**Name: Phan Tran Thanh Huy**

**ID: ITCSIU22056**

# Lab 5

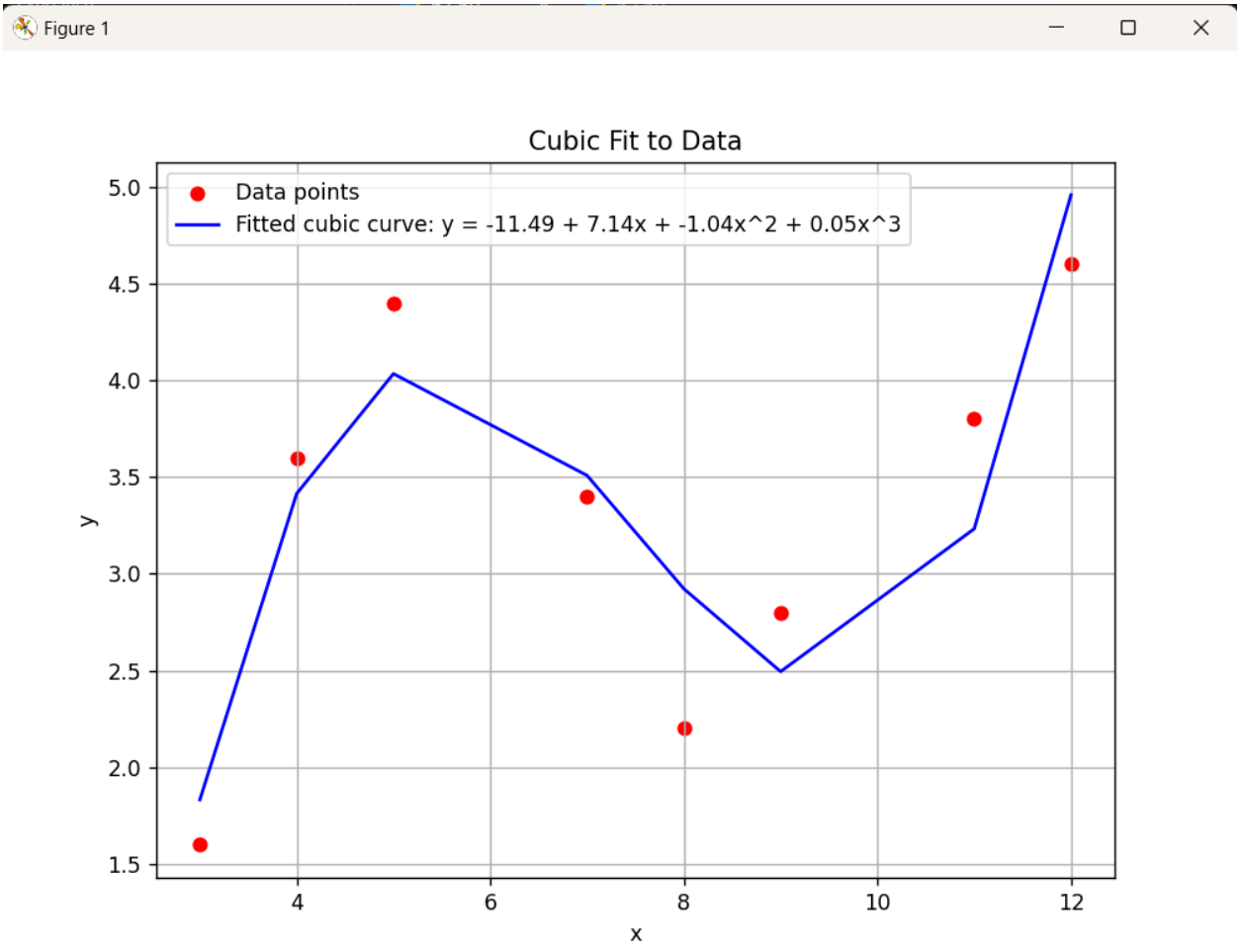
## Activity 1: Linear Regression

Code:

```python
import numpy as np

# Input data
x = np.array([3, 4, 5, 7, 8, 9, 11, 12])
y = np.array([1.6, 3.6, 4.4, 3.4, 2.2, 2.8, 3.8, 4.6])
n = len(x)

# Cubic polynomial: y = a + b*x + c*x^2 + d*x^3
X = np.column_stack((np.ones(n), x, x**2, x**3))

XtX = X.T @ X
Xty = X.T @ y
theta = np.linalg.inv(XtX) @ Xty

y_pred = X @ theta

# Calculate R² and Sy/x
ss_res = np.sum((y - y_pred) ** 2)
ss_tot = np.sum((y - np.mean(y)) ** 2)
r2 = 1 - ss_res / ss_tot

m = 4
syx = np.sqrt(ss_res / (n - m))

a, b, c, d = theta
print("Fitted cubic function: y = a + b*x + c*x^2 + d*x^3")
print(f"a = {a:.4f}")
print(f"b = {b:.4f}")
print(f"c = {c:.4f}")
print(f"d = {d:.4f}")
print(f"R² = {r2:.4f}")
print(f"Standard Error (Sy/x) = {syx:.4f}")

```

Result:

```
PS E:\Homework\TMC\Lab5> & C:/Python312/python.exe e:/Homework/TMC/Lab5/p1.py
Coefficients:
a = 0.04667601649405476
b = -1.0412069207164207
c = 7.143817219173287
d = -11.488707178897203
Fitted cubic equation: ax^3 + bx^2 + cx + d
a = 0.0467
b = -1.0412
c = 7.1438
d = -11.4887
R² = 0.8290
Standard Error Sy/x = 0.5700
```



## Activity 2: Non-linear Regression

Code:

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

# Data points
x_data = np.array([0.1, 0.2, 0.4, 0.6, 0.9, 1.3, 1.5, 1.7, 1.8])
y_data = np.array([0.75, 1.25, 1.45, 1.25, 0.85, 0.55, 0.35, 0.28, 0.18])

# Model function
def model(x, alpha, beta):
    return alpha * x * (1 - np.exp(beta * x))

def compute_jacobian(x, alpha, beta):
    df_dalpha = x * (1 - np.exp(beta * x))
    df_dbeta = alpha * x**2 * np.exp(beta * x)
    return np.vstack((df_dalpha, df_dbeta)).T

# Residual vector D
def compute_residuals(x, y, alpha, beta):
    return y - model(x, alpha, beta)

# Jacobi method for solving Ax = b
def jacobi(A, b, tol=1e-6, max_iter=100):
    n = len(b)
    x = np.zeros_like(b)
    for it in range(max_iter):
        x_new = np.zeros_like(x)
        for i in range(n):
            s = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - s) / A[i, i]
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            return x_new
        x = x_new
    return x

# Nonlinear regression using Gauss-Newton style iterations
def nonlinear_fit(x, y, alpha_init, beta_init, iterations=10):
    alpha, beta = alpha_init, beta_init

    for i in range(iterations):
        D = compute_residuals(x, y, alpha, beta)
        Z = compute_jacobian(x, alpha, beta)
        ZT_Z = Z.T @ Z
        ZT_D = Z.T @ D

        delta = jacobi(ZT_Z, ZT_D)

        alpha += delta[0]
        beta += delta[1]

    return alpha, beta

# Initial guesses
alpha0 = 1.0
beta0 = -1.0

# Run the fitting
alpha_est, beta_est = nonlinear_fit(x_data, y_data, alpha0, beta0)
print(f"\nEstimated parameters:\n  α4 = {alpha_est:.5f}\n  β4 = {beta_est:.5f}")

# Plot the results
x_plot = np.linspace(min(x_data), max(x_data), 100)
y_plot = model(x_plot, alpha_est, beta_est)

plt.scatter(x_data, y_data, color='red', label='Data points')
plt.plot(x_plot, y_plot, color='blue', label=f'Fitted curve: y = {alpha_est:.3f}x(1 - exp({beta_est:.3f}x))')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Nonlinear Regression Fit')
plt.legend()
plt.show()
```

Result:

```
Estimated parameters:
  α4 = -1.52762
  β4 = 5.96930
```