Exercise 1

Nicolás Rojo Esteban

February 10, 2026

1 Theoretical Background

Polynomial regression is a statistical technique used to model the relationship between an independent variable x and a dependent variable y by fitting a polynomial function of degree n. This method extends linear regression by allowing the model to capture nonlinear relationships within the data. The general form of a polynomial regression model is:

$$y = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n + \varepsilon \tag{1}$$

where $c_0, c_1, ..., c_n$ are the polynomial coefficients, and ε represents the error term.

Least Squares Method

The least squares method is a mathematical approach used to find the best-fitting polynomial curve for a given set of data points. The goal is to determine the polynomial coefficients that minimize the total squared error between the predicted values and the observed data.

The error (or residual) for each data point is defined as:

$$\varepsilon_i = y_i - P(x_i) \tag{2}$$

To find the optimal coefficients, we minimize the sum of squared residuals:

$$S = \sum_{i=1}^{m} (y_i - P(x_i))^2 \tag{3}$$

This ensures that the chosen polynomial is as close as possible to the actual data points in terms of least squares error.

Expanding S:

$$S = \sum_{i=1}^{m} (y_i - (c_0 + c_1 x_i + c_2 x_i^2 + \cdots + c_n x_i^n))^2. \tag{4}$$

Taking the derivative with respect to c_k:

$$\frac{\partial S}{\partial c_k} = \sum_{i=1}^{m} 2(y_i - P(x_i)) \cdot \frac{\partial}{\partial c_k}(y_i - P(x_i)). \tag{5}$$

Since y_i is a constant, its derivative is zero. The derivative of $-P(x_i)$ with respect to c_k is simply $-x_i^k$, as only the term $c_k x_i^k$ depends on c_k:

$$\frac{\partial S}{\partial c_k} = \sum_{i=1}^{m} 2(y_i - P(x_i))(-x_i^k). \tag{6}$$

Rearranging:

$$\frac{\partial S}{\partial c_k} = -2\sum_{i=1}^{m} x_i^k (y_i - P(x_i)). \tag{7}$$

To minimize S, we set the derivative to zero:

$$-2\sum_{i=1}^{m} x_i^k (y_i - P(x_i)) = 0. \tag{8}$$

Dividing by -2:

$$\sum_{i=1}^{m} x_i^k y_i = \sum_{i=1}^{m} x_i^k P(x_i). \tag{9}$$

Substituting $P(x_i)$:

$$\sum_{i=1}^{m} x_i^k y_i = \sum_{i=1}^{m} x_i^k \left(c_0 + c_1 x_i + c_2 x_i^2 + \cdots + c_n x_i^n\right). \tag{10}$$

Expanding the sum:

$$\sum_{i=1}^{m} x_i^k y_i = c_0 \sum_{i=1}^{m} x_i^k + c_1 \sum_{i=1}^{m} x_i^{k+1} + c_2 \sum_{i=1}^{m} x_i^{k+2} + \cdots + c_n \sum_{i=1}^{m} x_i^{k+n}. \tag{11}$$

This results in a system of normal equations, which can be solved to determine the polynomial coefficients c_k.

This can be rewritten in matrix form as:

$$Ac = b \tag{12}$$

where:

- A is the design matrix given by:

$$A = \begin{bmatrix} m & \sum x_i & \sum x_i^2 & \dots & \sum x_i^n \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \dots & \sum x_i^{n+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \dots & \sum x_i^{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum x_i^n & \sum x_i^{n+1} & \sum x_i^{n+2} & \dots & \sum x_i^{2n} \end{bmatrix} \tag{13}$$

- c is the coefficient vector, containing the unknown polynomial coefficients:

$$c = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \tag{14}$$

- b is the right-hand side vector, containing the summation of products of y and powers of x:

$$b = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \\ \vdots \\ \sum x_i^n y_i \end{bmatrix} \tag{15}$$

Gauss-Jordan Elimination

To solve the system of equations $Ac = b$, the Gauss-Jordan elimination method is employed. This algorithm transforms the augmented matrix $[A|b]$ into its reduced row echelon form through a series of row operations. The process includes selecting a pivot element, normalizing the pivot row, and performing row reductions to eliminate nonzero elements in the pivot column. Once the matrix is in echelon form, the solution vector c is directly obtained.

A system of n linear equations with n unknowns can be written in matrix form as:

$$Ac = b \tag{16}$$

where A is an $n \times n$ coefficient matrix, c is the column vector of unknowns, and b is the column vector of constants.

To solve for c, we construct the augmented matrix $[A|b]$:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right] \tag{17}$$

The Gauss-Jordan elimination method transforms this matrix into the reduced row echelon form, where the left part becomes an identity matrix:

$$\left[\begin{array}{cccc|c} 1 & 0 & \dots & 0 & c_1 \\ 0 & 1 & \dots & 0 & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & c_n \end{array}\right] \tag{18}$$

Once in this form, the solution vector c can be read directly from the last column.

Coefficient of Determination (R^2)

The coefficient of determination (R^2) measures the goodness-of-fit of the polynomial model. It is computed as:

$$R^2 = 1 - \frac{SS_{\text{residual}}}{SS_{\text{total}}} \tag{19}$$

where SS_{residual} represents the sum of squared residuals, and SS_{total} is the total sum of squares. An R^2 value close to 1 indicates a strong fit between the model and the observed data.

2 Code:

[1]:

```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display, Math
```

[2]:

```
class RegressionAnalysis:
    def __init__(self, filename):
        """
        Initializes the class, first asks for the delimiter and header rows,␣
↪then loads the data,
        asks for the degree, and finally solves the system.
        """
        self.delimiter = input("Enter the delimiter used in the file (comma ',',␣
↪semicolon ';', tab '\\t', etc.): ").strip()

        while True:
            try:
                self.skiprows = int(input("Enter the number of header rows to␣
↪skip (e.g., 1 if there is a row with column names): ").strip())
                if self.skiprows >= 0:
                    break
                else:
                    print("The number of header rows cannot be negative.")
            except ValueError:
                print("Please enter a valid integer.")

        self.data_x, self.data_y = self.load_csv_data(filename)
        while True:
            try:
                self.degree = int(input("Up to what degree do you want to create the matrix (0-5): ").strip())
                if 0 <= self.degree <= 5:
                    break
                else:
                    print("Error: The degree must be between 0 and 5.")
            except ValueError:
                print("Error: Please enter a valid integer.")

        self.augmented_matrix = self.set_augmented_matrix(self.degree)
        self.coefficients = self.gauss_jordan(self.augmented_matrix)[:, -1]

        self.coefficients_np = np.polyfit(self.data_x, self.data_y, self.degree)
        self.coefficients_np = self.coefficients_np[::-1]

        self.equation = r"$y = "
        for i, coef in enumerate(self.coefficients):
            sign = " + " if coef >= 0 and i > 0 else " - " if coef < 0 else ""
            if i == 0:
                self.equation += f"{sign}{abs(coef):.4f}"
            elif i == 1:
                self.equation += f"{sign}{abs(coef):.4f}x"
            else:
                self.equation += f"{sign}{abs(coef):.4f}x^{{{i}}}"
        self.equation += "$"

        self.equation_np = r"$y = "
        for i, coef in enumerate(self.coefficients_np):
            sign = " + " if coef >= 0 and i > 0 else " - " if coef < 0 else ""
            if i == 0:
                self.equation_np += f"{sign}{abs(coef):.4f}"
            elif i == 1:
                self.equation_np += f"{sign}{abs(coef):.4f}x"
            else:
                self.equation_np += f"{sign}{abs(coef):.4f}x^{{{i}}}"
        self.equation_np += "$"

        self.r2 = self.calculate_r2(self.evaluate_polynomial, self.coefficients)
        self.r2_np = self.calculate_r2(self.evaluate_polynomial, self.coefficients_np)

    def load_csv_data(self, filename):
        """
        Loads data from a CSV file with the specified delimiter and header rows.
        """
        try:
            data = np.loadtxt(filename, delimiter=self.delimiter, skiprows=self.skiprows)

            if data.shape[1] < 2:
                raise ValueError("The file must have at least two columns.")

            print("Data successfully loaded.")
            return data[:, 0], data[:, 1]

        except FileNotFoundError:
            print(f"Error: The file '{filename}' was not found.")
            exit()
        except ValueError as e:
            print(f"Error: {e}")
            exit()
        except Exception as e:
            print(f"Unexpected error: {e}")
            exit()

    def set_augmented_matrix(self, degree):
        """
        Constructs the augmented matrix for polynomial regression.
        """
        A = np.zeros((degree + 1, degree + 1))
        b = np.zeros((degree + 1, 1))

        for i in range(degree + 1):
            for j in range(degree + 1):
                A[i, j] = np.sum(self.data_x ** (i + j))
            b[i] = np.sum(self.data_y * (self.data_x ** i))

        augmented_matrix = np.hstack((A, b))
        return augmented_matrix

    def gauss_jordan(self, matrix):
        """
        Applies the Gauss-Jordan method to solve a system of equations.
        """
        matrix = matrix.astype(float)
        n = len(matrix)

        for i in range(n):
            if matrix[i][i] == 0:
                print("Error: Zero pivot found, the system may not have a unique solution.")
                exit()

            matrix[i] = matrix[i] / matrix[i][i]

            for j in range(n):
                if i != j:
                    factor = matrix[j][i]
                    matrix[j] = matrix[j] - matrix[i] * factor

        return matrix

    def display_data(self):
        """
        Displays the loaded data, augmented matrix, and the fitted polynomial␣
↪coefficients.
        """
        np.set_printoptions(suppress=True, precision=6)

        print("\nCurrent data:")
        print("x:", self.data_x)
        print("y:", self.data_y)
        print("\nAugmented matrix:")
        print(self.augmented_matrix)
        print("\nFitted polynomial coefficients:")
        display(Math(self.equation))
        print(f"\nCoefficient of determination:")
        display(Math(f"$R^{{2}} = {self.r2:.4f}$"))
        print("\nFitted polynomial coefficients (calculated with numpy):")
        display(Math(self.equation_np))
        print(f"\nCoefficient of determination (calculated with numpy):")
        display(Math(f"$R^{{2}} = {self.r2_np:.4f}$"))

    def evaluate_polynomial(self, x, coefficients):
        """
        Evaluates the fitted polynomial at a given point x using the provided␣
↪coefficients.
        """
        return sum(coefficients[i] * (x ** i) for i in range(len(coefficients)))

    def calculate_r2(self, eval_func, coefficients):
        """
        Calculates the coefficient of determination R^2 to evaluate the model␣
↪fit.
        """
        y_poly = np.array([eval_func(x, coefficients) for x in self.data_x])
        y_mean = np.mean(self.data_y)
        # Sum of squares
        ss_total = np.sum((self.data_y - y_mean) ** 2)
        ss_residual = np.sum((self.data_y - y_poly) ** 2)

        # Calculate R^2
        r2 = 1 - (ss_residual / ss_total)
        return r2

    def plot_fit(self):
        """
        Plots the original data and the polynomial fit curve.
        """
        x_fit = np.linspace(min(self.data_x), max(self.data_x), 1000)
        y_fit = np.array([self.evaluate_polynomial(x, self.coefficients) for x␣
↪in x_fit])
        y_fit_np = np.array([self.evaluate_polynomial(x, self.coefficients_np)␣
↪for x in x_fit])

        plt.figure(figsize=(10, 6))
        plt.scatter(self.data_x, self.data_y, color='red', label='Original data')
        plt.plot(x_fit, y_fit, color='blue', label=f'Polynomial degree {self.
↪degree} (My code)')
        plt.plot(x_fit, y_fit_np, color='orange', linestyle='--',␣
↪label=f'Polynomial degree {self.degree} (numpy.polyfit)')


        legend = plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

        x_text, y_text = 0.70, 0.70  # Ajuste relativo a la figura

        plt.figtext(x_text, y_text,
                    f"Equation (My code):\n{self.equation}\n$R^2 = {self.r2:.
↪4f}$",
                    fontsize=7, bbox=dict(facecolor='white', alpha=0.6),␣
↪ha="left")

        plt.figtext(x_text, y_text - 0.1,
                    f"Equation (numpy.polyfit):\n{self.equation_np}\n$R^2 =␣
↪{self.r2_np:.4f}$",
                    fontsize=7, bbox=dict(facecolor='white', alpha=0.6),␣
↪ha="left")

        plt.xlabel("X")
        plt.ylabel("Y")
        plt.title("Polynomial Regression Fit")
        plt.grid(True)
        plt.tight_layout()
        plt.show()
```

[3]:

```
file = "datos_5.csv"
mol = RegressionAnalysis(file)
mol.display_data()
mol.plot_fit()
```

```
Enter the delimiter used in the file (comma ',', semicolon ';', tab '\t', etc.):
,
Enter the number of header rows to skip (e.g., 1 if there is a row with column
names):  1
```

```
Data successfully loaded.
```

```
Up to what degree do you want to create the matrix (0-5):  3
```

```
Current data:
x: [0. 1. 2. 3. 4. 5.]
y: [0. 2. 4. 1. 0. 1.]

Augmented matrix:
[[    6.    15.    55.   225.     8.]
 [   15.    55.   225.   979.    18.]
 [   55.   225.   979.  4425.    52.]
 [  225.   979.  4425. 20515.   186.]]

Fitted polynomial coefficients:
```

$y = -0.2540 + 5.3360x - 2.4563x^2 + 0.2870x^3$

```
Coefficient of determination:
```

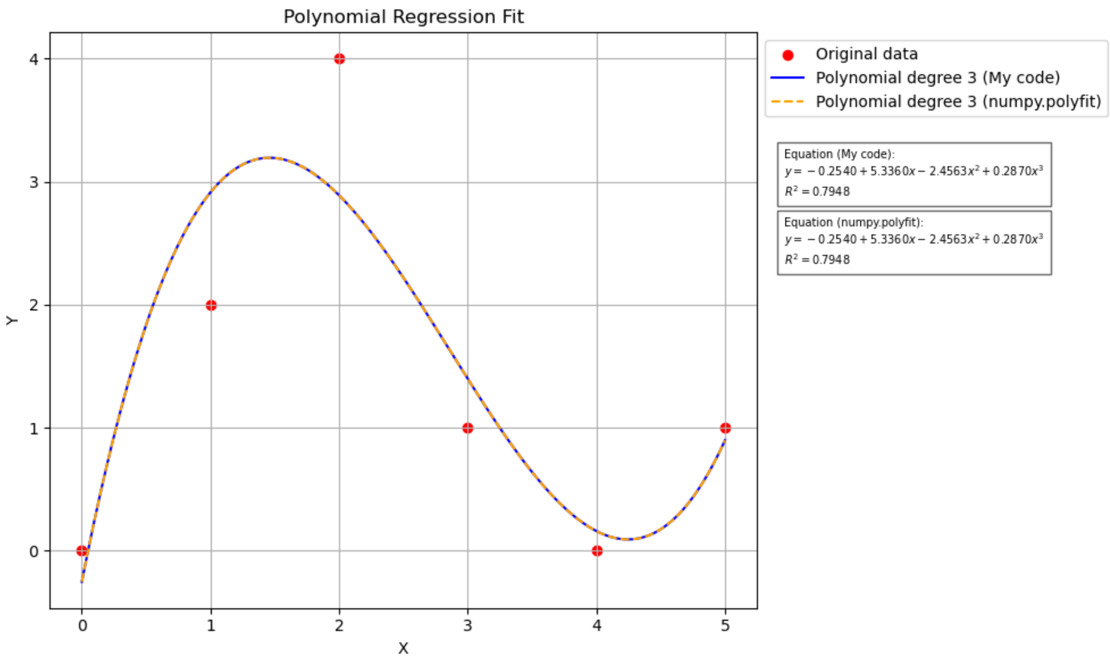
$R^2 = 0.7948$

```
Fitted polynomial coefficients (calculated with numpy):
```

$y = -0.2540 + 5.3360x - 2.4563x^2 + 0.2870x^3$

```
Coefficient of determination (calculated with numpy):
```

$R^2 = 0.7948$



[4]:
```
file = "datos_6.csv"
mol = RegressionAnalysis(file)
mol.display_data()
mol.plot_fit()
```

```
Enter the delimiter used in the file (comma ',', semicolon ';', tab '\t', etc.):
,
Enter the number of header rows to skip (e.g., 1 if there is a row with column
names):  1

Data successfully loaded.

Up to what degree do you want to create the matrix (0-5):  3


Current data:
x: [1. 2. 3. 5. 6.]
y: [1. 4. 4. 1. 2.]

Augmented matrix:
[[    5.    17.    75.   377.    12.]
 [   17.    75.   377.  2019.    38.]
 [   75.   377.  2019. 11177.   150.]
 [  377.  2019. 11177. 63075.   698.]]

Fitted polynomial coefficients:
```

$$y = -7.2353 + 11.3683x - 3.4622x^2 + 0.3039x^3$$

```
Coefficient of determination:
```

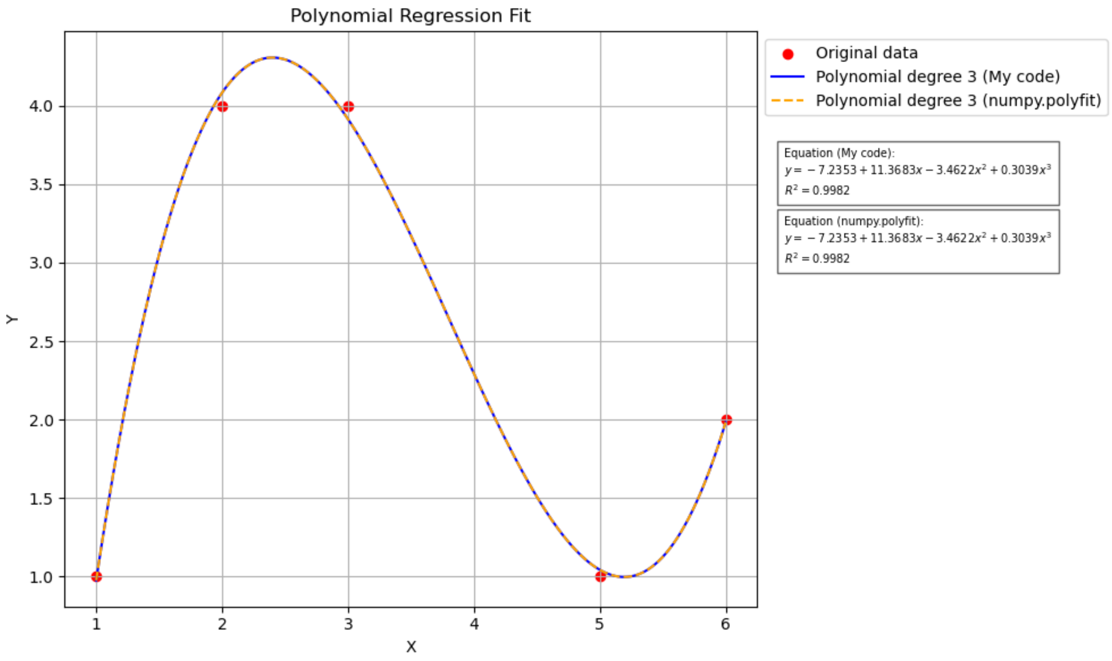
$$R^2 = 0.9982$$

```
Fitted polynomial coefficients (calculated with numpy):
```

$$y = -7.2353 + 11.3683x - 3.4622x^2 + 0.3039x^3$$

```
Coefficient of determination (calculated with numpy):
```

$$R^2 = 0.9982$$



[5]:
```python
file = "datos_7.csv"
mol = RegressionAnalysis(file)
mol.display_data()
mol.plot_fit()
```

```
Enter the delimiter used in the file (comma ',', semicolon ';', tab '\t', etc.):
,
Enter the number of header rows to skip (e.g., 1 if there is a row with column
names):  1

Data successfully loaded.

Up to what degree do you want to create the matrix (0-5):  3,

Error: Please enter a valid integer.
```

```
Up to what degree do you want to create the matrix (0-5):  3


Current data:
x: [-3. -1.  0.  1.  2.]
y: [4. 1. 0. 2. 5.]

Augmented matrix:
[[   5.   -1.   15.  -19.   12.]
 [  -1.   15.  -19.   99.   -1.]
 [  15.  -19.   99. -211.   59.]
 [ -19.   99. -211.  795.  -67.]]

Fitted polynomial coefficients:
```

$$y = 0.4313 + 0.3518x + 0.7992x^2 + 0.0943x^3$$

```
Coefficient of determination:
```

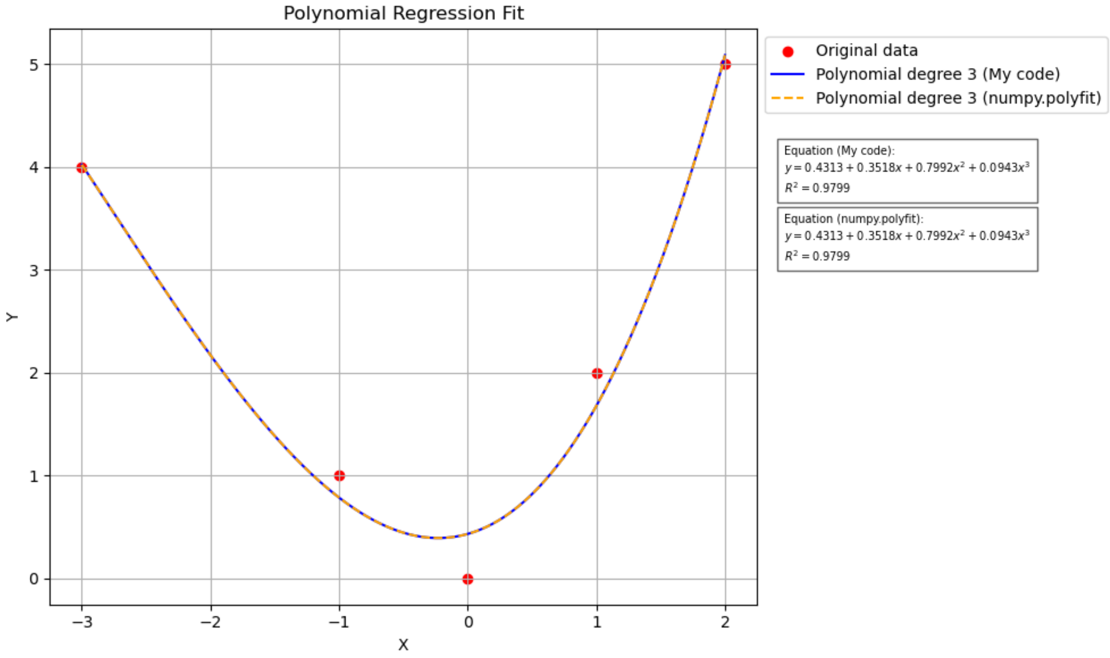
$$R^2 = 0.9799$$

```
Fitted polynomial coefficients (calculated with numpy):
```

$$y = 0.4313 + 0.3518x + 0.7992x^2 + 0.0943x^3$$

```
Coefficient of determination (calculated with numpy):
```

$$R^2 = 0.9799$$



[6]:

```python
file = "datos_8.csv"
mol = RegressionAnalysis(file)
mol.display_data()
mol.plot_fit()
```

```
Enter the delimiter used in the file (comma ',', semicolon ';', tab '\t', etc.):
,
Enter the number of header rows to skip (e.g., 1 if there is a row with column
names):  1
```

```
Data successfully loaded.
```

```
Up to what degree do you want to create the matrix (0-5):  3
```

```
Current data:
x: [-7. -3.  1.  2.  4.]
y: [ 2.  0. -1.  3.  6.]

Augmented matrix:
[[     5.     -3.     79.   -297.     10.]
 [    -3.     79.   -297.   2755.     15.]
 [    79.   -297.   2755. -15993.    205.]
 [  -297.   2755. -15993. 122539.   -279.]]
```

```
Fitted polynomial coefficients:
```

$$y = -0.4261 + 0.4855x + 0.2327x^2 + 0.0161x^3$$

```
Coefficient of determination:
```

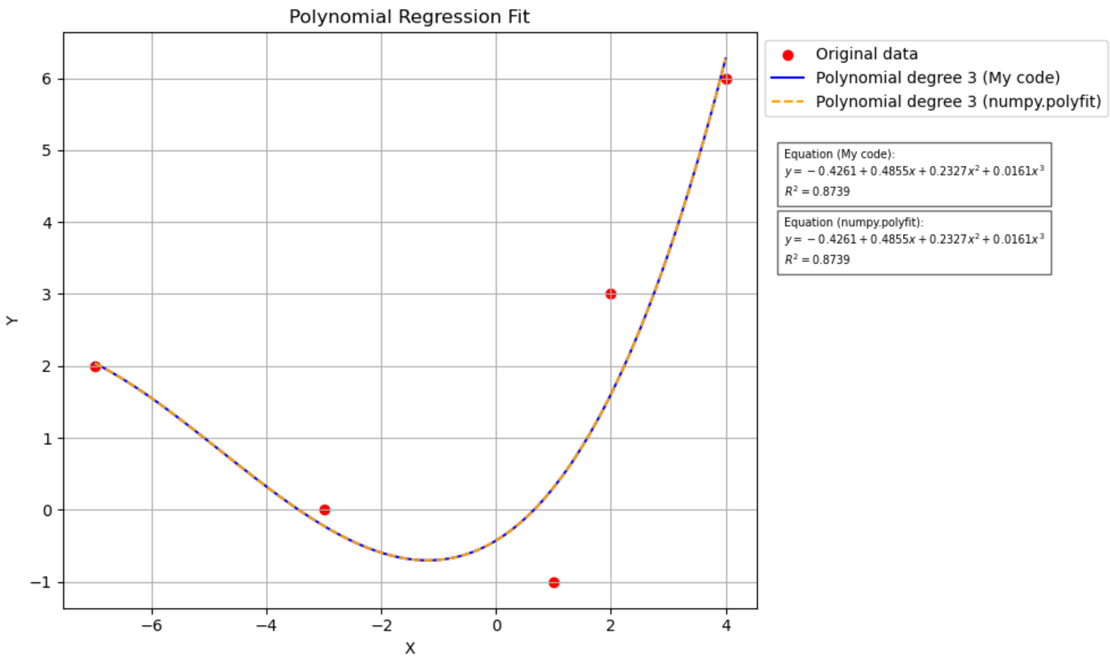
$$R^2 = 0.8739$$

```
Fitted polynomial coefficients (calculated with numpy):
```

$$y = -0.4261 + 0.4855x + 0.2327x^2 + 0.0161x^3$$

```
Coefficient of determination (calculated with numpy):
```

$$R^2 = 0.8739$$



[7]:

```
file = "datos_9.csv"
mol = RegressionAnalysis(file)
mol.display_data()
mol.plot_fit()

print(f"Approximate value of cos(π/4): {mol.evaluate_polynomial(np.pi/4, mol.
 ↪coefficients)}")

print(f"Exact value of cos(π/4): {np.cos(np.pi / 4)}")

print(f"Absolute error: {abs(mol.evaluate_polynomial(np.pi/4, mol.coefficients)␣
 ↪- np.cos(np.pi / 4)):.6f}")
```

```
Enter the delimiter used in the file (comma ',', semicolon ';', tab '\t', etc.):
,
Enter the number of header rows to skip (e.g., 1 if there is a row with column
names):  1

Data successfully loaded.

Up to what degree do you want to create the matrix (0-5):  2


Current data:
x: [-1.570796 -1.047198  0.        1.047198  1.570796]
y: [0.  0.5 1.  0.5 0. ]
```

```
Augmented matrix:
[[ 5.        0.        7.128048  2.       ]
 [ 0.        7.128048  0.        0.       ]
 [ 7.128048  0.       14.581299  1.096623]]
```

```
Fitted polynomial coefficients:
```

$$y = 0.9660 - 0.0000x - 0.3970x^2$$

```
Coefficient of determination:
```

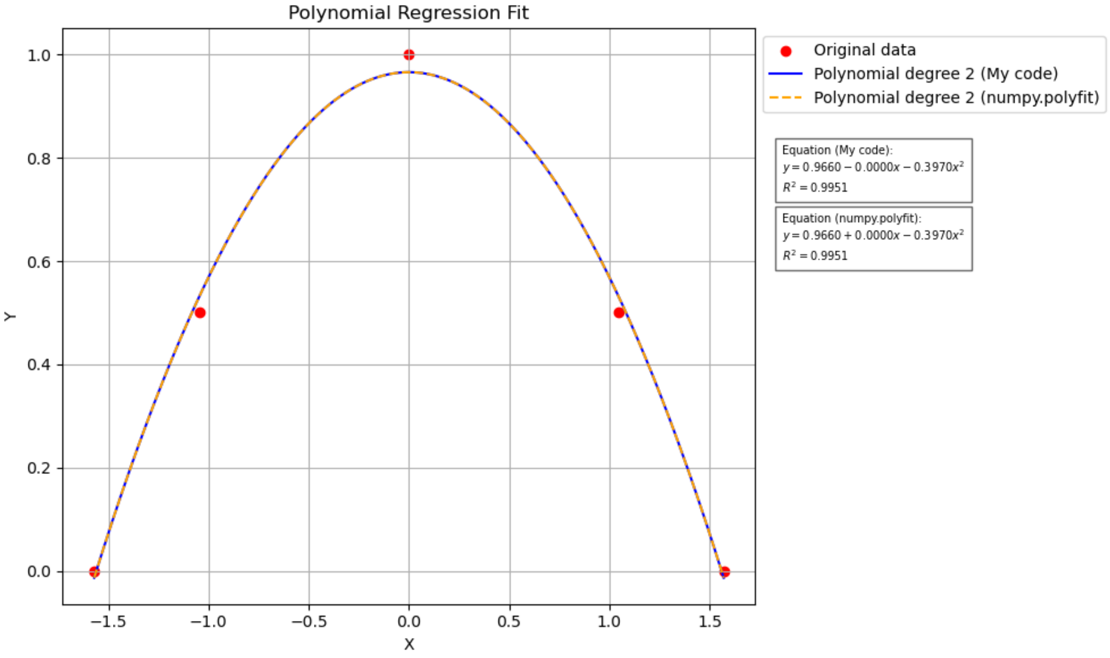
$$R^2 = 0.9951$$

```
Fitted polynomial coefficients (calculated with numpy):
```

$$y = 0.9660 + 0.0000x - 0.3970x^2$$

```
Coefficient of determination (calculated with numpy):
```

$$R^2 = 0.9951$$



```
Approximate value of cos(π/4): 0.7210884353741496
Exact value of cos(π/4): 0.7071067811865476
Absolute error: 0.013982
```

[ ]:

3 Explanation of the Code for Exercise 1

In this exercise, several functions have been implemented within a class. This class, named `RegressionAnalysis`, takes a single argument: the filename containing the data points to be fitted to the polynomial. The data file must follow a specific format: it may or may not include a header (e.g., "x, y"). The code prompts the user to specify how many header lines are present and skips them. Below the header, the numerical data must be structured in two columns, separated by a chosen delimiter. The code also asks the user to specify this delimiter, which can be a dot, a comma, a tab, etc.

When the class is initialized, the code stores several variables: the delimiter, the number of header lines, the x and y data points, and the desired polynomial degree, which must be between 0 and 5. Additionally, it computes and stores various elements: the polynomial coefficients obtained using the Gauss-Jordan method, the polynomial coefficients calculated with a built-in `numpy` function, the LaTeX-formatted polynomial equations for both methods, and the R^2 coefficient for both the Gauss-Jordan and `numpy` solutions.

The next key component of the code is the `load_csv_data` function, which is responsible for reading the data while skipping the specified number of header lines and using the previously obtained delimiter. Furthermore, if the data file does not follow the expected format, the function raises an error to ensure the format of the input data is correct.

The `set_augmented_matrix` function takes the polynomial degree as an argument and constructs the augmented matrix required for polynomial regression using the least squares method. To achieve this, it first defines a square matrix of dimensions $(degree + 1) \times (degree + 1)$, since the polynomial includes terms from x^0 to x^n, and a column vector of the same size that stores the independent terms of the equation system. The elements of the matrix are computed from the summation of powers of the x values, following the least squares formulation, where each element $A[i, j]$ is given by $A_{i,j} = \sum x_k^{i+j}$, while the values of the column vector b are obtained as $b_i = \sum y_k x_k^i$, where x_k and y_k are the input data points. Finally, the augmented matrix is constructed by concatenating the square matrix A with the column vector b using `numpy.hstack()`, allowing the equation system to be solved through the Gauss-Jordan elimination method to obtain the polynomial regression coefficients.

The `gauss_jordan` function performs the Gauss-Jordan elimination method to solve a system of linear equations. It takes an augmented matrix as input and applies row operations to transform it, allowing direct extraction of the solution. The function first ensures the matrix is in floating-point, then iterates through each row, selecting the diagonal element as the pivot. If the pivot is zero, the function terminates, as the system may not have a unique solution. Otherwise, it normalizes the pivot row by dividing all its elements by the pivot value to ensure a leading coefficient of 1. Then, it eliminates all other nonzero elements in the column by subtracting appropriate multiples of the pivot row from the remaining rows. This process is repeated for each row until the left-hand side forms an identity matrix and the solution vector is in the last column, which are the polynomial coefficients required for the regression.

The `display_data` function is responsible for presenting the information computed during the polynomial regression process. It first ensures numerical values are displayed with 6 decimals and without scientific notation. Then, it prints the loaded dataset, including the x and y values, followed by the augmented matrix, it displays the polynomial equations obtained from both the Gauss-Jordan elimination method and with numpy, formatted in LaTeX for readability. Finally, it prints the coefficient of determination R^2 for both methods, allowing a quantitative comparison of the regression's accuracy.

The `evaluate_polynomial` function computes the value of a polynomial at a given point x using as an argument the set of coefficients of the polynomial. It iterates through the coefficient list, multiplying each coefficient by the corresponding power of x, following the standard polynomial expansion formula $P(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$, where c_i represents the coefficient of x^i. The function returns the sum of these terms, effectively evaluating the polynomial at any point x.

The `calculate_r2` function takes as arguments a polynomial evaluation function and a list of coefficients. It computes the coefficient of determination R^2. The function first evaluates the polynomial at each input x using the provided coefficients, obtaining the predicted values. Then, it calculates the total sum of squares (SST) as $SS_{\text{total}} = \sum(y_i - \bar{y})^2$, where y_i are the actual data points and $\bar{y}$ is their mean. The residual sum of squares (SSR) is computed as $SS_{\text{residual}} = \sum(y_i - P(x_i))^2$, where $P(x_i)$ represents the predicted values. Finally, R^2 is calculated as $R^2 = 1 - \frac{SS_{\text{residual}}}{SS_{\text{total}}}$.

The `plot_fit` function visualizes the polynomial regression results by plotting the original data points and the fitted polynomial curves. It does not take any arguments. The function first generates a set of evenly spaced x values to ensure a smooth curve. It then evaluates the polynomial at these points using both the Gauss-Jordan computed coefficients and those obtained via `numpy.polyfit`. The original data points are plotted as a scatter plot, while the fitted curves are drawn using solid and dashed lines for the respective methods. Additionally, a legend is placed in the upper right corner. The function positions the polynomial equations and their corresponding R^2 values below the legend using `plt.figtext`. Finally, the function applies grid lines, labels the axes, and formats the layout for optimal visualization.

4 Discussion of Results and Comparison with `numpy.polyfit`

4.1 Accuracy of the Implemented Algorithm

For all datasets, the polynomial coefficients obtained using the implemented Gauss-Jordan method exactly match those computed with `numpy.polyfit` up to the fourth decimal place. This confirms the correctness of the algorithm in solving polynomial regression problems via the least squares method.

The coefficient of determination (R^2) values indicate how well the polynomial models fit the data:

- **Dataset 5:** $R^2 = 0.7948$ – moderate fit, suggesting some variance in the data is not captured by the polynomial.
- **Dataset 6:** $R^2 = 0.9982$ – excellent fit, indicating the polynomial explains almost all the variance.
- **Dataset 7:** $R^2 = 0.9799$ – strong fit, with minimal deviation from the observed data.
- **Dataset 8:** $R^2 = 0.8739$ – good fit, though some deviations are present.
- **Dataset 9:** $R^2 = 0.9951$ – nearly perfect fit, with very little error.

These values shows that the polynomial degree chosen in some cases is enough to model the data such as for the dataset 6 and 9 and for other datasets a higher polynomial degree would fit better the data such as for dataset 5 and 8 .

Finally for the dataset 9 the polynomial approximation of $\cos(\pi/4)$ resulted in a value of 0.7211, compared to the exact value of 0.7071, yielding an absolute error of 0.01398. This small error demonstrates that the polynomial model provides a reasonable approximation of the cosine function within the given data range.