(B)Using PYTHON show how the

following is achieved(PRAC-

TICAL)

i.Differentiation

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import derivative

def f(x):
    return x**3 - 2*x**2 + 3*x - 1

x = np.linspace(-2, 3, 100)
y = f(x)
dy = derivative(f, x, dx=1e-6)

plt.figure(figsize=(10, 6))
plt.plot(x, y, label='f(x)')
plt.plot(x, dy, label="f'(x)")
plt.legend()
plt.title('Function and its Derivative')
plt.xlabel('x')
plt.ylabel('y')
```

```
plt.grid(True)
plt.show()
```

ii.Numerical integration

```
import numpy as np
from scipy import integrate

def f(x):
    return x**2

a, b = 0, 2  # integration limits
result, error = integrate.quad(f, a, b)

print(f"The integral of x^2 from {a} to {b} is approximately {result:.4f}")
print(f"The estimated error is {error:.4e}")
```

iii.Curve Fitting

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

def func(x, a, b, c):
    return a * np.exp(-b * x) + c
```

```
# Generate sample data
x = np.linspace(0, 4, 50)
y = func(x, 2.5, 1.3, 0.5) + 0.2 * np.random.normal(size=len(x))

# Fit the function
popt, _ = curve_fit(func, x, y)

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(x, y, label='data')
plt.plot(x, func(x, *popt), 'r-', label='fit: a=%5.3f, b=%5.3f, c=%5.3f' % tuple(popt))
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Curve Fitting Example')
plt.show()
```

iv. Linear Regression

```
    import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate sample data
x = np.array([1, 2, 3, 4, 5]).reshape((-1, 1))
y = np.array([2, 4, 5, 4, 5])
```

```
# Create and fit the model
model = LinearRegression()
model.fit(x, y)

# Make predictions
x_pred = np.array([0, 6]).reshape((-1, 1))
y_pred = model.predict(x_pred)

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(x, y, color='blue', label='Data')
plt.plot(x_pred, y_pred, color='red', label='Linear Regression')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Linear Regression Example')
plt.show()

print(f"Slope: {model.coef_[0]:.2f}")
print(f"Intercept: {model.intercept_:.2f}")
```

v. Spline Interpolation

```
    import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy.interpolate import CubicSpline

# Generate sample data
x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([1, 3, 3, 4, 2, 5])

# Create the cubic spline
cs = CubicSpline(x, y)

# Generate points for a smooth curve
x_smooth = np.linspace(0, 5, 200)
y_smooth = cs(x_smooth)

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(x, y, color='red', label='Data points')
plt.plot(x_smooth, y_smooth, label='Cubic Spline')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Cubic Spline Interpolation')
plt.grid(True)
plt.show()
```

(C) A smart robotic agent with a laser Scanner is doing a quick quality check on holes drilled in a rectangular plate. The centers of the hole in the plate describes the path the arm needs to take, and the hole centers are located on a Cartesian coordinate system as shown

X (in) 2.00 4.25 5.25 7.81 9.20 10.60

Y(in) 7.2 7.1 6.0 5.0 3.5 5.0

If the laser scanner is traversing from x=2.00 to x =4.25 in a linear path, what is the value of y at x= 4.0 using the linear spline formula , show how this problem can be solved using PYTHON (PRACTICAL)

```
import numpy as np
# Given data
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])

# Linear interpolation function
def linear_interpolation(x0, x1, y0, y1, x):
    return y0 + (x - x0) * (y1 - y0) / (x1 - x0)

# Find the appropriate segment
for i in range(len(x) - 1):
    if x[i] <= 4.0 <= x[i+1]:
        y_interpolated = linear_interpolation(x[i], x[i+1], y[i], y[i+1], 4.0)
        break

print(f"The interpolated y value at x = 4.0 is {y_interpolated}")
```

(G) Write a program to show how the trapezoidal rule of integration works in PYTHON

(PRACTICAL)

```
 import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x**2  # Example function: f(x) = x^2

def trapezoidal_rule(f, a, b, n):
    x = np.linspace(a, b, n+1)
    y = f(x)
    h = (b - a) / n

    integral = h * (0.5 * y[0] + 0.5 * y[-1] + np.sum(y[1:-1]))

    return integral, x, y

# Set integration limits and number of trapezoids
a, b = 0, 2
n = 10

# Calculate the integral using the trapezoidal rule
result, x, y = trapezoidal_rule(f, a, b, n)
```

```
# Plot the function and trapezoids
plt.figure(figsize=(10, 6))
plt.plot(x, y, 'b-', label='f(x) = x^2')
plt.fill_between(x, 0, y, alpha=0.3)
for i in range(len(x) - 1):
    plt.plot([x[i], x[i], x[i+1], x[i+1]], [0, f(x[i]), f(x[i+1]), 0], 'r-', linewidth=1)
plt.title(f'Trapezoidal Rule: ∫[{a},{b}] x^2 dx ≈ {result:.4f}')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()

print(f"Approximate integral: {result:.4f}")
print(f"Actual integral: {(b**3 - a**3) / 3:.4f}")
```