**MUSILA JOSHUA.**

**INTE/MG/2265/09/22.**

**Assignment INTE 316:**

**b)**

**i)** import numpy as np

from scipy.misc import derivative

# Define the function

def f(x):

return x\*\*3 + 2\*x\*\*2 + 3\*x + 4

# Calculate the derivative at a specific point

x\_point = 2.0

deriv = derivative(f, x\_point, dx=1e-6)

print(f"The derivative of f at x = {x\_point} is approximately {deriv}")

ii)

import numpy as np

from scipy.integrate import quad

# Define the function to integrate

def g(x):

return np.sin(x) \* np.exp(-x\*\*2)

# Integrate from 0 to infinity

integral, error = quad(g, 0, np.inf)

print(f"The integral of g from 0 to infinity is approximately {integral}")

**iii)**

import numpy as np

from scipy.optimize import curve\_fit

import matplotlib.pyplot as plt

# Example data points

x\_data = np.array([1, 2, 3, 4, 5])

y\_data = np.array([1.2, 1.9, 3.1, 3.9, 5.1])

# Define a polynomial function to fit

def poly(x, a, b, c):

return a \* x\*\*2 + b \* x + c

# Fit the curve

params, covariance = curve\_fit(poly, x\_data, y\_data)

# Plot the data and the fitted curve

plt.scatter(x\_data, y\_data, label='Data')

plt.plot(x\_data, poly(x\_data, \*params), label='Fitted Curve', color='red')

plt.legend()

plt.show()

print(f"Fitted parameters: {params}")

iv)

import numpy as np

import matplotlib.pyplot as plt

# Example data points

x = np.array([1, 2, 3, 4, 5])

y = np.array([2.1, 2.9, 3.6, 4.5, 5.1])

# Perform linear regression

A = np.vstack([x, np.ones(len(x))]).T

m, c = np.linalg.lstsq(A, y, rcond=None)[0]

# Plot the data and the fitted line

plt.scatter(x, y, label='Data')

plt.plot(x, m\*x + c, 'r', label='Fitted Line')

plt.legend()

plt.show()

print(f"Slope: {m}, Intercept: {c}")

**v**)

import numpy as np

from scipy.interpolate import CubicSpline

import matplotlib.pyplot as plt

# Example data points

x = np.array([0, 1, 2, 3, 4, 5])

y = np.array([0, 0.8, 0.9, 0.1, -0.8, -1.0])

# Create a cubic spline interpolation

cs = CubicSpline(x, y)

# Generate fine x points for plotting the spline

x\_fine = np.linspace(0, 5, 100)

y\_fine = cs(x\_fine)

# Plot the original data and the interpolated spline

plt.scatter(x, y, label='Data')

plt.plot(x\_fine, y\_fine, label='Cubic Spline', color='red')

plt.legend()

plt.show()

**c)**

﻿import numpy as np

# \_Given the exact points

X = np.Array([2.00, 4.25, 5.25, 7.81, 9.20, 10.60])

Y = np.Array([7.2, 7.1, 6.0, 5.0, 3.5, 5.0])

x\_target = 4.0

def linear\_interpolate(x0, y0, x1, y1, x):

go back y0 (y1 - y0) \* (x - x0) / (x1 - x0)

for i in variety(len(X) - 1):

if X[i] <= x\_target <= X[i 1]:

y\_target = linear\_interpolate(X[i], Y[i], X[i 1], Y[i 1], x\_target)

break

print(f'The interpolated fee of y at x = x\_target is y\_target:.2f')

**g**)

import numpy as np

def trapezoidal\_rule(f, a, b, n):

"""

Approximate the integral of f from a to b by the trapezoidal rule.

Parameters:

f: function to integrate

a: lower limit of integration

b: upper limit of integration

n: number of trapezoids

Returns:

float: approximation of the integral

"""

h = (b - a) / n

integral = 0.5 \* (f(a) + f(b))

for i in range(1, n):

integral += f(a + i \* h)

integral \*= h

return integral

# Example function to integrate

def f(x):

return x\*\*2

# Integration parameters

a = 0 # lower limit

b = 1 # upper limit

n = 100 # number of trapezoids

# Compute the integral

result = trapezoidal\_rule(f, a, b, n)

print(f"Approximate integral of f(x) from {a} to {b} using {n} trapezoids: {result}")