

Data Analysis : Linear Systems

Contents

1	Linear Systems: From Equations to Matrices	3
1.1	A Simple Motivating Example	3
1.2	Geometric View	3
1.3	Matrix Representation	4
1.4	Solving Linear Systems via Matrix Decompositions: LU and QR	4
1.4.1	LU Decomposition: Forward and Backward Substitution .	5
1.4.2	QR Decomposition: Orthogonality for Stability	5
2	Linear Systems and Linear Regression	7
2.1	Problem Setup	7
2.2	Objective of Linear Regression	9
2.3	Matrix Formulation	9
3	Types of Linear Systems	11
3.1	Square Full-Rank Systems ($m = n$)	11
3.2	Overdetermined Systems ($m > n$)	11
3.3	Underdetermined Systems ($m < n$)	11
4	Least Squares Problems	14
4.1	Problem Statement	14
4.2	Intuition: Projection Onto the Column Space	14
4.3	Theorem (Orthogonality Condition)	15
4.4	Derivation via Expansion of the Objective	15
4.5	Pseudo-Inverse and Explicit Solution	16
4.6	Computing the Pseudo-Inverse in Practice	16
4.7	Example: Polynomial Regression	17
5	Underdetermined Linear Systems and Regularization	19
5.1	Tikhonov (L2) Regularization (Ridge Regression)	19
5.2	Other Types of Regularization	20
6	Exercises	22
6.1	Polynomial Models and Linear Systems	22
6.1.1	Case 1: Square System — Exact Polynomial Interpolation	22

6.1.2	Case 2: Overdetermined System — Least Squares Polynomial Fit	23
6.1.3	Case 3: Underdetermined System — Too Few Data, Too Many Coefficients	24
7	Exercises	25
7.1	Exercises: Linear Systems	25
7.2	Exercises: Types of Linear Systems	26
7.3	Exercises: Least Squares	26
7.4	Exercises: Regularized Linear Regression	27
7.5	Ethics Questions	27
8	Practical Session Question	29

1 Linear Systems: From Equations to Matrices

A **linear system** is a collection of equations in which each equation expresses a linear relationship between a set of unknown variables. Linear systems arise whenever multiple constraints, measurements, or interactions relate several unknown quantities. Such systems are very important in different domains such as biology, physics, chemistry, and engineering.

1.1 A Simple Motivating Example

Consider a metabolite whose production is regulated by two genes, denoted a and b . Let x and y represent the (unknown) regulatory strengths of genes a and b , respectively. These quantities may be positive or negative, corresponding to activation or repression. Moreover, let us assume that the gene expressions and the metabolite level are represented as a deviations from a baseline (and thus can be negative).

Assume that the metabolite level M responds *linearly* to the expression levels of the two genes. In other words, for a given experiment with gene expression levels (E_a, E_b) , the resulting metabolite concentration M satisfies the model

$$M = x E_a + y E_b.$$

We perform two experiments in which genes a and b are expressed at different controlled levels. Suppose the measured metabolite concentrations are as follows:

$$\begin{cases} \text{Experiment 1: } M_1 = x \cdot 2 + y \cdot (-1) = 3, \\ \text{Experiment 2: } M_2 = x \cdot 1 + y \cdot 4 = 10. \end{cases}$$

Thus, the regulatory strengths x and y must satisfy the linear system:

$$\begin{cases} 2x - y = 3, \\ x + 4y = 10. \end{cases}$$

Each equation corresponds to a distinct experimental condition, and each provides a linear constraint relating the unknown regulatory influences x and y to the observed metabolite concentration. Our goal is to recover the pair (x, y) that explains both observations simultaneously.

1.2 Geometric View

Each equation describes a straight line in the plane. A solution corresponds to the point where the lines intersect: And thus a linear system is simply the problem of finding where several geometric constraints meet. In higher dimensions, instead of lines, we intersect planes, hyperplanes, or more abstract linear constraints. The central idea remains the same.

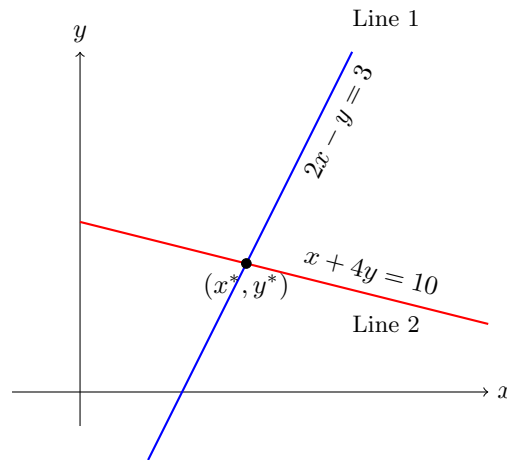


Figure 1: Geometric interpretation of the previous linear system: each equation defines a line; the solution is their intersection.

1.3 Matrix Representation

To write the system more compactly, we collect:

- the coefficients of the variables into a matrix A ,
- the unknowns into a vector x ,
- the right-hand sides into a vector b .

For the example above:

$$\mathbf{A} = \begin{pmatrix} 2 & -1 \\ 1 & 4 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 3 \\ 10 \end{pmatrix}.$$

The system becomes

$$\boxed{\mathbf{Ax} = \mathbf{b}}.$$

This compact formulation is the starting point for numerical linear algebra, matrix decompositions, regression, PCA, and many more operations that we will discover together during this lecture.

1.4 Solving Linear Systems via Matrix Decompositions: LU and QR

Once a system is written in matrix form $Ax = b$, solving it efficiently and reliably becomes a major task. In practice, and mainly in the case of large systems, direct algebraic manipulation numerically unstable and untractable. Instead,

numerical linear algebra relies on **matrix decompositions** that express A as dot products of simpler matrices. Two important decompositions are the so-called **LU** and **QR** decompositions.

1.4.1 LU Decomposition: Forward and Backward Substitution

Under some conditions, matrix A can be factored as

$$A = LU,$$

where:

- L is a **lower triangular matrix** with ones on the diagonal,
- U is an **upper triangular matrix**.

Solving $Ax = b$ becomes a two-step process:

$$LUx = b.$$

1. Solve **forward substitution**:

$$Ly = b.$$

2. Solve **backward substitution**:

$$Ux = y.$$

This strategy is extremely efficient: both triangular solves take only a quadratic number of operations, and the factorization $A = LU$ is only computed once.

1.4.2 QR Decomposition: Orthogonality for Stability

For solving linear systems another important factorization is the QR decomposition. Any full-rank matrix $A \in \mathbb{R}^{m \times n}$ (with $m \geq n$) can be factored as

$$A = QR,$$

where:

- Q is an $m \times m$ **orthogonal** (or unitary) matrix
- R is an $m \times n$ matrix whose top $n \times n$ block is **upper triangular**

If we solve a full-rank square system ($m = n$), this reduces to

$$A = QR, \quad Q^\top Q = I.$$

To solve $Ax = b$:

$$QRx = b \quad \Rightarrow \quad Rx = Q^\top b.$$

Since R is upper triangular, this system is solved by backward substitution.

QR is numerically more stable than LU for certain problems because orthogonal matrices preserve lengths: $\|Qx\| = \|x\|$. Given these properties, QR is often used to solve ill-conditioned systems and overdetermined systems ($m > n$) using least-squares regression (explained in the next sections), as well as for iterative numerical methods (e.g., algorithms for eigenvalues).

Summary:

A **linear system** is written in matrix form as

$$Ax = b,$$

where solving the system means finding the vector x that satisfies all linear constraints. Geometrically, each equation represents a hyperplane, and the solution is their intersection. Efficient numerical solution relies on transforming the system into simpler triangular forms.

Two classical factorizations are used in practice:

- **LU decomposition** ($A = LU$): efficient for *square* systems and ideal when solving repeatedly for different right-hand sides.
- **QR decomposition** ($A = QR$): numerically more stable for *rectangular* or overdetermined systems, and the standard tool for **least-squares** problems.

2 Linear Systems and Linear Regression

Imagine you have a set of observations, each observation could be a single input value x_i , or a vector of values (often called features). For instance, each data point may represent several measured quantities, or several nonlinear transformations of the same underlying variable.

For example, let us imagine that we want to understand how several measurable cellular quantities jointly influence the expression of a target gene. For each biological sample i , we record:

- the concentration of a transcription factor TF1,
- the concentration of TF2,
- the cell's metabolic state via the ATP level
- and the observed expression level of a target gene y_i .

We suspect that the target gene expression depends on these quantities in a way that is approximately linear (once we apply appropriate transformations). Thus, for each sample i , we represent the biological state as a vector

$$x_i = (\text{TF1}_i, \text{TF2}_i, \text{ATP}_i)$$

and we aim to predict gene expression with a model of the form

$$y_i \approx c_1 \text{TF1}_i + c_2 \text{TF2}_i + c_3 \text{ATP}_i + c_4.$$

This leads naturally to a linear regression problem: indeed, the coefficients c_1, \dots, c_4 quantify the influence of each biological factor on the gene expression level. The matrix formulation $Ac = y$ captures all observations at once. A schematic representation of the linear regression task is represented in Figure 2.

Notice that even when the model involves features that correspond to nonlinear transformations of the data (e.g. x^2 , $\log x$, $\sin x$), as long as the coefficients appear linearly, the method remains a *linear* regression. In Figure 3, we represent a linear model $f_c(x) = c_1 \sin(x) + c_2 x^2$, which combines two nonlinear transformations of the input variable x through a weighted linear sum.

2.1 Problem Setup

We consider:

- m **observations**, indexed by $i = 1, \dots, m$.
- For each observation, an **input vector**

$$x_i = (x_{i1}, x_{i2}, \dots, x_{ip}) \in \mathbb{R}^p,$$

representing p measured or computed quantities.

- Corresponding outputs $y_i \in \mathbb{R}$.

Thus, each data point is now a pair (x_i, y_i) with x_i multidimensional.

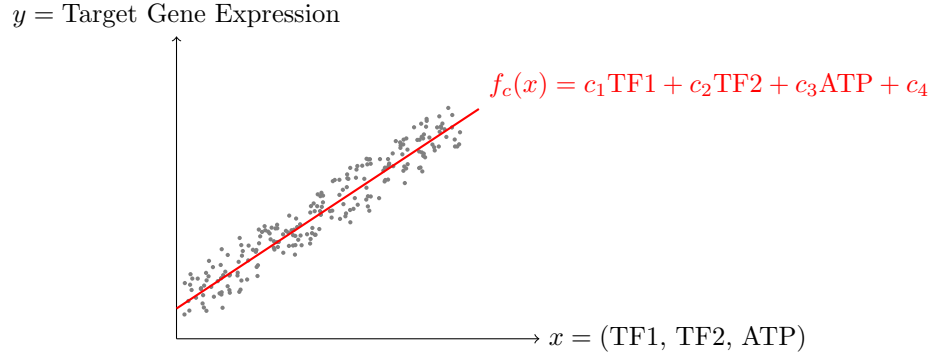


Figure 2: Schematic representation of the linear regression task: the x axis represent a 3D space $x = (\text{TF1}, \text{TF2}, \text{ATP})$, and the y axis represent simply the Target Gene Expression. Gray dots represent the experimental observations and the red line represents the linear model.

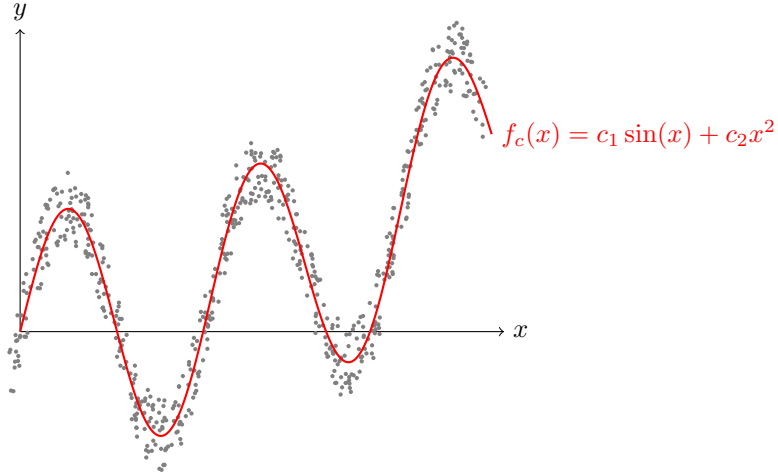


Figure 3: Schematic illustration of a linear regression task using two features derived as nonlinear transformations of a variable x , i.e., $\sin(x)$ and x^2 .

Feature Transformations. The components of x_i do not have to be the raw measured data. They can also be obtained from arbitrary transformations:

$$\phi_1(x_i), \phi_2(x_i), \dots, \phi_n(x_i),$$

where each ϕ_j extracts or computes a feature, for instance we can take:

- raw components: $x_{i1}, x_{i2},$
- polynomial features: $x_{i1}^2, x_{i1}x_{i2},$
- nonlinear transforms: $\log(x_{i2} + 1), \sin(x_{i1}),$

Thus, even a complex nonlinear model can be turned into a *linear* regression problem by defining appropriate feature functions ϕ_j .

2.2 Objective of Linear Regression

We assume a model of the form

$$f_c(x_i) = c_1 \phi_1(x_i) + c_2 \phi_2(x_i) + \dots + c_n \phi_n(x_i),$$

with $c = (c_1, \dots, c_n)^T$ unknown. The goal is to choose c so that $f_c(x_i)$ approximates y_i for all observations:

$$y_i \approx f_c(x_i).$$

The model is **linear in the parameters**, regardless of the complexity of the features. Typically, the “best approximation” means choosing c that minimizes the squared error *SSE*:

$$SSE = \sum_{i=1}^m (y_i - f_c(x_i))^2.$$

2.3 Matrix Formulation

We collect all feature evaluations into the **design matrix**

$$A = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \dots & \phi_n(x_m) \end{pmatrix}.$$

We also define:

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}, \quad c = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}.$$

The prediction equations for all observations can be written compactly as

$$Ac \approx y$$

In the special (idealized) case where the model fits perfectly, the system becomes the linear system

$$Ac = y.$$

This matrix formulation naturally leads to the different cases studied later: square systems, overdetermined systems (least squares), underdetermined systems (multiple solutions), and regularized regression.

Summary:

Linear regression provides a systematic framework for modeling the relationship between input variables (features) and an output variable by assuming that the prediction is a linear combination of feature transformations. By applying possibly nonlinear mappings such as $\phi_1(x) = 1$, $\phi_2(x) = x$, $\phi_3(x) = \sin(x)$, or $\phi_4(x) = x^2$, the model can capture complex behaviors while remaining **linear in the parameters**.

Given a set of observations, all feature evaluations are assembled into the *design matrix* A , and the regression task reduces to solving (exactly or approximately) the linear system

$$Ac \approx y,$$

where c contains the unknown coefficients.

3 Types of Linear Systems

Given a linear system, depending on how many equations (m) and unknowns (n) we have, we face fundamentally different scenarios.

3.1 Square Full-Rank Systems ($m = n$)

A system is **square** if the matrix A is $n \times n$. It is **full-rank** if its columns are linearly independent (i.e., $\det(A) \neq 0$). A square full-rank system has:

- exactly one solution,
- stable and well-defined algebraic structure.

Intuitively, each equation provides a unique, non-redundant constraint. Geometrically, this corresponds to n hyperplanes intersecting at exactly one point, as show in Figure 1.2. To solve such a system, we can use the methods detailed in the first Section of this chapter (i.e., using exact solvers based on LU or QR decomposition)

3.2 Overdetermined Systems ($m > n$)

A system is overdetermined if it has more equations than unknowns. This is a very common scenario in experimental sciences. Intuitively, this reflects the case where the system contains many constraints or measurements that could be noisy, inconsistent, or redundant. Figure ?? depicts a 2D representation of an overdetermined system. In general, no solution satisfies all equations exactly, instead we search for a coefficients vector c that best fits all constraints in an approximate sense: we aim at finding c such that Ac is *as close as possible* to the target vector y . This leads to the classic **least-squares problem**:

$$x^* = \arg \min_c \|Ac - y\|^2.$$

This problem can be solved using the techniques that will be studied in the next section.

3.3 Underdetermined Systems ($m < n$)

A system is underdetermined if it has more unknowns than equations. In this case, the system has an **infinite number of solutions**. And hence **additional constraints** are needed to select a meaningful solution. Intuitively, a system is underdetermined when we lack of information. Geometrically, an underdetermined system corresponds to an entire subspace, instead of a single point, and has thus an infinite number of points laying in it. This occurs in real world scenarios, where regression methods are applied to very high-dimensional datasets,

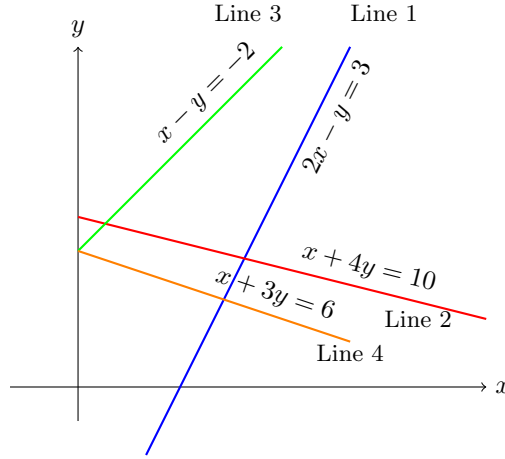


Figure 4: Geometric interpretation of an overdetermined linear system with more equations (lines) than variables. The exact solution (intersection of all the lines) do not exist.

with more features than experiments. This scenario is common in genomics or transcriptomics for instance.

Since the system has infinitely many solutions, we enforce extra constraints, that regularize the solution and make the problem well-defined. In practice different possibilities exist, for example:

- **Ridge regression** (L2 penalty): chooses smallest-norm solution.

$$x^* = \arg \min_x \|Ac - y\|^2 + \lambda \|c\|_2^2$$

- **LASSO** (L1 penalty): chooses sparse solutions, favoured in biology when only a few features matter.

$$x^* = \arg \min_x \|Ac - y\|^2 + \lambda \|x\|_1$$

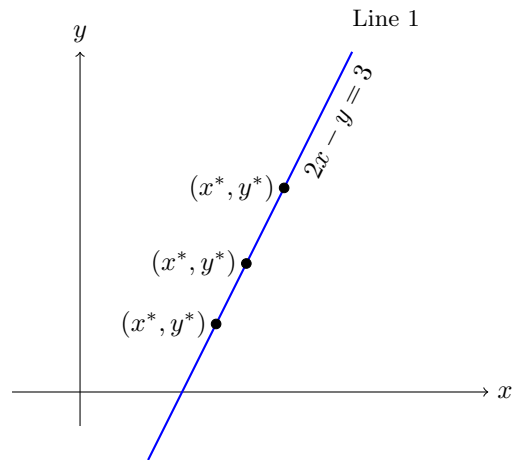


Figure 5: Geometric interpretation of an underdetermined linear system with less equations (lines) than variables. There exist an infinite number of possible solutions (all the points in the line).

Summary:

Depending on the shape and properties of the matrix A , the linear system $Ac = y$ can fall into one of the following categories:

System Type	Characteristics	Recommended Method
Square full-rank ($m = n$)	unique solution	LU or QR decomposition
Overdetermined ($m > n$)	No exact solution	Least-squares approximate solution
Underdetermined ($m < n$)	infinitely many solutions	regularization (ridge, LASSO, ...)

4 Least Squares Problems

In many real situations, the Linear system $Ac = y$ is **overdetermined** and does *not* have an exact solution.

$$\nexists c \text{ such that } Ac = y \iff y \notin \text{Im}(A).$$

The idea of *least squares* is to choose c so that Ac is as close as possible to y in the Euclidean norm. This gives an *approximate* solution that best fits the data. This approximate solution **minimizes** the distance with respect to the real solution.

4.1 Problem Statement

Given:

- $A \in \mathbb{R}^{m \times n}$ with $m > n$ (more equations than unknowns: **overdetermined system**)
- $y \in \mathbb{R}^m$

We seek $c \in \mathbb{R}^n$ minimizing the distance between Ac and y :

$$\|y - Ac\|_2^2.$$

Equivalently, let us define the **residual vector**

$$r = y - Ac,$$

and solve

$$\min_c \|r\|_2^2.$$

Notice that minimizing the norm of the residual or the euclidean distance between the approximations and the true values is equivalent.

4.2 Intuition: Projection Onto the Column Space

The columns of A span a subspace $\text{Im}(A) \subset \mathbb{R}^m$. Every vector of the form Ac lies in this subspace.

If $y \notin \text{Im}(A)$, the best we can do is project y onto $\text{Im}(A)$. The least-squares solution c^* is precisely the coefficient vector such that Ac^* is the orthogonal projection of y into $\text{Im}(A)$. Thus, at the minimum:

$$r = y - Ac^* \perp \text{Im}(A).$$

This orthogonality condition is the geometric foundation of least squares.

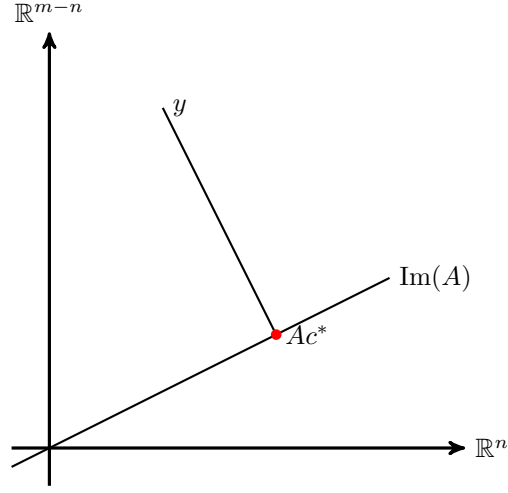


Figure 6: Least squares: the residual $r = y - Ac^*$ is orthogonal to $\text{Im}(A)$.

4.3 Theorem (Orthogonality Condition)

Let $A \in \mathbb{R}^{m \times n}$ with $m > n$, $R \in \mathbb{C}^m$, c^* minimizes $\|y - Ac\|_2^2$ if and only if

$$r = y - Ac^* \perp \text{Im}(A).$$

Equivalently:

$$\forall z \in \mathbb{R}^n, \quad (Az)^\top r = 0.$$

Since this holds for all z , we obtain:

$$A^\top r = 0.$$

Because $r = y - Ac^*$, we obtain the **normal equations**:

$$A^\top Ac^* = A^\top y.$$

4.4 Derivation via Expansion of the Objective

Another way to derive the normal equations is to expand the squared norm:

$$\begin{aligned} \|Ac - y\|_2^2 &= (Ac - y)^\top (Ac - y) \\ &= c^\top A^\top Ac - c^\top A^\top y - y^\top Ac + y^\top y. \end{aligned}$$

Differentiating with respect to c :

$$\frac{\partial}{\partial c} ((Ac - y)^\top (Ac - y)) = (A^\top A + (A^\top A)^\top) c - 2A^\top y.$$

Since $A^\top A$ is Hermitian:

$$A^\top A = (A^\top A)^\top,$$

so the critical point satisfies:

$$\boxed{A^\top A c^* = A^\top y}$$

This reproduces the normal equations.

4.5 Pseudo-Inverse and Explicit Solution

If A has full column rank (i.e., $\text{Rank}(A) = n$), then A^*A is invertible, and the solution is

$$\boxed{c^* = (A^*A)^{-1}A^\top y = A^+ y},$$

where A^+ is the **Moore–Penrose pseudo-inverse**. It maps $y \in \mathbb{R}^m$ to the least-squares optimal coefficients in \mathbb{R}^n .

4.6 Computing the Pseudo-Inverse in Practice

Although the pseudo-inverse is formally defined as $A^+ = (A^\top A)^{-1}A^\top$, this expression is rarely used directly in practice. The reason is that forming $A^\top A$ squares the condition number of A , which can lead to severe numerical instability, especially when A is ill-conditioned or nearly rank-deficient. Instead, the pseudo-inverse is typically computed using **matrix factorizations** that are more stable and efficient.

QR Decomposition If $A \in \mathbb{R}^{m \times n}$ with full column rank and $m \geq n$, we compute the QR decomposition

$$A = QR,$$

where $Q \in \mathbb{R}^{m \times n}$ has orthonormal columns and $R \in \mathbb{R}^{n \times n}$ is upper triangular. The least-squares solution then satisfies

$$Rc = Q^\top y,$$

which can be solved efficiently by backward substitution. In this case, the pseudo-inverse can be written as

$$A^+ = R^{-1}Q^\top.$$

This approach avoids forming $A^\top A$ and is numerically stable.

Singular Value Decomposition For general matrices (possibly rank-deficient or ill-conditioned), the most robust approach is the Singular Value Decomposition:

$$A = U\Sigma V^\top.$$

The pseudo-inverse is then given by

$$A^+ = V\Sigma^+U^\top,$$

where Σ^+ is obtained by inverting the nonzero singular values. This formulation: reveals the effective rank of A , allows safe handling of small singular values and provides the minimum-norm least-squares solution.

Both methods improve numerical stability, avoid squaring the condition number, naturally handles rank deficiency and are thus supported by highly optimized numerical libraries. For these reasons, modern scientific computing libraries *never* compute pseudo-inverses via $(A^\top A)^{-1}A^\top$, but rely instead on QR or SVD-based algorithms.

4.7 Example: Polynomial Regression

Let m observations (x_i, y_i) be given. We want to fit a polynomial of degree $n - 1$:

$$p(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}.$$

The least-squares objective is:

$$\min_c \sum_{i=1}^m |p(x_i) - y_i|^2.$$

This corresponds to the linear system:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}.$$

Since $m > n$, this is an **overdetermined** system, and the polynomial coefficients are obtained via:

$$c^* = A^+y.$$

This example illustrates how least squares provides a principled method to fit noisy data with a model that cannot exactly satisfy all measurements.

Summary:

Least squares problems arise when a linear system $Ac = y$ is **overdetermined** ($m > n$) and no exact solution exists. The goal is to find the coefficient vector c^* that minimizes the residual norm

$$\|y - Ac\|_2^2,$$

thus producing the best approximation of the data in the Euclidean sense.

Geometrically, the least-squares solution corresponds to the **orthogonal projection** of y onto the column space $\text{Im}(A)$. At the optimum, the residual $r = y - Ac^*$ is orthogonal to $\text{Im}(A)$, which leads to the **normal equations**

$$A^\top Ac^* = A^\top y.$$

When A has full column rank, the solution is unique and can be written explicitly using the **Moore–Penrose pseudo-inverse**:

$$c^* = A^+ y = (A^\top A)^{-1} A^\top y.$$

In practice, least-squares solutions and pseudo-inverses are computed using QR or SVD decompositions, as they are numerically more stable and robust than directly forming $(A^\top A)^{-1} A^\top$.

5 Underdetermined Linear Systems and Regularization

Consider a linear system $Ac = y$ with $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, where $m < n$. Such a system is called **underdetermined**: there are fewer equations than unknowns. In geometric terms:

- Each equation corresponds to a hyperplane in \mathbb{R}^n .
- With only m hyperplanes, we cannot isolate a single point unless $m = n$.
- Their intersection is typically a whole affine subspace of dimension at least $n - m$.

If the system is consistent (i.e. $y \in \text{Im } A$), the solution set is

$$c = c_0 + z, \quad z \in \ker(A)$$

where c_0 is any particular solution and $\ker(A)$ is nontrivial (dimension $\geq n - m$). Hence, underdetermined systems have infinitely many solutions.

If $y \notin \text{Im } (A)$, then the system does not admit an exact solution, and we must instead solve a **least-squares problem** or regularized problem. In applications this means that many different coefficient vectors c explain the data equally well. The model is **not identifiable** without adding constraints. Solutions become unstable: tiny perturbations in y yield completely different c . Underdetermination is not a rare theoretical edge-case but a typical scenario in modern data analysis. It happens frequently in biology (gene expression: n genes $\gg m$ samples), signal processing (high dimensional signals), machine learning with high-dimensional features (n large, m small)...

Several strategies exist to tackle this problem. In general we impose additional constraints. For example we can enforce non-negativity constraints ($c_i \geq 0$), sparsity constraints, smoothness constraints, structural assumptions (group sparsity, monotonicity, etc.). We also often apply regularization in order to modify the optimization problem so that only one solution is optimal. This is the standard approach in machine learning, inverse problems, and statistics. Different regularization techniques have been proposed so-far, with different properties (e.g., LASSO, Tikhonov, ElasticNet, OMP).

5.1 Tikhonov (L2) Regularization (Ridge Regression)

In the underdetermined case, the system $Ac = y$ does not have a unique solution or may not have any solution. We therefore solve instead:

$$c_\lambda = \arg \min_c \{ \|Ac - y\|_2^2 + \lambda \|c\|_2^2 \},$$

where $\lambda > 0$ controls the strength of penalization.

The term $\|Ac - y\|^2$ ensures that we stay close to the data. The term $\lambda\|c\|^2$ penalizes large coefficients. When there are infinitely many c fitting the data, the regularizer selects the one with smallest norm. As $\lambda \rightarrow 0$, we recover the minimal-norm solution. Tikhonov regularization ensures **uniqueness**, stabilizes the solution, handles noise in data, resolves underdetermination.

Let us derive the **Tikhonov Normal Equations**. To do so, let us minimize the function:

$$J(c) = \|Ac - y\|_2^2 + \lambda\|c\|_2^2.$$

Expand the terms:

$$\|Ac - y\|_2^2 = (Ac - y)^\top (Ac - y).$$

Compute the gradient with respect to c :

$$\nabla_c J(c) = 2A^\top (Ac - y) + 2\lambda c.$$

Set the gradient to zero:

$$A^\top (Ac - y) + \lambda c = 0.$$

Rearrange:

$$A^\top Ac + \lambda c = A^\top y.$$

This yields the **Tikhonov normal equations**:

$$(A^\top A + \lambda I)c = A^\top y.$$

Since $\lambda > 0$, the matrix

$$A^\top A + \lambda I$$

is always invertible. Thus, the problem is **always well-posed**, even when A is rank-deficient or $m < n$. The solution is then:

$$c_\lambda = (A^\top A + \lambda I)^{-1} A^\top y.$$

Why does Tikhonov Regularization Solve the Problem? The term λI makes $A^\top A + \lambda I$ strictly positive definite. Hence the inverse of $A^\top A + \lambda I$ always exists. Moreover, Small singular values of A (which could cause instability) are shifted upward by λ , i.e., $\sigma_i(A^\top A + \lambda I) = \sigma_i(A)^2 + \lambda$. Therefore, regularization improves conditioning and stabilizes the solution.

5.2 Other Types of Regularization

Although Tikhonov (L2) is the most classical and analytically convenient, other forms of regularization are widely used.

LASSO regularization replaces the L2 penalty by a L1 regularization.

$$\min_c \|Ac - y\|_2^2 + \lambda \|c\|_1.$$

This promotes sparsity in c , and hence LASSO is useful for feature selection in high-dimensional data. Unlike Tikhonov regularization, this technique could suffer from numerical instabilities.

Elastic Net regression combines of L1 and L2 penalties:

$$\|Ac - y\|_2^2 + \alpha \|c\|_1 + \beta \|c\|_2^2.$$

By combining both penalties, the Elastic Net aims at promoting sparsity while keeping a numerically stable method.

Constraints-Based Regularization

$c \geq 0$ (non negative), $\|c\|_0 \leq k$ (OMP), monotonicity constraints, etc ...

These approaches encode domain knowledge into the model.

Summary:

Underdetermined linear systems arise when $m < n$, meaning there are fewer equations than unknowns. Such systems admit infinitely many solutions or none at all, and the solution set is unstable: many coefficient vectors c fit the data equally well, making the model non-identifiable. To recover a unique and stable solution, additional constraints or regularization must be imposed.

Tikhonov (L2) regularization addresses this by solving

$$c_\lambda = \arg \min_c \{\|Ac - y\|_2^2 + \lambda \|c\|_2^2\},$$

which leads to the normal equations

$$(A^\top A + \lambda I)c = A^\top y.$$

Since $A^\top A + \lambda I$ is always invertible for $\lambda > 0$, the problem becomes well-posed and numerically stable.

Other regularization strategies (LASSO, Elastic Net, or domain-specific constraints such as non-negativity or sparsity limits) provide additional flexibility, especially in high-dimensional settings common in modern biological and data-driven applications.

6 Exercises

6.1 Polynomial Models and Linear Systems

Polynomial fitting is a natural setting to illustrate the three possible types of linear systems (square, overdetermined, and underdetermined) depending on the relation between the number of data points and the degree of the polynomial.

6.1.1 Case 1: Square System — Exact Polynomial Interpolation

Consider m data points (x_i, y_i) with all x_i distinct. There exists a unique polynomial of degree $m - 1$,

$$p(x) = c_0 + c_1x + \cdots + c_{m-1}x^{m-1},$$

such that

$$p(x_i) = y_i, \quad i = 1, \dots, m.$$

This yields the square linear system

$$Ac = y,$$

where A is the so-called $m \times m$ Vandermonde matrix

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^{m-1} \end{pmatrix}.$$

Since A is invertible when all x_i are distinct,

$$\boxed{c = A^{-1}y}.$$

Python Implementation

- Generate the conditions:

$$x = \left(\frac{1}{s}, \dots, \frac{10}{s}\right), \quad s = 10.$$

- Generate noisy observations:

$$y = 30 - x + 3 * x^3 + \epsilon \quad \text{with: } \forall \epsilon_i \in \epsilon : \quad \epsilon_i \sim \mathcal{N}(0, 10)$$

- Construct $A = \text{np.vander}(x)$.
- Compute $c = A^{-1}y$ using `np.linalg.inv`.
- Generate new points x' and evaluate $p(x')$.
- Plot the interpolation curve and the original points.

Useful Python functions: `np.arange`, `np.random.randn`, `np.vander`, `np.linalg.inv`.

Questions:

- What happens when the degree becomes too large?
- Try changing the spacing of x_i — what changes?

6.1.2 Case 2: Overdetermined System — Least Squares Polynomial Fit

Interpolation requires $m = n$, but in practice we often have $m > n$: more data points than polynomial coefficients.

Let $m = 10$ points and fit a polynomial of degree $n - 1$ with $n < m$.

This yields an **overdetermined** system:

$$Ac \approx y, \quad A \in \mathbb{R}^{m \times n}, \quad m > n.$$

There is no exact solution, hence we compute the least squares solution:

$$c = A^+ y = (A^* A)^{-1} A^* y.$$

Python Implementation

- Generate x and y as in the previous section.
- Build the truncated Vandermonde matrix:

$$A = \text{np.vander}(x, N = n) \quad (n < m).$$

- Compute the least-squares solution:

$$c = A^+ y, \quad A^+ = \text{np.linalg.pinv}(A).$$

- Evaluate the fitted polynomial on new points.
- Plot the noisy data and the fitted curve.

Useful functions: `np.vander`, `np.linalg.pinv`, `np.linalg.lstsq`.

Questions:

- Does increasing n reduce the residuals?
- Compare the coefficients of the fitted polynomial and the real ones for different values n .
- What is the risk of choosing n too large?

6.1.3 Case 3: Underdetermined System — Too Few Data, Too Many Coefficients

Let us now take the opposite situation, i.e., more polynomial coefficients than data points :

$$n > m$$

Example:

- $m = 5$ points,
- fit a polynomial of degree $n - 1 = 9$ ($n = 10$).

Then

$$A \in \mathbb{R}^{5 \times 10}, \quad \text{rank}(A) = 5 < 10.$$

This yields an **underdetermined system**: infinitely many exact interpolating polynomials satisfy $Ac = y$.

Tikhonov Regularization

To select a meaningful solution (e.g., smoothest, smallest coefficients), we use **Tikhonov regularization** (ridge regression):

$$\min_c (\|Ac - y\|_2^2 + \lambda \|c\|_2^2).$$

Normal equations:

$$(A^*A + \lambda I)c = A^*y.$$

This ensures a unique solution because $A^*A + \lambda I$ is invertible for all $\lambda > 0$.

Python Implementation

1. Generate x and y as in the previous case, but take only 5 points, and all columns from the Vandermonde matrix.
2. Show numerically that A is rank-deficient.
3. Compute the ridge-regression solution:

$$c_\lambda = (A^*A + \lambda I)^{-1} A^*y.$$

4. Plot the resulting polynomial.

Questions:

- How does the solution change as you vary λ ?
- What happens as $\lambda \rightarrow 0$?
- Why is regularization essential in high-degree polynomial models?

7 Exercises

This section provides exercises for each major part of the course. The goal is to test intuition, mathematical understanding, and the ability to apply linear algebra tools to real biological or data-science problems.

7.1 Exercises: Linear Systems

1. **Geometric interpretation.** Explain in words what it means to solve the system $Ax = b$. Discuss the geometric meaning when $A \in \mathbb{R}^{2 \times 2}$ versus $A \in \mathbb{R}^{2 \times 3}$.

2. **Hyperplane intersection.** Consider the system:

$$\begin{cases} x + y + z = 3, \\ 2x - y + z = 1. \end{cases}$$

Describe the set of solutions geometrically.

3. **Linear dependence.** Analyse the system

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} x = \begin{pmatrix} 3 \\ 6 \end{pmatrix}.$$

Explain why the equations are redundant and why infinitely many solutions exist.

4. **Matrix formulation.** Write the system:

$$4x_1 - x_2 = 0, \quad -2x_1 + 3x_2 = 5$$

in matrix form. Identify A , x , and b .

5. **Row scaling.** Consider $Ax = b$. If one row of A and the corresponding entry of b are multiplied by a constant, does the solution change? Justify.
6. **Rank computation.** Compute the rank of the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 1 & 0 \end{pmatrix}.$$

How many solutions can the system $Ax = b$ have?

7. **Consistency.** Determine whether the system $Ax = b$ is consistent for:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 3 \end{pmatrix}.$$

8. **Identify the system type.** For the matrices

$$A_1 \in \mathbb{R}^{3 \times 2}, \quad A_2 \in \mathbb{R}^{2 \times 3}, \quad A_3 \in \mathbb{R}^{3 \times 3},$$

explain whether the associated linear system could be underdetermined, overdetermined, or square.

7.2 Exercises: Types of Linear Systems

1. Give an example of a full-rank square system and explain how to solve it.
2. Construct an overdetermined system with no exact solution. Explain why no x satisfies all the equations.
3. Provide an underdetermined system with infinitely many solutions and parametrize the solution space.
4. For each matrix below, determine whether the system is: underdetermined, overdetermined, or square full rank:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \end{pmatrix}.$$

5. Explain why an overdetermined system typically has no exact solution.
6. Explain why an underdetermined system typically has infinitely many solutions.
7. Application: In gene expression inference, one often measures fewer conditions than the number of regulatory factors. Explain why this leads to an underdetermined system.
8. Construct a biological example of an overdetermined system.
9. Explain why LU decomposition is appropriate for square full-rank systems.
10. Explain why the pseudo-inverse is needed for non-square systems.

7.3 Exercises: Least Squares

1. Show that the residual vector $r = y - Ac$ is orthogonal to $\text{Im}A$ at the minimum of the least-squares problem.
2. Given

$$A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad y = \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix},$$

compute the least-squares solution.

3. Derive the normal equations from minimizing $\|y - Ac\|_2^2$.
4. Discuss why the matrix A^*A may be ill-conditioned.
5. Show that if A has full column rank then the least-squares solution is unique.

6. For a polynomial of degree 2, write the least-squares matrix for data (x_i, y_i) , $i = 1, \dots, m$.
7. Discuss the risk of overfitting in least-squares polynomial fitting.
8. Apply the SVD to the normal equations and give the corresponding expression for the pseudo-inverse matrix
9. Apply the QR decomposition to the normal equations and give the corresponding expression for the pseudo-inverse matrix

7.4 Exercises: Regularized Linear Regression

1. The Tikhonov regularization solves

$$\min_c \|y - Ac\|_2^2 + \lambda \|c\|_2^2$$

derive the associated normal equations $(A^*A + \lambda I)c = A^*y$.

2. Explain why $A^*A + \lambda I$ is always invertible for $\lambda > 0$.
3. Application: In ridge regression for gene expression prediction, explain what the λ parameter controls.
4. Compare the solutions of the unregularized and regularized systems for a simple underdetermined example.
5. One of your colleague affirms that it is totally wrong to apply Tikhonov regularization to an overdetermined system, what do you think about that?
6. Discuss why regularization helps when columns of A are nearly linearly dependent.
7. Explain the effect of increasing λ on the magnitude of coefficients.
8. Formalize the LASSO optimization problem and explain how it differs from ridge regression.
9. Explain why regularization is essential in high-dimensional biological datasets (e.g., transcriptomics).

7.5 Ethics Questions

1. What are the ethical risks of using ill-conditioned linear models in medicine?
2. Search the definition of overfitting in the context of regression models. Explain it using the concepts studied during this lecture.
3. In biological data analysis, how can overfitting lead to incorrect clinical decisions?

4. Let us imagine a company that uses an algorithm that aims at creating black-box polynomial regression model that reduces the residual norm almost to zero. Whould you apply such a tool to bio-medical data? Why?
5. You colleague decides to use regularization to "smooth" noisy data that may contain rare events in biomedical data. What do you think about that?
6. What biases may arise from using least squares when data contain outliers? Why?
7. Why must researchers disclose when a model is underdetermined and relies heavily on priors or regularization?

8 Practical Session Question

In [1], the authors focus on the visual system of the fruit fly (*Drosophila melanogaster*) and develop a comprehensive genomic, and computational resource that maps gene expression profiles onto well-characterized neural cell types. This resource enables a deeper understanding of how the molecular properties of individual neurons relate to neural circuit organization and function.

In this study, the authors identify several genes of particular interest. In this practical, we will focus on the following subset: Vmat, Nos, AstA, ple, DAT, VACHT, VGlut, Gad1, AstC, Pdfr, GluClalpha, Oamb, para.

Download the gene expression dataset from: <https://www.ebi.ac.uk/gxa/experiments/E-GEOD-116969/Downloads> (use the file containing *expression values across all genes (TPMs)*). This dataset provides the expression level of each gene (rows) across multiple cell types (columns). Restrict the analysis to columns corresponding to neuronal cell types only.

From the FlyEnrichr platform: <https://maayanlab.cloud/FlyEnrichr/#stats>, we have prepared two JSON files. Each file contains, for a given target gene (key), the list of transcription factors (TFs) that have been experimentally or computationally associated with its regulation (values). The two files are derived from distinct databases: `TF2DNA_2018` and `TranscriptionFactors_fromDroID_2015`.

For each target gene, your goal is to construct a linear regression model that explains its expression across neuronal cell types as a function of the expression levels of its associated transcription factors.

Address the following questions:

- Is it always possible to build such a linear model?
- Which type of regression would be appropriate in each situation?
- What issues arise when an unsuitable regression method is applied?
- Fit the models for each gene, compute and compare the norms of the residuals (errors), and analyze the resulting regression coefficients.
- Implement the Orthogonal Matching Pursuit algorithm to perform an ℓ_0 regularized linear regression (the pseudocode is provided hereafter).
- Now try to build regression models providing all tfs at once. Explore different values for the OMP's level of sparsity (for example between 1 and 80), and analyze the error as well as coefficients vectors c .
- Discuss how the choice of regularization affects interpretability, and predictive performance.

Orthogonal Matching Pursuit (OMP) aims at computing an approximate solution to the sparse linear regression problem

$$\min_c \|y - Xc\|_2^2 \quad \text{subject to} \quad \|c\|_0 \leq N,$$

where $\|c\|_0$ denotes the ℓ_0 pseudo-norm, i.e. the number of nonzero components of the vector c . At each step, OMP selects the column of X that is most correlated with the current residual and then recomputes the least-squares solution restricted to the selected predictors.

Input: $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times p}$, sparsity level N

• **Initialization:**

$$r \leftarrow y, \quad \Lambda \leftarrow \emptyset$$

• **For** $k = 1, \dots, N$:

- Select the index most correlated with the residual:

$$\lambda \leftarrow \arg \max_{j \notin \Lambda} \frac{|X_j^\top r|}{\|X_j\|_2}$$

- Update the active set:

$$\Lambda \leftarrow \Lambda \cup \{\lambda\}$$

- Solve the restricted least-squares problem:

$$c_\Lambda \leftarrow \arg \min_c \|X_\Lambda c - y\|_2^2, \quad c_j = 0 \text{ for } j \notin \Lambda$$

- Update the prediction and residual:

$$\hat{y} \leftarrow Xc, \quad r \leftarrow y - \hat{y}$$

Return: the sparse coefficient vector c

Bonus: in real-world applications, it is crucial to assess the *generalization ability* of predictive models. Investigate common strategies for model evaluation (e.g. cross-validation, train/test splits, bootstrapping), and propose a concrete methodology to assess generalization performance in this context (no implementation is required).

References

- [1] F. P. Davis, A. Nern, S. Picard, M. B. Reiser, G. M. Rubin, S. R. Eddy, and G. L. Henry. A genetic, genomic, and computational resource for exploring neural circuit function. *elife*, 9:e50901, 2020.