Linear Regression

PI - Introduction & Supervised Learning

Tanish Patel, Hrishikesh Kalola

May 2025

1 Introduction to Linear Regression

1.1 What is Regression?

Regression is a fundamental concept in supervised machine learning used to model the relationship between a dependent variable and one or more independent variables. The primary goal is to predict a continuous output variable based on one or more input features. Unlike classification, which deals with discrete categories, regression deals with quantities that vary along a continuum, such as temperature, prices, or age.

Mathematically, a regression problem aims to find a function $f : R^n \to R$ such that:

$$y \approx f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n)$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ represents the feature vector and y is the target variable.

1.2 What is Linear Regression?

Linear regression is the simplest form of regression that assumes a linear relationship between the input variables and the output. That is, the output variable y can be expressed as a weighted sum of the input variables plus a bias term:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

Here:

- β_0 is the intercept term.
- $\beta_1, \ldots, \beta_n$ are the coefficients (weights) of the model.
- ε is the error term accounting for noise or model imperfections.

Linear regression tries to estimate the values of $\beta_0, \beta_1, \ldots, \beta_n$ such that the predicted outputs are as close as possible to the actual outputs.

1.3 Use-Cases of Linear Regression

Linear regression is used extensively in both academia and industry due to its simplicity, interpretability, and wide applicability. Some common use-cases include:

- Predicting house prices based on size, location, and number of rooms.
- Estimating demand for a product based on historical sales data.
- Modeling the relationship between advertising spend and revenue.
- Forecasting stock prices and financial metrics.

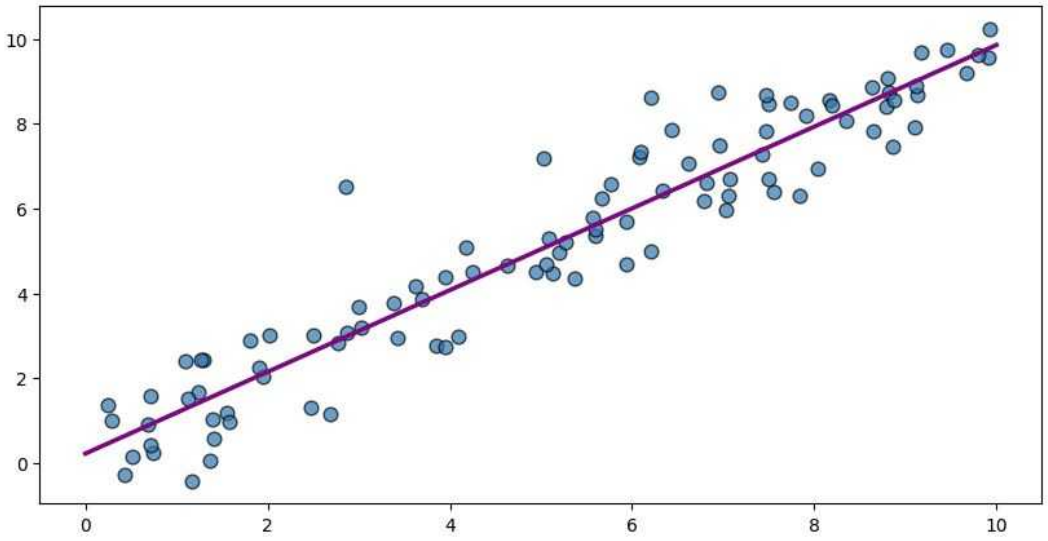

Figure 1: Simple Linear Regression: Fitting a line to data points

1.4 Types of Linear Regression

1. **Simple Linear Regression:** Involves one independent variable. The model has the form:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

   where the relationship is modeled as a straight line in 2D space.

2. **Multiple Linear Regression:** Involves two or more independent variables. The model generalizes to:

$$y = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \varepsilon$$

   which represents a hyperplane in higher-dimensional space.

3. **Polynomial Regression (Brief Mention):** Although technically not linear in the variables, it is linear in the parameters. The model includes terms like x^2, x^3, etc., and is covered in a later section.

1.5 Visualization of Linear Regression

A simple linear regression model can be visualized as fitting a straight line through a scatterplot of data points (fig 1). The goal is to minimize the vertical distance (residuals) between the observed data points and the predicted line.

1.6 Why Study Linear Regression?

Linear regression serves as the foundation for understanding more advanced regression techniques and many other machine learning models. It also provides an interpretable baseline that can be used to assess the effectiveness of more complex models.

Furthermore, the mathematical underpinnings of linear regression—such as optimization, gradient descent, and matrix algebra—play a vital role in the study of machine learning theory.

2 Mathematical Foundations of Linear Regression

2.1 Equation of a Line

In its simplest form, the equation of a straight line in two-dimensional space is:

$$y = mx + c$$

where:

- m is the slope of the line, representing the rate of change of y with respect to x.

- c is the y-intercept, the point at which the line crosses the y-axis.

In linear regression, this equation is generalized to model the relationship between a dependent variable and one or more independent variables.

2.2 General Linear Regression Model

For a dataset with n features (independent variables) and m samples (data points), the linear regression model can be expressed as:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in} + \varepsilon_i \quad \text{for } i = 1, 2, \ldots, m$$

Here:

- x_{ij} represents the j-th feature of the i-th observation.
- β_j are the model parameters (coefficients) to be learned.
- ε_i is the residual error for the i-th observation.

In matrix notation, this can be written more compactly as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where:

- $\mathbf{y} \in R^{m \times 1}$ is the vector of observed outputs,
- $\mathbf{X} \in R^{m \times (n+1)}$ is the design matrix with a column of ones for the intercept,
- $\boldsymbol{\beta} \in R^{(n+1) \times 1}$ is the parameter vector,
- $\boldsymbol{\varepsilon} \in R^{m \times 1}$ is the error vector.

2.3 Key Assumptions of Linear Regression

Linear regression relies on several critical assumptions. Violation of these can lead to misleading estimates and poor predictive performance.

1. **Linearity:** The relationship between predictors and the target variable is assumed to be linear.
2. **Independence:** Observations are assumed to be independent of each other. Violation can lead to autocorrelated errors.
3. **Homoscedasticity:** The variance of residuals (errors) is constant across all levels of the independent variables.
4. **Normality of Residuals:** The residuals are normally distributed (especially important for inference, not prediction).
5. **No Multicollinearity:** Predictor variables are not highly linearly correlated with each other. High multicollinearity can make coefficient estimates unstable and difficult to interpret.

2.4 Interpretation of Coefficients

Each coefficient β_j represents the expected change in the output variable y for a one-unit increase in the input variable x_j, holding all other variables constant.

- If $\beta_j > 0$, the feature has a positive relationship with the output.
- If $\beta_j < 0$, the feature has a negative relationship.
- If $\beta_j = 0$, the feature does not influence the output.

2.5 Geometric Viewpoint

In geometric terms, linear regression fits a hyperplane in n-dimensional feature space that minimizes the perpendicular (orthogonal) projection errors onto the subspace spanned by the independent variables.

This geometric insight underpins the derivation of the Ordinary Least Squares (OLS) solution, which we'll explore in the next section.

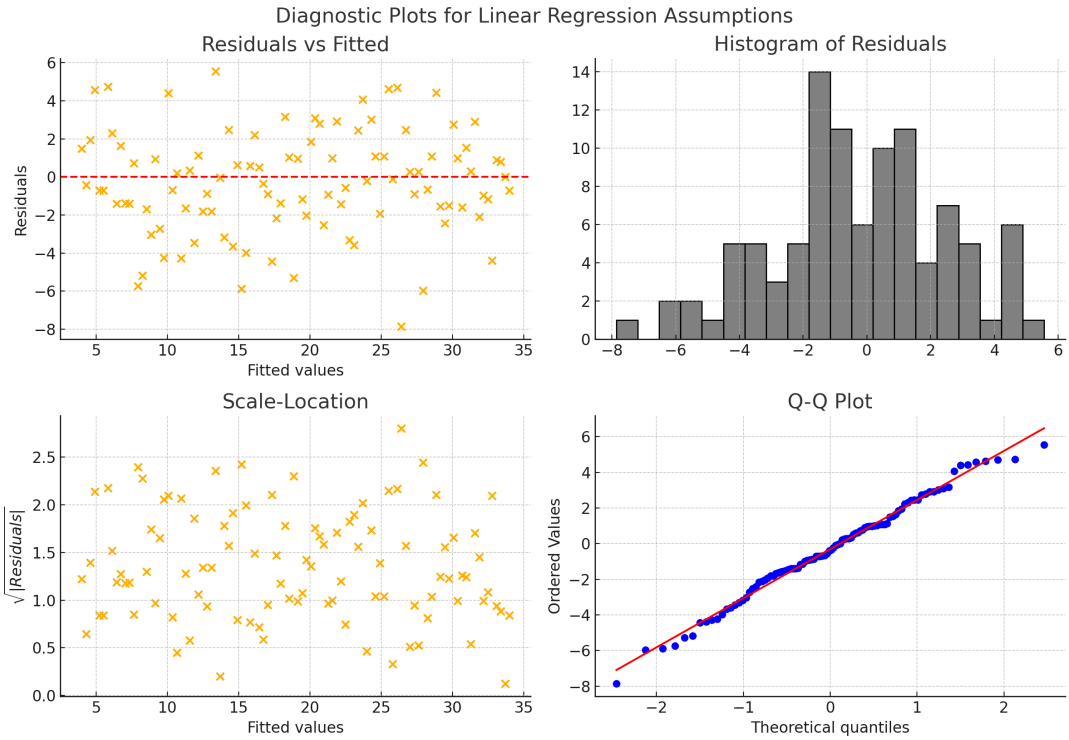

Figure 2: Diagnostic plots to assess assumptions in linear regression

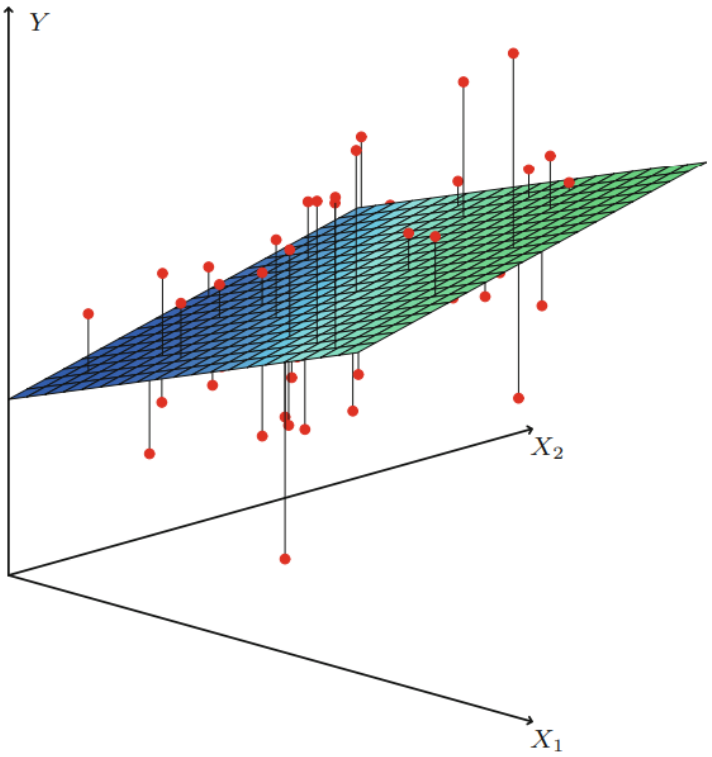

Figure 3: Linear regression as projection of $\mathbf{y}$ onto the column space of $\mathbf{X}$

3 Model Training and Estimation

In this section, we delve into the mathematical machinery that allows us to estimate the parameters of a linear regression model. The goal is to find the optimal set of coefficients $\boldsymbol{\beta}$ that best fit the data.

3.1 Ordinary Least Squares (OLS)

Objective

The most common method for estimating the parameters of a linear regression model is the **Ordinary Least Squares (OLS)** method. The idea is to minimize the sum of squared differences between the observed values and the predicted values from the model.

Given a dataset with m observations and n features, the cost function (or loss function) is:

$$J(\boldsymbol{\beta}) = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} \left(y_i - \sum_{j=0}^{n} \beta_j x_{ij} \right)^2$$

In vectorized form:

$$J(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Closed-Form Solution

To find the optimal coefficients, we take the derivative of the cost function with respect to $\boldsymbol{\beta}$ and set it to zero:

$$\frac{\partial J}{\partial \boldsymbol{\beta}} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = 0$$

Solving this equation yields the normal equation:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

Assuming that $\mathbf{X}^T\mathbf{X}$ is invertible, the solution is:

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

This is known as the **closed-form OLS solution**. It directly gives the parameter estimates that minimize the residual sum of squares.

Python Implementation using NumPy

```
import numpy as np

# X is the design matrix with a column of ones for intercept
X = np.column_stack((np.ones(x.shape[0]), x))
y = y.reshape(-1, 1)

# OLS closed-form solution
beta_hat = np.linalg.inv(X.T @ X) @ X.T @ y
```

3.2 Cost Function: MSE and RMSE

The Mean Squared Error (MSE) is a commonly used cost function in regression, defined as:

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

The Root Mean Squared Error (RMSE) is its square root:

$$\text{RMSE} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2}$$

These metrics provide a quantitative measure of how well the model predictions align with the observed data.

3.3 Gradient Descent for Linear Regression

When the dataset is large or $\mathbf{X}^T\mathbf{X}$ is not invertible, we use iterative optimization algorithms like **Gradient Descent** to find the optimal $\boldsymbol{\beta}$.

Gradient Descent Update Rule

Initialize $\boldsymbol{\beta}$ randomly. For each iteration, update the parameters as:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \alpha \cdot \nabla J(\boldsymbol{\beta})$$

where α is the learning rate and the gradient is:

$$\nabla J(\boldsymbol{\beta}) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Python Implementation

```
def gradient_descent(X, y, lr=0.01, epochs=1000):
    m, n = X.shape
    beta = np.zeros((n, 1))
    for _ in range(epochs):
        gradient = -2 * X.T @ (y - X @ beta) / m
        beta -= lr * gradient
    return beta
```

3.4 Convergence and Learning Rate

The learning rate α controls how big the steps are during each update. Choosing α is critical:

- Too small: Convergence is very slow.
- Too large: The algorithm may overshoot and diverge.

In practice, learning rates are selected through experimentation or optimization techniques like learning rate schedules or adaptive optimizers.

4 Gradient Descent for Linear Regression

4.1 Motivation: Why Use Gradient Descent?

While the closed-form solution using Ordinary Least Squares (OLS) is elegant and efficient for small- to medium-sized datasets, it becomes computationally expensive for:

- Very large datasets (due to the matrix inversion of $\mathbf{X}^T\mathbf{X}$),
- Ill-conditioned matrices or when multicollinearity is present,
- Online learning where data arrives sequentially.

In such cases, **Gradient Descent** provides a scalable and general-purpose optimization algorithm to estimate the parameters iteratively without matrix inversion.

4.2 Intuition Behind Gradient Descent

Gradient descent attempts to minimize a cost function $J(\boldsymbol{\beta})$ by iteratively updating the parameters in the direction of the steepest descent (negative gradient). For linear regression, the cost function is the Mean Squared Error (MSE):

$$J(\boldsymbol{\beta}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \mathbf{x}_i^T\boldsymbol{\beta})^2$$

4.3 Gradient Descent Algorithm

The gradient of the cost function with respect to $\boldsymbol{\beta}$ is given by:

$$\nabla J(\boldsymbol{\beta}) = -\frac{2}{m}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

The parameters are updated iteratively using the rule:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \alpha \cdot \nabla J(\boldsymbol{\beta}^{(t)})$$

where $\alpha > 0$ is the **learning rate**, which controls the step size.

4.4 Algorithm Steps

1. Initialize $\boldsymbol{\beta}^{(0)}$ (often to zeros or small random values).
2. For each iteration t:
   - Compute predictions: $\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}^{(t)}$
   - Compute gradient: $\nabla J(\boldsymbol{\beta}^{(t)})$
   - Update weights: $\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \alpha \cdot \nabla J$
3. Repeat until convergence (change in J is below a threshold or maximum iterations reached).

4.5 Python Implementation

```
import numpy as np

def gradient_descent(X, y, lr=0.01, epochs=1000):
    m, n = X.shape
    beta = np.zeros((n, 1))
    history = []

    for i in range(epochs):
        y_pred = X @ beta
        gradient = -2 * X.T @ (y - y_pred) / m
        beta -= lr * gradient

        # Store the cost
        mse = np.mean((y - y_pred) ** 2)
        history.append(mse)

    return beta, history
```

4.6 Learning Rate and Convergence Behavior

The choice of learning rate α plays a critical role:

- **Too small:** Convergence is guaranteed but may be very slow.
- **Too large:** The algorithm might diverge or oscillate.

Strategies to choose an appropriate α:

- Grid search or cross-validation
- Learning rate decay
- Adaptive methods (e.g., Adam, RMSprop)

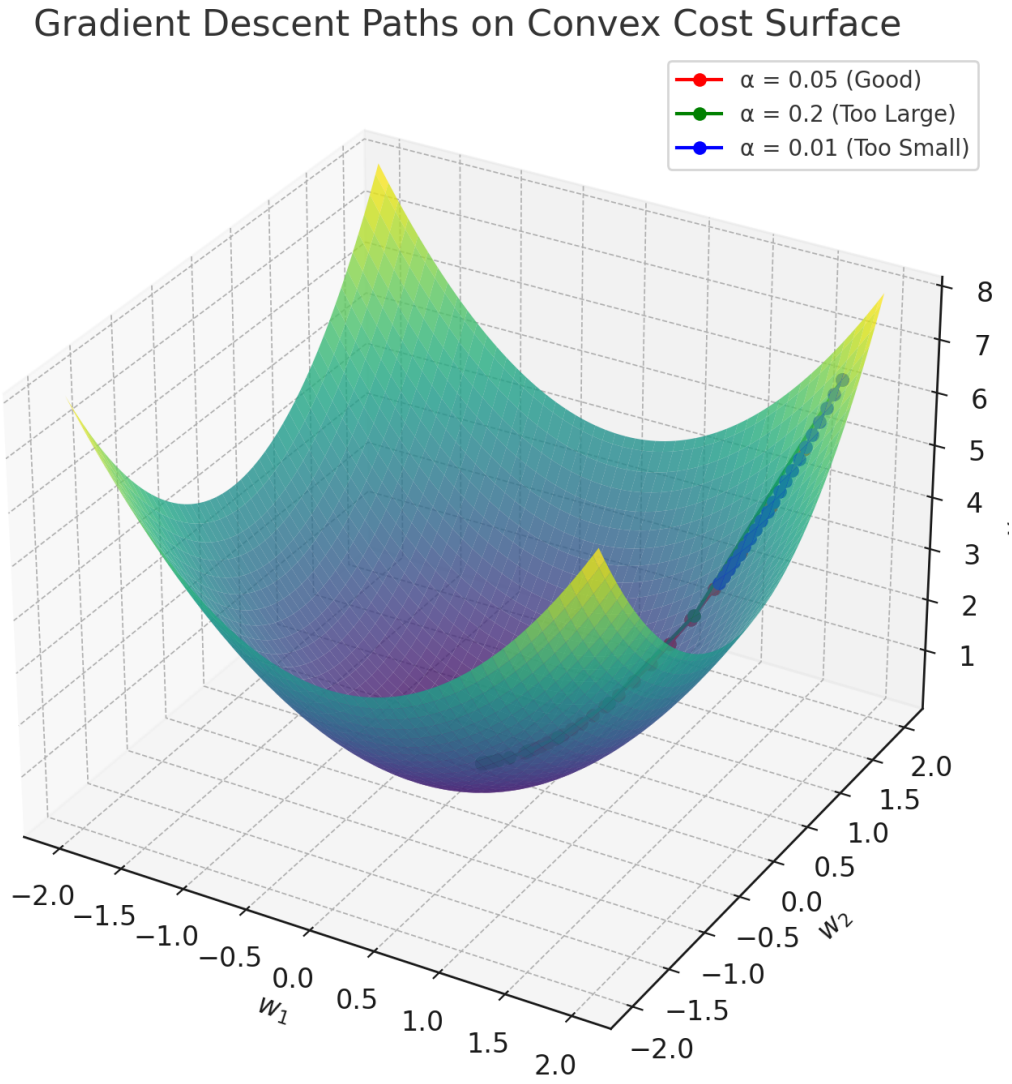

Figure 4: Effect of learning rate on convergence of gradient descent

4.7 Variants of Gradient Descent

- **Batch Gradient Descent:** Uses the entire dataset in each update (standard method above).
- **Stochastic Gradient Descent (SGD):** Updates parameters using a single sample at each step.
- **Mini-Batch Gradient Descent:** Uses a small subset (batch) of data for each update; balances convergence speed and variance.

Each variant has trade-offs in terms of speed, stability, and scalability. For large datasets, mini-batch or SGD is typically preferred.

5 Worked Example: Manual Linear Regression on a Small Dataset

To solidify the concepts of linear regression, we present a step-by-step example using a small dataset of 10 points.

5.1 Dataset

Let us consider the following dataset, where x is the independent variable and y is the dependent variable:

x	y
1	1.2
2	1.9
3	3.2
4	3.8
5	5.1
6	5.9
7	7.2
8	8.1
9	9.1
10	9.9

5.2 Goal

Fit a simple linear regression model of the form:

$$y = \beta_0 + \beta_1 x$$

5.3 Step 1: Compute Means

$$\bar{x} = \frac{1}{10}\sum_{i=1}^{10} x_i = 5.5, \quad \bar{y} = \frac{1}{10}\sum_{i=1}^{10} y_i = 5.64$$

5.4 Step 2: Compute Slope (β_1) and Intercept (β_0)

The formulas for the slope and intercept are:

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \quad \beta_0 = \bar{y} - \beta_1 \bar{x}$$

Calculating these from the data gives:

$$\beta_1 \approx 0.97, \quad \beta_0 \approx 0.31$$

5.5 Step 3: Regression Equation

Thus, the fitted line is:

$$\hat{y} = 0.31 + 0.97x$$

5.6 Step 4: Predictions and Residuals

x	y	$\hat{y}$	Residual $e = y - \hat{y}$
1	1.2	1.28	-0.08
2	1.9	2.25	-0.35
3	3.2	3.22	-0.02
4	3.8	4.19	-0.39
5	5.1	5.16	-0.06
6	5.9	6.13	-0.23
7	7.2	7.10	+0.10
8	8.1	8.07	+0.03
9	9.1	9.04	+0.06
10	9.9	10.01	-0.11

5.7 Step 5: Mean Squared Error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \approx 0.037$$

5.8 Visualization

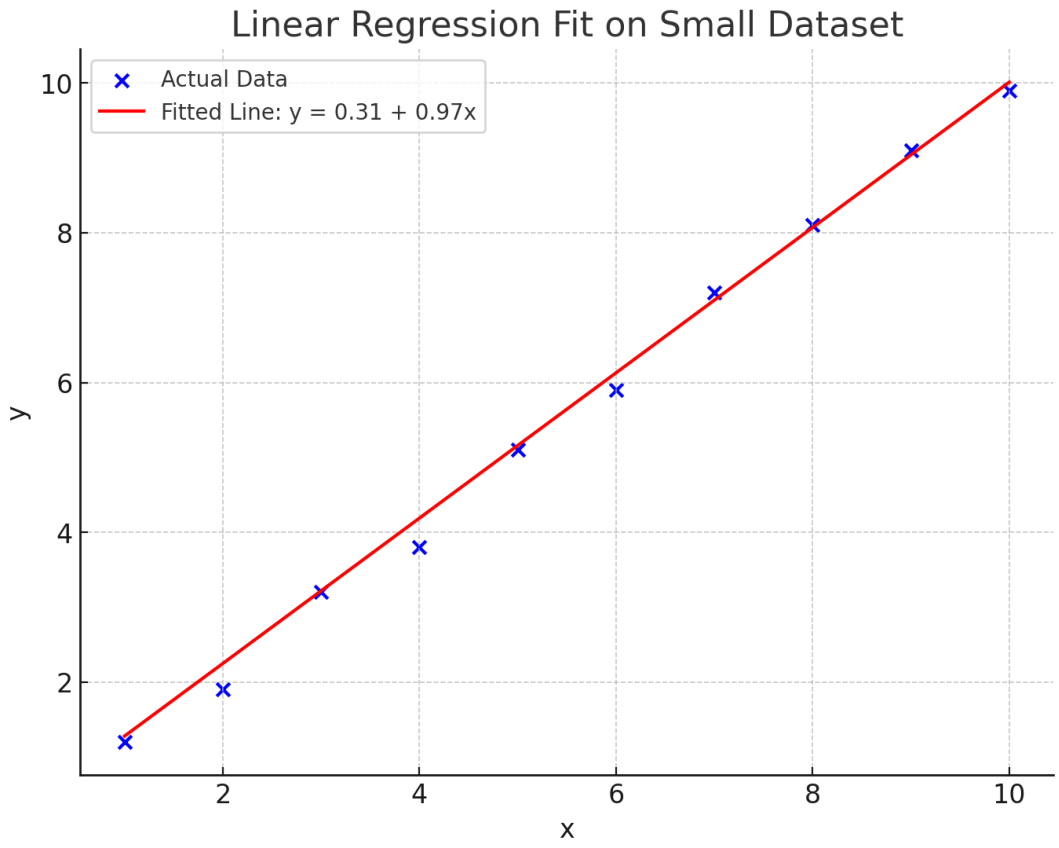

Figure 5: Linear Regression Fit on Small Dataset

This example demonstrates the core mechanics of linear regression and how the coefficients are learned from data using basic algebra and vector operations.

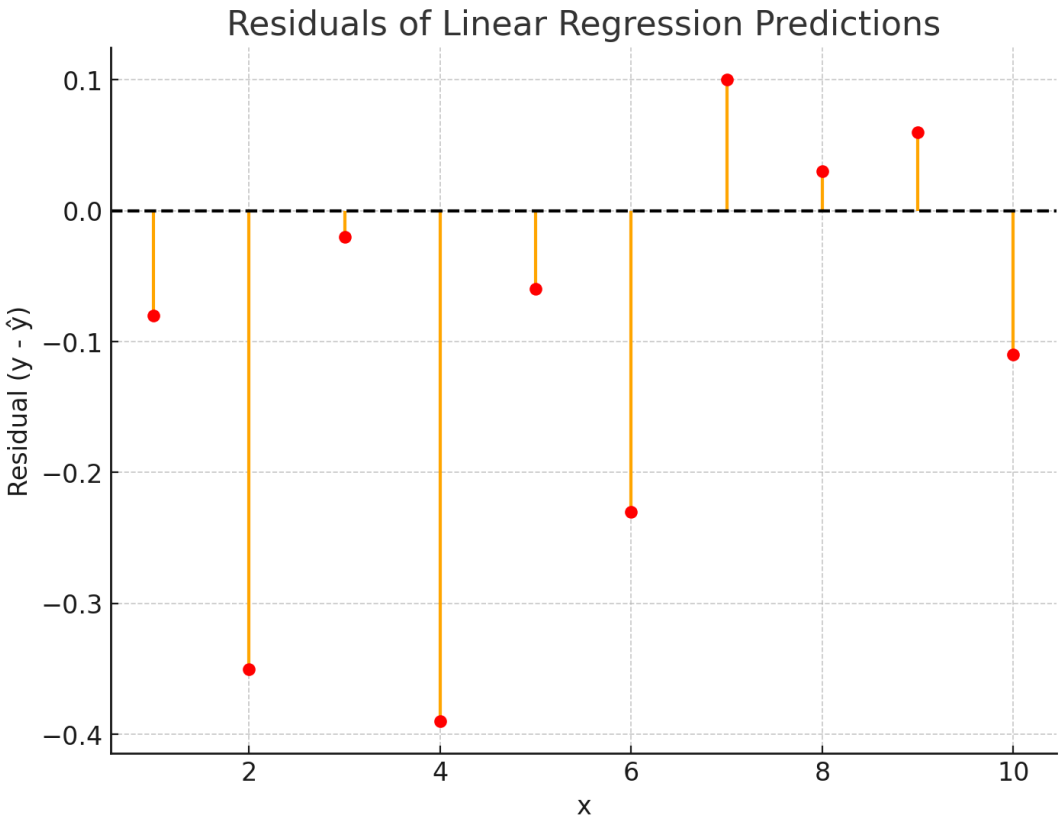

Figure 6: Residuals of Predictions