Linear Least Square Methods

Matej Staif
CTU–FIT
staifmat@fit.cvut.cz

July 4, 2025

1 Introduction

This project implements and compares three different computational approaches for regression models:

1. Pure Python (using only for-loops)
2. Python with NumPy
3. Python with Numba (JIT-compiled)[1]

Each engine implements four regression models:

1. Linear Regression (Least Square)
2. Ridge Regression (Least Square)
3. Lasso Regression (Coordinate Descent)
4. Elastic Net Regression (Coordinate Descent)

Linear and Ridge Regression are implemented from scratch, while Lasso and Elastic Net Regression utilize `sklearn` implementations for their coordinate descent optimization.

This project extends beyond the scope of the Linear Algebra II [2]. course at CTU FIT, where the least squares method was introduced theoretically. As an extension of the coursework, I explored practical implementation without relying on high-level machine learning libraries, demonstrating the mathematical foundations learned in class through code.

The project also provides performance benchmarking to demonstrate the computational advantages of different implementation strategies, particularly showcasing Numba's JIT compilation performance gains over pure Python implementations. Additionally, the program offers curve fitting capabilities for sixteen pre-selected functions, allowing users to fit various mathematical models to their datasets through an interactive menu system.

2 Curve Fitting and Visualization

Beyond regression models, the program can fit data to 16 predefined functions including polynomials (degrees 1-7), logarithmic functions, square and cube roots, inverse functions, and mixed transformations. After fitting, the program provides visualization capabilities to display the original data points alongside the fitted curves, allowing users to assess the quality of fit visually.

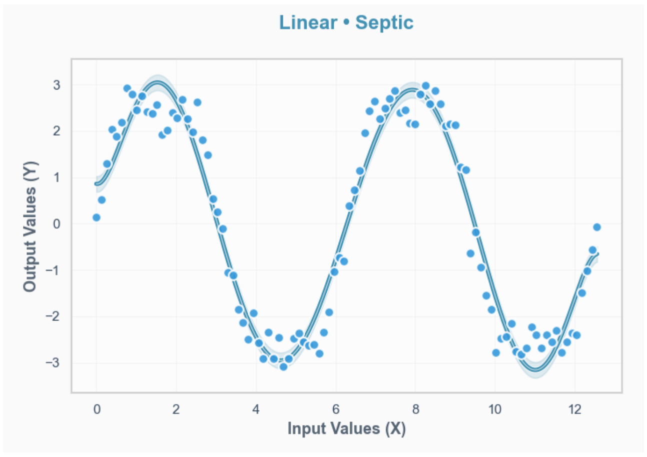


Figure 1: Septic (7th degree) polynomial regression fitting synthetic sinusoidal data. The fitted function $f(x) = c_7x^7 + c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$ using Linear Regression.

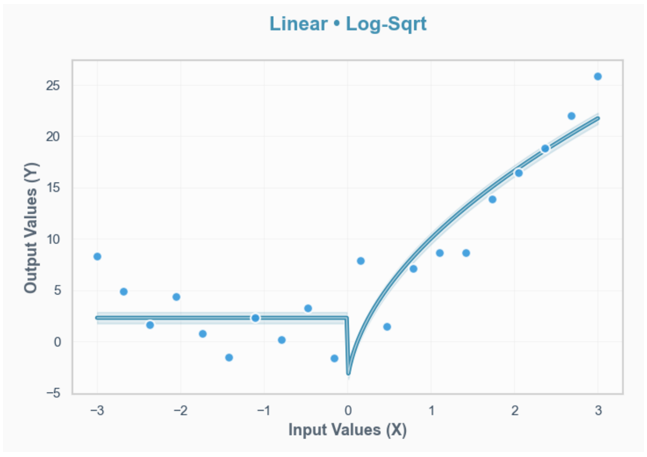


Figure 2: Example of fitting the log-square root function $f(x) = c_0 + c_1 \log(\max(x, \varepsilon)) + c_2\sqrt{\max(x, \varepsilon)}$ using linear regression on transformed features, where $\varepsilon = 10^{-10}$.

3 Input Data

The program accepts numerical data in CSV format. Additionally, users can create data through manual input for 2D points or multidimensional data. The program also supports generating synthetic data or, for the quickest option, using a pre-loaded example dataset.

4 Regression Models

4.1 Least Square Method

Least Squares Method solves a fundamental problem: how to estimate the relationship between a target variable (cz: vysvětlovanou proměnnou) Y and a set of features (cz: souborem příznaků) $X_1, X_2, \dots, X_p$. We assume a linear relationship of the form:

$$Y \approx w_0 + w_1 X_1 + w_2 X_2 + \dots + w_p X_p$$

where $w_0, w_1, \dots, w_p \in \mathbb{R}$ are model parameters that need to be estimated from the data.

4.1.1 Problem Formulation

The parameter w_0 is called the intercept and allows the model to have a non-zero value when all features are zero. Technically, we incorporate the intercept by creating an artificial feature X_0 that is always equal to one. We can then denote the parameter vector as $\mathbf{w} = (w_0, w_1, \dots, w_p)^T$ and the feature vector as $\mathbf{x} = (1, X_1, \dots, X_p)^T$, allowing us to write the model compactly as:

$$Y \approx \mathbf{w}^T \mathbf{x}$$

Given N observations, where the i-th observation contains the target variable value Y_i and feature vector $(x_{i1}, x_{i2}, \dots, x_{ip})$, we organize the data into a design matrix $\mathbf{X} \in \mathbb{R}^{N,(p+1)}$ and target vector $\mathbf{Y} \in \mathbb{R}^N$:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Np} \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{pmatrix}$$

The first column of matrix $\mathbf{X}$ contains ones to capture the intercept term.

4.1.2 Derivation of the Least Squares Solution

The estimate for the i-th observation is given by $\hat{Y}_i = \mathbf{w}^T \mathbf{x}_i$. We measure the model error using the Residual Sum of Squares (RSS):

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^{N} (Y_i - \mathbf{w}^T \mathbf{x}_i)^2 = ||\mathbf{Y} - \mathbf{X}\mathbf{w}||^2$$

For the error to be zero, we would need:

$$\mathbf{X}\mathbf{w} = \mathbf{Y}$$

This represents a system of N linear equations with $p+1$ unknowns. In typical applications, the number of observations N is significantly larger than the number of parameters $p+1$, so this system generally has no solution. Therefore, we seek the least squares solution that minimizes (rather than eliminates) the RSS.

4.1.3 Least squares solution

Since the system $\mathbf{X}\mathbf{w} = \mathbf{Y}$ typically has no solution, we formalize the concept of the best possible solution.

[Least squares solution] Given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and vector $\mathbf{b} \in \mathbb{R}^m$, we call $\mathbf{x} \in \mathbb{R}^n$ a **least squares solution** of the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ if for every $\mathbf{y} \in \mathbb{R}^n$ it holds:

$$||\mathbf{b} - \mathbf{A}\mathbf{x}|| \leq ||\mathbf{b} - \mathbf{A}\mathbf{y}||$$

4.1.4 Normal Equations

Now we can apply the theory of least squares solutions to our multidimensional linear regression problem. We seek to minimize the expression:

$$\text{RSS}(\mathbf{w}) = ||\mathbf{Y} - \mathbf{X}\mathbf{w}||^2$$

for design matrix $\mathbf{X} \in \mathbb{R}^{N,(p+1)}$ and target vector $\mathbf{Y} \in \mathbb{R}^N$.

From the general least squares theory, $\mathbf{w}$ minimizes the RSS function if and only if it satisfies the **normal equations**:

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{Y}$$

Furthermore, this system has a unique solution precisely when the columns of matrix $\mathbf{X}$ are linearly independent. This is equivalent to the condition that matrix $\mathbf{X}^T \mathbf{X}$ is invertible, and the unique solution is:

$$\hat{\mathbf{w}}^{(\text{OLS})} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

This vector $\hat{\mathbf{w}}^{(\text{OLS})}$ is called the ordinary least squares estimate.

[Linear regression via least squares] Let $\mathbf{X}$ be the design matrix and $\mathbf{Y}$ the target vector as defined above. For the OLS estimate $\hat{\mathbf{w}}^{(\mathrm{OLS})}$ of multidimensional linear regression parameters via least squares, it holds:

$$\mathbf{X}^T\mathbf{X}\hat{\mathbf{w}}^{(\mathrm{OLS})} = \mathbf{X}^T\mathbf{Y}$$

If additionally the columns of matrix $\mathbf{X}$ are linearly independent, then the solution of this equation is unique and:

$$\hat{\mathbf{w}}^{(\mathrm{OLS})} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

4.1.5 QR decomposition

Computing the inverse $(\mathbf{X}^T\mathbf{X})^{-1}$ in the normal equations can be numerically unstable. We can avoid this computation by using QR decomposition of matrix $\mathbf{X}$, which can be computed using numerically stable algorithms.

The key insight is that the least squares solution corresponds to the orthogonal projection of $\mathbf{Y}$ onto the column space of $\mathbf{X}$. If $\mathbf{X} = \mathbf{QR} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$ represents the full or reduced QR decomposition of $\mathbf{X}$, where $\hat{\mathbf{Q}}$ contains an orthonormal basis for the column space of $\mathbf{X}$, then:

[Least squares solution via QR decomposition] Let $\mathbf{X} \in \mathbb{R}^{N,p+1}$ be the design matrix and $\mathbf{Y} \in \mathbb{R}^N$ the target vector. Let $\mathbf{X} = \mathbf{QR} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$ be the full or reduced QR decomposition of matrix $\mathbf{X}$. Then $\hat{\mathbf{w}}^{(\mathrm{OLS})}$ satisfies:

$$\hat{\mathbf{R}}\mathbf{w} = \hat{\mathbf{Q}}^T\mathbf{Y}$$

and moreover:

$$\min_{\mathbf{w}\in\mathbb{R}^{p+1}} \mathrm{RSS}(\mathbf{w}) = ||\mathbf{Q}_0^T\mathbf{Y}||^2$$

where $\mathbf{Q} = \begin{pmatrix}\hat{\mathbf{Q}} & \mathbf{Q}_0\end{pmatrix}$.

The QR approach is numerically superior because:

- It avoids forming $\mathbf{X}^T\mathbf{X}$, which can amplify numerical errors
- QR decomposition algorithms are inherently stable
- The resulting upper triangular system $\hat{\mathbf{R}}\mathbf{w} = \hat{\mathbf{Q}}^T\mathbf{Y}$ can be solved efficiently by back substitution

4.2 Linear Regression

Linear regression is the standard OLS problem discussed in Section 4.1. It minimizes:

$$\mathrm{RSS}(\mathbf{w}) = ||\mathbf{Y} - \mathbf{X}\mathbf{w}||^2$$

The solution is obtained via normal equations as $\hat{\mathbf{w}}^{(\mathrm{OLS})} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$.

4.3 Ridge Regression

Ridge regression adds L_2 regularization to prevent overfitting. It minimizes:

$$\mathrm{RSS}_{\mathrm{Ridge}}(\mathbf{w}) = ||\mathbf{Y} - \mathbf{X}\mathbf{w}||^2 + \alpha||\mathbf{w}_{-0}||_2^2$$

where $\alpha > 0$ is the regularization parameter and $\mathbf{w}_{-0}$ denotes all coefficients except the intercept. The solution still uses least squares via modified normal equations:

$$(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I}_p)\mathbf{w} = \mathbf{X}^T\mathbf{Y}$$

where $\mathbf{I}_p$ has zeros in the first row/column (intercept is not penalized).

4.4 Lasso Regression

Lasso regression uses L_1 regularization for feature selection. It minimizes:

$$\mathrm{RSS}_{\mathrm{Lasso}}(\mathbf{w}) = ||\mathbf{Y} - \mathbf{X}\mathbf{w}||^2 + \alpha||\mathbf{w}_{-0}||_1$$

The L_1 norm $||\mathbf{w}||_1 = \sum_{j=1}^{p} |w_j|$ is not differentiable at zero, preventing the use of normal equations. Coordinate descent optimization is required, updating one coefficient at a time while holding others fixed.

4.5 Elastic Net Regression

Elastic Net combines L_1 and L_2 penalties. It minimizes:

$$\mathrm{RSS}_{\mathrm{EN}}(\mathbf{w}) =$$
$$= ||\mathbf{Y} - \mathbf{X}\mathbf{w}||^2 + \alpha[\rho||\mathbf{w}_{-0}||_1 + (1-\rho)||\mathbf{w}_{-0}||_2^2]$$

where $\rho \in [0, 1]$ controls the mix. Due to the L_1 component, coordinate descent is necessary. This method combines Ridge's grouping effect with Lasso's sparsity.

5 Numerical Stability Functionality

The implementation includes several features to ensure numerical stability:

5.1 Condition Number Analysis

The code computes and displays the condition number $\kappa(\mathbf{X}^T\mathbf{X}) = \lambda_{\max}/\lambda_{\min}$ for Linear and Ridge regression. This metric indicates how sensitive the solution is to small perturbations in the data. High condition numbers ($> 10^{10}$) suggest an ill-conditioned problem that may produce unreliable results.

5.2 Adaptive Algorithm Selection

The implementation uses normal equations $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$ by default for efficiency. However, when the condition number exceeds 10^6, it automatically switches to QR decomposition, which is numerically more stable for ill-conditioned matrices.

5.3 Additional Stability Measures:

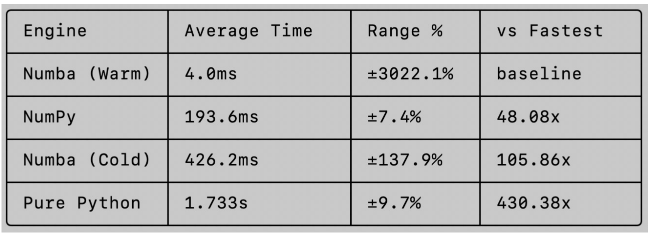
- Feature normalization to $[0, 1]$ range for polynomials of degree > 3
- Small regularization (10^{-10}) added to diagonal for singular matrices
- Specialized scaling for very high-degree polynomial features

5.4 Why Only Linear & Ridge Regression

Condition numbers are computed only for Linear and Ridge regression because they solve the least squares problem directly. Lasso and Elastic Net use coordinate descent optimization from sklearn, which handles numerical stability internally without computing $(\mathbf{X}^T\mathbf{X})^{-1}$.

6 Benchmark Performance

I compared 3 implementations, where I additionally examined the Python with Numba version in 2 cases: Python with Numba (Cold Start) and Python with Numba (Warm Start).



Engine	Average Time	Range %	vs Fastest
Numba (Warm)	4.0ms	±3022.1%	baseline
NumPy	193.6ms	±7.4%	48.08x
Numba (Cold)	426.2ms	±137.9%	105.86x
Pure Python	1.733s	±9.7%	430.38x

Figure 3: Result of Performance Benchmark with 100 runs

The table is displayed using the `tabulate` library and shows comprehensive performance comparison of four implementation approaches. The benchmark was conducted using Python's `timeit` module to ensure precise timing measurements. Each engine was tested with the complete regression pipeline including data transformation, model fitting, and prediction across multiple regression types (Linear, Ridge, Lasso, ElasticNet) and polynomial functions of degrees 1-7.

- **Average Time**: Mean execution time across 100 benchmark runs for each engine
- **Range %**: Performance stability metric — the smaller the percentage, the more consistent the execution times. High values indicate unpredictable performance with significant variations between runs.
- **vs Fastest**: Speed comparison relative to the fastest implementation (Numba Warm), showing how many times slower each approach is

Key findings reveal that Numba (Warm) achieves baseline performance at 4.0ms but exhibits extremely high range spread (±3022.1%). This wide range spread occurs because the execution time is so small that microsecond-level interruptions (garbage collection, system scheduling, cache misses) create large percentage variations in the highly optimized code.

Despite the wide range, Numba demonstrates exceptional average speed, being 430.38× faster than Pure Python and 48.08× faster than Python with NumPy implementation. The pre-compilation process works by using Numba's `@njit` decorator to compile Python functions to optimized machine code at runtime, eliminating Python interpreter overhead entirely.

NumPy shows remarkably stable performance (±7.4% range) through its consistent C-based vectorized operations, while Pure Python maintains reasonable stability (±9.7%) but remains the slowest approach. Numba (Cold) includes the JIT compilation penalty, making it 105.86× slower than warm start because it must first compile Python bytecode to optimized machine code. Once compiled, warm start executes pre-compiled code directly, eliminating compilation overhead.

7 Conclusion

The project began as an implementation of the basic least squares method (hence the project name emphasizing the linear least squares method). Subsequently, the work expanded to include Ridge regression, which also utilizes the least squares method, followed by the addition of Lasso and Elastic Net regression. It's important to note that these latter two methods do not use ordinary least squares (OLS) but instead employ coordinate descent optimization, which I did not implement from scratch but rather utilized through sklearn. I then implemented performance benchmarking and invested significant effort in creating an intuitive menu system and user-friendly interface, which I believe has greatly improved compared to my previous project *prg-precipitation-forecast-hmm*. I encountered numerous challenges towards the end of the project, particularly with bugs in the performance benchmark implementation, but by writing extensive unit tests, I was able to resolve all issues successfully.

Future extensions of this project could explore implementing a fourth computational approach for regression models and subsequently integrating it into the existing project ecosystem. Such an addition would provide valuable insights through performance benchmark comparisons with the current three implementations (Pure Python, NumPy, and Numba).

References

[1] Inc. Anaconda. Numba documentation. online, 2024. [cit. 2025–07–04] `https://numba.readthedocs.io/`.

[2] Ph.D. Ing. Daniel Dombek, Ph.D. RNDr. Luděk Kleprlík, Ph.D. Ing. Karel Klouda, and Ph.D. Ing. Jakub Šístek. Mathematical Script: Linear algebra II., CTU FIT. Mathematical Script available at courses.fit.cvut.cz, CTU FIT, 2025. [cit. 2025–07–04], `https://courses.fit.cvut.cz/BI-LA2/`.