Linear Regression

Linear Algebra and its Applications

Henry Kirveslahti

AI511 University of Southern Denmark

Oct 28 2025

Recap from yesterday

Given $\mathbf{y} \in \mathbb{R}^n$, design matrix $X \in \mathbb{R}^{n \times p}$ of column rank p with $n \geq p$, and $\boldsymbol{\varepsilon} \in \mathbb{R}^n$, the linear model reads

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

We want to find $\boldsymbol{\beta} \in \mathbb{R}^p$ that minimizes

$$\varepsilon^T \varepsilon = (y - X\beta)^T (y - X\beta)$$

This is given by the least squares solution:

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

Computational inconvenience: Inverting the *gram matrix* $X^T X$.

Weaseling out of the inversion

Suppose we could write $X = QR$ for some matrix Q that satisfies $Q^T Q = I$, and a matrix R that is in some way 'easy to invert'

The normal equation:

$$
\begin{aligned}
X^T y &= X^T X \beta & &|X = QR \\
R^T Q^T y &= R^T Q^T Q R \beta \\
R^T Q^T y &= R^T R \beta & &|\text{multiply from left with } R^{-T} \\
Q^T y &= R \beta.
\end{aligned}
$$

If R is easy to invert, we can recover β just by solving $z = Q^T y = R\beta$.

QR Decomposition

QR Decomposition:

$$X = QR$$

where:

- $Q \in \mathbb{R}^{n \times p}$: matrix with orthonormal columns
- $R \in \mathbb{R}^{p \times p}$: upper triangular matrix

Why is this useful?

- Columns of Q form an orthonormal basis for col(X)
- Makes solving least squares problems more numerically stable

Projection Matrix via QR:

$$P = X(X^T X)^{-1} X^T = QR(R^T R)^{-1} R^T Q^T$$

$$\Rightarrow \boxed{P = QQ^T}$$

This is easy!

QR Decomposition

- Note that $Q^T Q = I_p$ – this is a map from $\mathbb{R}^p$ to $\mathbb{R}^p$
- The projection matrix QQ^T is a map from $\mathbb{R}^n$ to $\mathbb{R}^n$ that factors through $\mathbb{R}^p$ - Not invertible

QR Decomposition

- Note that $Q^T Q = I_p$ – this is a map from $\mathbb{R}^p$ to $\mathbb{R}^p$
- The projection matrix QQ^T is a map from $\mathbb{R}^n$ to $\mathbb{R}^n$ that factors through $\mathbb{R}^p$ - Not invertible

Interpretation:

- The projection of y onto col(X) is: $\hat{y} = QQ^T y$
- QQ^T is the projection onto the subspace spanned by the columns of Q

Solving $R\boldsymbol{\beta} = \mathbf{z}$ via Back Substitution

Setup:

$$R\boldsymbol{\beta} = \mathbf{z}, \quad R \in \mathbb{R}^{p\times p} \text{ upper triangular, } \mathbf{z} \in \mathbb{R}^p.$$

Solving $R\boldsymbol{\beta} = \mathbf{z}$ via Back Substitution

Setup:

$$R\boldsymbol{\beta} = \mathbf{z}, \quad R \in \mathbb{R}^{p\times p} \text{ upper triangular, } \mathbf{z} \in \mathbb{R}^p.$$

Structure of the system:

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ 0 & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{pp} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_p \end{bmatrix}$$

Solving $R\boldsymbol{\beta} = \mathbf{z}$ via Back Substitution

Setup:

$$R\boldsymbol{\beta} = \mathbf{z}, \quad R \in \mathbb{R}^{p \times p} \text{ upper triangular, } \mathbf{z} \in \mathbb{R}^p.$$

Structure of the system:

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ 0 & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{pp} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_p \end{bmatrix}$$

Algorithm (Back Substitution):

$$\text{for } i = p, p-1, \ldots, 1: \quad \beta_i = \frac{1}{r_{ii}} \left(z_i - \sum_{j=i+1}^{p} r_{ij}\beta_j \right)$$

Solving $R\boldsymbol{\beta} = \mathbf{z}$ via Back Substitution

Algorithm (Back Substitution):

$$\text{for } i = p, p-1, \ldots, 1: \quad \beta_i = \frac{1}{r_{ii}} \left(z_i - \sum_{j=i+1}^{p} r_{ij} \beta_j \right)$$

Solving $R\boldsymbol{\beta} = \mathbf{z}$ via Back Substitution

Algorithm (Back Substitution):

$$\text{for } i = p, p-1, \dots, 1: \quad \beta_i = \frac{1}{r_{ii}} \left(z_i - \sum_{j=i+1}^{p} r_{ij}\beta_j \right)$$

Notes:

- We proceed *from bottom to top*, since each equation depends only on previously computed β_j for $j > i$.
- The method requires $r_{ii} \neq 0$ for all i (non-singular R).
- Computational cost: $\mathcal{O}(p^2)$.

Solving $R\boldsymbol{\beta} = \mathbf{z}$ via Back Substitution

Algorithm (Back Substitution):

$$\text{for } i = p, p-1, \ldots, 1: \quad \beta_i = \frac{1}{r_{ii}} \left(z_i - \sum_{j=i+1}^{p} r_{ij}\beta_j \right)$$

Notes:

- We proceed *from bottom to top*, since each equation depends only on previously computed β_j for $j > i$.
- The method requires $r_{ii} \neq 0$ for all i (non-singular R).
- Computational cost: $\mathcal{O}(p^2)$.

Interpretation: Back substitution is the natural way to solve triangular systems — it is equivalent to recursively eliminating known components from the system.

How to Compute the QR Decomposition

Goal: Given a full column rank matrix $X \in \mathbb{R}^{n \times p}$, find $Q \in \mathbb{R}^{n \times p}$, $R \in \mathbb{R}^{p \times p}$ such that:

$$X = QR$$

where:

- Columns of Q are orthonormal: $Q^T Q = I$
- R is upper triangular

Method: Classical Gram–Schmidt Orthogonalization

Given columns of $X = [x_1, x_2, \ldots, x_n]$:

1. Set $q_1 = \frac{x_1}{\|x_1\|}$
2. For $j = 2$ to n:
   - Subtract projections onto previous q_i's:

$$u_j = x_j - \sum_{i=1}^{j-1} \langle x_j, q_i \rangle q_i$$

   - Normalize: $q_j = \frac{u_j}{\|u_j\|}$

QR Decomposition of an $n \times p$ Matrix

$$Q = \begin{bmatrix} | & & | \\ q_1 & \cdots & q_p \\ | & & | \end{bmatrix} \in \mathbb{R}^{n\times p}, \quad R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ 0 & r_{22} & \cdots & r_{2p} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & r_{pp} \end{bmatrix} \in \mathbb{R}^{p\times p}.$$

The q_i are orthonormal vectors (basis) from the Gram-Schmidt process, and
$r_{i,j} =< q_i, x_j >= q_i^T x_j$ (the coefficients for the basis)

En lite visa om Gram-Schmidts Metod

There is a classic folk song on how to find an orthogonal basis [in Swedish]:

> Ta en delrumsbas M, och en vektor a;
>
> Projisera ner, ta dess residual;
>
> Normalisera, tillför den till M;
>
> Ta sen nästa vektor, börja om igen.

(The melody is that of *Itsy Bitsy Spider* [da: Lille Peter Edderkop], in case you were wondering)

Example Computation

Let's compute the QR decomposition of

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 0 \end{pmatrix}$$

Extending linear models

- Linear regression model needs only be linear in its parameters – but not in data!
- Given x, y, we could also just construct x, x^2, y and fit a polynomial regression $y = \beta_0 + \beta_1 x + \beta_2 x^2$ – the powers of x are just another column
- For example, we might take log transforms of the response and or the regressors
- Recall: Exponential function exp is a function f that satisfies $f(x+y) = f(x)f(y)$
- Then also $f(kx) = \underbrace{f(x)f(x)\ldots f(x)}_{k \text{ times}} = f(x)^k$
- The inverse, the logarithm, turns products into sums $\log(a)\log(b) = \log(a+b)$
- Consequently, $\log(x^k) = \log(\underbrace{xx\ldots x}_{k \text{ times}}) = k\log(x)$

Always interpret on the original scale of y

- If we log the response, we get a model

$$\log(y) = \beta_0 + \beta_1 x$$

  This is equivalent to (c.f. previous slide)

$$y = \exp(\beta_0)\exp(\beta_1 x) = a\exp(\beta_1 x)$$

- We can also take log of the regressor:

$$y = \beta_0 + \beta_1 \log(x)$$

  or

$$\exp(y) = \exp(\beta_0)x^{\beta_1} = ax^{\beta_1}$$

  (statistically speaking, it is usually better to interpret on the original y-scale - but this)

Log-log transform

- We can also fit a model on log-log scale:

$$\log(y) = \beta_0 + \beta_1 \log(x)$$

This is equivalent to

$$y = \exp(\beta_0)\exp(\beta_1 \log(x)) = ax^{\beta_1}$$

Statistical Digression: Interpreting coefficients under transformations

- Log transforming x doesn't usually need any extra interpretation (except that now the effect of β_1 is the effect of multiplicative change in x)
- However, log-transforming the response *does affect* the interpretation of $\beta_0 + \beta_1 x$ as the population mean – this is usually no longer accurate
- This is because if the errors ϵ_i have certain distribution with mean 0, the mean of the exponentiated distribution is not the exponentiation of the mean the distribution
- However, exponentiation does preserve the median, so we can interpret the fitted line as the population median.
- Example: If y is for example income, our model might say something about a typical worker (i.e. the median) and not about an average worker (which is skewed by mega-millionaires)

Core Linear Algebra Commands

Task	**Command**	**Description**
Matrix multiplication	`A @ B` or `np.dot(A, B)`	AB product
Transpose	`A.T`	Transpose of A
Inverse	`np.linalg.inv(A)`	Inverse of square A
Determinant	`np.linalg.det(A)`	Determinant of A
Rank	`np.linalg.matrix_rank(A)`	Rank of A
Norm	`np.linalg.norm(A)`	Vector or matrix norm
Trace	`np.trace(A)`	Sum of diagonal elements

Solving Linear Systems and Regression Models

Task	**Command**	**Description**
Solve $Ax = b$	`np.linalg.solve(A, b)`	Exact solution (square A)
Least squares	`np.linalg.lstsq(A, b)`	Minimizes $\|Ax - b\|^2$
Pseudo-inverse	`np.linalg.pinv(A)`	Moore–Penrose inverse
Normal equations	`np.linalg.inv(A.T @ A) @ A.T @ b`	Classic OLS formula

Matrix Decompositions in NumPy

Decomposition	Command	Description
QR	`np.linalg.qr(A)`	$A = QR$, Q orthogonal, R upper-triangular
Cholesky	`np.linalg.cholesky(A)`	$A = LL^\top$, A pos. definite
SVD	`np.linalg.svd(A)`	$A = U\Sigma V^\top$
Eigen	`np.linalg.eig(A)`	Eigenvalues/vectors of A
Symmetric eigendecomp.	`np.linalg.eigh(A)`	Stable for symmetric A
Rank	`np.linalg.matrix_rank(A)`	Detects linear dependence

Regression Workflow with NumPy

Task	Example Command	Description
Fit OLS manually	`beta = np.linalg.inv(X.T @ X) @ X.T @ y`	$(X^T X)^{-1} X^T y$
Fit safely	`np.linalg.lstsq(X, y)`	Stable least squares
Predict	`y_pred = X @ beta`	Compute fitted values
Residuals	`res = y - y_pred`	Model residuals
R-squared	`1 - np.sum(res**2)/np.sum((y-y.mean())**2)`	Fit quality
Back substitution	`np.linalg.solve(R, z)`	Solve $R\beta = z$

Diagnostics and Stability Tools

Concept	**Command**	**Description**
Condition number	`np.linalg.cond(A)`	Numerical stability check
Check symmetry	`np.allclose(A, A.T)`	Is A symmetric?
Positive definite	`np.all(np.linalg.eigvals(A) > 0)`	Eigenvalues > 0?
Orthogonality check	`np.allclose(Q.T @ Q, I)`	Verifies Q orthogonal
QR regression	`Q,R=scipy.linalg.qr(X, mode='economic')`	Stable OLS via QR
Solve via QR	`beta = scipy.linalg.solve_triangular(R, Q.T @ y)`	Solve $R\beta = Q^\top y$

NB: These are for Scipy 1.16 - if you are using older versions your mileage may vary. Always check documentation