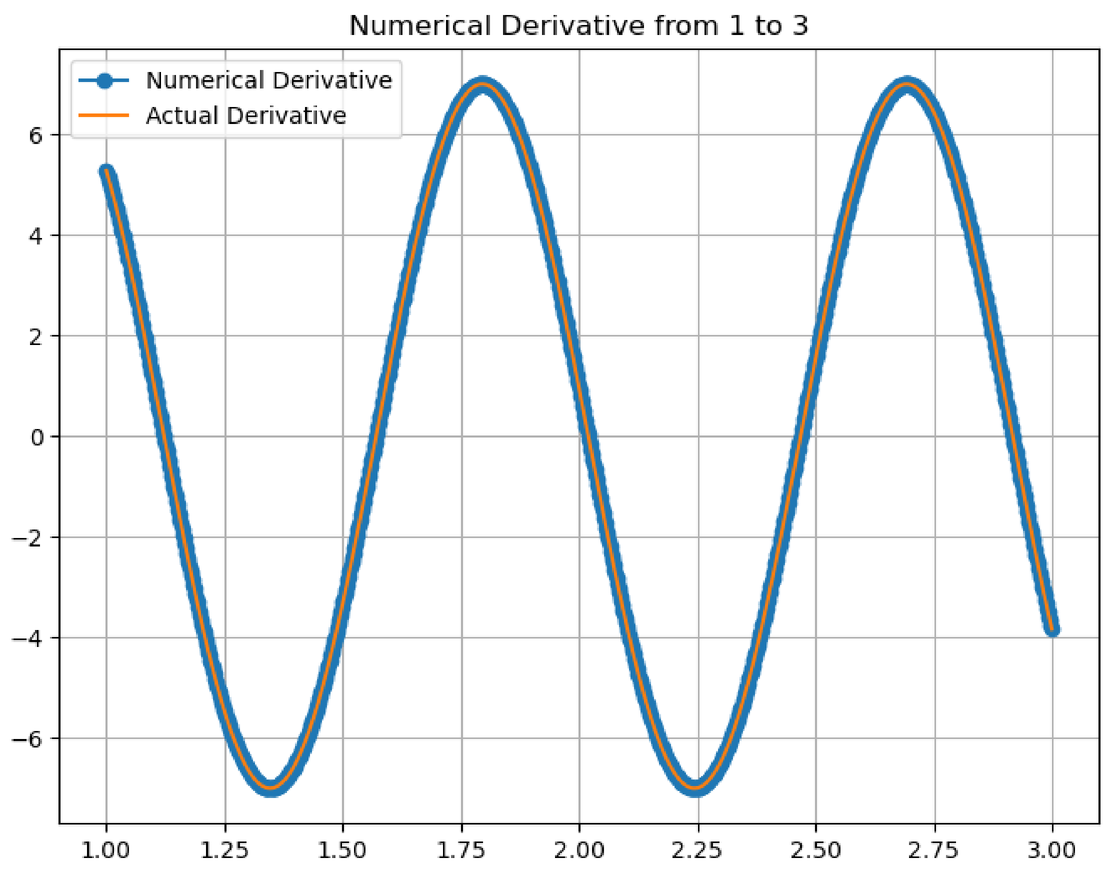


Finally I ran it again with 1000 intervals just to see how sin(7x) plays out. It's pretty close across the board.

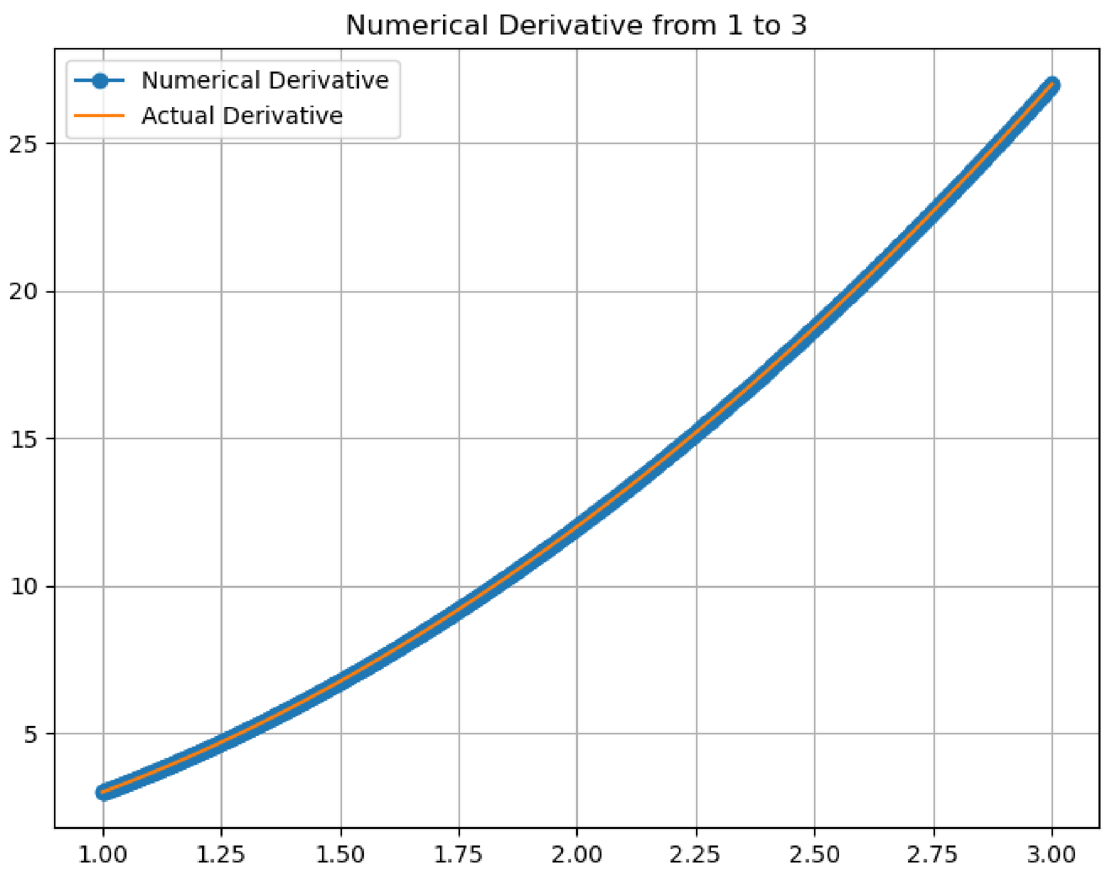
```
[5]: def plot_figure(x1,x2,num_points,f, f1):
         (xvals,yvals) = Numerical_Function_Derivative(x1,x2,num_points, f)
         actual_derivatives = [f1(x) for x in xvals]

         plt.figure(figsize=(8,6))
         plt.plot(xvals, yvals, label = 'Numerical Derivative', marker='o')
         plt.plot(xvals, actual_derivatives, label = 'Actual Derivative')
         plt.title("Numerical Derivative from " + str(x1) + " to " + str(x2))
         plt.grid(True)
         plt.legend()
         plt.show
     x1 = 1
     x2 = 3
     num_points = 1000
     funcs = [(a_f,a_f1), (b_f, b_f1), (c_f, c_f1), (d_f, d_f1)] #redo funcs to␣
      ↪include the actual derivatives
```

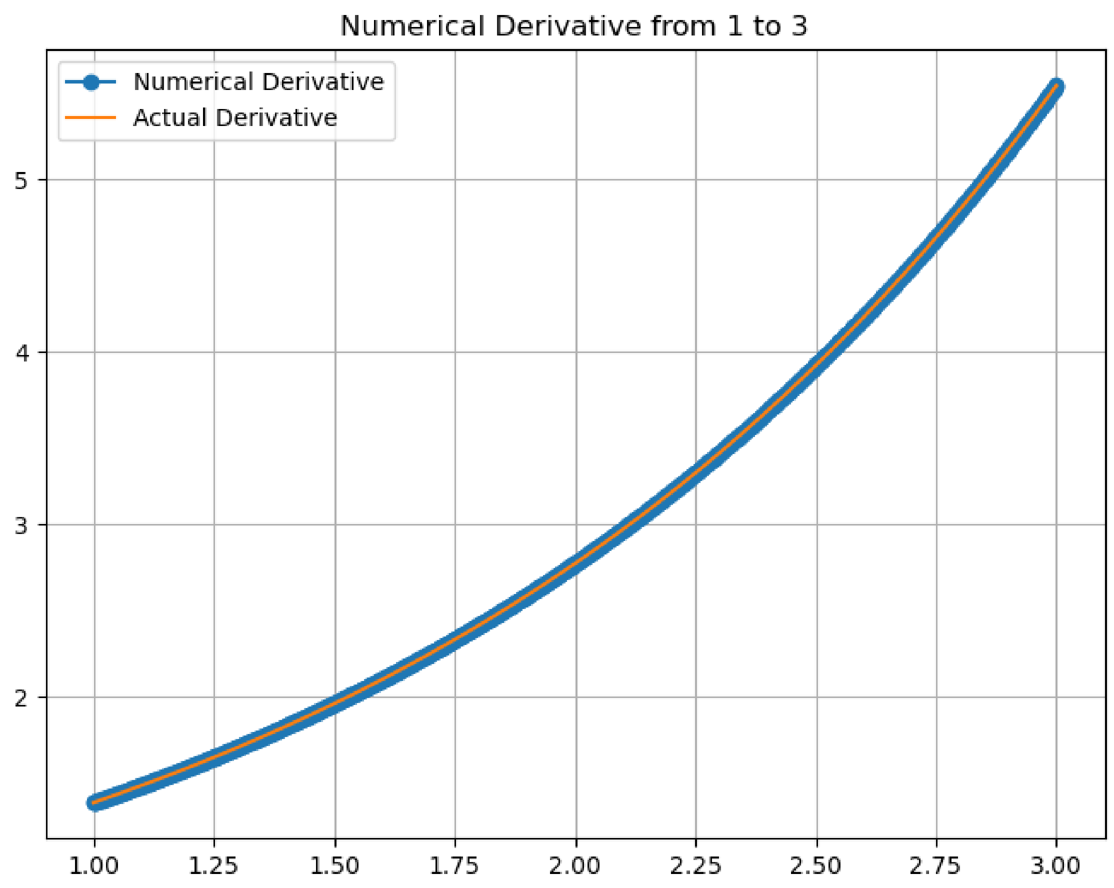
```python
for (f,f1) in funcs:
    plot_figure(x1,x2,num_points, f, f1)
```





Numerical Derivative from 1 to 3
Numerical Derivative
Actual Derivative
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


Numerical Derivative from 1 to 3
Numerical Derivative
Actual Derivative
25
20
15
10
5
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


Numerical Derivative from 1 to 3
Numerical Derivative
Actual Derivative
5
4
3
2
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00