Assignment 4

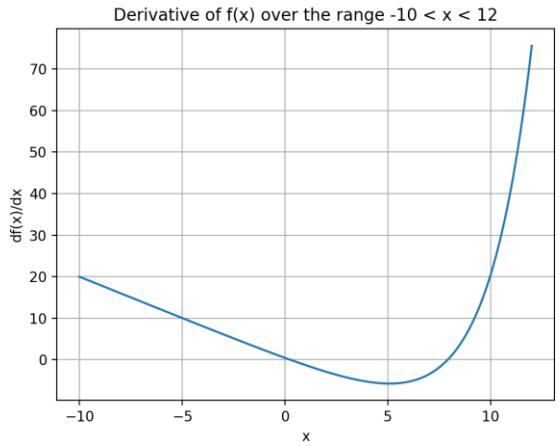
Omar Ahmed

UCID:30154382

Q1

```python
#Part 1
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

# Define the symbol
x = sp.symbols('x')

# Define the function
f = sp.exp(0.45*x) - x**2 + 5

# Differentiate the function
df = sp.diff(f, x)

# Lambdify the derivative for numerical calculations
df_lambdified = sp.lambdify(x, df, 'numpy')

# Plot df(x)/dx over the range -10 < x < 12
x_values = np.linspace(-10, 12, 400)
y_values = df_lambdified(x_values)

plt.plot(x_values, y_values)
plt.title("Derivative of f(x) over the range -10 < x < 12")
plt.xlabel("x")
plt.ylabel("df(x)/dx")
plt.grid(True)
plt.show()
```

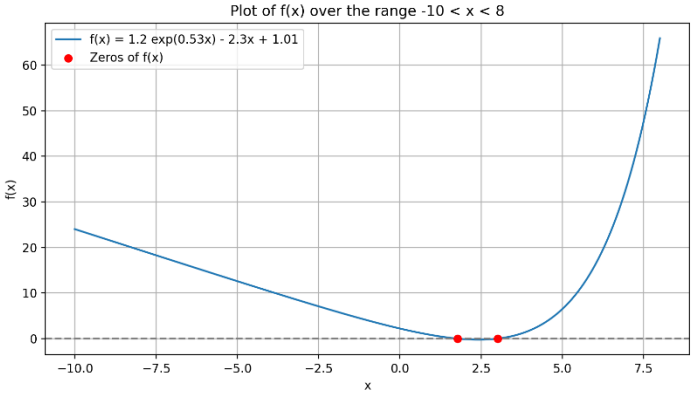
```python
#Part 2
import scipy.optimize as optimize
# Define a function for root finding based on the derivative
def df_numeric(x):
  return df_lambdified(x)
# Find roots (extremum points) using scipy.optimize.root_scalar
# The brackets are chosen based on the plot
root1 = optimize.root_scalar(df_numeric, bracket=[-5, 5]).root
root2 = optimize.root_scalar(df_numeric, bracket=[5, 10]).root
# Output the roots
root1, root2
```




```
(0.25202027083003, 7.910873522546913)
```

Q2

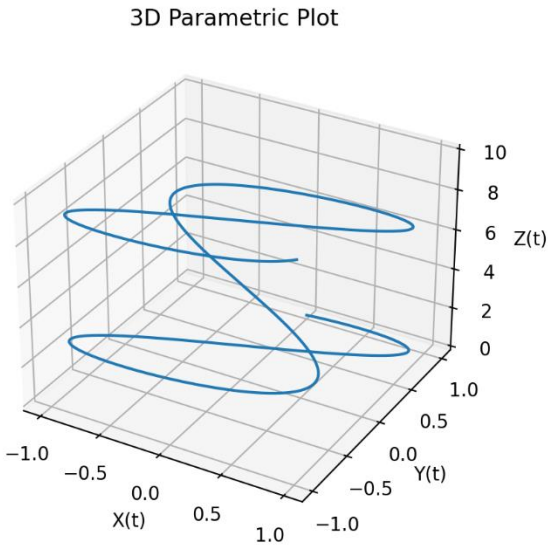
```python
# Define the symbol and the function
x = sp.symbols('x')
f_2 = 1.2 * sp.exp(0.53*x) - 2.3*x + 1.01

# Lambdify the function for numerical calculations
f_2_lambdified = sp.lambdify(x, f_2, 'numpy')

# Plot f_2(x) over the range -10 < x < 8
x_values_2 = np.linspace(-10, 8, 400)
y_values_2 = f_2_lambdified(x_values_2)
plt.figure(figsize=(10, 6))
plt.plot(x_values_2, y_values_2, label='f(x) = 1.2 exp(0.53x) - 2.3x + 1.01')
plt.axhline(0, color='gray', linestyle='--') # Adding a line at y=0 for reference
plt.title("Plot of f(x) over the range -10 < x < 8")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid(True)

# Define a function for root finding based on f_2
def f_2_numeric(x):
  return f_2_lambdified(x)

# Find zeros of the function
zero1 = optimize.root(f_2_numeric, -5).x[0]
zero2 = optimize.root(f_2_numeric, 5).x[0]

# Mark the zeros on the plot
plt.scatter([zero1, zero2], [0, 0], color='red', zorder=5, label='Zeros of f(x)')
plt.legend()
plt.show()
```



Q3

```python
#Question 3
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
from mpl_toolkits.mplot3d import Axes3D

# Define the parametric equations
def x(t): return np.sin(2*t)
def y(t): return np.cos(t)
def z(t): return t

# Create a 3D plot
t_values = np.linspace(0, 10, 1000)
x_values = x(t_values)
y_values = y(t_values)
z_values = z(t_values)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(x_values, y_values, z_values)
ax.set_xlabel('X(t)')
ax.set_ylabel('Y(t)')
ax.set_zlabel('Z(t)')
plt.title('3D Parametric Plot')
plt.show()
```

```python
# Define the arc length integrand
def arc_length_integrand(t):
  return np.sqrt((2*np.cos(2*t))**2 + (-np.sin(t))**2 + 1)

# Compute the arc length
arc_length, _ = quad(arc_length_integrand, 0, 10)
arc_length

# Rounding the computed arc length to three decimal places
arc_length_rounded = round(arc_length, 3)
arc_length_rounded
```



Q4

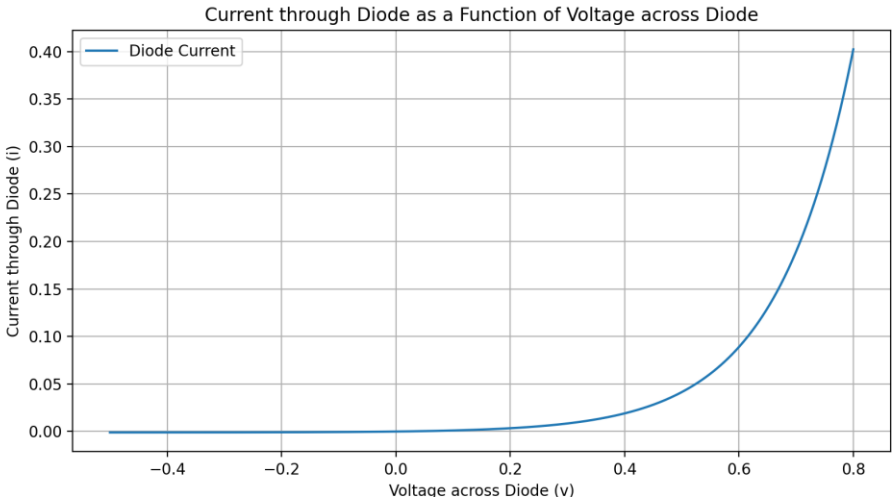
```python
#Question 4
from scipy.integrate import tplquad

# Define the integrand function
def integrand(x, y, z):
  return x * y * z**2

# Define the integration limits
# z goes from 0 to 3 (height of the prism)
# y goes from 0 to 1 - z/3, depending on z
# x goes from 0 to 1 - y, depending on y and z

# Perform the triple integral
result, error = tplquad(
  integrand, # The integrand function
  0, 3, # Limits for z
  lambda z: 0, lambda z: 1 - z/3, # Limits for y as functions of z
  lambda y, z: 0, lambda y, z: 1 - y # Limits for x as functions of y and z
)

# Result of the triple integration
result
result_rounded = round(result, 3)
result_rounded
```

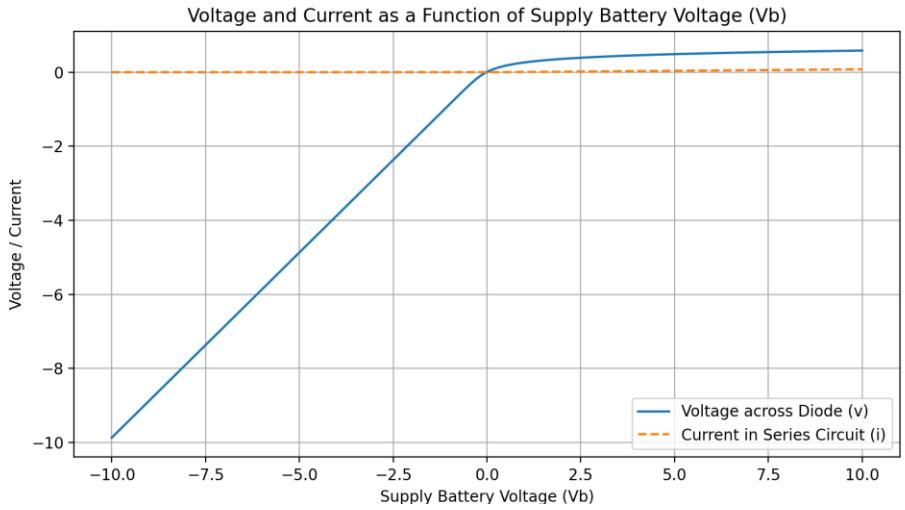
```
0.129
```

Q5

```python
#Question 5
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

# Constants
R = 120 # Resistance in ohms

# Define the function for the current-voltage relation of the diode
def diode_current(voltage):
  return 0.001 * (np.exp(7.5 * voltage) - 1.1)

# Define the function to find the root of
def func(v, Vb):
  i = diode_current(v)
  return Vb - (v + i * R)

# Plot the current through the diode i as a function of the voltage across the diode v
v_diode_plot = np.linspace(-0.5, 0.8, 400) # Voltage range where diode current changes significantly
i_diode_plot = diode_current(v_diode_plot)

plt.figure(figsize=(10, 6))
plt.plot(v_diode_plot, i_diode_plot, label='Diode Current')
plt.title('Current through Diode as a Function of Voltage across Diode')
plt.xlabel('Voltage across Diode (v)')
plt.ylabel('Current through Diode (i)')
plt.grid(True)
plt.legend()
plt.show()
```

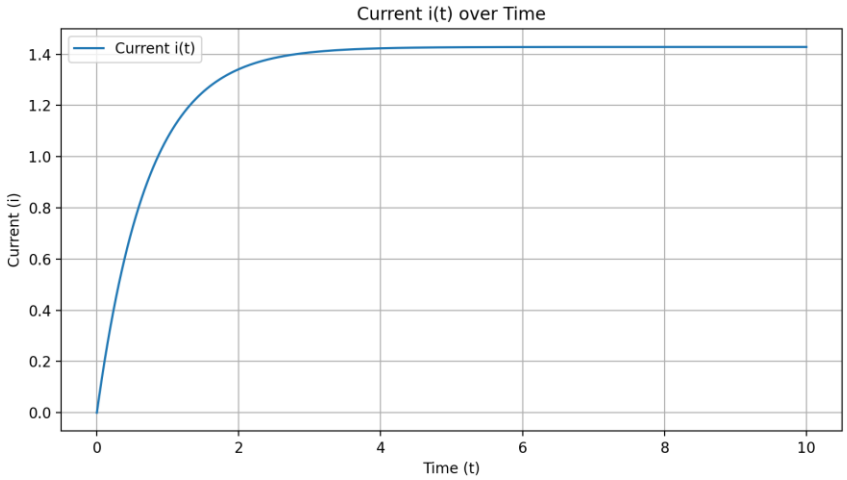
```python
# Plot the voltage across the diode and the current in the series circuit as Vb is varied
Vb_values = np.linspace(-10, 10, 400)
v_values = []
i_values = []

# Find the diode voltage and current for each battery voltage
for Vb in Vb_values:
  # Initial guess for the voltage across the diode
  v_guess = 0.65 if Vb > 0 else -0.65
  v_diode, = fsolve(func, v_guess, args=(Vb,))
  i_series = diode_current(v_diode)
  v_values.append(v_diode)
  i_values.append(i_series)

plt.figure(figsize=(10, 6))
plt.plot(Vb_values, v_values, label='Voltage across Diode (v)')
plt.plot(Vb_values, i_values, label='Current in Series Circuit (i)', linestyle='--')
plt.title('Voltage and Current as a Function of Supply Battery Voltage (Vb)')
plt.xlabel('Supply Battery Voltage (Vb)')
plt.ylabel('Voltage / Current')
plt.grid(True)
plt.legend()
plt.show()
```





Q6

```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Constants
L = 0.5 # Inductance in Henrys
R = 0.7 # Resistance in Ohms

# Define the DEQ (differential equation) to solve
def di_dt(i, t, L, R):
  v_t = 1 if t > 0 else 0 # Voltage source that switches from 0V to 1V at t=0
  return (-R/L) * i + (1/L) * v_t

# Time range to solve the DEQ
t = np.linspace(0, 10, 1000)

# Initial condition: i(0) = 0
i0 = [0]

# Solve the DEQ using scipy.integrate.odeint()
i_t = odeint(di_dt, i0, t, args=(L, R))

# Plot the current i(t)
plt.figure(figsize=(10, 6))
plt.plot(t, i_t, label='Current i(t)')
plt.title('Current i(t) over Time')
plt.xlabel('Time (t)')
plt.ylabel('Current (i)')
plt.grid(True)
plt.legend()
plt.show()
```



Q7

```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Constants for the mass-spring system
M = 1.3 # Mass in kilograms
k = 150 # Spring stiffness in N/m
g = 9.8 # Acceleration due to gravity in m/s^2

# Define the state space formulation of the second-order DEQ
def state_space(w, t, M, k):
  x, x_dot = w
  return [x_dot, -k/M * x + g]
# Initial conditions: x(0) = 0 (displacement), x_dot(0) = 0 (velocity)
w0 = [0, 0]
# Time range to solve the DEQ
t = np.linspace(0, 5, 1000)
# Solve the DEQ using scipy.integrate.odeint()
sol = odeint(state_space, w0, t, args=(M, k))
# Extracting displacement
x = sol[:, 0]

# Plot the weight displacement x(t)
plt.figure(figsize=(10, 6))
plt.plot(t, x, label='Displacement x(t)')
plt.title('Weight Displacement Over Time')
plt.xlabel('Time (t)')
plt.ylabel('Displacement (x)')
plt.grid(True)
plt.legend()
plt.show()
```