Chapter 4

Linear equations and least-squares

Contents (class version)

4.0 Introduction to linear equations

Source material for this chapter includes [1, §6.1, 8.1–8.4, 4.1, 5.2]. L§6.1

Solving a **system of linear equations** arises in numerous applications. Mathematically, a system of M equations in N unknowns is usually written

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \quad \boldsymbol{A} \in \mathbb{F}^{M \times N}, \quad \boldsymbol{x} \in \mathbb{F}^{N}, \quad \boldsymbol{y} \in \mathbb{F}^{M}. \tag{4.1}$$

Typically $\boldsymbol{A}$ is known from some type of (linear) modeling, $\boldsymbol{y}$ corresponds to some type of measurements, and the goal is to solve for $\boldsymbol{x}$.

However, despite the equals sign in the classic expression (4.1), in practice often there does not *exist* any $\boldsymbol{x}$ that yields equality, particularly when $\boldsymbol{A}$ is tall, so the notation "$\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{y}$" would be more realistic.

Sometimes there is not a *unique* $\boldsymbol{x}$ that satisfies (4.1), in particular whenever $\boldsymbol{A}$ is wide. Linear algebra texts focus on examining when a solution $\boldsymbol{x}$ exists and is unique for (4.1).

Solving $\boldsymbol{Ax} = \boldsymbol{y}$ (Read)

[1, Theorem 6.1] reviews the principal existence and uniqueness results for (4.1).

1. There *exists* a solution $\boldsymbol{x}$ iff $\boldsymbol{y} \in \mathcal{R}(\boldsymbol{A})$.
2. There *exists* a solution $\boldsymbol{x}$ for all $\boldsymbol{y} \in \mathbb{R}^M$ iff $\mathcal{R}(\boldsymbol{A}) = \mathbb{F}^M$.
3. A solution $\boldsymbol{x}$ is *unique* iff $\mathcal{N}(\boldsymbol{A}) = \{\boldsymbol{0}\}$.
4. There exists a unique solution for all $\boldsymbol{y} \in \mathbb{R}^M$ iff $\boldsymbol{A}$ is $M \times M$ and non-singular (invertible), *i.e.*, none of the eigenvalues or singular values of $\boldsymbol{A}$ are zero.
5. There is at most one solution for all $\boldsymbol{y} \in \mathbb{R}^M$ iff $\boldsymbol{A}$ has linearly independent columns, *i.e.*, $\mathcal{N}(\boldsymbol{A}) = \{\boldsymbol{0}\}$, and this is possible only if $M \geq N$.
6. The homogeneous system $\boldsymbol{Ax} = \boldsymbol{0}$ has a nontrivial (*i.e.*, nonzero) solution iff $\mathsf{rank}(\boldsymbol{A}) < N$.

(You should read and verify these points.)

To understand #3, suppose $\boldsymbol{x}_0 \in \mathcal{N}(\boldsymbol{A})$ and $\boldsymbol{x}_0 \neq \boldsymbol{0}$ and suppose we have a solution $\boldsymbol{Ax}_1 = \boldsymbol{y}$. Then $\boldsymbol{x}_2 = \boldsymbol{x}_1 + \boldsymbol{x}_0$ is also a solution because $\boldsymbol{Ax}_2 = \boldsymbol{Ax}_1 + \boldsymbol{Ax}_0 = \boldsymbol{y} + \boldsymbol{0}$.

When $\boldsymbol{A}$ is wide ($M < N$), $\mathcal{N}(\boldsymbol{A})$ is nontrivial and there are (infinitely) many solutions to (4.1). One must design some way to choose among all those solutions.

When $\boldsymbol{A}$ is square ($M = N$) and full rank (and hence invertible), then there is a unique solution $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}$. We say that "the number of equations" equals "the number of unknowns."

Linear regression and machine learning

In **linear regression**, we are given a set of training data consisting of M input (feature) vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_M \in \mathbb{F}^N$ and corresponding responses $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M \in \mathbb{F}$ and we want to find coefficients / weights $\boldsymbol{x} \in \mathbb{F}^N$ such that $\boldsymbol{a}_m^T \boldsymbol{x} \approx y_m$. Stacking up the input vectors into a matrix $\boldsymbol{A}$ and the response variables into a vector $\boldsymbol{y}$ yields the following matrix-vector form:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} \approx \begin{bmatrix} \boldsymbol{a}_1^T \\ \vdots \\ \boldsymbol{a}_M^T \end{bmatrix} \boldsymbol{x}, \quad \textit{i.e.}, \quad \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon} \in \mathbb{F}^M$ denotes noise or other errors.

We want to determine ("learn") the weight vector, or coefficients, $\boldsymbol{x}$ so that given some future feature vector $\boldsymbol{a}$, we can **predict** the corresponding (unknown) response by simply computing $\boldsymbol{a}^T \boldsymbol{x}$.

This is a classical problem in statistics and it is key "machine learning" method that one usually should consider (for estimation/regression problems) before considering more complicated methods.

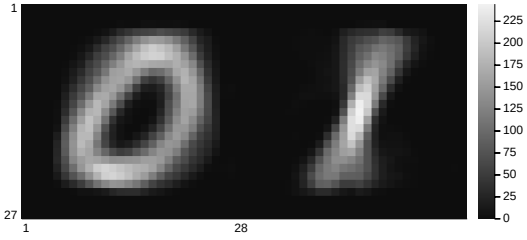
The notation for linear regression varies by discipline:
in statistics: $\boldsymbol{y} \approx \boldsymbol{X}\boldsymbol{\beta}$; in machine learning: $\boldsymbol{y} \approx \boldsymbol{X}\boldsymbol{w}$; linear algebra: $\boldsymbol{y} \approx \boldsymbol{A}\boldsymbol{x}$.

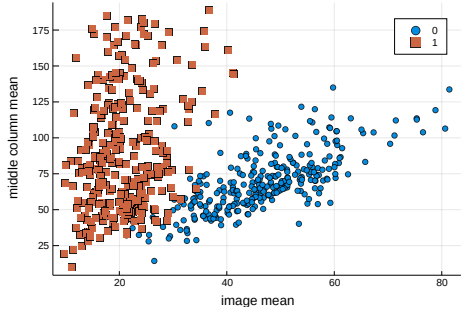
The variables here have many different names:

- $\boldsymbol{y}$: response, labels, regressand, endogenous variable, measured variable, criterion variable, dependent variable, predicted variable, ...
- $\boldsymbol{a}$ (rows of $\boldsymbol{A}$) : features, regressors, exogenous variables, explanatory variables, covariates, input variables, predictor variables, independent variables, ...
- $\boldsymbol{x}$: parameter vector, regression coefficients, unknown, ...
- $\boldsymbol{\varepsilon}$: noise, disturbance, error, ...

Example. Predict child's height from parent's height. [wiki]

Example. Hand-written digit classification using two "hand-crafted" features (plus a bias or offset).
Here $\boldsymbol{A}$ is $3 \times (28 \cdot 27) = 3 \times 756$.





4.1 Linear least-squares estimation

L§8.1

In a typical situation where $M > N$, called "tall" or "over-determined," one can show that $\mathcal{R}(\boldsymbol{A}) \neq \mathbb{F}^M$. So there will be (infinitely) many $\boldsymbol{y}$ (those not in $\mathcal{R}(\boldsymbol{A})$) for which no solution $\boldsymbol{x}$ exists.

Thus, when $M > N$, instead if insisting on *exactly* solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$, usually we look for *approximate* solutions where $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{y}$, To do this, we must quantify "approximately equal."

The most important and common approximate solution is to use **linear least-squares** (**LLS**) estimation or fitting, where we find an estimate $\hat{\boldsymbol{x}}$ that "best fits" the data $\boldsymbol{y}$ using a Euclidean norm distance as follows:

(4.2)

- $\hat{\boldsymbol{x}}$ is called the **linear least-squares estimate** (or solution)
  (The "hat" or caret above $\boldsymbol{x}$ is often used to denote an estimate.)
- $\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}$ is called the **residual**
- $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2$ is the Euclidean norm of the residuals for a candidate solution $\boldsymbol{x}$ and is a measure of the "error" of the fit. It is sometimes called the "goodness of fit" even though a larger value means a worse fit!
- $\arg\min_{\boldsymbol{x}}$ means that we see the argument $\hat{\boldsymbol{x}}$ that minimizes the fitting error; the minimum value of the fitting error: $\min_{\boldsymbol{x} \in \mathbb{F}^N} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ usually is of less interest.
  In words we often say "we minimized the squared error" but we really mean "we found a solution $\hat{\boldsymbol{x}}$ that

minimized the squared error."

Although this technique is ancient, it remains one of the most important linear algebra methods for signal processing and data analysis in general.

Example. Suppose we observe noisy samples of signal: L§8.3

$$y_m = s(t_m) + \epsilon_m, \quad m = 1, \dots, M$$

and we believe that the signal is a cubic polynomial:

$$s(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3$$

with unknown coefficients. In matrix-vector form:

$$\boldsymbol{y} \approx \boldsymbol{A}\boldsymbol{x}, \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} 1 & t_1 & t_1^2 & t_1^3 \\ 1 & t_2 & t_2^2 & t_2^3 \\ \vdots & & & \vdots \\ 1 & t_M & t_M^2 & t_M^3 \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}.$$

A figure on p. 4.9 illustrates this problem. Here the "features" are the time points t_m raised to different powers, and we call it **polynomial regression**.

To find the coefficient vector $\boldsymbol{x}$, one option would be to take just $M = 4$ samples. As long as we pick 4 distinct t_m points then one can show (using the fact that monomials are linearly independent functions) that the 4×4 matrix $\boldsymbol{A}_4$ has **linearly independent** columns. Thus $\boldsymbol{A}_4$ is **invertible** and we could use $\hat{\boldsymbol{x}} = \boldsymbol{A}_4^{-1} \boldsymbol{y}$ as an estimate of the coefficients.

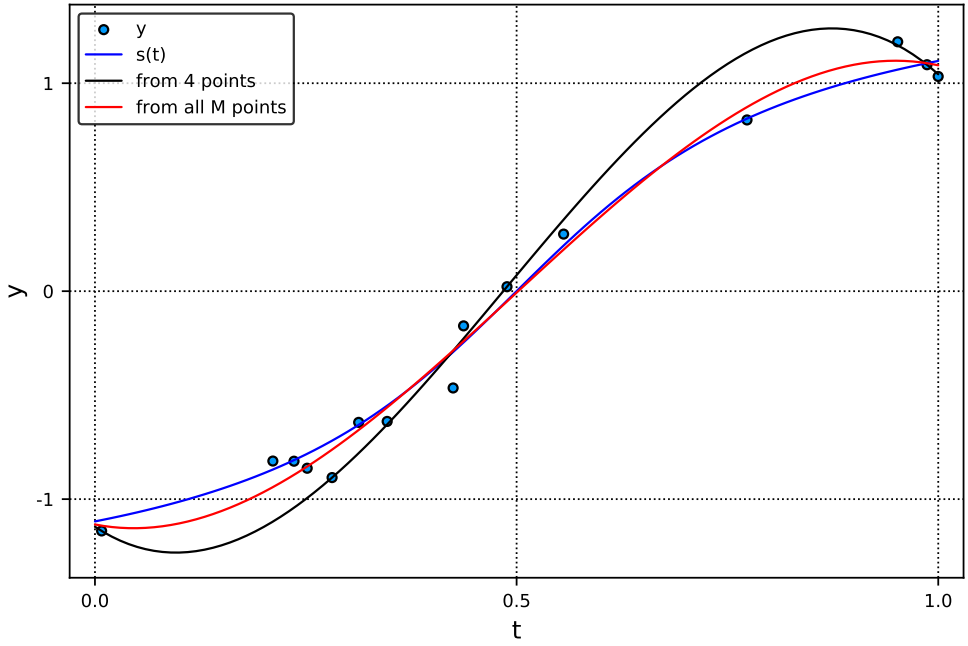
However, in the presence of noise in the data, this approach would give very noisy and unreliable estimates of the coefficients.

Instead, it is preferable to use all $M \gg 4$ samples and estimate $\hat{\boldsymbol{x}}$ using linear least-squares.

For higher polynomial orders, it is more stable to use **orthogonal polynomials** as the basis, instead of monomials. We use monomials here for simplicity in this example.

For a demo, see
https://web.eecs.umich.edu/~fessler/course/551/julia/demo/04_ls_fit1.html
https://web.eecs.umich.edu/~fessler/course/551/julia/demo/04_ls_fit1.ipynb

The following curves illustrate that fitting a cubic polynomial using all $M \gg 4$ points via (4.2) is better than using just 4 points with $\boldsymbol{A}_4^{-1}$. (One can prove that statement statistically; see EECS 564.)



A key line in the demo code is `xh = A \ y` and next we explain in detail the very important mathematical foundation behind that computation.

Minimization and gradients

The cost function in (4.2) is differentiable, so we use derivatives to find a minimizer.

Review of 1D minimization by example (Read)

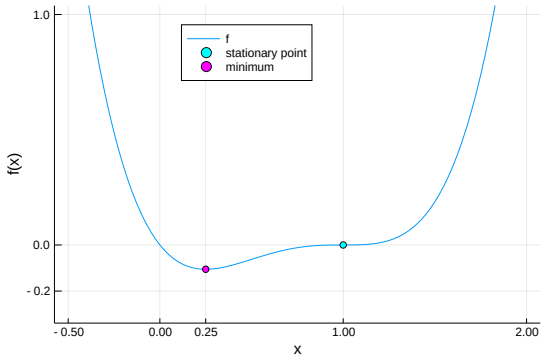
To find the minimizer of a function $f : \mathbb{R} \mapsto \mathbb{R}$ such as

$$f(x) = x(x-1)^3,$$

you simply take the **derivative** and set it equal to zero:

$$0 = \dot{f}(x) = (x-1)^2(4x-1).$$

This case has two roots at $x = 1$ and $x = 1/4$. Because $f(1) = 0$ and $f(1/4) < 0$ it looks like $x = 1/4$ is the minimizer.



To be sure that $x = 1/4$ is a minimizer, we also take the second derivative:

$$\ddot{f}(x) = 6(2x^2 - 3x + 1) \implies \ddot{f}(1/4) = 9/4 > 0,$$

so $x = 1/4$ is at least a local minimizer. Checking $\pm\infty$ verifies. This example required extra work because f is non-convex. For convex differentiable functions, simply finding a point x where $\dot{f}(x) = 0$ is sufficient.

Calculating gradients

Similarly, to solve the minimization problem (4.2), we will find a zero of its gradient.

Consider the **affine** function $f_1 : \mathbb{R}^N \mapsto \mathbb{R}$ defined for a vector $\boldsymbol{v} \in \mathbb{R}^N$ and $b \in \mathbb{R}$ by

$$f_1(\boldsymbol{x}) \triangleq b + \boldsymbol{v}'\boldsymbol{x} = b + \sum_{n=1}^{N} v_n x_n.$$

The (column) **gradient** of this function is the vector of partial derivatives:

$$\nabla f_1(\boldsymbol{x}) =$$

(Read)

Consider the **quadratic** function $f_2 : \mathbb{R}^N \mapsto \mathbb{R}$ defined for a matrix $\boldsymbol{M} \in \mathbb{R}^{N \times N}$ by

$$f_2(\boldsymbol{x}) \triangleq \frac{1}{2}\boldsymbol{x}'\boldsymbol{M}\boldsymbol{x} = \frac{1}{2}\sum_{k=1}^{N}\sum_{l=1}^{N} M_{kl} x_k x_l.$$

The partial derivative of this function w.r.t. x_n is

$$\frac{\partial}{\partial x_n} f_2(\boldsymbol{x}) = \frac{\partial}{\partial x_n}\frac{1}{2}\sum_{k=1}^{N}\sum_{l=1}^{N} M_{kl} x_k x_l.$$

I deliberately chose different letters k, l for the summations different from n to avoid mishaps.

Using linearity of differentiation:

$$\begin{aligned}
\frac{\partial}{\partial x_n} f_2(\boldsymbol{x}) &= \frac{1}{2} \sum_{k=1}^N \sum_{l=1}^N M_{kl} \left(\frac{\partial}{\partial x_n} x_k x_l \right) \\
&= \frac{1}{2} \sum_{k=1}^N \sum_{l=1}^N M_{kl} \left(x_l \mathbb{I}_{\{n=k\neq l\}} + x_k \mathbb{I}_{\{n=l\neq k\}} + 2 x_n \mathbb{I}_{\{n=k=l\}} \right) \\
&= \frac{1}{2} \left(\sum_{k=1}^N \sum_{l=1}^N M_{kl} x_l \mathbb{I}_{\{n=k\neq l\}} + \sum_{k=1}^N \sum_{l=1}^N M_{kl} x_k \mathbb{I}_{\{n=l\neq k\}} + 2 \sum_{k=1}^N \sum_{l=1}^N M_{kl} x_n \mathbb{I}_{\{n=k=l\}} \right) \\
&= \frac{1}{2} \left(\sum_{l=1}^N M_{nl} x_l \mathbb{I}_{\{n\neq l\}} + \sum_{k=1}^N M_{kn} x_k \mathbb{I}_{\{n\neq k\}} + 2 M_{nn} x_n \right) \\
&= \frac{1}{2} \sum_{l=1}^N M_{nl} x_l + \frac{1}{2} \sum_{k=1}^N M_{kn} x_k \\
&= \frac{1}{2} [\boldsymbol{M}\boldsymbol{x}]_n + \frac{1}{2} [\boldsymbol{M}'\boldsymbol{x}]_n,
\end{aligned}$$

where $[\boldsymbol{A}\boldsymbol{x}]_i$ denotes the ith element of the vector $\boldsymbol{A}\boldsymbol{x}$.

Now write the **gradient** of this function in column form by stacking up all the partial derivatives into a vector:

$$\nabla f_2(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f_2 \\ \vdots \\ \frac{\partial}{\partial x_N} f_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} [\boldsymbol{M}\boldsymbol{x}]_1 + [\boldsymbol{M}'\boldsymbol{x}]_1 \\ \vdots \\ [\boldsymbol{M}\boldsymbol{x}]_N + [\boldsymbol{M}'\boldsymbol{x}]_N \end{bmatrix} = \frac{1}{2} \left(\boldsymbol{M}\boldsymbol{x} + \boldsymbol{M}'\boldsymbol{x} \right).$$

Summary of key gradients needed

- constant: $\boxed{\nabla c = \mathbf{0}}$
- linear: $\boxed{\nabla \boldsymbol{v}' \boldsymbol{x} = \boldsymbol{v}}$
- quadratic: $\boxed{\nabla \frac{1}{2} \boldsymbol{x}' \boldsymbol{M} \boldsymbol{x} = \frac{1}{2} \left(\boldsymbol{M} + \boldsymbol{M}' \right) \boldsymbol{x}}$ and if $\boldsymbol{M}$ is symmetric then $\boxed{\nabla \frac{1}{2} \boldsymbol{x}' \boldsymbol{M} \boldsymbol{x} = \boldsymbol{M} \boldsymbol{x}.}$

The LLS problem

Now consider the LLS cost function corresponding to (4.2) for the real-valued case with $f : \mathbb{R}^N \mapsto \mathbb{R}$:

(4.3)

Applying the linearity of differentiation and the two gradients derived above, the (column) **gradient** of the cost function f w.r.t. $\boldsymbol{x}$ is:

$$\nabla f(\boldsymbol{x}) = \tag{4.4}$$

Complex case (Read)

The above derivation is applicable only when $\boldsymbol{x}$ and $\boldsymbol{y}$ and $\boldsymbol{A}$ are all real-valued. When they are complex-valued, then strictly speaking the LLS cost function f is not differentiable! Instead of working with the **gradient**, in the complex case we work with the **conjugate cogradient**, using methods from **Wirtinger calculus** [2] [3] [4]. One intermediate step above is different:

$$f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 = \frac{1}{2} \left(\boldsymbol{x}' \boldsymbol{A}' \boldsymbol{A} \boldsymbol{x} - 2 \operatorname{real}\{\boldsymbol{y}' \boldsymbol{A} \boldsymbol{x}\} + \boldsymbol{y}' \boldsymbol{y}\right),$$

but conveniently the final expression for the **conjugate cogradient** is identical to (4.4):

$$\nabla f(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}).$$

The only difference is that here $\boldsymbol{A}'$ truly denotes a **Hermitian transpose**, whereas in (4.4) it is simply an ordinary matrix transpose.

We use (4.4) hereafter, knowing that it applies to both real and complex-valued problems.
And we will call it the "**gradient**" even though technically it is the "**conjugate cogradient**" in the complex case.

Solving LLS using the normal equations

L§8.2

One can show that the LLS cost function corresponding to (4.2), *i.e.*,

$$f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 ,$$

where $f : \mathbb{F}^N \mapsto \mathbb{R}$, is a **convex function**. Because f is convex and differentiable, a vector $\hat{\boldsymbol{x}}$ is a minimizer of f if and only if $\hat{\boldsymbol{x}}$ is a point where the **gradient** of f is zero. We have shown that the gradient of f w.r.t. $\boldsymbol{x}$ is

$$\nabla f(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}).$$

Setting this gradient to zero, *i.e.*, $\nabla f(\boldsymbol{x})|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} = \boldsymbol{0}$, and rearranging yields the **normal equations**:

$$\tag{4.5}$$

The set of minimizers of f is the set of points $\hat{\boldsymbol{x}}$ satisfying (4.5).

Note that we started with a $M \times N$ matrix $\boldsymbol{A}$ in (4.1) and (4.2), but the normal equations *always* have a square $N \times N$ **Gram matrix**.

- If $\boldsymbol{A}$ is a wide matrix ($M < N$), then the rank of the $N \times N$ Gram matrix $\boldsymbol{A}'\boldsymbol{A}$ is at most M, so $\boldsymbol{A}'\boldsymbol{A}$ is singular. We will use the SVD of $\boldsymbol{A}$ to analyze this case **under-determined** case on p. 4.31.
- If $\boldsymbol{A}$ is tall or square *and* has linearly independent columns, *i.e.*, has rank N, then $\boldsymbol{A}'\boldsymbol{A}$ is invertible.

Practical LLS solution using backslash

When $\boldsymbol{A}$ has full column rank, the Gram matrix $\boldsymbol{A}'\boldsymbol{A}$ is invertible and the LLS cost function has a unique minimizer given by the solution to the normal equations (4.5). Specifically:

$$\boldsymbol{A} \in \mathbb{F}^{M \times N} \text{ and } \underbrace{\mathsf{rank}(\boldsymbol{A}) = N}_{\implies M \geq N} \implies \hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \mathbb{F}^N} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 = \qquad (4.6)$$

This expression is useful for analysis, but it is not the best computational approach!

1. It might be tempting to implement (4.6) in JULIA using one of the two following code snippets. Practical question: which of the following is likely to require less computation?

   A: `xh = inv(A'*A) * (A' * y)`

   B: `xh = inv(A'*A) * A' * y`

   C: Both always use the same computation.

Using `inv(A'A)` is computationally inefficient. Do not use it (at least not for large problem sizes)!
(An exception is applications where we must solve LLS problems for many $\boldsymbol{y}$ with the same $\boldsymbol{A}$ (as in HW).

A more efficient approach uses the **QR decomposition** of the matrix $\boldsymbol{A}$, without ever forming the Gram matrix $\boldsymbol{A}'\boldsymbol{A}$. In JULIA code it is simply: `xh = A \ y` or `xh = \(A, y)` , spoken as "$\boldsymbol{A}$ backslash $\boldsymbol{y}$."
It is also more efficient than applying **Gaussian elimination** to (4.5).
Next we turn to the SVD for more insight.

Solving LLS problems using the compact SVD

L§8.4

The **compact SVD** provides an insightful way to analyze the LLS problem (4.2). It is also a reasonable way to solve LLS problems where M and N are not too large. (Large problems require iterative methods, discussed later.) Using a compact SVD $\boldsymbol{A} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r'$ from (3.12), the LLS cost function (4.3) becomes:

$$\begin{aligned} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 &= \\ &= \\ &= \\ &= \end{aligned}$$

where we **completed the square** and used the fact that $\|\boldsymbol{U}_r \boldsymbol{z}_r\|_2 = \|\boldsymbol{z}_r\|_2$ because $\boldsymbol{U}$ is unitary.

Term 2 is independent of $\boldsymbol{x}$, so to minimize the LLS cost function we want must minimize Term 1, *i.e.*,

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{\Sigma}_r \boldsymbol{V}_r' \boldsymbol{x} - \boldsymbol{U}_r' \boldsymbol{y}\|_2^2 \,. \tag{4.7}$$

If $r = 0$ (*i.e.*, if $\boldsymbol{A} = \boldsymbol{0}$) then Term 1 vanishes, so we focus on the usual case where $1 \leq r \leq N$ hereafter.

Although (4.7) initially might look more complicated than the original LLS problem, it is actually simpler because $\boldsymbol{\Sigma}_r$ is invertible and $\boldsymbol{V}_r$ has orthonormal columns. By inspection (without taking any gradients!) one possible solution to (4.7) is:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{\Sigma}_r \boldsymbol{V}_r' \boldsymbol{x} - \boldsymbol{U}_r' \boldsymbol{y}\|_2^2 = \qquad (4.8)$$

because this solution makes Term 1 identically zero.
Verification: $\boldsymbol{\Sigma}_r \boldsymbol{V}_r' \hat{\boldsymbol{x}} = \boldsymbol{\Sigma}_r \underbrace{\boldsymbol{V}_r'(\boldsymbol{V}_r}_{\boldsymbol{I}} \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r' \boldsymbol{y}) = \underbrace{\boldsymbol{\Sigma}_r \boldsymbol{\Sigma}_r^{-1}}_{\boldsymbol{I}} \boldsymbol{U}_r' \boldsymbol{y} = \boldsymbol{U}_r' \boldsymbol{y}$.
Recall that $\boldsymbol{\Sigma}_r$ is $r \times r$ and contains the r *nonzero* singular values of $\boldsymbol{A}$ along its diagonal, so it is invertible.

If $r = N$, *i.e.*, if $\boldsymbol{A}$ has full column rank, then $\boldsymbol{V}_r = \boldsymbol{V}$ and the solution (4.8) is the *unique* minimizer.

However, if $r < N$, then there are multiple minimizers. All minimizers are given by

$$\hat{\boldsymbol{x}} = \qquad (4.9)$$

where $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_r & \boldsymbol{V}_0 \end{bmatrix}$ and $\boldsymbol{z}_0$ is *any* vector of the appropriate length!
Any such solution makes Term 1 identically zero because:
$\boldsymbol{\Sigma}_r \boldsymbol{V}_r' \hat{\boldsymbol{x}} = \boldsymbol{\Sigma}_r \boldsymbol{V}_r' \left(\boldsymbol{V}_r \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r' \boldsymbol{y} + \boldsymbol{V}_0 \boldsymbol{z}_0 \right) = \boldsymbol{U}_r' \boldsymbol{y} + \boldsymbol{\Sigma}_r \underbrace{\boldsymbol{V}_r' \boldsymbol{V}_0}_{} \boldsymbol{z}_0 = \boldsymbol{U}_r' \boldsymbol{y}$.

2. When $\boldsymbol{A}$ is $M \times N$, what is the length of the vector $\boldsymbol{z}_0$ in (4.9)?
A: r B: $N - r$ C: $M - r$ D: N E: M **??**

Letting $\boldsymbol{x} = \boldsymbol{V}\boldsymbol{z} = \begin{bmatrix} \boldsymbol{V}_r & \boldsymbol{V}_0 \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_r \\ \boldsymbol{z}_0 \end{bmatrix}$, so $\boldsymbol{z}$ denotes the coordinates of $\boldsymbol{x}$ in the $\boldsymbol{V}$ coordinate system, another way of writing the general solution in (4.9) is

$$\hat{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{V}_r & \boldsymbol{V}_0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r' \boldsymbol{y} \\ \boldsymbol{z}_0 \end{bmatrix} = \boldsymbol{V} \hat{\boldsymbol{z}}, \quad \hat{\boldsymbol{z}} = \begin{bmatrix} \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r' \boldsymbol{y} \\ \hline \boldsymbol{z}_0 \end{bmatrix} = \begin{bmatrix} [\boldsymbol{U}_r' \boldsymbol{y}]_1 / \sigma_1 \\ \vdots \\ [\boldsymbol{U}_r' \boldsymbol{y}]_r / \sigma_r \\ \hline \boldsymbol{z}_0 \end{bmatrix}. \tag{4.10}$$

This is a general expression for "the" LLS solution in terms of a compact SVD of $\boldsymbol{A}$ and the data $\boldsymbol{y}$.

The arbitrary choice of $\boldsymbol{z}_0$ may seem unsettling. We will address that concern shortly.

There are two ways that $\boldsymbol{A}$ can have rank $r < N$, leading to a non-unique solution:

- If $\boldsymbol{A}$ is square or tall but has
- If $\boldsymbol{A}$ is wide, because then

When $\boldsymbol{A}$ is tall with full rank $r = N$, the (unique) LLS solution is

(Read)

Example. Return to the example of fitting a cubic polynomial on p. 4.7. In this case $M \gg N = 4$. As long as at least 4 of the $\{t_m\}$ values are distinct, the fact that monomials are linearly independent functions implies that $\boldsymbol{A}$ has full rank, *i.e.*, $r = N = 4$. In this (typical) case, the SVD simplifies to

$$\underbrace{\boldsymbol{A}}_{M \times 4} = \underbrace{\boldsymbol{U}}_{M \times M} \underbrace{\boldsymbol{\Sigma}}_{M \times 4} \underbrace{\boldsymbol{V}'}_{4 \times 4} = \underbrace{\boldsymbol{U}_4}_{M \times 4} \underbrace{\boldsymbol{\Sigma}_4}_{4 \times 4} \underbrace{\boldsymbol{V}_4'}_{4 \times 4}, \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_4 \\ \hline \boldsymbol{0}_{(M-4) \times 4} \end{bmatrix} = \left[\begin{array}{cccc} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \\ 0 & 0 & 0 & \sigma_4 \\ \hline \multicolumn{4}{c}{\boldsymbol{0}_{(M-4) \times 4}} \end{array}\right].$$

The solution $\hat{\boldsymbol{x}}$ in (4.9) simplifies to

$$\hat{\boldsymbol{x}} = \boldsymbol{V} \boldsymbol{\Sigma}_4^{-1} \boldsymbol{U}_4' \boldsymbol{y}. \tag{4.11}$$

Because $r = N = 4$ here, this is a final and unique LLS solution; there is no arbitrary vector $\boldsymbol{z}_0$ to select.

Practical implementation (Read)

Returning to the JULIA demo for this example, where $\boldsymbol{A}$ is tall and $r = N$, we can implement this SVD-based solution using the following code.

Start with an SVD:

```
U,s,V = svd(A)
```

Recall that for this **economy SVD** version:

- `U` is $M \times N$ instead of the usual $M \times M$, *i.e.*, is $\boldsymbol{U}_r = \boldsymbol{U}_N$ in (3.12)
- `s` is a vector of the $(\sigma_1, \ldots, \sigma_N)$ values, so $\boldsymbol{\Sigma}_r = \boldsymbol{\Sigma}_N =$ `Diagonal(s)` because $r = N$ here.

Using this economy SVD, two mathematically equivalent forms of the LS solution are:

- `xh = V * Diagonal(1 ./ s) * (U' * y)` which looks like $\hat{\boldsymbol{x}} = \boldsymbol{V} \boldsymbol{\Sigma}_4^{-1} \boldsymbol{U}_4' \boldsymbol{y}$ in (4.11)
- `xh = V * ( (1 ./ s) .* (U' * y) )` which looks like (4.10).

Why the parentheses `(U' * y)` ? Saves computation because $\boldsymbol{U}_N'$ is wide.

When $\boldsymbol{A}$ is tall with full rank ($r = N$), the compact SVD (used in our mathematical analysis) and the economy SVD (returned by `svd`) are identical, *i.e.*, $\boldsymbol{U}_r = \boldsymbol{U}_N$, $\boldsymbol{\Sigma}_r = \boldsymbol{\Sigma}_N$, and $\boldsymbol{V}_r = \boldsymbol{V}_N = \boldsymbol{V}$.

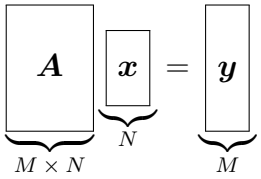
Uniqueness of LLS solution (reprise) (Read)

If $\boldsymbol{A}$ is $M \times N$, then by the definition of rank, $\mathsf{rank}(\boldsymbol{A}) = N$ iff $\boldsymbol{A}$ has linearly independent columns. Having $M \geq N$ is a necessary condition for linear independence of the columns of $\boldsymbol{A}$.

Conversely, if $M < N$ then $\mathsf{rank}(\boldsymbol{A}) \neq N$ because $\mathsf{rank}(\boldsymbol{A}) \leq \min(M, N) = M < N$.

Over-determined (tall) case

In the frequent case where $M > N$ we have (graphically):



In words, there are more equations than unknowns, called an **over-determined system**. Rarely is there an exact solution in this case due to noise in the data $\boldsymbol{y}$, so "best fit" solutions like LLS (4.2) are used instead.

When $M \geq N$ and $\boldsymbol{A}$ has rank $r = N$, the solution (4.9), (4.10) has no arbitrary terms and simplifies to:

$$\underbrace{\hat{z}_n = \frac{[\boldsymbol{U}'\boldsymbol{y}]_n}{\sigma_n}}_{n = 1, \dots, N} \implies \hat{\boldsymbol{z}} = \boldsymbol{\Sigma}_N^{-1} \boldsymbol{U}_N' \boldsymbol{y}, \qquad \hat{\boldsymbol{x}} = \boldsymbol{V}\hat{\boldsymbol{z}} \implies \boxed{\hat{\boldsymbol{x}} = \boldsymbol{V}\boldsymbol{\Sigma}_N^{-1}\boldsymbol{U}_N'\boldsymbol{y}}\,. \tag{4.12}$$

This is the *unique* solution to (4.2) when $\mathsf{rank}(\boldsymbol{A}) = N$.

Over-determined full-rank case using SVD

The expression (4.12) uses the **compact SVD** and it is useful to also write it in terms of the **full SVD**:

$$\hat{\boldsymbol{x}} = \boldsymbol{V}\boldsymbol{\Sigma}_N^{-1}\boldsymbol{U}_N'\boldsymbol{y} = \boldsymbol{V}\left[\begin{array}{c|c} \boldsymbol{\Sigma}_N^{-1} & \boldsymbol{0}_{N\times(M-N)} \end{array}\right]\begin{bmatrix}\boldsymbol{U}_N' \\ \boldsymbol{U}_0'\end{bmatrix}\boldsymbol{y} = \boldsymbol{V}\left[\begin{array}{c|c} \boldsymbol{\Sigma}_N^{-1} & \boldsymbol{0}_{N\times(M-N)} \end{array}\right]\boldsymbol{U}'\boldsymbol{y}.$$

This form is suboptimal for *computation* because the term $\boldsymbol{U}_0'\boldsymbol{y}$ is computed but then multiplied by $\boldsymbol{0}$.

An even more concise expression is

$$M \geq N \text{ and } \mathsf{rank}(\boldsymbol{A}) = N \implies \qquad (4.13)$$

where $\boldsymbol{\Sigma}^+ \triangleq \left[\begin{array}{c|c} \boldsymbol{\Sigma}_N^{-1} & \boldsymbol{0}_{N\times(M-N)} \end{array}\right]$ denotes the **Moore-Penrose pseudoinverse** of $\boldsymbol{\Sigma} = \left[\begin{array}{c} \boldsymbol{\Sigma}_N \\ \hline \boldsymbol{0}_{(M-N)\times N} \end{array}\right]$.

The next page discusses this new term in detail and explains $\boldsymbol{A}^+$.

4.2 Moore-Penrose pseudoinverse

L§4.1

Define. In general, for any matrix $\boldsymbol{A} \in \mathbb{F}^{M \times N}$, the **Moore-Penrose pseudoinverse** of $\boldsymbol{A}$, denoted $\boldsymbol{A}^+ \in \mathbb{F}^{N \times M}$, is a generalization of the usual notation of matrix inverse that satisfies the following four properties:

- (weaker than $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}$)
-
- (symmetry)
-

Properties. (For proofs see [wiki].) (Another notation is $\boldsymbol{A}^\dagger$.) L§4.3

- $\boldsymbol{A}^+$ is unique and $(\boldsymbol{A}^+)^+ = \boldsymbol{A}$
- If $\boldsymbol{A}$ is invertible, then $\boldsymbol{A}^+ = \boldsymbol{A}^{-1}$
- $\boldsymbol{A}^+ = (\boldsymbol{A}'\boldsymbol{A})^+\boldsymbol{A}'$
- If $\boldsymbol{A}$ has full column rank (linearly independent columns), then $\boldsymbol{A}^+ =$
  In this case, $\boldsymbol{A}^+$ is a **left inverse** because $\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{I}_N$.
- $\boldsymbol{A}^+ = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{A}')^+$
- If $\boldsymbol{A}$ has full row rank, (linearly independent rows), then $\boldsymbol{A}^+ = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{A}')^{-1}$.
  In this case, $\boldsymbol{A}^+$ is a **right inverse** because $\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{I}_M$.
- $\boldsymbol{0}_{M \times N}^+ = \boldsymbol{0}_{N \times M}$ so in particular $0^+ = 0$
- $(\boldsymbol{A}')^+ = (\boldsymbol{A}^+)'$

Pseudo-inverse and matrix products

Caution: in general $(\boldsymbol{AB})^+ \neq \boldsymbol{B}^+\boldsymbol{A}^+$, so be careful with matrix products. However, for $\boldsymbol{B} \in \mathbb{F}^{K \times L}$:

- (P1) If $\boldsymbol{Q}$ is a $M \times K$ matrix with **orthonormal columns**, *i.e.*, $\boldsymbol{Q}'\boldsymbol{Q} = \boldsymbol{I}_{K \times K}$, then $(\boldsymbol{QB})^+ = \boldsymbol{B}^+\boldsymbol{Q}'$.
  Partial proof:
  - $(\boldsymbol{QB})(\boldsymbol{QB})^+(\boldsymbol{QB}) = \boldsymbol{QB}(\boldsymbol{B}^+\boldsymbol{Q}')\boldsymbol{QB} = \boldsymbol{QBB}^+\boldsymbol{B} = \boldsymbol{QB}$
  - $(\boldsymbol{QB})^+\boldsymbol{QB}(\boldsymbol{QB})^+ = (\boldsymbol{B}^+\boldsymbol{Q}')\boldsymbol{QB}(\boldsymbol{B}^+\boldsymbol{Q}') = \boldsymbol{B}^+\boldsymbol{BB}^+\boldsymbol{Q}' = \boldsymbol{B}^+\boldsymbol{Q}' = (\boldsymbol{QB})^+$, etc.
- (P2) If $\boldsymbol{Q}$ is a $L \times N$ matrix with **orthonormal rows**, *i.e.*, $\boldsymbol{QQ}' = \boldsymbol{I}_{L \times L}$, then $(\boldsymbol{BQ})^+ = \boldsymbol{Q}'\boldsymbol{B}^+$.
  - $(\boldsymbol{BQ})(\boldsymbol{BQ})^+(\boldsymbol{BQ}) = \boldsymbol{BQ}(\boldsymbol{Q}'\boldsymbol{B}^+)\boldsymbol{BQ} = \boldsymbol{BB}^+\boldsymbol{BQ} = \boldsymbol{BQ}$
  - $(\boldsymbol{BQ})^+(\boldsymbol{BQ})(\boldsymbol{BQ})^+ = (\boldsymbol{Q}'\boldsymbol{B}^+)\boldsymbol{BQ}(\boldsymbol{Q}'\boldsymbol{B}^+) = \boldsymbol{Q}'\boldsymbol{B}^+\boldsymbol{BB}^+ = \boldsymbol{Q}'\boldsymbol{B}^+ = (\boldsymbol{BQ})^+$, etc.
- (P3) If $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ has full column rank (linearly independent columns)
  and $\boldsymbol{B} \in \mathbb{F}^{N \times K}$ has full row rank (linearly independent rows), then $(\boldsymbol{AB})^+ = \boldsymbol{B}^+\boldsymbol{A}^+$.

Fortunately, these product properties serve our needs.

A special case of the first two preceding results is when $\boldsymbol{B} = \boldsymbol{I}$, leading to:

- (P4) If $\boldsymbol{Q}$ is a matrix with **orthonormal columns**, then $\boldsymbol{Q}^+ = \boldsymbol{Q}'$.
- (P5) If $\boldsymbol{Q}$ is a matrix with **orthonormal rows**, then $\boldsymbol{Q}^+ = \boldsymbol{Q}'$.

In the context of the **compact SVD**, (P4) implies that $\boldsymbol{U}_r^+ = \boldsymbol{U}_r'$ and $\boldsymbol{V}_r^+ = \boldsymbol{V}_r'$.
Because $\boldsymbol{U}_r$ has orthonormal columns and $\boldsymbol{V}_r'$ has orthonormal rows and $\boldsymbol{\Sigma}_r$ is invertible, it follows that:

$$\boldsymbol{A} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r' \implies \boldsymbol{A}^+ = (\boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r')^+ = (\boldsymbol{U}_r(\boldsymbol{\Sigma}_r\boldsymbol{V}_r'))^+ \overset{\text{P1}}{=} (\boldsymbol{\Sigma}_r\boldsymbol{V}_r')^+\boldsymbol{U}_r' \overset{\text{P2}}{=} \boldsymbol{V}_r\boldsymbol{\Sigma}_r^+\boldsymbol{U}_r' = \boldsymbol{V}_r\boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}_r'.$$

Example. A pseudo-inverse example of particular interest when working with the **SVD** is that of a **rectangular diagonal matrix**:

$$\boldsymbol{\Sigma} = \underbrace{\left[\begin{array}{c|c} \boldsymbol{\Sigma}_r & \boldsymbol{0}_{r \times (N-r)} \\ \hline \boldsymbol{0}_{(M-r) \times r} & \boldsymbol{0}_{(M-r) \times (N-r)} \end{array}\right]}_{M \times N} \quad \Longrightarrow \quad \boldsymbol{\Sigma}^+ = \underbrace{\left[\begin{array}{c|c} & \boldsymbol{0}_{r \times (M-r)} \\ \hline \boldsymbol{0}_{(N-r) \times r} & \end{array}\right]}_{\quad}.$$

Exercise. Verify that this $\boldsymbol{\Sigma}^+$ satisfies the four conditions for a pseudo-inverse on p. 4.24.

An important special case is when $r = N$, *i.e.*, $\boldsymbol{A}$ has full column rank, in which case:

$$\boldsymbol{\Sigma} = \underbrace{\left[\begin{array}{c} \boldsymbol{\Sigma}_N \\ \hline \boldsymbol{0}_{(M-N) \times N} \end{array}\right]}_{M \times N} \quad \Longrightarrow \quad \boldsymbol{\Sigma}^+ = \quad$$

Noting that $\boldsymbol{\Sigma}^+ \boldsymbol{\Sigma} = \boldsymbol{I}_N$ in the tall full-rank case, it is trivial to verify the four defining properties for this special case.

Caution: in general $\boldsymbol{A}^+ \boldsymbol{A} \neq \boldsymbol{I}_N$ and $\boldsymbol{A} \boldsymbol{A}^+ \neq \boldsymbol{I}_M$.

Pseudo-inverse and SVD

Using the orthogonal matrix product properties of the pseudo-inverse yields the following **SVD** property:

$$\underbrace{\boldsymbol{A}}_{M \times N} = \underbrace{\boldsymbol{U}}_{M \times M} \underbrace{\boldsymbol{\Sigma}}_{M \times N} \underbrace{\boldsymbol{V}'}_{N \times N} \implies \underbrace{\boldsymbol{A}^+}_{N \times M} = \qquad (4.14)$$

The **compact SVD** version of pseudo-inverse is also useful:

$$\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}' = \left[\begin{array}{c|c} \boldsymbol{V}_r & \boldsymbol{V}_0 \end{array} \right] \boldsymbol{\Sigma}^+ \left[\begin{array}{c} \boldsymbol{U}_r' \\ \hline \boldsymbol{U}_0' \end{array} \right] \implies \qquad (4.15)$$

Here is one sanity check for the pseudo-inverse expression (4.14):

$$\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\underbrace{\boldsymbol{V}'\boldsymbol{V}}\boldsymbol{\Sigma}^+\underbrace{\boldsymbol{U}'\boldsymbol{U}}\boldsymbol{\Sigma}\boldsymbol{V}' = \boldsymbol{U}\underbrace{\boldsymbol{\Sigma}\boldsymbol{\Sigma}^+\boldsymbol{\Sigma}}\boldsymbol{V}' = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}' = \boldsymbol{A}.$$

The key expression (4.14) simplifies to the usual matrix inverse in the rare case where $\boldsymbol{A}$ is square and invertible: $\boldsymbol{A}^{-1} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}'$. (However, for invertible matrices often we do not need to use the SVD.)

3. LLS solution $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{y}$ lies in which of the four fundamental spaces of $\boldsymbol{A}$?

A: $\mathcal{N}(\boldsymbol{A})$ B: $\mathcal{N}^\perp(\boldsymbol{A})$ C: $\mathcal{R}(\boldsymbol{A})$ D: $\mathcal{R}^\perp(\boldsymbol{A})$ E: None of these. **??**

Warm-up questions

4. If $\boldsymbol{A}$ has linearly independent columns, then $\|\boldsymbol{A}^+ \boldsymbol{A}\boldsymbol{x}\| = \|\boldsymbol{x}\|$.
A: True B: False **??**

5. Let $\boldsymbol{A}$ have **full SVD** $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$ and **compact SVD** $\boldsymbol{A} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r'$, where we partition unitary matrix $\boldsymbol{V}$ as usual as $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_r & \boldsymbol{V}_0 \end{bmatrix}$. Which of the following is the tuple $(\mathcal{N}(\boldsymbol{V}'), \mathcal{N}(\boldsymbol{V}_r'))$?
A: $(\boldsymbol{0}, \boldsymbol{0})$ B: $(\boldsymbol{0}, \mathcal{R}(\boldsymbol{V}_0))$ C: $(\mathcal{R}(\boldsymbol{V}_0), \boldsymbol{0})$ D: $(\mathcal{R}(\boldsymbol{V}_0), \mathcal{R}(\boldsymbol{V}_0))$ E: None of these. **??**

6. If matrix $\boldsymbol{A}$ has **SVD** $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}' = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r'$, then two expressions for its **pseudo-inverse** are:

$$\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}' = \boldsymbol{V}_r\boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}_r'.$$

Which of the following is the **null space** of $\boldsymbol{A}^+$, *i.e.*, $\mathcal{N}(\boldsymbol{A}^+)$?
A: $\mathcal{N}(\boldsymbol{A})$ B: $\mathcal{N}^\perp(\boldsymbol{A})$ C: $\mathcal{R}(\boldsymbol{A})$ D: $\mathcal{R}^\perp(\boldsymbol{A})$ E: None of these. **??**

7. If $\boldsymbol{A} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r'$ then $\boldsymbol{A}^+ = \boldsymbol{V}_r\boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}_r'$ is an SVD of $\boldsymbol{A}^+$.
A: True B: False **??**

Projector / idempotent preview (Read)

The following matrix is called an **orthogonal projector** (see p. 4.56):

$$\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A}')} = \boldsymbol{P}_{\mathcal{N}^\perp(\boldsymbol{A})} \triangleq \boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}'\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}' = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{\Sigma}\boldsymbol{V}' = \boldsymbol{V}_r\boldsymbol{V}_r'.$$

Because $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A}')}\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A}')} = \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A}')}$, it is called **idempotent**.

Likewise the following matrix is also a projector (and also idempotent):

$$\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})} = \boldsymbol{P}_{\mathcal{N}^\perp(\boldsymbol{A}')} \triangleq \boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'\boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}' = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^+\boldsymbol{U}' = \boldsymbol{U}_r\boldsymbol{U}_r'.$$

Relating pseudo-inverse and normal equations (Read)

The following equalities (properties of the pseudo-inverse) follow:

$$\boldsymbol{A}'\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}' \text{ because } \boldsymbol{A}'\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{V}_r\boldsymbol{\Sigma}_r\boldsymbol{U}_r'\left(\boldsymbol{U}_r\boldsymbol{U}_r'\right) = \boldsymbol{V}_r\boldsymbol{\Sigma}_r\boldsymbol{U}_r' = \boldsymbol{A}'$$
$$\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}' = \boldsymbol{A}' \text{ because } \boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}' = \left(\boldsymbol{V}_r\boldsymbol{V}_r'\right)\boldsymbol{V}_r\boldsymbol{\Sigma}_r\boldsymbol{U}_r' = \boldsymbol{V}_r\boldsymbol{\Sigma}_r\boldsymbol{U}_r' = \boldsymbol{A}'.$$

Multiplying the first of these two equalities by $\boldsymbol{y}$ yields:

$$\boldsymbol{A}'\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{y} = \boldsymbol{A}'\boldsymbol{y} \implies \boldsymbol{A}'\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{A}'\boldsymbol{y},$$

so the pseudo-inverse solution $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{y}$ satisfies the **normal equations** always, regardless of the rank of $\boldsymbol{A}$.

LLS solution using pseudo-inverse

Using (4.13), we write the LLS estimate (4.13) particularly concisely in the full-rank case as follows:

$$\boldsymbol{A} \in \mathbb{F}^{M \times N} \text{ and } \underbrace{\text{rank}(\boldsymbol{A}) = N}_{\implies M \geq N} \implies \hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \mathbb{F}^N} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 = \qquad (4.16)$$

This is a wonderfully elegant form on paper, but it is not the best computational approach!
We could implement (4.16) in JULIA using the pseudo-inverse function: `xh = pinv(A) * y;`

Alternatively we could use the `inv` form discussed on p. 4.16: `xh = inv(A'*A) * (A'*y);`
However, both these approaches are computationally inefficient!
We rarely use `pinv` or `inv` in practice (at least for large problem sizes) unless we must solve LLS problems for many different $\boldsymbol{y}$ with the same $\boldsymbol{A}$ (as in HW).

Otherwise, the backslash approach `xh = A \ y` discussed on p. 4.16 with its more efficient **QR decomposition** is preferable.

Nevertheless, the SVD is very helpful conceptually, especially in the under-determined case.

If $\boldsymbol{A}$ does not have full column rank, then `A \ y` will produce an error message like `SingularException`.
Then one should first think about why $\boldsymbol{A}$ does not have linearly dependent columns, and then perhaps consider using `pinv`, as discussed in the next section, or one of the regularized solutions discussed after that.

4.3 Linear least-squares estimation: Under-determined case

We focused previously on cases where $M \geq N$ and $\boldsymbol{A}$ has full rank, and derived the concise LLS solution based on the pseudo-inverse (4.16). Now we examine cases where $M < N$, called **under-determined**, as well as cases where $\boldsymbol{A}$ is tall but rank deficient *i.e.*, $\mathsf{rank}(\boldsymbol{A}) < N$. In these cases there is not a unique minimizer to the LLS cost function $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$. Using the **compact SVD** of $\boldsymbol{A}$, we showed in (4.9) that any minimizer has the form

$$\hat{\boldsymbol{x}} = \underbrace{\boldsymbol{V}_r \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r' \boldsymbol{y}}_{\hat{\boldsymbol{x}}_{\mathcal{R}} \in \mathcal{R}(\boldsymbol{V}_r) = \mathcal{N}^\perp(\boldsymbol{A})} + \underbrace{\boldsymbol{V}_0 \hat{\boldsymbol{z}}_0}_{\hat{\boldsymbol{x}}_{\mathcal{N}} \in \mathcal{N}(\boldsymbol{A})} = \qquad (4.17)$$

The choice of $\hat{\boldsymbol{z}}_0$ is arbitrary because the residual is invariant to the nullspace component:

$$\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y} = \boldsymbol{A}(\boldsymbol{V}_r \hat{\boldsymbol{z}}_r + \boldsymbol{V}_0 \hat{\boldsymbol{z}}_0) - \boldsymbol{y} = \boldsymbol{A}(\hat{\boldsymbol{x}}_{\mathcal{R}} + \hat{\boldsymbol{x}}_{\mathcal{N}}) - \boldsymbol{y} = \boldsymbol{A}\hat{\boldsymbol{x}}_{\mathcal{R}} + \boldsymbol{0} - \boldsymbol{y} = \boldsymbol{A}\hat{\boldsymbol{x}}_{\mathcal{R}} - \boldsymbol{y}.$$

In other words, for any $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}_{\mathcal{R}} + \hat{\boldsymbol{x}}_{\mathcal{N}}$ where $\hat{\boldsymbol{x}}_{\mathcal{N}} \in \mathcal{N}(\boldsymbol{A})$, the error $\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}\|_2^2$ is identical.

When $r < N$, the LLS solution is *not* unique because we can choose $\hat{\boldsymbol{z}}_0$ arbitrarily, or equivalently we can choose any $\hat{\boldsymbol{x}}_{\mathcal{N}} \in \mathcal{N}(\boldsymbol{A})$. The LLS criterion by itself is insufficient to identify a unique best $\hat{\boldsymbol{x}}$ in under-determined (or rank-deficient) problems.

Conversely, in the full rank case where $\mathsf{rank}(\boldsymbol{A}) = r = N$, the matrix $\boldsymbol{V}_0$ does not exist and $\mathcal{N}(\boldsymbol{A}) = \{\boldsymbol{0}\}$ so the LLS solution is unique.

If $\boldsymbol{A}$ is wide, *i.e.*, $M < N$, can we have $\mathsf{rank}(\boldsymbol{A}) = N$?
No, because $\mathsf{rank}(\boldsymbol{A}) \leq \min(M, N) = M < N$ per (3.9). So wide cases are always under-determined.

To summarize:

- When $M \geq N$ and $\boldsymbol{A}$ has full rank N, the LLS solution is unique and is $\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y}$.
- Otherwise (*i.e.*, if $M < N$ or if $\boldsymbol{A}$ has rank less than N), the LLS solution is not unique and any $\hat{\boldsymbol{x}}$ of the form $\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y} + \hat{\boldsymbol{x}}_{\mathcal{N}}$ where $\hat{\boldsymbol{x}}_{\mathcal{N}} \in \mathcal{N}(\boldsymbol{A})$ is "equally good" from the point of view of the LLS cost function $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$.

Dealing with non-uniqueness

To pin down a unique solution in the under-determined case, we must introduce some additional criterion to select on $\hat{\boldsymbol{x}}$ from the (infinitely) many candidates of the form $\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y} + \hat{\boldsymbol{x}}_{\mathcal{N}}$.

A modern way to do this is to use a **sparsity** model and seek a sparse solution, *e.g.*:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \text{nonsparsity}(\boldsymbol{x}) \text{ s.t. } \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \leq \epsilon.$$

But this is an advanced topic for later in the course!

Instead we focus for now on the "classical" approach of choosing the **minimum norm** solution, *i.e.*, the LLS minimizer where $\|\hat{\boldsymbol{x}}\|_2^2$ is the smallest. This is a "double minimization" problem, because (conceptually) we first find all the minimizers of $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$, and then among those minimizers we pick the one where $\|\boldsymbol{x}\|_2^2$ is the smallest.

But first we give a geometric interpretation of the LLS solution.

Orthogonality principle

To examine LLS geometrically, recall from the **normal equations** (4.5) that any LLS solution must satisfy

$$\boldsymbol{A}'\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{A}'\boldsymbol{y} \implies \boldsymbol{A}'(\boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}}) = \boldsymbol{0} \implies \boldsymbol{A}'\boldsymbol{r} = \boldsymbol{0} \implies \boldsymbol{z}'\boldsymbol{A}'\boldsymbol{r} = 0 \implies$$

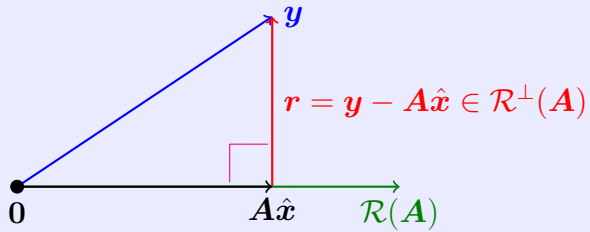
where the **residual** after fitting is denoted $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}}$.

(The normal equations are a **necessary condition** for $\hat{\boldsymbol{x}}$ to be a minimizer, regardless of the rank of $\boldsymbol{A}$.)

The following **orthogonality principle** of LLS estimation is a direct consequence of the above:

$$\langle \boldsymbol{A}\boldsymbol{z},\, \boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}} \rangle = (\boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}})'\boldsymbol{A}\boldsymbol{z} = 0, \text{ i.e.,} \tag{4.18}$$

In words, the **residual** is **perpendicular** to $\mathcal{R}(\boldsymbol{A})$. There is an important geometric interpretation:



Terminology that (unfortunately) is often used inter-changeably:

- $\boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}}$ is called the **residual** (from fitting)
- $\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_{\text{true}}$ is called the **error** in the data
- $\hat{\boldsymbol{x}} - \boldsymbol{x}_{\text{true}}$ is called the **error** in the parameter estimate.

Often one must determine from context which meaning of "error" is meant.

If $\boldsymbol{y} \in \mathcal{R}(\boldsymbol{A})$ (which rarely happens for noisy data when $M > N$), then there is a solution with zero residual $\boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}}$. However, the parameter error $\hat{\boldsymbol{x}} - \boldsymbol{x}_{\text{true}}$ may still be nonzero!

Alternate proof of optimality (Read)

Here is an alternate derivation that uses the orthogonality principle (4.18) to confirm that $\hat{\boldsymbol{x}}$ in (4.17) is optimal for LLS estimation (for $\boldsymbol{A}$ of any size or rank).

Suppose $\boldsymbol{x}_? = \boldsymbol{A}^+\boldsymbol{y} + \hat{\boldsymbol{x}}_{\mathcal{N}} + \boldsymbol{z} = \hat{\boldsymbol{x}} + \boldsymbol{z}$ for some arbitrary vector $\boldsymbol{z} \in \mathbb{F}^N$.
Could this $\boldsymbol{x}_?$ be a better estimator?

Using the norm of a sum in (1.17) and the orthogonality principle (4.18), the squared error criterion for such an $\boldsymbol{x}_?$ has the following lower bound:

$$\begin{aligned}\|\boldsymbol{A}\boldsymbol{x}_? - \boldsymbol{y}\|_2^2 &= \|\boldsymbol{A}(\hat{\boldsymbol{x}} + \boldsymbol{z}) - \boldsymbol{y}\|_2^2 = \|(\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}) + \boldsymbol{A}\boldsymbol{z}\|_2^2 \\ &= \|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}\|_2^2 + \underbrace{2\,\mathrm{real}\{(\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y})'\boldsymbol{A}\boldsymbol{z}\}}_{0} + \underbrace{\|\boldsymbol{A}\boldsymbol{z}\|_2^2}_{\geq 0} \geq \|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}\|_2^2 ,\end{aligned}$$

where the lower bound is achieved by $\boldsymbol{z} = \boldsymbol{0}$.
In words, the LLS fit is best when $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{y} + \hat{\boldsymbol{x}}_{\mathcal{N}}$ for any vector $\hat{\boldsymbol{x}}_{\mathcal{N}} \in \mathcal{N}(\boldsymbol{A})$.

Mathematically, the set of LLS solutions is the sum of a vector and a subspace (the nullspace of $\boldsymbol{A}$):

$$\left\{\tilde{\boldsymbol{x}} \in \mathbb{F}^N \;:\; \|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{y}\|_2^2 \leq \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 ,\; \forall \boldsymbol{x} \in \mathbb{F}^N\right\} = \quad\quad$$
$$\triangleq \left\{\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{y} + \hat{\boldsymbol{x}}_{\mathcal{N}} \;:\; \hat{\boldsymbol{x}}_{\mathcal{N}} \in \mathcal{N}(\boldsymbol{A})\right\} .$$

The sum of a vector plus a subspace is called a **linear variety** or a **flat**.

Minimum-norm LS solution via pseudo-inverse

L§8.1

We have seen that the set of optimal LLS estimates is $\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y} + \mathcal{N}(\boldsymbol{A})$.

- If $\boldsymbol{A}$ has full column rank, then $\mathcal{N}(\boldsymbol{A}) = \boldsymbol{0}$ and we have a unique solution $\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y}$
- If $\boldsymbol{A}$ does not have full column rank, then we want to pick one choice from the set of LLS estimates.

The classical way to pick one of the many possible estimates in the under-determined case is to choose the one with minimum norm.

Fact. The **minimum norm LLS solution** is:

$$\hat{\boldsymbol{x}} \triangleq \underset{\boldsymbol{x} \in \{\boldsymbol{A}^+ \boldsymbol{y} + \mathcal{N}(\boldsymbol{A})\}}{\arg\min} \|\boldsymbol{x}\|_2^2 = \qquad (4.19)$$

This is a kind of "double minimization" because first we found a set of candidate solutions by finding minimizers of $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$, and then we solve a different minimization problem involving $\|\boldsymbol{x}\|_2^2$ to select one final solution from that set of candidates.

Proof. If $\boldsymbol{x} = \boldsymbol{A}^+ \boldsymbol{y} + \hat{\boldsymbol{x}}_{\mathcal{N}}$ where $\hat{\boldsymbol{x}}_{\mathcal{N}} \in \mathcal{N}(\boldsymbol{A})$, then $\hat{\boldsymbol{x}}_{\mathcal{N}} \in \mathsf{span}(\boldsymbol{V}_0)$, where $\boldsymbol{V} = \left[\begin{array}{c|c} \boldsymbol{V}_r & \boldsymbol{V}_0 \end{array} \right]$. (Read)
Using (4.15): $\boldsymbol{A}^+ \boldsymbol{y} = \boldsymbol{V}_r \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r' \boldsymbol{y} \in \mathcal{R}(\boldsymbol{V}_r)$.
Because $\boldsymbol{V}$ is unitary, the columns of $\boldsymbol{V}_r$ and $\boldsymbol{V}_0$ are orthogonal. Thus $\boldsymbol{A}^+ \boldsymbol{y} \perp \hat{\boldsymbol{x}}_{\mathcal{N}}$. Using (1.17):

$$\|\boldsymbol{x}\|_2^2 = \left\|\boldsymbol{A}^+ \boldsymbol{y} + \hat{\boldsymbol{x}}_{\mathcal{N}}\right\|_2^2 = \left\|\boldsymbol{A}^+ \boldsymbol{y}\right\|_2^2 + 2\,\mathrm{real}\left\{\hat{\boldsymbol{x}}_{\mathcal{N}}' \boldsymbol{A}^+ \boldsymbol{y}\right\} + \|\hat{\boldsymbol{x}}_{\mathcal{N}}\|_2^2 = \left\|\boldsymbol{A}^+ \boldsymbol{y}\right\|_2^2 + \|\hat{\boldsymbol{x}}_{\mathcal{N}}\|_2^2 \geq \left\|\boldsymbol{A}^+ \boldsymbol{y}\right\|_2^2,$$

where the minimum is achieved when $\hat{\boldsymbol{x}}_{\mathcal{N}} = \boldsymbol{0}$. Thus the minimum norm solution is $\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y}$. □

The set over which we minimize in (4.19) is $\{\boldsymbol{A}^+\boldsymbol{y} + \mathcal{N}(\boldsymbol{A})\}$.

8. What is the cardinality of this set when $\mathsf{rank}(\boldsymbol{A}) = N$?
A: 0 B: 1 C: r D: N E: ∞ **??**

9. What is the cardinality of this set when $M < N$?
A: 0 B: 1 C: r D: N E: ∞ **??**

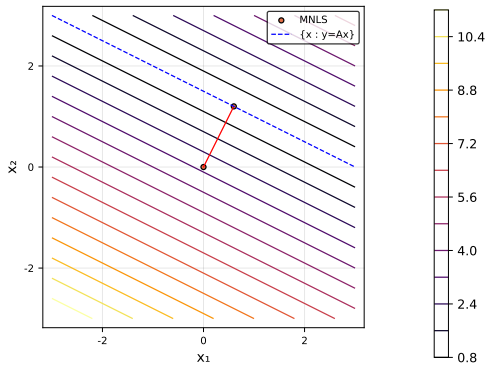
In summary we have the following fortuitous situation.

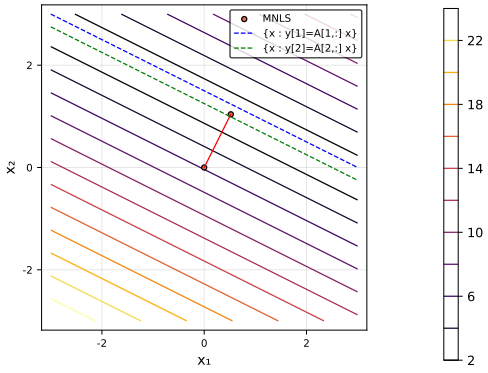
- If $\boldsymbol{A}$ has full column rank, then $\mathcal{N}(\boldsymbol{A}) = \boldsymbol{0}$ and we have a unique solution $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{y}$.
- If $\boldsymbol{A}$ does not have full column rank, then there are multiple LLS estimates and the one with smallest Euclidean norm is $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{y}$.

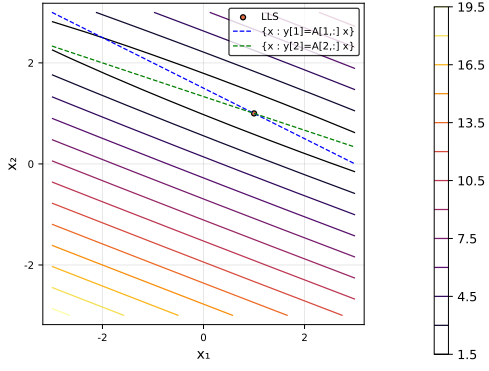
Hence the **pseudo-inverse** solution has been very popular historically, and remains important today, except in many highly under-determined problems of the kind known as **compressed sensing**.

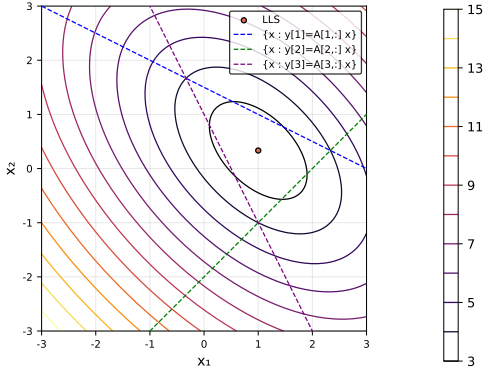
There are quantum-computing methods for the pseudo-inverse solution.

Example. To visualize the cost function (4.3) and solution $\hat{\boldsymbol{x}}$ see:
https://web.eecs.umich.edu/~fessler/course/551/julia/demo/04_ls_cost1.html
https://web.eecs.umich.edu/~fessler/course/551/julia/demo/04_ls_cost1.ipynb
and plots on next page.


MNLS
{x : y=Ax}
x₂
x₁
10.4
8.8
7.2
5.6
4.0
2.4
0.8


MNLS
{x : y[1]=A[1,:] x}
{x : y[2]=A[2,:] x}
x₂
x₁
22
18
14
10
6
2


LLS
{x : y[1]=A[1,:] x}
{x : y[2]=A[2,:] x}
x₂
x₁
19.5
16.5
13.5
10.5
7.5
4.5
1.5


LLS
{x : y[1]=A[1,:] x}
{x : y[2]=A[2,:] x}
{x : y[3]=A[3,:] x}
x₂
x₁
15
13
11
9
7
5
3

Interpreting the pseudo-inverse solution (Read)

Using (4.14), we can interpret the pseudo-inverse solution

$$\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y} = \boldsymbol{V} \boldsymbol{\Sigma}^+ \boldsymbol{U}' \boldsymbol{y}$$

as a cascade of three separate transformations:

$$\boldsymbol{y} \overset{\boldsymbol{U}'}{\longmapsto} \tilde{\boldsymbol{y}} \overset{\boldsymbol{\Sigma}^+}{\longmapsto} \boldsymbol{z} \overset{\boldsymbol{V}}{\longmapsto} \hat{\boldsymbol{x}}.$$

The geometric interpretation is similar to what we did for the SVD except "in reverse" and with $\boldsymbol{\Sigma}^+$.

To interpret the residual, note that

$$\begin{aligned}
\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{y} - \boldsymbol{A}\boldsymbol{A}^+\boldsymbol{y} = (\boldsymbol{I} - \boldsymbol{A}\boldsymbol{A}^+)\boldsymbol{y} = (\boldsymbol{I} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^+\boldsymbol{U}')\boldsymbol{y} = \boldsymbol{U}(\boldsymbol{I} - \boldsymbol{\Sigma}\boldsymbol{\Sigma}^+)\boldsymbol{U}'\boldsymbol{y} \\
= \left[\begin{array}{c|c} \boldsymbol{U}_r & \boldsymbol{U}_0 \end{array}\right] \left(\boldsymbol{I} - \left[\begin{array}{c|c} \boldsymbol{I} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{0} \end{array}\right] \right) \left[\begin{array}{c|c} \boldsymbol{U}_r & \boldsymbol{U}_0 \end{array}\right]' \boldsymbol{y} = \left[\begin{array}{c|c} \boldsymbol{U}_r & \boldsymbol{U}_0 \end{array}\right] \left[\begin{array}{c|c} \boldsymbol{0} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{I} \end{array}\right] \left[\begin{array}{c|c} \boldsymbol{U}_r & \boldsymbol{U}_0 \end{array}\right]' \boldsymbol{y} = \boldsymbol{U}_0 \boldsymbol{U}_0' \boldsymbol{y}
\end{aligned}$$

$$\implies \|\boldsymbol{r}\|_2^2 = \|\boldsymbol{U}_0 \boldsymbol{U}_0' \boldsymbol{y}\|_2^2 = \boldsymbol{y}' \boldsymbol{U}_0 \boldsymbol{U}_0' \left(\boldsymbol{U}_0 \boldsymbol{U}_0' \boldsymbol{y}\right) = \boldsymbol{y}' \boldsymbol{U}_0 \boldsymbol{U}_0' \boldsymbol{y} = \|\boldsymbol{U}_0' \boldsymbol{y}\|_2^2 ,$$

where from p. 3.37, $\mathcal{R}^\perp(\boldsymbol{A}) = \mathsf{span}(\boldsymbol{U}_0)$. Thus the residual norm squared comes from the portion of $\boldsymbol{y}$ in $\mathcal{R}^\perp(\boldsymbol{A})$, consistent with our earlier picture about the orthogonality principle.

4.4 Truncated SVD solution

We have seen that the minimum-norm LLS minimizer of $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ is the pseudo-inverse solution:

$$\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y} = \boldsymbol{V}\boldsymbol{\Sigma}^+ \boldsymbol{U}'\boldsymbol{y} = \boldsymbol{V}_r \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r' \boldsymbol{y} =$$

This solution is mathematically elegant, but in practice sometimes it works very poorly.
This section provides one remedy based on the **truncated SVD**.

Example. Consider an application where $r = \mathsf{rank}(\boldsymbol{A}) = 2$ with singular values $\sigma_1 = 1,\ \sigma_2 = 10^{-8}$.
Then the pseudo-inverse solution here is:

$$\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y} = \sum_{k=1}^{2} \frac{1}{\sigma_k} \boldsymbol{v}_k \left(\boldsymbol{u}_k' \boldsymbol{y}\right) = \frac{1}{\sigma_1} \boldsymbol{v}_1 (\boldsymbol{u}_1' \boldsymbol{y}) + \frac{1}{\sigma_2} \boldsymbol{v}_2 \left(\boldsymbol{u}_2' \boldsymbol{y}\right) = (1)\, \boldsymbol{v}_1 (\boldsymbol{u}_1' \boldsymbol{y}) + (10^8)\, \boldsymbol{v}_2 \left(\boldsymbol{u}_2' \boldsymbol{y}\right).$$

The pseudo-inverse solution "blows up" when any of the σ_k values are "too small" relative to the others, *i.e.*, if σ_k / σ_1 is near the **floating-point precision** limits.

Condition number

Such problems are called **poorly conditioned** because the $\boldsymbol{A}$ has an undesirably large **condition number**, defined as

$$\kappa(\boldsymbol{A}) \triangleq \begin{cases} \infty, & M < N \\ \square, & M \geq N \end{cases} = \begin{cases} \infty, & M < N \\ \square, & M \geq N \end{cases} = \square = \square \tag{4.20}$$

Caution: an alternate definition of condition number is σ_1 / σ_r, *e.g.*, [5, p. 69], but we will use (4.20).
Caution: Many sources consider only the case where $\boldsymbol{A}$ is square; hence the expression $\boldsymbol{A}'\boldsymbol{A}$ above.

This problem motivates the **truncated SVD** solution where we discard any singular values that are "too small" and write:

$$\hat{\boldsymbol{x}}_K = \square \tag{4.21}$$

where we choose fewer terms, $K < r$, such that $\sigma_K > \delta > 0$ for some tolerance δ.

Practical implementation of truncated SVD solution (Read)

The tolerance δ may depend on factors such as the noise level in the data, the size of $\boldsymbol{A}$, and whether one is using half, single, double or extended precision variables.

In JULIA, these variable types are `Float16`, `Float32`, `Float64`, and `BigFloat`.

JULIA's `pinv` has an optional second argument for specifying the tolerance.
The default value (see HW) is `eps(real(float(one(eltype(M)))))*minimum(size(A))`

The backslash function in JULIA `xh = A \ y` does not have any tolerance parameter, so one must use `pinv` instead of backslash to control the tolerance for poorly conditioned problems.

Alternatively, one can use some other method to improve the condition number, such as **Tikhonov regularization**, also called **ridge regression**, described later on p. 4.46.

Low-rank approximation interpretation of truncated SVD solution

L§8.2

One way to interpret the truncated SVD solution (4.21) is as follows.

- First we form a **low-rank approximation** $\boldsymbol{A}_K$ of $\boldsymbol{A}$, defined as

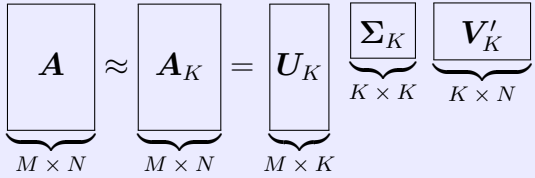
$$\boldsymbol{A}_K = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K' =$$

  with $K < r \leq \min(M, N)$. (See Ch. 6 for more details about such approximations.)
- Then we express the pseudo-inverse LLS solution using that approximation:

$$\hat{\boldsymbol{x}} =$$

  which is the same expression as (4.21). This approach is also called **principal component regression** [6].

We visualize a low-rank approximation as follows.



Noise effects and perturbations (Read)

There are two sources of perturbations in LLS problems that can degrade the estimate $\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{y}$.

- Additive noise: $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}$
- Errors in the model $\boldsymbol{A}$.
  The $\boldsymbol{A}$ we use for estimation might differ from the "true model" $\boldsymbol{A}_{\text{true}}$. Define $\Delta \boldsymbol{A} \triangleq \boldsymbol{A}_{\text{true}} - \boldsymbol{A}$.

[5, Theorem 6.12, p. 69]. (See [7].) Let $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ with $M \geq N$ and $\mathsf{rank}(\boldsymbol{A}) = N$.

Let $\kappa \triangleq \sigma_1 / \sigma_N = \|\boldsymbol{A}\|_2 / \sigma_N$ denote the condition number of $\boldsymbol{A}$, using the matrix 2-norm defined in (2.5). Assume that the model perturbations are not "too large" as follows:

$$\eta \triangleq \frac{\|\Delta \boldsymbol{A}\|_2}{\sigma_N} = \kappa \epsilon_A < 1, \quad \epsilon_A \triangleq \frac{\|\Delta \boldsymbol{A}\|_2}{\|\boldsymbol{A}\|_2}.$$

If the perturbed matrix $\boldsymbol{A} + \Delta \boldsymbol{A}$ has full rank, then the solution perturbation $\boldsymbol{\delta \hat{x}} = \hat{\boldsymbol{x}} - \boldsymbol{x}_{\text{true}}$ has the following bound in terms of the residual $\boldsymbol{r} = \boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}$:

$$\|\boldsymbol{\delta \hat{x}}\|_2 \leq \frac{\kappa}{1 - \eta} \left(\epsilon_A \|\boldsymbol{x}\|_2 + \frac{\|\boldsymbol{\varepsilon}\|_2}{\|\boldsymbol{A}\|_2} + \epsilon_A \kappa \frac{\|\boldsymbol{r}\|_2}{\|\boldsymbol{A}\|_2} \right).$$

See also [7–11].

- In the (full-rank) square case where $M = N$, the residual is $\boldsymbol{r} = \boldsymbol{0}$, so the solution error $\|\boldsymbol{\delta}\hat{\boldsymbol{x}}\|_2$ depends on the condition number κ.
- In the usual over-determined case where $M \geq N$ and $\boldsymbol{y} \notin \mathcal{R}(\boldsymbol{A})$, then the solution error is proportional to κ^2 so it is particularly important to try to keep κ small.

This bound is a motivation for using the truncated SVD where σ_1/σ_K is better (lower) than σ_1/σ_N.

Challenge. Find an error bound that depends on the truncated SVD where $\mathsf{rank}(\boldsymbol{A}) = K$, instead of the full SVD where $\mathsf{rank}(\boldsymbol{A}) = N$ was assumed above.

Tikhonov regularization aka ridge regression

A drawback of the **truncated SVD** solution to LLS problems is that it requires one to compute an SVD of $\boldsymbol{A}$, which can be impractical for large problems. An alternate approach to address ill-conditioned problems is to use **Tikhonov regularization**, also known as **ridge regression**.

We saw that the pseudo-inverse solution to LLS problems involves $1/\sigma_k$ terms that can "blow up" for small singular values, leading to very large values of $\hat{\boldsymbol{x}}$. Instead of directly modifying the singular values, **Tikhonov regularization** modifies the LS cost function to include a term that discourages the estimate from having excessively high values:

$$\hat{\boldsymbol{x}}_\beta = \arg\min_{\boldsymbol{x} \in \mathbb{F}^N} \tag{4.22}$$

where $\beta > 0$ is a **regularization parameter** that one must **tune** to trade-off between how well $\hat{\boldsymbol{x}}_\beta$ fits the data and how high is the energy of $\hat{\boldsymbol{x}}_\beta$.

By combining terms, we can rewrite the Tikhonov estimate (4.22) as:

$$\hat{\boldsymbol{x}}_\beta = \arg\min_{\boldsymbol{x} \in \mathbb{F}^N} \left\| \begin{bmatrix} \boldsymbol{A} \\ \sqrt{\beta} \boldsymbol{I} \end{bmatrix} \boldsymbol{x} - \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2 = \arg\min_{\boldsymbol{x} \in \mathbb{F}^N} \left\| \tilde{\boldsymbol{A}} \boldsymbol{x} - \tilde{\boldsymbol{y}} \right\|_2^2, \qquad \tilde{\boldsymbol{A}} \triangleq \begin{bmatrix} \\ \\ \end{bmatrix}, \quad \tilde{\boldsymbol{y}} \triangleq \begin{bmatrix} \\ \end{bmatrix}.$$

10. If $\boldsymbol{A}$ is $M \times N$, how many rows does $\tilde{\boldsymbol{A}}$ have?

A: M B: N C: $M + N$ D: $2M$ E: $2N$ **??**

11. What is the rank of $\tilde{\boldsymbol{A}}$?

A: r B: M C: N D: $M + N$ E: None of these. **??**

In this simplified form, we know that the (unique!) LLS solution to (4.22) is

$$\hat{\boldsymbol{x}}_\beta = \tilde{\boldsymbol{A}}^+ \tilde{\boldsymbol{y}} =$$

If N is large, this solution requires inverting a $N \times N$ matrix (actually, solving a $N \times N$ system of equations) which is impractical. Instead we usually apply an optimization approach (such as a **conjugate gradient method**) directly to (4.22). Nevertheless, the closed-form expression for the solution is useful for analysis.

In particular it is insightful to examine the solution in terms of an **SVD** $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$. Here,

$$\boldsymbol{A}'\boldsymbol{A} + \beta \boldsymbol{I} =$$

so

$$\hat{\boldsymbol{x}}_\beta = (\boldsymbol{A}'\boldsymbol{A} + \beta \boldsymbol{I})^{-1} \boldsymbol{A}' \boldsymbol{y} =$$

$$=$$

If $\beta \to 0$ then the ratio $\dfrac{\sigma_k}{\sigma_k^2 + \beta} \to \dfrac{1}{\sigma_k}$, unless $\sigma_k = 0$ (which never happens for $k = 1, \dots, r$).
So $\hat{\boldsymbol{x}}_\beta \to \boldsymbol{A}^+ \boldsymbol{y}$ as $\beta \to 0$.

Regularization parameter selection (Read)

An important practical question is how does one choose the regularization parameter β. Unsupervised methods include **cross validation** and **Stein's unbiased risk estimate** (**SURE**). One can also pursue supervised approaches when training data is available. The details are beyond the scope of this course, but the foundations from this course are essential for understanding such methods. ♦♦

4.5 Summary of LLS solution methods in terms of SVD

This chapter has discussed three different ways of solving the linear least-squares (LLS) problem (4.2), each of which have SVD expressions as follows.

- "Optimal" solution (minimum-norm LLS):

$$\hat{\boldsymbol{x}} = \sum_{k=1}^{r} \frac{1}{\sigma_k} \boldsymbol{v}_k \left(\boldsymbol{u}_k' \boldsymbol{y} \right)$$

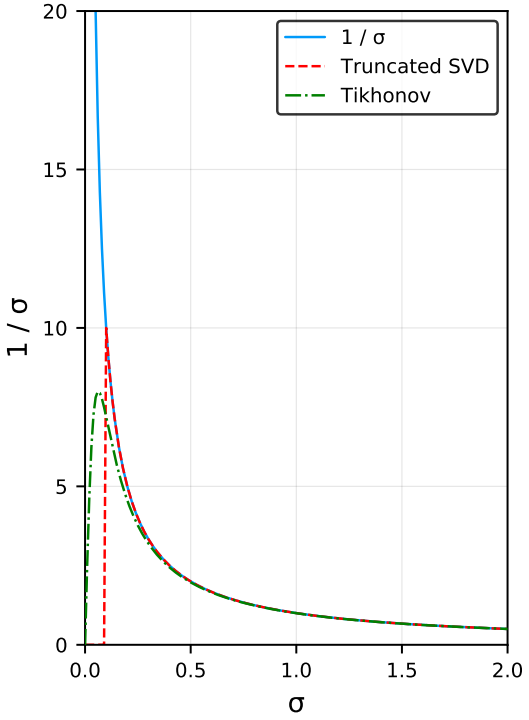
- Truncated SVD:

$$\hat{\boldsymbol{x}} = \sum_{k=1}^{K<r} \frac{1}{\sigma_k} \boldsymbol{v}_k \left(\boldsymbol{u}_k' \boldsymbol{y} \right)$$

- Tikhonov regularized:

$$\hat{\boldsymbol{x}} = \sum_{k=1}^{r} \left(\frac{\sigma_k}{\sigma_k^2 + \beta} \right) \boldsymbol{v}_k \left(\boldsymbol{u}_k' \boldsymbol{y} \right)$$

The SVD is a key tool for understanding LLS problems.



4.6 Frames and tight frames

A LS minimization problem $\arg\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2$ is the easiest to solve when $\boldsymbol{A}$ is unitary because in that case $\hat{\boldsymbol{x}} = \boldsymbol{A}'\boldsymbol{y}$. We now discuss a generalization of unitary matrices that leads to equally easy solutions.

For simplicity we focus on the case of a finite number of vectors in a finite-dimensional vector space [13]. The concepts generalize to a countable collection of vectors in a general Hilbert space.

Define. A collection of M vectors $\{\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_M\}$ in $\mathbb{F}^N$ is called a **frame** in $\mathbb{F}^N$ iff there exist real numbers $0 < \alpha \le \beta < \infty$, called the **frame bounds**, such that

$$\alpha \|\boldsymbol{x}\|_2^2 \le \sum_{m=1}^{M} |\langle \boldsymbol{\phi}_m, \boldsymbol{x} \rangle|^2 \le \beta \|\boldsymbol{x}\|_2^2, \quad \forall \boldsymbol{x} \in \mathbb{F}^N. \tag{4.23}$$

In other words, if we arrange those vectors into a $N \times M$ matrix $\boldsymbol{\Phi} \triangleq \begin{bmatrix} \boldsymbol{\phi}_1 & \dots & \boldsymbol{\phi}_M \end{bmatrix}$, then the collection of vectors is a **frame** iff there exist real numbers $0 < \alpha \le \beta < \infty$ such that

$$\alpha \|\boldsymbol{x}\|_2^2 \le \|\boldsymbol{\Phi}'\boldsymbol{x}\|_2^2 \le \beta \|\boldsymbol{x}\|_2^2, \quad \forall \boldsymbol{x} \in \mathbb{F}^N. \tag{4.24}$$

For brevity, we call such a $\boldsymbol{\Phi}$ a **frame**.

The upper bound is important in infinite-dimensional inner product spaces, but is not very informative in $\mathbb{F}^N$.

12.

If $\alpha > 0$, then what is β here in terms of the **singular values** of $\mathbf{\Phi}$?
A: σ_1 B: σ_1^2 C: σ_r^2 D: σ_M E: None of these

??

When is the upper frame bound β positive? Always unless $\mathbf{\Phi} = \mathbf{0}$.

So in $\mathbb{F}^N$, the key to whether $\mathbf{\Phi}$ is a frame or not depends on existence of $\alpha > 0$.

Properties of a frame

- Fact. $\alpha > 0$ in (4.24) iff $\mathbf{\Phi}$ has full **row rank** (*i.e.*, $\mathsf{rank}(\mathbf{\Phi}) = N$).
- Thus $\mathbf{\Phi}$ must be wide (usually) or square, *i.e.*, $M \geq N$.
- Thus $\alpha > 0 \implies \mathbf{\Phi} = \boldsymbol{U} \boldsymbol{\Sigma}_N \boldsymbol{V}_N'$ (**compact SVD** is same as **economy SVD**) where $\sigma_N > 0$, because $\boldsymbol{U}_r = \boldsymbol{U}_N = \boldsymbol{U}$ and $\boldsymbol{V}_r = \boldsymbol{V}_N$.
- So $\mathbf{\Phi}^+ = \boldsymbol{V}_N \boldsymbol{\Sigma}_N^{-1} \boldsymbol{U}' = \mathbf{\Phi}'(\mathbf{\Phi}\mathbf{\Phi}')^{-1}$ and $\mathbf{\Phi}\mathbf{\Phi}^+ = \boldsymbol{I}_N$.

Now suppose we have a vector $\boldsymbol{x} \in \mathbb{F}^N$ that we would like to express as a linear combination of the frame vectors $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_M\}$, *i.e.*, we want to write

$$\boldsymbol{x} = \mathbf{\Phi} \boldsymbol{c}$$

for some coefficient vector $\boldsymbol{c} \in \mathbb{F}^M$. In the usual case where $\mathbf{\Phi}$ is wide, there will be numerous possible coefficient vectors $\boldsymbol{c}$ for which $\|\boldsymbol{x} - \mathbf{\Phi}\boldsymbol{c}\|_2 = 0$. The unique such $\boldsymbol{c}$ having minimum norm is given by the Moore-Penrose pseudo-inverse:

$$\boldsymbol{c} = \mathbf{\Phi}^+ \boldsymbol{x} = \mathbf{\Phi}'(\mathbf{\Phi}\mathbf{\Phi}')^{-1} \boldsymbol{x}.$$

This is not too exciting yet because of the expensive matrix inverse. So we impose more conditions on $\mathbf{\Phi}$.

Tight frame

Define. A matrix $\boldsymbol{\Phi}$ is a **tight frame** [14–16] iff $\boldsymbol{\Phi}$ is a **frame** with $\alpha = \beta$, *i.e.*,

$$\alpha \|\boldsymbol{x}\|_2^2 = \|\boldsymbol{\Phi}'\boldsymbol{x}\|_2^2 = \sum_{m=1}^{M} |\langle \boldsymbol{\phi}_m, \boldsymbol{x} \rangle|^2, \quad \forall \boldsymbol{x} \in \mathbb{F}^N. \tag{4.25}$$

If $\boldsymbol{\Phi}$ is a **tight frame**, then

- $\boldsymbol{\Phi}\boldsymbol{\Phi}' = \alpha \boldsymbol{I}_N$, so $\alpha = \sigma_1^2 = \cdots = \sigma_N^2$, where $\{\sigma_k\}$ denotes the singular values of $\boldsymbol{\Phi}$,
- its pseudo-inverse is simply $\boldsymbol{\Phi}^+ = \frac{1}{\alpha}\boldsymbol{\Phi}' = \frac{1}{\sigma_1^2}\boldsymbol{\Phi}'$.

Proof. Using (4.25):

$$\alpha \|\boldsymbol{x}\|_2^2 = \|\boldsymbol{\Phi}'\boldsymbol{x}\|_2^2 \implies \boldsymbol{x}'(\alpha \boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}')\boldsymbol{x} = 0, \ \forall \boldsymbol{x} \in \mathbb{F}^N \implies \alpha \boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}' = \boldsymbol{0} \implies \alpha = \mathsf{eig}\{\boldsymbol{\Phi}\boldsymbol{\Phi}'\} = \left\{\sigma_k^2\right\},$$

because the singular values are the square roots of those eigenvalues. □

Normally, finding σ_1 requires an SVD, which is expensive for large problems. But for a tight frame,

$$\|\boldsymbol{\Phi}'\boldsymbol{e}_1\|_2 = \sqrt{\alpha} = \sigma_1$$

so here we can find σ_1 by a simple matrix-vector multiplication and a norm.

Furthermore, one can show that $\boldsymbol{\Phi}'\boldsymbol{\Phi} \preceq \alpha \boldsymbol{I}$, *i.e.*, $\boldsymbol{I} - \boldsymbol{\Phi}'\boldsymbol{\Phi} \succeq \boldsymbol{0}$, using the SVD $\boldsymbol{\Phi} = \boldsymbol{U} \begin{bmatrix} \sqrt{\alpha}\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \boldsymbol{V}'$. (HW)

Parseval tight frame

Define. A matrix $\boldsymbol{\Phi}$ is a **Parseval tight frame** [14, 15] iff $\boldsymbol{\Phi}$ is a **tight frame** with $\alpha = \beta = 1$, *i.e.*,

$$\|\boldsymbol{x}\|_2^2 = \|\boldsymbol{\Phi}'\boldsymbol{x}\|_2^2 = \sum_{m=1}^{M} |\langle \boldsymbol{\phi}_m, \boldsymbol{x} \rangle|^2, \quad \forall \boldsymbol{x} \in \mathbb{F}^N. \tag{4.26}$$

What are the singular values σ_1 and σ_N of $\boldsymbol{\Phi}$ in this case? $\sigma_1 = \cdots = \sigma_N = 1$

Properties of Parseval tight frames

If $\boldsymbol{\Phi}$ is a **Parseval tight frame**, then
- $\boldsymbol{\Phi}\boldsymbol{\Phi}' = \boldsymbol{I}_N$,
- its pseudo-inverse is simply $\boldsymbol{\Phi}^+ = \boldsymbol{\Phi}'$.

Proof. Using (4.26):

$$\|\boldsymbol{x}\|_2^2 = \|\boldsymbol{\Phi}'\boldsymbol{x}\|_2^2 \implies \boldsymbol{x}'(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}')\boldsymbol{x} = 0, \ \forall \boldsymbol{x} \in \mathbb{F}^N \implies \boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}' = \boldsymbol{0}.$$

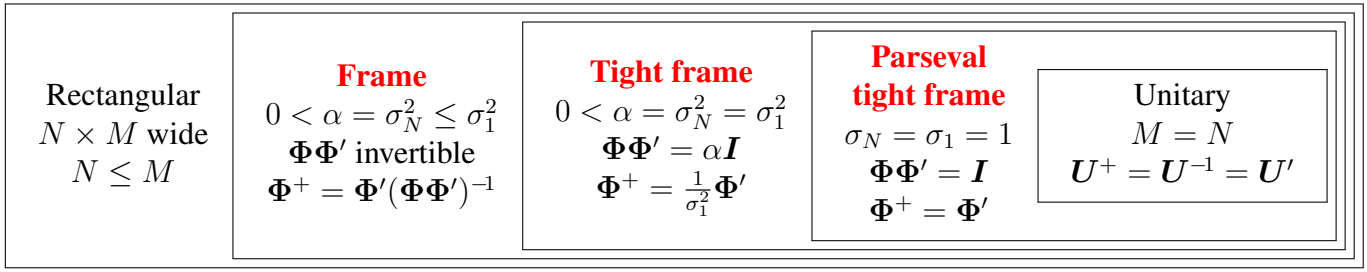
13. The converse also holds, *i.e.*, if $\boldsymbol{\Phi}\boldsymbol{\Phi}' = \boldsymbol{I}_N$, then $\boldsymbol{\Phi}$ is a **Parseval tight frame**. (?)
A: True B: False **??**

14. Every **unitary matrix** is a **tight frame**. (?)
A: True B: False **??**

Example. A simple **Parseval tight frame** is the "Mercedes Benz" frame: $\boldsymbol{\Phi} = \sqrt{\frac{2}{3}} \begin{bmatrix} 0 & -\sqrt{3}/2 & \sqrt{3}/2 \\ 1 & -1/2 & -1/2 \end{bmatrix}$.
One can verify that $\boldsymbol{\Phi}\boldsymbol{\Phi}' = \boldsymbol{I}_2$.

15. Every **Parseval tight frame** is a **unitary matrix**. (?)
A: True B: False **??**

Venn diagram of frames



Another way to construct a **tight frame** is to combine multiple unitary matrices:

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{U}_1 & \dots & \boldsymbol{U}_K \end{bmatrix}.$$

This construction is fairly common in signal processing, *e.g.*, combining orthogonal wavelet transforms with other transforms.

16. If $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are $N \times N$ unitary matrices, what is the frame bound of $\boldsymbol{\Phi} \triangleq \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix}$?

A: 1 B: 2 C: N D: $2N$ E: None **??**

- Frames have numerous uses in signal processing [14–17].
- In the usual case where $\boldsymbol{\Phi}$ is wide, they are considered "robust redundant signal representations" [18].
- Frame theory underpins recent advances in image denoising and attempts to provide insights into CNNs by relating them to perfect reconstruction filter banks [19] [20] [21].
- Solving LLS problems for a **Parseval tight frame** $\boldsymbol{\Phi}$ is as easy as for a unitary matrix, because $\boldsymbol{\Phi}^+ = \boldsymbol{\Phi}'$. So given a vector $\boldsymbol{x}$ in $\mathbb{F}^N$, the representation of $\boldsymbol{x}$ using the (typically linearly dependent!) set of "atoms" $\{\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_M\}$ for which the coefficient vector has minimum 2-norm is

  $$\boldsymbol{x} = \boldsymbol{\Phi}\boldsymbol{c}, \qquad \boldsymbol{c} = \boldsymbol{\Phi}'\boldsymbol{x}.$$

  In this sense, Parseval tight frames are generalizations of orthonormal bases and unitary matrices.
- Other minimum-norm representations are also of interest, such as [22]

  $$\arg\min_{\boldsymbol{c}: \boldsymbol{x} = \boldsymbol{\Phi}\boldsymbol{c}} \|\boldsymbol{c}\|_p.$$

4.7 Projection and orthogonal projection

Idempotent matrix

L§7.1

Define. A (square) matrix $\boldsymbol{P}$ is called a **projection matrix** iff $\boldsymbol{P}^2 = \boldsymbol{P}\boldsymbol{P} = \boldsymbol{P}$.
Such a (square) matrix is also called an **idempotent matrix** .

Example. $\boldsymbol{P} = \begin{bmatrix} 1 & a \\ 0 & 0 \end{bmatrix} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}$, $\boldsymbol{V} = \begin{bmatrix} 1 & -a/\sqrt{1+a^2} \\ 0 & 1/\sqrt{1+a^2} \end{bmatrix}$, $\boldsymbol{\Lambda} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ is idempotent for any a.

All eigenvalues of an idempotent matrix are either 0 or 1.

Proof. Suppose $\boldsymbol{x}$ is an eigenvector of an idempotent matrix $\boldsymbol{P}$, *i.e.*, $\boldsymbol{P}\boldsymbol{x} = \lambda\boldsymbol{x}$. Multiplying both sides by $\boldsymbol{P}$ yields $\boldsymbol{P}(\boldsymbol{P}\boldsymbol{x}) = \lambda(\boldsymbol{P}\boldsymbol{x}) \implies \boldsymbol{P}\boldsymbol{x} = \lambda^2\boldsymbol{x}$. Combining we have $\lambda^2 = \lambda$, so $\lambda = 0$ or $\lambda = 1$. □

Fact. Every projection matrix is **diagonalizable** [wiki].

17. The converse of that property also holds, *i.e.*, if $\boldsymbol{A}$ is a (square) diagonalizable matrix with eigenvalues that are all either 0 or 1, then $\boldsymbol{A}$ is always idempotent. (?)
A: True B: False **??**

??

Example. For any matrix $\boldsymbol{A}$, the matrix $\boldsymbol{P} = \boldsymbol{A}\boldsymbol{A}^+$ is **idempotent**,
because $\boldsymbol{P}^2 = (\boldsymbol{A}\boldsymbol{A}^+)(\boldsymbol{A}\boldsymbol{A}^+) = \boldsymbol{A}(\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+) = \boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{P}$.

Orthogonal projection matrix

Our primary interest will be the subset of projection matrices that are (Hermitian) symmetric matrices.

Define. A (square) matrix $\boldsymbol{P}$ is called an **orthogonal projector** or **orthogonal projection matrix** iff $\boldsymbol{P}$ is **idempotent** and $\boldsymbol{P}$ is **Hermitian**.

Caution: An **orthogonal projection matrix** typically is not an **orthogonal matrix**!
If $\boldsymbol{P} = \boldsymbol{P}' = \boldsymbol{P}^2$ is an **orthogonal projection matrix** and if $\boldsymbol{P}$ is also an **orthogonal matrix** then $\boldsymbol{I} = \boldsymbol{P}'\boldsymbol{P} = \boldsymbol{P}^2 = \boldsymbol{P}$. So the only matrix that is both is the identity matrix $\boldsymbol{I}$.

Example. $\boldsymbol{P} = \frac{1}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \left(\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}\right)\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\left(\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}\right).$

18. Every **orthogonal projection matrix** has a **unitary eigendecomposition**. (?)
A: True B: False **??**

19. If x has unit norm, then xx' is an orthogonal projection matrix. (?)
A: True B: False **??**

20. Every orthogonal projection matrix $\boldsymbol{P}$ is positive semidefinite. (?)
A: True B: False **??**

Alternative explanation: if $\boldsymbol{P}$ is an orthogonal projection matrix, *i.e.*, both idempotent and Hermitian, then:

$$\boldsymbol{P} =$$

More generally, if $\boldsymbol{Q}$ is any (possibly non-square) matrix with orthonormal columns, then $\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{Q}'$ is an orthogonal projection matrix, because

- $\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{Q}'$ is Hermitian
- $\boldsymbol{P}^2 = (\boldsymbol{Q}\boldsymbol{Q}')(\boldsymbol{Q}\boldsymbol{Q}') = \boldsymbol{Q}\boldsymbol{Q}' = \boldsymbol{P}$, because $\boldsymbol{Q}'\boldsymbol{Q} = \boldsymbol{I}$.

21. Is the converse true? Can a $N \times N$ **orthogonal projection matrix** $\boldsymbol{P}$ be written as $\boldsymbol{Q}\boldsymbol{Q}'$ for some matrix $\boldsymbol{Q}$ with orthonormal columns?

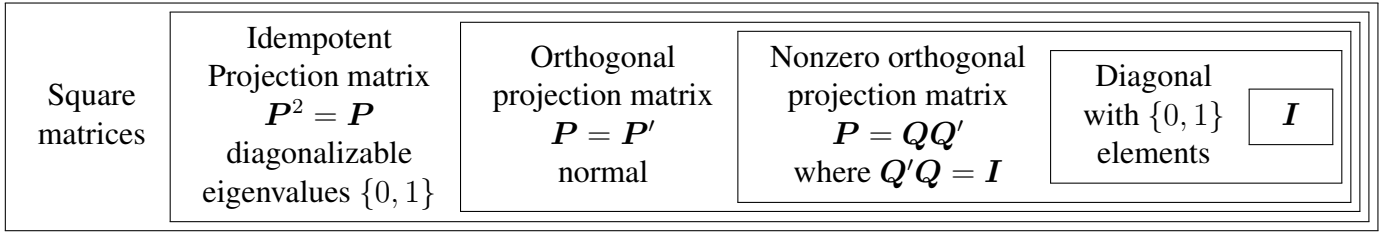
A: Yes, always.
B: Yes, if $\mathsf{rank}(\boldsymbol{P}) \geq 1$.
C: Yes, if $\mathsf{rank}(\boldsymbol{P}) \leq N - 1$.
D: Only if $\boldsymbol{P}$ is nonsingular.
E: No.

??

22. When writing a nonzero **orthogonal projection matrix** $\boldsymbol{P}$ as $\boldsymbol{QQ}'$, the $\boldsymbol{Q}$ is unique.
A: True B: False **??**

23. The $N \times N$ **Gram matrix** $\boldsymbol{G} = \boldsymbol{\Phi}'\boldsymbol{\Phi}$ of a $N \times M$ **Parseval tight frame** $\boldsymbol{\Phi}$ is an **orthogonal projection matrix**. (?)
A: True B: False **??**

The following **Venn diagram** summarizes relationships related to **projection** matrices.



Convex sets

Define. A nonempty set $\mathcal{C}$ in a vector space $\mathcal{V}$ is a **convex set** iff

$$\boldsymbol{x}, \boldsymbol{z} \in \mathcal{C} \implies \alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{z} \in \mathcal{C}, \quad \forall 0 \le \alpha \le 1.$$

A **linear combination** $\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{z}$ for any $0 \le \alpha \le 1$ is a **convex combination**.

It follows by induction that if $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K \in \mathcal{C}$ and $\alpha_1, \ldots, \alpha_K \ge 0$ and $\sum_{k=1}^K \alpha_k = 1$, then $\sum_{k=1}^K \alpha_k \boldsymbol{x}_k \in \mathcal{C}$. Such a sum is also called a **convex combination**.

Example. Any **subspace** $\mathcal{S}$ in a vector space $\mathcal{V}$ is a **convex set**.
If $\mathcal{V}$ is a finite-dimensional vector space, then any subspace is a **closed** convex set.

Example. The **nonnegative orthant** $\mathbb{R}_+^N \triangleq \left\{ \boldsymbol{x} \in \mathbb{R}^N \ : \ \boldsymbol{x} \ge \boldsymbol{0} \right\}$ is a convex set.

Example. The ball of radius $r > 0$ in $\mathbb{F}^N$, defined as $\mathcal{B}_r \triangleq \left\{ \boldsymbol{x} \in \mathbb{F}^N \ : \ \|\boldsymbol{x}\|_2 \le r \right\}$, is a convex set. (HW)

Projection onto convex sets

Finding the nearest point in a set is an important operation.
Ch. 3 defined the **projection** of a point $\boldsymbol{v}$ onto a set $\mathcal{S}$ as

$$\mathcal{P}_{\mathcal{S}}(\boldsymbol{v}) \triangleq \arg\min_{\boldsymbol{s} \in \mathcal{S}} \|\boldsymbol{v} - \boldsymbol{s}\| .$$

For general sets $\mathcal{S}$, the "arg min" in this definition need not be unique.
However, if $\mathcal{C}$ is a **closed convex set** in a finite-dimensional vector space $\mathcal{V}$,
then for any $\boldsymbol{v} \in \mathcal{V}$ the nearest point in $\mathcal{C}$ exists and is unique.
(See EECS 600 or IOE 611.) Thus the "arg min" in the definition of $\mathcal{P}_{\mathcal{C}}$ above is well-defined.

We have used the term **projection** in two ways here.

- One use is in defining a $N \times N$ **projection matrix** $\boldsymbol{P}$ that has the property that $\boldsymbol{P}^2 = \boldsymbol{P}$.
  In other words, for any $\boldsymbol{x} \in \mathbb{F}^N$: $\boldsymbol{P}(\boldsymbol{P}\boldsymbol{x}) = \boldsymbol{P}^2\boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}$.
- Another use is in defining the **projection** onto a (typically convex) set $\mathcal{P}_{\mathcal{C}}(\cdot)$.

One connection between the two uses is that if we project a point $\boldsymbol{v}$ onto a set $\mathcal{C}$ to get $\hat{\boldsymbol{v}} = \mathcal{P}_{\mathcal{C}}(\boldsymbol{v})$ and then project $\hat{\boldsymbol{v}}$ onto $\mathcal{C}$ we get the same point: $\hat{\boldsymbol{v}} = \mathcal{P}_{\mathcal{C}}(\hat{\boldsymbol{v}})$.
In other words, for any $\boldsymbol{v} \in \mathcal{V}$: $\mathcal{P}_{\mathcal{C}}(\mathcal{P}_{\mathcal{C}}(\boldsymbol{v})) = \mathcal{P}_{\mathcal{C}}(\boldsymbol{v})$.
Put concisely, we have an expression very similar to $\boldsymbol{P}^2 = \boldsymbol{P}$, hence the similarity in terminology:

$$\mathcal{P}_{\mathcal{C}} \circ \mathcal{P}_{\mathcal{C}} = \mathcal{P}_{\mathcal{C}}.$$

Projection onto a subspace

Now we move towards using projection as a tool for solving signal processing problems.

Let $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N$ denote any collection of N vectors, each in $\mathbb{F}^M$. The span of this set defines a subspace:

$$\mathcal{S} \triangleq \mathsf{span}(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\}) .$$

Equivalently (because we are working in $\mathbb{F}^M$ here) we can group the vectors into a $M \times N$ matrix and then describe the subspace as the **range** of this matrix:

$$\mathcal{S} = \mathcal{R}(\boldsymbol{A}) \triangleq \left\{ \boldsymbol{A}\boldsymbol{x} \ : \ \boldsymbol{x} \in \mathbb{F}^N \right\}, \qquad \boldsymbol{A} \triangleq \begin{bmatrix} \boldsymbol{a}_1 & \ldots & \boldsymbol{a}_N \end{bmatrix} .$$

As introducted in Ch. 3, in many applications (including handwritten digit recognition via nearest subspace) we are given some test vector $\boldsymbol{y}$ and we want to find the vector in a subspace $\mathcal{S}$ that is closest to $\boldsymbol{y}$:

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{s} \in \mathcal{S} = \mathcal{R}(\boldsymbol{A})}{\arg\min} \ \|\boldsymbol{y} - \boldsymbol{s}\|_2 .$$

The resulting vector $\hat{\boldsymbol{y}}$ is called the **projection** of the point $\boldsymbol{y}$ onto the subspace $\mathcal{S}$, and the process (operation) of finding that closest point is called **projecting** the point $\boldsymbol{y}$ onto the subspace $\mathcal{S}$, as introduced in Ch. 3.

The key to solving this problem is to use the fact that every point in the subspace $\mathcal{S}$ has the form $\boldsymbol{A}\boldsymbol{x}$ for some $\boldsymbol{x} \in \mathbb{F}^N$. Thus $\hat{\boldsymbol{y}} = \boldsymbol{A}\hat{\boldsymbol{x}}$ for some $\hat{\boldsymbol{x}} \in \mathbb{F}^N$, and we just have to find that best $\hat{\boldsymbol{x}}$ to get $\hat{\boldsymbol{y}}$. In other words, the closest point or projection problem is equivalent to:

$$\hat{\boldsymbol{y}} =$$

because every $\boldsymbol{s} \in \mathcal{S}$ is $\boldsymbol{s} = \boldsymbol{A}\boldsymbol{x}$ for some $\boldsymbol{x} \in \mathbb{F}^N$.

This form involves solving a LS problem, and a solution to that part is $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{y}$, so

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{s} \in \mathcal{S} = \mathcal{R}(\boldsymbol{A})}{\arg\min} \|\boldsymbol{y} - \boldsymbol{s}\|_2 =$$

In other words:

$$\mathcal{P}_{\mathcal{R}(\boldsymbol{A})}(\boldsymbol{y}) = \boldsymbol{A}\boldsymbol{A}^+\boldsymbol{y}.$$

24. If $\boldsymbol{A}$ does not have full rank, then there will be other LS solutions, *i.e.*, $\hat{\boldsymbol{x}}$ is not unique. Thus, $\hat{\boldsymbol{y}}$ is not unique. (?)

A: True B: False **??**

Practical implementation

Reiterating, the point on the subspace $\mathcal{R}(\boldsymbol{A})$ that is closest to the point $\boldsymbol{y}$ is $\hat{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{A}^+\boldsymbol{y}$

If we need to compute this just once, for one $\boldsymbol{A}$ and one $\boldsymbol{y}$, then using code like `A * (A \ y)` is fine if $\boldsymbol{A}$ is small enough for backslash to work. If $\boldsymbol{A}$ is large, then we use an iterative method to compute the LS coefficients $\hat{\boldsymbol{x}}$ first, then compute the **projection** $\hat{\boldsymbol{y}} = \boldsymbol{A}\hat{\boldsymbol{x}}$.

The parentheses are essential in `A * (A \ y)` because `*` and `\` have the same operator precedence in JULIA.

But in most applications $\boldsymbol{A}$ is fixed (after some training or modeling process) and we will need to perform projection for many different "test" $\boldsymbol{y}$ vectors. In such cases, it is more efficient to use an SVD at the beginning (as part of the training process) to save computation at the test stage. Specifically, when $r > 0$:

$$\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})} \triangleq \boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r'\boldsymbol{V}_r\boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}_r' = \boldsymbol{U}_r\boldsymbol{U}_r'.$$

So a practical implementation is something like the following JULIA code for $r > 0$:

```
(U,s,V) = svd(A)
r = sum(s .> threshold)
Ur = U[:,1:r]
projector = (y) -> Ur * (Ur'*y)
```

We do the SVD once, then after that we need only use simple matrix-vector multiplies to perform projection.

Note that $\boldsymbol{U}_r$ is an orthonormal basis for $\mathcal{R}(\boldsymbol{A})$ and $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})} = \boldsymbol{U}_r\boldsymbol{U}_r'$. This is not a coincidence!

Orthonormal vs non-orthonormal bases for a subspace (Read)

More generally, if $\boldsymbol{Q}$ is a matrix whose columns are orthonormal, then those columns form an orthonormal basis for a subspace, namely $\mathcal{R}(\boldsymbol{Q})$, and the projection of a vector $\boldsymbol{y}$ onto that subspace is simply

$$\mathcal{P}_{\mathcal{R}(\boldsymbol{Q})}(\boldsymbol{y}) = \boldsymbol{P}_{\mathcal{R}(\boldsymbol{Q})}\boldsymbol{y} = \boldsymbol{Q}(\boldsymbol{Q}'\boldsymbol{y}).$$

So **orthonormal bases** for a subspace are extremely convenient for computation, because subspace projection using such bases requires mere matrix-vector multiplication.

If a basis $\boldsymbol{B}$ is not orthonormal, then to compute the projection we would need

$$\boldsymbol{P}_{\mathcal{R}(\boldsymbol{B})}\boldsymbol{y} = \boldsymbol{B}(\boldsymbol{B}^{+}\boldsymbol{y}),$$

as derived above, which is much more expensive in general because $\boldsymbol{B}^{+}$ usually requires an SVD.

Orthogonality principle revisited (Read)

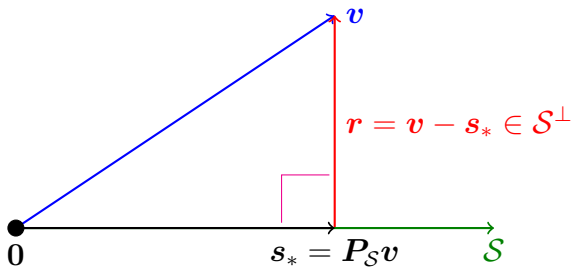
Another version of the **orthogonality principle** is the following. Let $\mathcal{S}$ denote any subspace of a vector space $\mathcal{V}$, and $\boldsymbol{v} \in \mathcal{V}$ be any point in that vector space. If the closest point in $\mathcal{S}$ to $\boldsymbol{v}$ is

$$\boldsymbol{s}_* = \arg\min_{\boldsymbol{s} \in \mathcal{S}} \|\boldsymbol{v} - \boldsymbol{s}\|_2 = \boldsymbol{P}_{\mathcal{S}} \boldsymbol{v},$$

then the **orthogonality principle** says that the residual vector $\boldsymbol{r} = \boldsymbol{v} - \boldsymbol{s}_*$ is perpendicular to the entire subspace $\mathcal{S}$, *i.e.*,

$$\langle \boldsymbol{v} - \boldsymbol{s}_*, \boldsymbol{s} \rangle = 0, \quad \forall \boldsymbol{s} \in \mathcal{S}.$$

The following figure illustrates this principle.



Note that $\boldsymbol{r} = \boldsymbol{v} - \boldsymbol{s}_* = \boldsymbol{v} - \boldsymbol{P}_{\mathcal{S}} \boldsymbol{v} = (\boldsymbol{I} - \boldsymbol{P}_{\mathcal{S}}) \boldsymbol{v} = \boldsymbol{P}_{\mathcal{S}}^{\perp} \boldsymbol{v}, \quad \boldsymbol{P}_{\mathcal{S}}^{\perp} \triangleq \boldsymbol{I} - \boldsymbol{P}_{\mathcal{S}}.$

Proof sketch from [wiki]: For $\boldsymbol{s} \in \mathcal{S}$ and any $\alpha \in \mathbb{F}$: $0 \le \|(\boldsymbol{s}_* + \alpha \boldsymbol{s}) - \boldsymbol{v}\|_2^2 - \|\boldsymbol{s}_* - \boldsymbol{v}\|_2^2$
$= 2 \operatorname{real}\{\alpha \langle \boldsymbol{s}_* - \boldsymbol{v}, \boldsymbol{s} \rangle\} + |\alpha|^2 \|\boldsymbol{s}\|_2^2 \implies \langle \boldsymbol{s}_* - \boldsymbol{v}, \boldsymbol{s} \rangle = 0.$

Projection onto a subspace's orthogonal complement

Recall that in a finite-dimensional vector space $\mathcal{V}$, if $\mathcal{S}$ is a subspace of $\mathcal{V}$, then

$$\mathcal{V} =$$

Clearly $\boldsymbol{P}_{\mathcal{V}} = \boldsymbol{I}$, so it follows that

$$\boldsymbol{I} =$$

Thus the projection onto the subspace's **orthogonal complement** $\mathcal{S}^\perp$ is simply:

$$\boldsymbol{P}_{\mathcal{S}^\perp} =$$

If $\boldsymbol{P}$ is any projection matrix, then as a notation convention we define

$$\boldsymbol{P}^\perp \triangleq$$

It then follows that

$$\boldsymbol{P}_{\mathcal{S}^\perp} =$$

In words, $\boldsymbol{P}_{\mathcal{S}}^\perp$ is the orthogonal projector onto $\mathcal{S}^\perp$, the orthogonal complement of the subspace $\mathcal{S}$.

Example. If $\mathcal{S} = \mathsf{span}(\{\mathbf{1}_n\})$ then $\boldsymbol{P}_{\mathcal{S}}^\perp = \boldsymbol{I} - \mathbf{1}_n \mathbf{1}_n^+ = \boldsymbol{I} - \frac{1}{n}\mathbf{1}_n \mathbf{1}_n'$.
The matrix-vector product $\boldsymbol{y} = \boldsymbol{P}_{\mathcal{S}}^\perp \boldsymbol{x}$ returns a vector $\boldsymbol{y}$ having zero mean, so $\boldsymbol{y}$ is orthogonal to $\mathbf{1}_n$.

Projectors and the four fundamental subspaces

Recall the SVD anatomy of a $M \times N$ matrix with rank $0 < r < \min(M,N)$: L§7.1.1

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}' = \sum_{k=1}^{\min(M,N)} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' = \sum_{k=1}^{r} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k' = \left[\begin{array}{c|c} \boldsymbol{U}_r & \boldsymbol{U}_0 \end{array}\right] \left[\begin{array}{c|c} \boldsymbol{\Sigma}_r & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{0} \end{array}\right] \left[\begin{array}{c|c} \boldsymbol{V}_r & \boldsymbol{V}_0 \end{array}\right]'.$$

The four fundamental subspaces have the following **orthonormal bases** and **orthogonal projectors**.

space in terms of $\boldsymbol{A}$	subspace in terms of $\boldsymbol{A}'$	equivalent subspace matrix	orthonormal basis	orthogonal projector
$\mathbb{F}^N$	$\mathcal{N}(\boldsymbol{A})$	$\mathcal{R}^\perp(\boldsymbol{A}')$	$\boldsymbol{V}_0$	$\boldsymbol{V}_0\boldsymbol{V}_0' = \boldsymbol{I} - \boldsymbol{V}_r\boldsymbol{V}_r'$
$\mathbb{F}^N$	$\mathcal{N}^\perp(\boldsymbol{A})$	$\mathcal{R}(\boldsymbol{A}')$	$\boldsymbol{V}_r$	$\boldsymbol{V}_r\boldsymbol{V}_r'$
$\mathbb{F}^M$	$\mathcal{R}(\boldsymbol{A})$	$\mathcal{N}^\perp(\boldsymbol{A}')$	$\boldsymbol{U}_r$	$\boldsymbol{U}_r\boldsymbol{U}_r'$
$\mathbb{F}^M$	$\mathcal{R}^\perp(\boldsymbol{A})$	$\mathcal{N}(\boldsymbol{A}')$	$\boldsymbol{U}_0$	$\boldsymbol{U}_0\boldsymbol{U}_0' = \boldsymbol{I} - \boldsymbol{U}_r\boldsymbol{U}_r'$

I list two forms for the projectors involving $\boldsymbol{U}_0$ and $\boldsymbol{V}_0$ because if we have stored "only" the **compact SVD**, then we will need to use $\boldsymbol{U}_r$ and $\boldsymbol{V}_r$.

25. $\mathcal{N}^\perp(\boldsymbol{A}') \oplus \mathcal{R}^\perp(\boldsymbol{A}) = \mathbb{F}^M$. (?)

A: True B: False **??**

Warm-up questions: projections

26. If $\boldsymbol{Q}$ is a matrix with orthonormal columns and $\boldsymbol{D}$ is diagonal matrix whose elements are all 0 or 1, then $\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}'$ is an **orthogonal projection matrix**. (?)

A: True B: False **??**

27. Let rank r matrix $\boldsymbol{A}$ have **SVD** $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$ and **compact SVD** $\boldsymbol{A} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r'$, where we partition unitary matrix $\boldsymbol{V}$ as usual as $\boldsymbol{V} = \begin{bmatrix}\boldsymbol{V}_r & \boldsymbol{V}_0\end{bmatrix}$. Which of the following matrices is $\boldsymbol{P}_{\mathcal{N}(\boldsymbol{A}'\boldsymbol{A})}$?

A: $\boldsymbol{V}_0\boldsymbol{V}_0'$ B: $\boldsymbol{V}_r\boldsymbol{V}_r'$ C: $\boldsymbol{V}_0\boldsymbol{V}_r'$ D: $\boldsymbol{V}_r\boldsymbol{V}_0'$ E: None of these. **??**

28. Continuing the previous problem, and assuming the rank r is much smaller than the matrix dimensions, which of the following JULIA snippets is the most efficient way to implement $\boldsymbol{P}_{\mathcal{R}^\perp(\boldsymbol{A}^+)}\boldsymbol{x}$ after the line `U,s,V = svd(A)` ?

A: `V[:,1:r] * (V[:,1:r]' * x)` B: `V[:,1:r] * V[:,1:r]' * x`

C: `x - V[:,1:r] * V[:,1:r]' * x` D: `x - V[:,1:r] * (V[:,1:r]' * x)`

E: `V[:,(r+1):end] * (V[:,(r+1):end]' * x)`

??

This last question is a "mini review" that uses elements from Ch. 1 (multiplication order), Ch. 2 (SVD), Ch. 3 (subspaces), Ch. 4 (pseudo-inverse, orthogonal projection), and JULIA!

Binary classifier design using least-squares (Read)

Linear LS can be used as a simple tool for designing a binary classifier via supervised learning.

Given M feature vectors $\boldsymbol{v}_i \in \mathbb{F}^N$ and corresponding binary class labels $y_i = \pm 1$. We want to find a weight vector $\boldsymbol{x}$ such that $\langle \boldsymbol{x}, \boldsymbol{v}_i \rangle$ has the same sign as y_i. Equivalently, we want $\operatorname{sign}(\langle \boldsymbol{x}, y_i \boldsymbol{v}_i \rangle) = 1$.

Form a $M \times N$ data matrix from the training feature vectors and a corresponding label vector:

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{v}_1' \\ \vdots \\ \boldsymbol{v}_M' \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix}.$$

Then solve a least-squares problem (or regularized variant thereof):

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \left\| \tilde{\boldsymbol{A}} \boldsymbol{x} - \boldsymbol{y} \right\|_2 .$$

Because each element of $\boldsymbol{y}$ is ± 1, the diagonal matrix $\boldsymbol{D} = \mathsf{Diag}\{\boldsymbol{y}\}$ is unitary and $\boldsymbol{D}^{-1} = \boldsymbol{D}' = \boldsymbol{D}$. Using unitary invariance of the 2-norm, and the fact that $\boldsymbol{y} = \boldsymbol{D}\boldsymbol{1}$, an equivalent formulation is

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \left\| \boldsymbol{D} \tilde{\boldsymbol{A}} \boldsymbol{x} - \boldsymbol{1}_M \right\|_2 = \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \left\| \boldsymbol{A} \boldsymbol{x} - \boldsymbol{1}_M \right\|_2 , \qquad \boldsymbol{A} \triangleq \begin{bmatrix} y_1 \boldsymbol{v}_1' \\ \vdots \\ y_M \boldsymbol{v}_M' \end{bmatrix}.$$

After "learning" the regression weights $\hat{\boldsymbol{x}}$, the classifier for a subsequent test data point $\boldsymbol{v} \in \mathbb{R}^N$ is simply

$$\text{sign}(\boldsymbol{v}' \hat{\boldsymbol{x}}) \,.$$

It is somewhat unconventional to use LS for a data vector $\boldsymbol{y}$ that is binary (± 1 values), whereas **logistic regression** (see Ch. 8) is truly designed for such data. Nevertheless, the LS approach is simple and fast and can work adequately in some cases. This approach will be explored in Discussion.

4.8 Summary

Key points about solving LS problems $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$

- Every minimizer $\hat{\boldsymbol{x}}$ satisfies the **normal equations** $\boldsymbol{A}'\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{A}'\boldsymbol{y}$
- If $\boldsymbol{A}$ has full column rank then $\boldsymbol{A}'\boldsymbol{A}$ is invertible and the unique solution is $\hat{\boldsymbol{x}} = (\boldsymbol{A}'\boldsymbol{A})^{-1}\boldsymbol{A}'\boldsymbol{y}$
- The **minimum-norm LS solution** is always $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{y}$
- If $\boldsymbol{A}$ is **unitary** or a **Parseval tight frame**, then $\boldsymbol{A}^+ = \boldsymbol{A}'$
- The projection of a point $\boldsymbol{y}$ onto $\mathcal{R}(\boldsymbol{A})$ is $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{y} = \boldsymbol{U}_r\boldsymbol{U}_r'\boldsymbol{y}$ when $r > 0$ and is $\boldsymbol{0}$ when $r = 0$.

Bibliography

[1] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005 (cit. on pp. 4.2, 4.3).

[2] T. Adali, P. J. Schreier, and L. L. Scharf. "Complex-valued signal processing: the proper way to deal with impropriety". In: *IEEE Trans. Sig. Proc.* 59.11 (Nov. 2011), 5101–25 (cit. on p. 4.14).

[3] L. Sorber, M. V. Barel, and L. D. Lathauwer. "Unconstrained optimization of real functions in complex variables". In: *SIAM J. Optim.* 22.3 (2012), 879–98 (cit. on p. 4.14).

[4] J. Liu, C. Garcia-Cardona, B. Wohlberg, and W. Yin. "First- and second-order methods for online convolutional dictionary learning". In: *SIAM J. Imaging Sci.* 11.2 (Jan. 2018), 1589–628 (cit. on p. 4.14).

[5] L. Eldén. *Matrix methods in data mining and pattern recognition*. Errata: http://users.mai.liu.se/larel04/matrix-methods/index.html http://users.mai.liu.se/larel04/matrix-methods/. Soc. Indust. Appl. Math., 2007 (cit. on pp. 4.41, 4.44).

[6] L. Mor-Yosef and H. Avron. "Sketching for principal component regression". In: *SIAM J. Matrix. Anal. Appl.* 40.2 (Jan. 2019), 454–85 (cit. on p. 4.43).

[7] Per-AAke Wedin. "Perturbation theory for pseudo-inverses". In: *BIT Numerical Mathematics* 13.2 (June 1973), 217–32 (cit. on p. 4.44).

[8] G. W. Stewart. "On the perturbation of pseudo-inverses, projections and linear least squares problems". In: *SIAM Review* 19.4 (1977), 634–62 (cit. on p. 4.44).

[9] G. W. Stewart. “Stochastic perturbation theory”. In: *SIAM Review* 32.4 (1990), 579–610 (cit. on p. 4.44).

[10] L. Meng and B. Zheng. “The optimal perturbation bounds of the Moore-Penrose inverse under the Frobenius norm”. In: *Linear Algebra and its Applications* 432.4 (Feb. 2010), 956–63 (cit. on p. 4.44).

[11] L. Meng and B. Zheng. “New multiplicative perturbation bounds of the Moore-Penrose inverse”. In: *Linear and Multilinear Algebra* 63.5 (2015), 1037–48 (cit. on p. 4.44).

[12] Per-AAke Wedin. “On the almost rank deficient case of the least squares problem”. In: *BIT Numerical Mathematics* 13.3 (Sept. 1973), 344–54.

[13] S. F. D. Waldron. *An introduction to finite tight frames*. Springer, 2018 (cit. on p. 4.50).

[14] J. Kovacevic and A. Chebira. “Life beyond bases: the advent of frames (Part I)”. In: *IEEE Sig. Proc. Mag.* 24.4 (July 2007), 86–104 (cit. on pp. 4.52, 4.53, 4.55).

[15] J. Kovacevic and A. Chebira. “Life beyond bases: the advent of frames (Part II)”. In: *IEEE Sig. Proc. Mag.* 24.5 (Sept. 2007), 115–25 (cit. on pp. 4.52, 4.53, 4.55).

[16] B. Adcock and D. Huybrechs. “Frames and numerical approximation”. In: *SIAM Review* 61.3 (2019), 443–73 (cit. on pp. 4.52, 4.55).

[17] J. Kovacevic and A. Chebira. “An introduction to frames”. In: *Found. & Trends in Sig. Pro.* 2.1 (2008), 1–94 (cit. on p. 4.55).

[18] A. Rahimi, G. Zandi, and B. Daraby. “Redundancy of fusion frames in Hilbert spaces”. In: *Complex Analysis and Operator Theory* 10.3 (Mar. 2016), 545–65 (cit. on p. 4.55).

[19] R. Yin, T. Gao, Y. M. Lu, and I. Daubechies. “A tale of two bases: local-nonlocal regularization on image patches with convolution framelets”. In: *SIAM J. Imaging Sci.* 10.2 (Jan. 2017), 711–50 (cit. on p. 4.55).

[20] E. Kang and J. C. Ye. “Framelet denoising for low-dose CT using deep learning”. In: *Proc. IEEE Intl. Symp. Biomed. Imag.* 2018, 311–4 (cit. on p. 4.55).

[21] J. C. Ye, Y. Han, and E. Cha. “Deep convolutional framelets: A general deep learning framework for inverse problems”. In: *SIAM J. Imaging Sci.* 11.2 (Jan. 2018), 991–1048 (cit. on p. 4.55).

[22] M. Akcakaya and V. Tarokh. “A frame construction and a universal distortion bound for sparse representations”. In: *IEEE Trans. Sig. Proc.* 56.6 (June 2008), 2443–50 (cit. on p. 4.55).