How to Easily Perform Least Squares Fitting with NumPy’s polyfit()

Authored by

stats writer

November 27, 2025

The methodology known as Least squares fitting represents a core component of statistical modeling, providing a rigorous way to estimate the parameters of a model that minimizes the discrepancies between observed data and the model's predictions. Specifically, this technique seeks the optimal line or curve that reduces the sum of the squared vertical distances (residuals) from the data points to the proposed function. Utilizing the high-performance capabilities of the NumPy library is paramount for executing this process efficiently within the Python ecosystem, allowing analysts to handle large datasets and complex regression tasks with speed.

In NumPy, there are two specialized functions designed to tackle fitting problems. While `numpy.polyfit()` is often used for simple curve fitting, taking the x and y coordinates along with a specified polynomial degree as parameters, the more generalized and powerful tool is `numpy.linalg.lstsq()`. For instance, to fit a linear model (a line), the degree parameter for `polyfit()` would be set to 1, whereas fitting a quadratic curve (a parabola) requires a degree of 2. Regardless of the function used, the ultimate output is an array containing the numerical coefficients that define the function which best minimizes the error across the entire dataset.

1. Theoretical Foundations of the Least Squares Method

The fundamental objective of the method of least squares is to identify the unique regression line that minimizes the overall error in parameter estimation. This method is defined mathematically by solving for the regression parameters ($beta_0$ and $beta_1$) that ensure the sum of the squared residuals (e_i^2) is the smallest possible value. This process yields a statistically efficient estimate of the linear relationship between the independent variable (X) and the dependent variable (Y), forming the backbone of standard linear regression.

When implemented in NumPy, specifically through the `linalg.lstsq()` function, we are solving a system of equations that defines this minimization problem. This approach is highly effective because it provides a closed-form solution for the parameters, meaning no iterative optimization is necessary for standard linear models. The ability of NumPy's linear algebra module to quickly solve these systems makes it the preferred tool for high-volume data processing and rapid model prototyping, offering superior performance compared to manual implementation of the normal equations.

2. Step 1: Preparing Data for Regression Analysis

The success of any statistical analysis hinges on careful data preparation. In the context of linear least squares fitting, we must define our data points using two distinct arrays corresponding to the independent variable (X) and the dependent variable (Y). These arrays, created using NumPy's array initialization functions, serve as the primary inputs for our model. It is essential that the data points are correctly paired, reflecting the empirical observations being modeled.

For our example, we define ten paired observations that exhibit a clear positive linear trend. These initial steps are critical for structuring the problem correctly, as `linalg.lstsq()` requires these variables to be passed in a specific matrix format--where the independent variables are ultimately used to construct the design matrix, and the dependent variables form the target vector.

import numpy as np

#define x and y arrays
x = np.array()

y = np.array()

3. Step 2: Executing the Least Squares Fit using linalg.lstsq()

To perform the linear least squares fit using `numpy.linalg.lstsq()`, we must transform our raw data arrays into a design matrix, A, suitable for solving the linear system $Ax = b$. For linear regression, A must include a column of ones to account for the intercept term ($beta_0$). This structural requirement ensures that the calculation finds both the slope ($beta_1$) associated with X and the constant baseline value.

The construction of this design matrix involves using `np.ones()` to generate the intercept column and `np.vstack()` to combine it with our X array. Crucially, the result must be transposed (using `.T`) so that the data is structured correctly: observations running down the rows and variables (X and the intercept column) running across the columns. Once the design matrix (A) and the target vector (Y) are ready, they are passed into the `lstsq()` function, which performs the core computation of minimizing the squared residuals to determine the optimal coefficients. We include the argument `rcond=None` to ensure compatibility with modern NumPy versions and to suppress related warnings.

#perform least squares fitting
np.linalg.lstsq(np.vstack().T, y, rcond=None)

array()

4. Step 3: Extracting and Interpreting the Regression Coefficients

The output of the `linalg.lstsq()` function is a tuple, and the first element, indexed by , contains the resulting array of estimated parameters, or coefficients. These values define the fitted regression line derived through the minimization of the squared error. In the context of the linear model, the first value corresponds to the slope of the line, and the second value corresponds to the

intercept.

For our calculation, the resulting array translates directly into the model's parameters:

The first parameter is the **Slope** ($beta_1$): **0.969** (rounded).
The second parameter is the **Intercept** ($beta_0$): **7.767** (rounded).

Using these calculated values, we can formally construct the equation that best describes the linear relationship within our observed data, providing a mathematical representation of the line of best fit:

? = 7.767 + 0.969x

5. Step 4: Practical Interpretation of the Model Results

The derived equation allows for meaningful interpretation of the underlying data relationship. The estimated slope and intercept quantify the nature and magnitude of the association between X and Y based on the principles of least squares fitting.

The interpretation of the model parameters is as follows:

The **Intercept (7.767)**: This value represents the predicted average value of the dependent variable, Y, when the independent variable, X, is exactly zero. Our model predicts that when $X = 0$, the average value for Y is **7.767**. This is the starting point of the regression line on the Y-axis.
The **Slope (0.969)**: This is the rate of change. It signifies that for every one-unit increase in the independent variable, X, the dependent variable, Y, is predicted to increase by an average of **0.969** units. Since the slope is positive, we confirm a positive correlation between the variables.

This equation is now a predictive tool. For instance, if we want to predict the value of Y when $X=10$, we substitute this value into our derived model:

? = 7.767 + 0.969x
? = 7.767 + 0.969(10)
? = 17.457

Thus, for an X value of 10, the model predicts the corresponding Y value to be **17.457**.

6. The Advantage of Using linalg.lstsq() for Complex Models

While `numpy.polyfit()` is convenient for simple polynomial fitting, `linalg.lstsq()` offers greater flexibility, especially when transitioning to more complex statistical models. This function is

not limited to linear or polynomial relationships; it is designed to solve any system where the relationship between the parameters and the dependent variable is linear (even if the resulting curve is non-linear, such as fitting sinusoidal functions or exponentials by transforming the variables).

This capability stems from the user's ability to define the design matrix, A, explicitly. By constructing A with columns corresponding to different basis functions (e.g., X^2, $log(X)$, or interaction terms), `linalg.lstsq()` can accurately determine the optimal coefficients for these advanced models using the same core least squares minimization principle. This makes it a foundational function for any analyst performing general linear modeling beyond basic straight-line fitting.

7. Resources for Further Exploration

The practical application of the least squares method in NumPy provides a robust basis for conducting statistical analysis. To fully appreciate the mathematical rigor and statistical implications of this technique, additional theoretical review is highly recommended.

For a detailed and accessible visual breakdown of how the least squares fitting process physically determines the line of best fit, please refer to the following video resource. This visual aid complements the technical implementation demonstrated using the NumPy code.

Refer to the video below for a simple explanation of least squares fitting:

To further expand your skills in data manipulation and modeling within Python, the following tutorials cover other common and advanced tasks that utilize NumPy's extensive mathematical capabilities: