MACHINE EXERCISE 3: GENERAL LEAST SQUARES

As a student of the University of the Philippines, I pledge to act ethically and uphold the value of honor and excellence.

I understand that suspected misconduct on this Assignment will be reported to the appropriate office and if established, will result in disciplinary action in accordance with University rules, policies and procedures. I may work with others only to the extent allowed by the Instructor.

Name: Jeryl Salas
Student Number: 202321128

Write a MATLAB code that generates the *Nth* degree polynomial least squares matrix.

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 & \cdots & \sum x_i^N \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \cdots & \sum x_i^{N+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \cdots & \sum x_i^{N+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum x_i^N & \sum x_i^{N+1} & \sum x_i^{N+2} & \cdots & \sum x_i^{2N} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \\ \vdots \\ \sum x_i^N y_i \end{bmatrix}$$

INPUT:

1. **(x,y) dataset**
2. **Degree of polynomial**

OUTPUT:

Least squares matrix

CODE:

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


def conjugate_gradient(A, B, x_0, max_iterations=1000, tolerance=1e-8):
    # We use conjugate gradient from our second lesson to find the coefficient values to be placed in
matrix x
    x = x_0
    r = B - np.dot(A, x)
    p = r
    rsold = np.dot(r, r)

    for i in range(max_iterations):
        Ap = np.dot(A, p)
        alpha = rsold / np.dot(p, Ap)
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = np.dot(r, r)

        if np.linalg.norm(r) < tolerance: # if the norm of r is lower than tolerance, we break the loop
```

```
            print(f"Convergence achieved in {i+1} iterations using conjugate gradient.")
            return x

        beta = rsnew / rsold # compute for beta and p
        p = r + beta * p
        rsold = rsnew # use the new r for the next iter

    print("Maximum iterations reached without convergence.")
    return x # reutrn matrix x

def generate_polynomial_matrix(x_values, y_values, degree):
    x_points = len(x_values)
    A = []
    B = []
    x = []


    for N in range(degree + 1):
        # For building of A matrix
        row = []
        for i in range(degree + 1):
            if N == 0 and i == 0: # this is the N element in the [0,0] portion of matrix A
                row.append(x_points)
            else:
                sum_x = sum(x ** (N + i) for x in x_values)
                row.append(sum_x)

        A.append(row)

        # For building of B matrix
        if N == 0:
            sum_y = sum(y for y in y_values)
            B.append(sum_y)
        else:
            sum_xy = 0
            for x, y in zip(x_values, y_values):
                sum_xy += (x**N) * y
            B.append(sum_xy)

    # For building of x matrix
    A = np.array(A)
    B = np.array(B)
    x_0 = np.random.rand(degree + 1) # Initializing x_0 values for faster convergence in conjugate
gradient

    x = conjugate_gradient(A, B, x_0)

    return A, B, x # Return matrix A, B, and x

def format_matrix(matrix):
    formatted_matrix = np.array(matrix)  # Convert to NumPy array for formatting
    formatted_matrix = np.around(formatted_matrix, decimals=4)  # Limit decimals to 4 places
    return formatted_matrix

def polynomial_fit(x_matrix, degree):
    x = sp.symbols('x')

    for d in range(degree):
        if d == 0:
            y = x_matrix[0]
```

```python
        else:
            y += x_matrix[d]*x**d
    return y

def main():
    # Input for degree of polynomial
    degree = int(input("Enter the degree of the polynomial: "))

    # Input for x, y dataset
    num_points = int(input("Enter the number of data points: "))
    x_data = []
    y_data = []
    for i in range(num_points):
        x = float(input(f"Enter x{i+1}: "))
        y = float(input(f"Enter y{i+1}: "))
        x_data.append(x)
        y_data.append(y)

    x_values = x_data
    y_values = y_data

    A, B, x = generate_polynomial_matrix(x_values, y_values, degree)
    A = format_matrix(A)

    # Removing scientific notations when printing
    np.set_printoptions(suppress=True)

    print(f"\nGenerated {degree} degree polynomial least squares matrix:")
    print("\033[1mA matrix: \033[0m")
    print(A, "\n")
    print("\033[1mx matrix: \033[0m")
    print(x, "\n")
    print("\033[1mB matrix: \033[0m")
    print(B, "\n")
    print("\033[1my equation: \033[0m")
    y = polynomial_fit(x, degree)
    print(y, "\n")

    # We create the polynomial curve using the coefficients created from matrix x
    poly_curve = np.poly1d(np.flip(x))

    # We first use the scatterplot
    plt.figure(figsize=(8, 6))
    plt.scatter(x_values, y_values, label='Data Points')

    # Plot the curve
    x_curve = np.linspace(min(x_values), max(x_values), 100)  # X values for the curve
    y_curve = poly_curve(x_curve)  # Corresponding Y values for the curve
    plt.plot(x_curve, y_curve, label='Polynomial Curve', color='red')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Polynomial Fit')
    plt.legend()
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    main()
```

INPUT:

```
Enter the degree of the polynomial: 2
Enter the number of data points: 5
Enter x1: 1
Enter y1: 3.2939
Enter x2: 2
Enter y2: 4.2699
Enter x3: 4
Enter y3: 7.1749
Enter x4: 5
Enter y4: 9.3008
Enter x5: 8
Enter y5: 20.259
```

OUTPUT:

A matrix:

```
[[   5.    20.   110.]
 [  20.   110.   710.]
 [ 110.   710. 4994.]]
```

x matrix:

```
[ 3.39936    -0.25677333  0.29488333]
```

B matrix:

```
[  44.2985  249.1093 1664.2679]
```

y equation:

```
3.39935999999999 - 0.256773333333328*x
```