Optimization, Positive Definiteness, and Least Squares

Graduate Quantitative Economics and Datascience

Jesse Perla

jesse.perla@ubc.ca

University of British Columbia



Table of contents

- Overview
- Optimization and Curvature
- Positive Definite Matrices
- Cholesky and Covariance Matrices
- Least Squares and the Normal Equations
- Collinearity and Identification
- Regularization
- Review Questions



Overview

Motivation

- Least squares is the canonical quadratic optimization problem in datascience, economics, and ML, and our first taste of optimization
- A quadratic has a unique minimum only if its Hessian is positive definite, so we start with curvature and definiteness
- Positive definite matrices also have a Cholesky decomposition, which solves the normal equations and turns a covariance matrix into correlated random draws
- Full column rank gives a unique sample least-squares coefficient, the sample analogue of identification
  - → Collinear or underdetermined data breaks it, and `lstsq` will still return an answer
- Regularization adds curvature to restore a unique solution, at the cost of interpretation



Extra Materials

- scikit-learn ridge regression

Packages

This section uses the following packages:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from numpy.linalg import cond, matrix_rank, norm
from scipy.linalg import solve, eigvalsh
from scipy.linalg import cholesky, solve_triangular
np.random.seed(526)
```

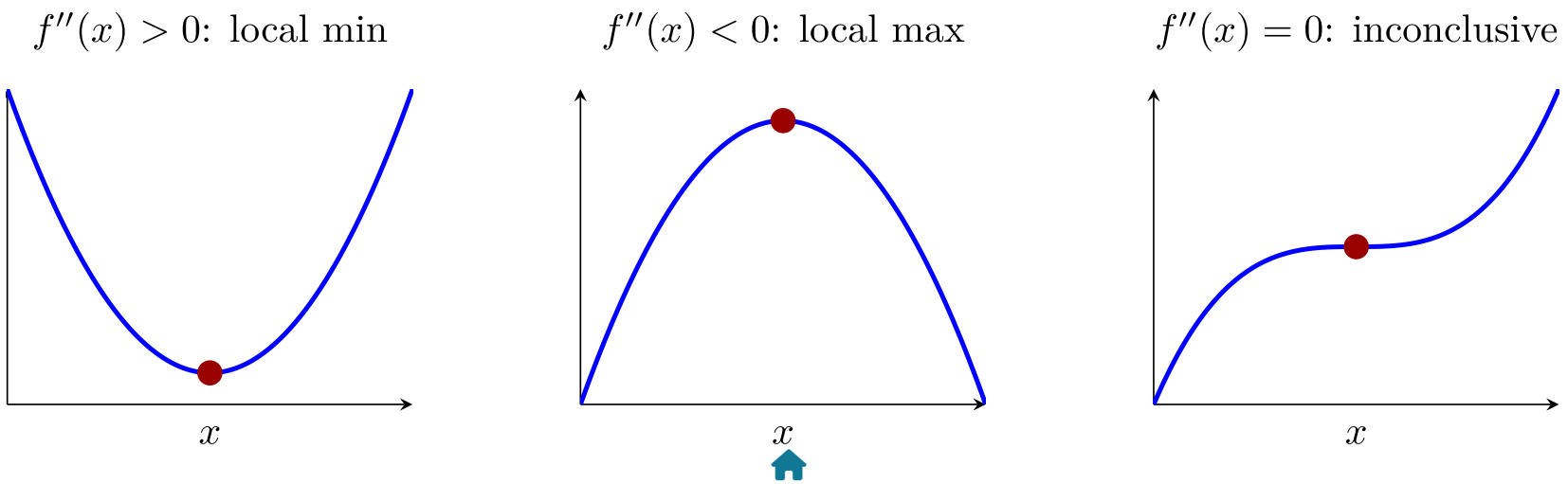


Optimization and Curvature



First and Second Order Conditions in Optimization

- For univariate unconstrained optimization $\min_x f(x)$
- The FONC was $f'(x) = 0$.
  - → But this might not be a valid solution! Or there might be many
- The second order condition gives us more information and provides sufficient conditions
  - → if $f''(x) > 0$, then x is a local minimum; if $f''(x) < 0$, then x is a local maximum.
  - → if $f''(x) = 0$ the test is inconclusive



Related Univariate Concepts

- Recall in your math prep that for a univariate function $f(x)$, we have:
  - → $f(x)$ is **convex** if $f''(x) \geq 0$ for all x in the domain.
  - → $f(x)$ is **concave** if $f''(x) \leq 0$ for all x in the domain.
  - → $f(x)$ is **strictly convex** if $f''(x) > 0$ for all x in the domain.
  - → $f(x)$ is **strictly concave** if $f''(x) < 0$ for all x in the domain.
- We will generalize these concepts for thinking about multivariate functions
  - → Local behavior, x' such that $|x - x'| < \epsilon$, for some ϵ "balls"

Quadratic Functions in Higher Dimensions

- Recall univariate function $f(x) = \frac{a}{2}x^2 + bx + c$ for $x \in \mathbb{R}$.
- General quadratic for $x \in \mathbb{R}^N$ requires cross-terms ($a_{12}x_1x_2, a_{11}x_1^2$ etc.) and linear terms (e.g, b_1x_1, b_2x_2)
- An example form is $f(x) = \frac{1}{2}x^\top Ax + b^\top x + c$
  - → For symmetric matrix A, vector b, and scalar c
  - → Not unique, but this form gives convenient gradients and Hessians

Gradients of Quadratic Forms

- Univariate: $f'(x) \equiv \nabla f(x) = ax + b$ and $f''(x) \equiv \nabla^2 f(x) = a$
- Multivariate: $\nabla f(x) = Ax + b$ and $\nabla^2 f(x) = A$
  - → $\nabla f(x)$ is the gradient vector at x
  - → $\nabla^2 f(x)$ is the Hessian matrix at x
  - → Quadratic forms are special since the Hessian is constant (not x dependent)



Positive Definite Matrices



Definiteness from Vectors

Note

For a symmetric matrix $A \in \mathbb{R}^{N \times N}$, consider $x^\top Ax$ for $x \in \mathbb{R}^N$.

Classification	Definition
Positive definite	$x^\top Ax > 0$ for every $x \neq \mathbf{0}$
Negative definite	$x^\top Ax < 0$ for every $x \neq \mathbf{0}$
Positive semi-definite	$x^\top Ax \geq 0$ for every x
Negative semi-definite	$x^\top Ax \leq 0$ for every x
Indefinite	Positive for some vectors and negative for others

A definite matrix is also semi-definite with the same sign. **Semi-definite but not definite** means equality holds for some nonzero vector.



Positive Definite Matrices

- A symmetric matrix A is positive definite if $x^\top Ax > 0$ for all $x \neq 0$
- Useful in many areas, such as covariance matrices. Example

```
A = np.array([[1, 2],
              [2, 5]])
x = np.array([0, 1]) # can't really check for all x
print(f"x^T A x = {x.T @ A @ x}")
```

```
x^T A x = 5
```

- Example of a symmetric matrix that is not positive definite

```
A = np.array([[1, 2],
              [2, 0]])
print(f"x^T A x = {x.T @ A @ x}")  # one counterexample is enough
```

```
x^T A x = 0
```

- We can check these with eigenvalues



QUESTION Checking Positive Definite with Vectors

```
A = np.array([[1, 2],
              [2, 1]])
x = np.array([1, 1])
print(x.T @ A @ x)  # prints 6
```

Is A positive definite?

A. Yes, since $x^\top Ax = 6 > 0$
B. No, it is negative definite
C. Need more vectors or an alternative method, and might be negative definite
D. Need more vectors or an alternative method, but cannot be negative definite

Using Eigenvalues

- You cannot check $x^\top Ax > 0$ for all x. Instead check whether the "stretching" is positive in every eigenvector direction
- Symmetric A is positive definite iff all eigenvalues are strictly positive, and positive semi-definite iff all are weakly positive
  - → `eigvalsh` is specialized for symmetric matrices and returns real eigenvalues
  - → A positive semi-definite matrix that is not positive definite has a zero eigenvalue, so it is singular

```
A_eigs = eigvalsh(np.array([[1, 2],
                            [2, 5]]))
print(f"eigs = {A_eigs}, pos-def? {np.all(A_eigs > 0)}")
A_eigs = eigvalsh(np.array([[1, 0],
                            [0, 0]]))
print(f"eigs = {A_eigs}, pos-def? {np.all(A_eigs > 0)}, \
pos-semi-def? {np.all(A_eigs >= 0)}")
```

```
eigs = [0.17157288 5.82842712], pos-def? True
eigs = [0. 1.], pos-def? False, pos-semi-def? True
```



QUESTION Checking Positive Definite with Eigenvalues

For a symmetric matrix A, we obtain

```
print(eigvalsh(A))  # prints [1. 2. 5.]
```

How should we classify A?

A. Positive definite
B. Positive semi-definite, but not positive definite
C. Indefinite
D. Singular

Negative Definite Matrices

- Negative definite flips the inequalities: all eigenvalues strictly (weakly) negative for negative (semi-)definite. Think of a concave vs. convex function

```
A = -1 * np.array([[3, -0.1],
                   [-0.1, 2]])
A_eigs = eigvalsh(A)
print(A_eigs)
is_negative_definite = np.all(A_eigs < 0)
is_negative_semi_definite = np.all(A_eigs <= 0)
print(f"neg-def? {is_negative_definite}, "
      f"neg-semi-def? {is_negative_semi_definite}")
```

```
[-3.00990195 -1.99009805]
neg-def? True, neg-semi-def? True
```



Negative Semi-Definite Matrix

- Semi-definite, but not definite requires the matrix to not be full rank
- At least one zero eigenvalue is necessary and sufficient for a matrix to be singular

```
A = np.array([[-1, -1],
              [-1, -1]])
A_eigs = eigvalsh(A)
print(A_eigs)
is_negative_definite = np.all(A_eigs < 0)
is_negative_semi_definite = np.all(A_eigs <= 0)
print(f"neg-def? {is_negative_definite}, "
      f"neg-semi-def? {is_negative_semi_definite}")
```

```
[-2.  0.]
neg-def? False, neg-semi-def? True
```



Indefinite Matrices

- If a matrix has both positive and negative eigenvalues, it is indefinite
- For example,

```
A = np.array([[1, -1],
              [-1, -1]])
A_eigs = eigvalsh(A)
print(A_eigs)
```

```
[-1.41421356  1.41421356]
```



QUESTION Classify from the Eigenvalues

For a symmetric 3×3 matrix A

```
A_eigs = eigvalsh(A)
print(A_eigs)            # [-2.  0.  0.]
print(np.all(A_eigs < 0))
```

Which is true, and what does the last line print?

A. Negative definite, `True`
B. Negative semi-definite and singular, `False`
C. Indefinite, `False`
D. Negative semi-definite and invertible, `True`

QUESTION A Tiny Negative Eigenvalue

A symmetric matrix A has the reported eigenvalues

$$\left[-10^{-17},\ 0.5,\ 1.1\right].$$

How would you categorize A? What if I told you this was solved in `float64`?



Necessary and Sufficient Conditions for Optimization

- The FONC of $\min_x \left\{ \frac{1}{2} x^\top A x + b^\top x + c \right\}$ is
  - → $\nabla f(x^*) = Ax^* + b = \mathbf{0}$
  - → If invertible, then $x^* = -A^{-1}b$, where uniqueness will be related to identification
  - → Same condition for minimization and maximization?
- Need second order sufficient condition in general to check min vs. max vs. neither
  - → $\nabla^2 f(x^*)$ is positive definite for a unique minimum, semi-definite for a minimum with local multiplicity where some directions are "flat"
  - → Replace with negative definite/semi-definite for maxima
  - → If neither, then x^* is neither a minimum nor a maximum (e.g., a saddle point)

Strict Concavity/Convexity

- Univariate $\frac{1}{2}ax^2 + bx + c$: $a > 0$ strictly convex, $a < 0$ strictly concave, $a = 0$ linear
- Multivariate, with A in place of a:
  - → Positive definite: strictly convex. Negative definite: strictly concave
  - → Semi-definite: weakly convex or concave, zero curvature along the eigenvectors with zero eigenvalue
  - → Indefinite: neither



QUESTION Does Positive Semi-Definite Guarantee a Minimum?

Consider the unconstrained problem

$$\min_{x \in \mathbb{R}^2} f(x), \qquad f(x) = \tfrac{1}{2}x_1^2 + x_2.$$

The Hessian has eigenvalues 0 and 1.

What can we conclude?

A. A unique minimum at the origin
B. Every point with $x_1 = 0$ is a minimizer
C. No minimum: the objective is unbounded below
D. The origin is a saddle point



Example of Constructing Quadratic Form

- For $f(x_1, x_2) = -3x_1^2 + x_1x_2 - \frac{1}{2}x_2^2 + x_2 + 5$.
- Let $x \equiv \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and use canonical $f(x) = \frac{1}{2}x^\top Ax + b^\top x + c$.
  - → Then $A \equiv \begin{bmatrix} -6 & 1 \\ 1 & -1 \end{bmatrix}$ (negative definite), $b \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, and $c \equiv 5$.
  - → Gradient: $\nabla f(x) = Ax + b = \begin{bmatrix} -6x_1 + x_2 \\ x_1 - x_2 + 1 \end{bmatrix}$.
  - → Hessian: $\nabla^2 f(x) = A$.

```
eigs(A) = [-6.1925824 -0.8074176]
neg-def? True
```



Indefinite Example

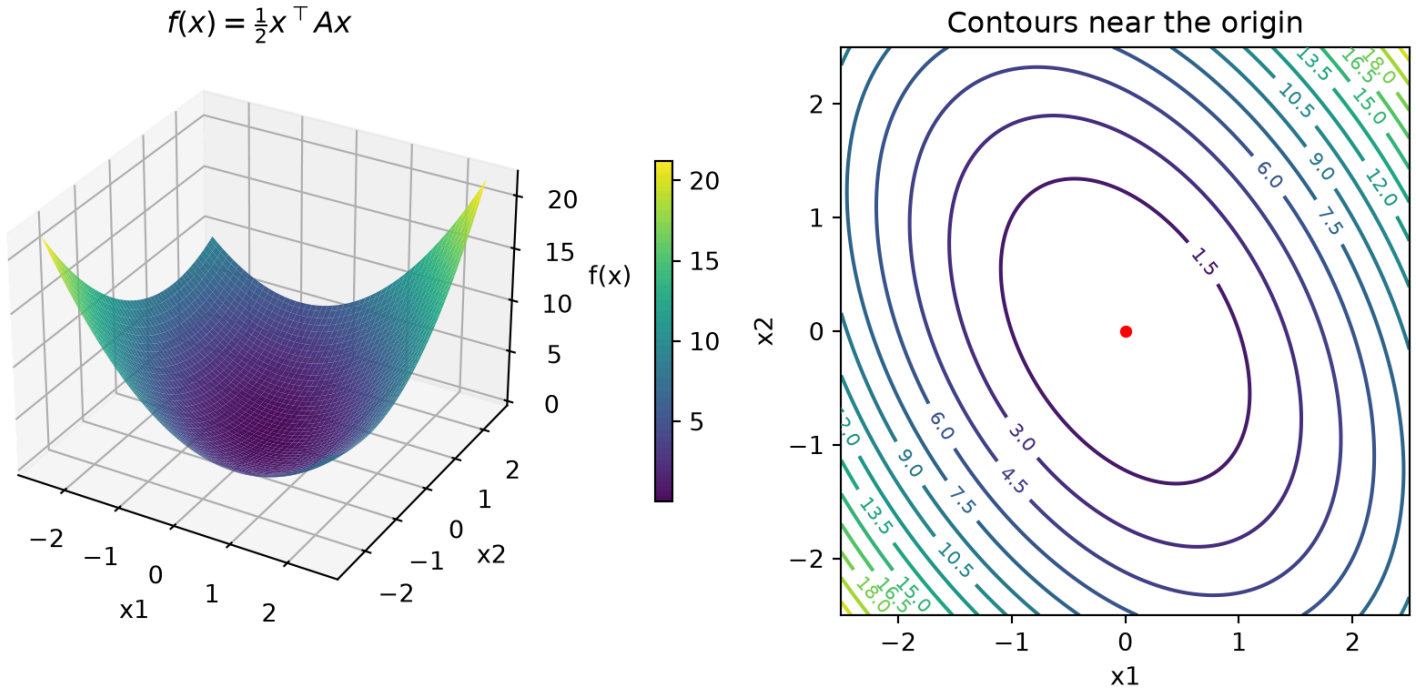
- $f(x_1, x_2) = -3x_1^2 + x_1x_2 + x_2 + 5$
  - → $A = \begin{bmatrix} -6 & 1 \\ 1 & 0 \end{bmatrix}$, $b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, and $c = 5$

```
A = np.array([[-6, 1],
              [1, 0]])
A_eigs = eigvalsh(A)
print(f"eigs(A) = {A_eigs}")
```

```
eigs(A) = [-6.16227766  0.16227766]
```

Shape of Positive Definite Function

- For $A = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}$
- This has a **unique minimum** (at $(0, 0)$, since no "affine" term, b)

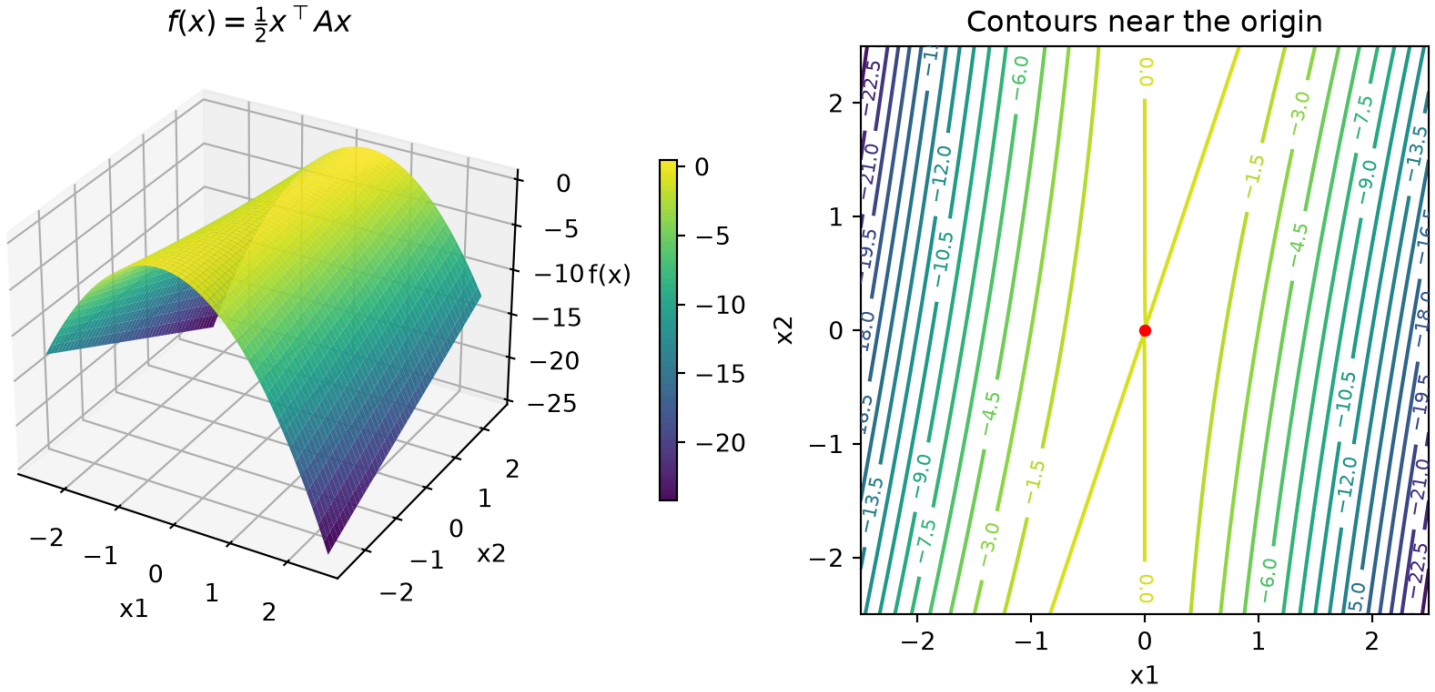


Shape of Negative Semi-Definite Function

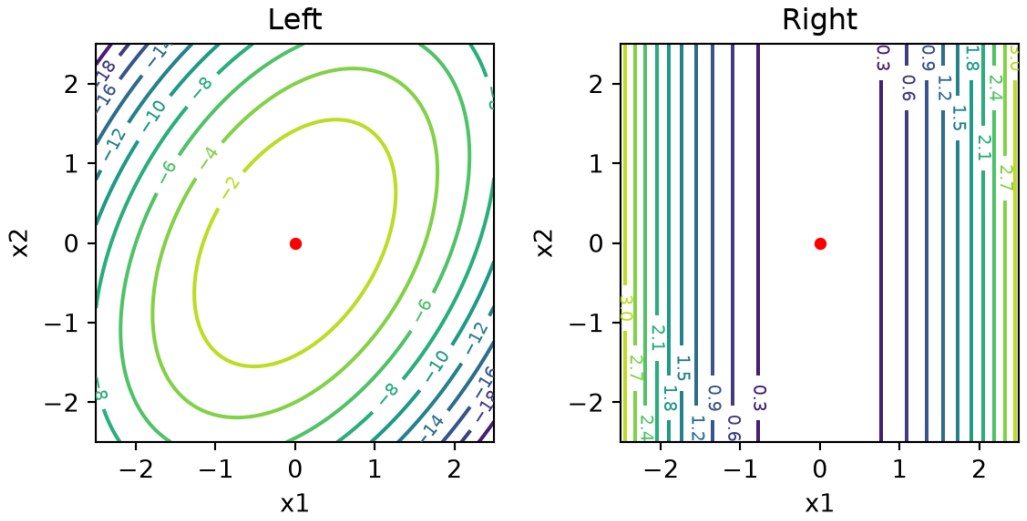
- For our $A = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$
- Note that this **does not** have a unique maximum: every point with $x_1 + x_2 = 0$ is a maximizer
- Maxima rather than minima since negative rather than positive semi-definite

Shape of an Indefinite Function

- For our $A = \begin{bmatrix} -6 & 1 \\ 1 & 0 \end{bmatrix}$
- No maxima or minima since indefinite



QUESTION Classify Eigenvalues from Contours



A. neg def, pos semi-def
B. neg def, pos def
C. neg semi-def, pos semi-def
D. pos def, neg semi-def

Cholesky and Covariance Matrices



Cholesky Decomposition

- A symmetric positive definite matrix has a symmetric version of the LU, with the same triangular factor on both sides
- Called a Cholesky decomposition: $A = LL^\top$ for a lower triangular matrix L.
- Equivalently, could find $A = U^\top U$ for an upper triangular matrix U

```
A = np.array([[1, 2],
              [2, 5]])
L = cholesky(A, lower=True) # lower=False gives the upper factor
print(L)
print(f"L*L^T =\n{L @ L.T}")
```

```
[[1. 0.]
 [2. 1.]]
L*L^T =
[[1. 2.]
 [2. 5.]]
```



Solving Positive Definite Systems

- With $A = LL^\top$, $Ax = LL^\top x = b$ is two triangular systems:
  1. Let $y \equiv L^\top x$ and solve $Ly = b$ by forward substitution.
  2. Solve $L^\top x = y$ by backward substitution.
- `solve` with `assume_a="pos"` does exactly this

```
A = np.array([[1, 2],
              [2, 5]])
b = np.array([1, 4])
print(solve(A, b, assume_a="pos"))

L = cholesky(A, lower=True)
y = solve_triangular(L, b, lower=True)
x = solve_triangular(L.T, y, lower=False)
print(x)
```

```
[-3.  2.]
[-3.  2.]
```



Cholesky for Covariance Matrices

- Covariance matrices are positive-definite, semi-definite if degenerate
- Key property of Gaussian random variables:
  - → $X \sim \mathcal{N}(\mu, \Sigma)$ for $\mu \in \mathbb{R}^N, \Sigma \in \mathbb{R}^{N \times N}$
  - → $X = \mu + AZ$ for $Z \sim \mathcal{N}(\mathbf{0}_N, I_N)$ where $AA^\top = \Sigma$
- One such A is the Cholesky factor of Σ
- Simulate correlated shocks: draw independent ones and multiply by A

Built-in Multivariate Normal

- Check the sample mean and covariance against μ and Σ with a norm

```
mu = np.array([1.0, 2.0])
Sigma = np.array([[1.0, 0.8],
                  [0.8, 2.0]])
N = 10000
X = np.random.multivariate_normal(mu, Sigma, size=N)
print(f"mean =\n{np.mean(X, axis=0)}")
print(f"cov =\n{np.cov(X, rowvar=False)}")
print(f"||mean - mu|| = {norm(np.mean(X, axis=0) - mu):.4f}")
print(f"||cov - Sigma|| = {norm(np.cov(X, rowvar=False) - Sigma):.4f}")
```

```
mean =
[1.00890785 1.99134466]
cov =
[[1.01849699 0.81993007]
 [0.81993007 2.01162319]]
||mean - mu|| = 0.0124
||cov - Sigma|| = 0.0357
```

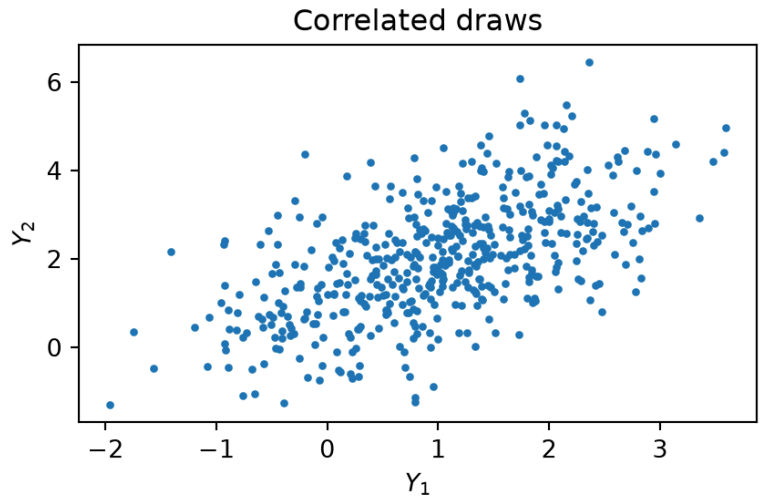


Simulating with the Cholesky

- **Column vectors:** $Z \sim \mathcal{N}(\mathbf{0}, I)$ and $Y = \mu + LZ \sim \mathcal{N}(\mu, \Sigma)$, where $\Sigma = LL^\top$.
- **Row storage:** transpose to get $Y^\top = \mu^\top + Z^\top L^\top$; the code stores one draw per row.

```
L = cholesky(Sigma, lower=True)  # Sigma = L @ L.T
Z = np.random.randn(N, 2)  # one independent draw per row
Y = mu + Z @ L.T  # mu broadcasts across rows
print(f"L =\n{L}")
mean_error = norm(Y.mean(axis=0) - mu)
cov_error = norm(np.cov(Y, rowvar=False) - Sigma)
print(f"Mean error: {mean_error:.4f}")
print(f"Cov error: {cov_error:.4f}")
fig, ax = plt.subplots(figsize=(5, 2.8))
ax.scatter(Y[:500, 0], Y[:500, 1], s=5)
ax.set(xlabel="$Y_1$", ylabel="$Y_2$",
       title="Correlated draws")
plt.show()
```

```
L =
[[1.         0.        ]
 [0.8        1.16619038]]
Mean error: 0.0034
Cov error: 0.0239
```



Why it Works

- With $Y = \mu + LZ$ and $\mathbf{Var}(Z) = I$, we have $\mathbb{E}[Y] = \mu$ and

$$\mathbf{Var}(Y) = L\,\mathbf{Var}(Z)L^\top = LIL^\top = LL^\top = \Sigma$$

- Nothing here required L to be triangular: any A with $AA^\top = \Sigma$ works
  - → The symmetric square root $\Sigma^{1/2} = Q\Lambda^{1/2}Q^\top$ from the eigendecomposition $\Sigma = Q\Lambda Q^\top$ is another choice
  - → The Cholesky is the cheapest to compute and to apply since it is triangular
- If Σ is only positive semi-definite the Cholesky may fail, but a rank-deficient A still exists



Least Squares and the Normal Equations



Least Squares

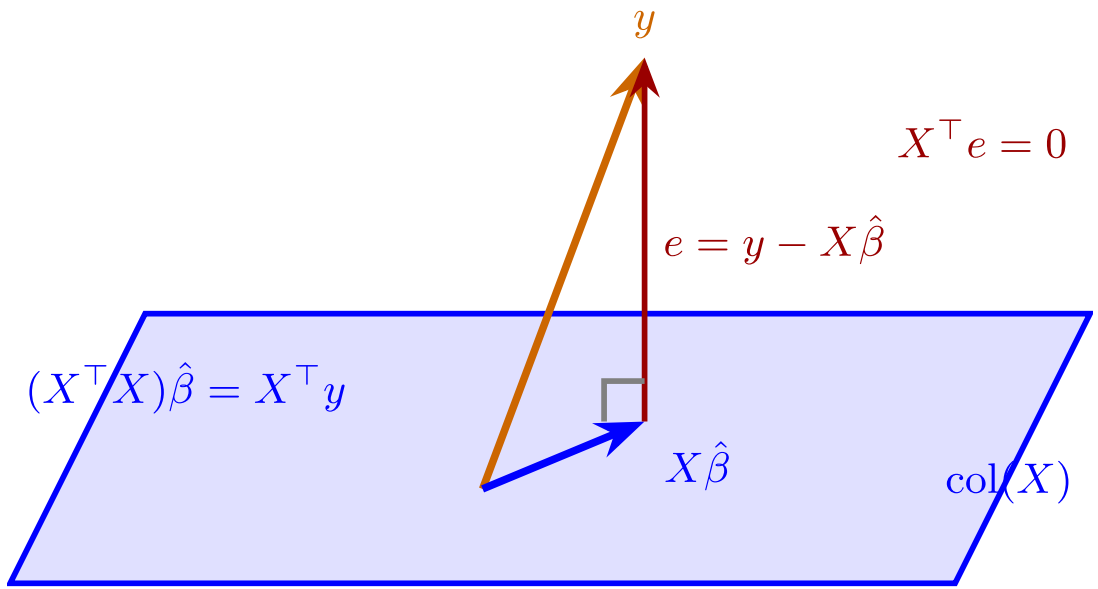
Given a matrix $X \in \mathbb{R}^{N \times M}$ and a vector $y \in \mathbb{R}^N$, we want to find $\beta \in \mathbb{R}^M$ such that

$$\min_{\beta} ||y - X\beta||^2, \text{ that is,}$$

$$\min_{\beta} \sum_{n=1}^{N} (y_n - X_n \cdot \beta)^2$$

Where X_n is the nth row. Take FOCs and rearrange to get

$$(X^\top X)\beta = X^\top y$$



Solving the Normal Equations

- The X is often referred to as the "design matrix" and $X^\top X$ as the Gram matrix of the features (columns)
- Can form $A = X^\top X$ and $b = X^\top y$ and solve $A\beta = b$.
  - → Or invert $X^\top X$ to get

$$\beta = (X^\top X)^{-1} X^\top y$$

  - → Note that $X^\top X$ is symmetric and, if X has full column rank ($\mathbf{rank}(X) = M$), positive definite
- The Hessian of $||y - X\beta||_2^2$ is $2X^\top X$: positive definite when $\mathbf{rank}(X) = M$, so the normal equations give the unique minimum

QUESTION What Must Be True of the Residuals?

Let X have full column rank, and let $\hat{\beta}$ solve

$$\min_{\beta} ||y - X\beta||_2^2.$$

with FONC: $(X^\top X)\hat{\beta} = X^\top y$.

For $e = y - X\hat{\beta}$, which statement **must** hold?

A. $e = \mathbf{0}$
B. $y^\top e = 0$
C. The residuals sum to zero, whether or not an intercept is included
D. $X^\top e = \mathbf{0}$: the residual vector is orthogonal to every column of X

Solving Regression Models in Practice

- `lstsq` uses better algorithms (more numerically stable, same "order", possibly slightly slower)
- One algorithm uses another factoring, the QR decomposition: $X = QR$ for Q orthogonal and R upper triangular. See QR Decomposition
  - → LU decomposition is Gaussian elimination, QR decomposition is Gram-Schmidt orthogonalization
- Stata also uses a QR decomposition under the hood.
- For applied work, use Stata, R, or `statsmodels` (integrates with `pandas` and `seaborn`)
  - → See statsmodels docs for R-style notation and QuantEcon OLS Notes
- For well conditioned problems (i.e., $\mathbf{cond}(X^\top X)$ small), a Cholesky decomposition on $X^\top X$ then solving $(X^\top X)\beta = X^\top y$ may be faster but less numerically stable



Example of LLS using Scipy

- `lstsq`, or the normal equations with `assume_a="pos"` so it can use a Cholesky

```
N, M = 100, 5
X = np.random.randn(N, M)
beta = np.random.randn(M)
y = X @ beta + 0.05 * np.random.randn(N)
beta_hat, residuals, rank, s = scipy.linalg.lstsq(X, y)
beta_hat_normal = solve(X.T @ X, X.T @ y, assume_a="pos")
print(f"beta = {beta}\nlstsq = {beta_hat}\nnormal = {beta_hat_normal}")
```

```
beta = [-2.10378865  1.08556707 -2.78329393 -0.68383148  2.16714994]
lstsq = [-2.10292981  1.08372188 -2.77870538 -0.67543023  2.17412533]
normal = [-2.10292981  1.08372188 -2.77870538 -0.67543023  2.17412533]
```

Collinearity and Identification



Collinearity in “Tall” Matrices

- Tall $\mathbb{R}^{N \times M}$ “design matrices” have $N > M$ and are “overdetermined”
- The rank of a matrix is full rank if all columns are linearly independent
- You can only uniquely determine M parameters from this sample with M linearly independent columns

```
X = np.array([[1, 2],
              [2, 5],
              [3, 7]]) # 3 observations, 2 variables
X_col = np.array([[1, 2],
                  [2, 4],
                  [3, 6]]) # all proportional
print(f"rank(X) = {matrix_rank(X)}, rank(X_col) = {matrix_rank(X_col)}")
```

```
rank(X) = 2, rank(X_col) = 1
```

Collinearity and Estimation

- Forming the Gram matrix squares the condition number: $\mathbf{cond}(X^\top X) = \mathbf{cond}\,(X)^2$
- If X is not full rank, then $X^\top X$ is not invertible. For example:

```
print(f"cond(X) = {cond(X):.1f}, cond(X'X) = {cond(X.T@X):.1f}")
print(f"cond(X_col'X_col) = {cond(X_col.T@X_col):.3g}")
```

```
cond(X) = 53.1, cond(X'X) = 2819.3
cond(X_col'X_col) = 1.1e+16
```

- Note that when you start doing operations on matrices, numerical error creeps in, so you will not get an exact number
- The rule-of-thumb with condition numbers is that if it is 1×10^k then you lose about k digits of precision.
- Given the singular matrix, this means a continuum of β will solve the problem



`lstsq` Solves it? Careful on Interpretation!

- Since $X_{\text{col}}^{\top} X_{\text{col}}$ is singular, we cannot use `solve(X_col.T@X_col, X_col.T@y)`
- The key is that in the case of non-full rank, you cannot uniquely determine individual parameters from this sample

  → Having low residuals is not enough to conclude anything about identification of parameters

```
y = np.array([5.0, 10.1, 14.9])
beta_hat, residuals, rank, s = scipy.linalg.lstsq(X_col, y)
print(f"beta_hat_col = {beta_hat}")
print(f"rank={rank}, cols={X_col.shape[1]}")
print(f"norm(X_col @ beta_hat - y)={norm(X_col @ beta_hat - y)}")
```

```
beta_hat_col = [0.99857143 1.99714286]
rank=1, cols=2
norm(X_col @ beta_hat - y)=0.13887301496588172
```

Numerical Rank is a Judgement Call

- `lstsq` reports the number of singular values above a cutoff relative to the largest
- A collinear column gives a singular value that is rounding noise, not exactly zero, and the matching eigenvalue of $X^\top X$ can even come out negative

```
print(f"singular values of X_col = {s}")
print(f"eigenvalues of X_col'X_col = {eigvalsh(X_col.T @ X_col)[::-1]}")
```

```
singular values of X_col = [8.36660027e+00 8.88178420e-16]
eigenvalues of X_col'X_col = [70.  0.]
```

- With random data the noise can land on either side of the cutoff: the same collinear design then reports full rank on some draws, and the returned eta has entries of order 10^{12} or more
- Always look at the singular values or eigenvalues, never just `rank`, before trusting a coefficient vector



Fat Design Matrices

- Fat $\mathbb{R}^{N \times M}$ "design matrices" have $N < M$ and are "underdetermined"
- Less common in econometrics, but useful to understand the structure
- A continuum of $\beta \in \mathbb{R}^M$ solve this problem: an affine subspace of dimension $M - \text{rank}(X)$

```
X = np.array([[1, 2, 3],
              [0, 5, 7]]) # 2 rows, 3 variables
y = np.array([5, 10])
beta_hat, residuals, rank, s = scipy.linalg.lstsq(X, y)
print(f"beta_hat = {beta_hat}, rank={rank}")
print(f"norm(X @ beta_hat - y) = {norm(X @ beta_hat - y):.2g}")
```

```
beta_hat = [0.8 0.6 1. ], rank=2
norm(X @ beta_hat - y) = 4e-15
```

QUESTION How Many Minimizers?

For each shape of $X \in \mathbb{R}^{N \times M}$, does $\min_\beta \|y - X\beta\|_2^2$ have a unique minimizer?

	5×3, **rank 3**	3×5, **rank 3**	5×3, **rank 2**
A.	unique	continuum	continuum
B.	unique	unique	continuum
C.	unique	continuum	unique
D.	continuum	continuum	continuum

Which Solution?

- Residuals are zero here because $\mathbf{rank}(X) = N$: enough parameters to fit perfectly
- Given the multiple solutions, the `lstsq` is giving

$$\min_{\beta} ||\beta||_2^2 \text{ s.t. } X\beta = y$$

- i.e., the “smallest” coefficients which interpolate the data exactly
- With enough parameters to interpolate (X, y), OLS achieves zero residuals.



QUESTION Which Other β Fits Exactly?

```
X = np.array([[1, 2, 3],
              [0, 5, 7]])
y = np.array([5, 10])
beta_hat, *_ = scipy.linalg.lstsq(X, y)
print(np.allclose(X @ beta_hat, y))  # prints True
z_hat = np.array([1.0, 7.0, -5.0])
print(X @ z_hat)          # prints [0. 0.]
```

A. $\hat{\beta}$ and $\hat{z}$

B. $\hat{\beta}$ only

C. $\hat{\beta} + c\hat{z}$ for all $c \in \mathbb{R}$

Which describes the solutions to $X\beta = y$?



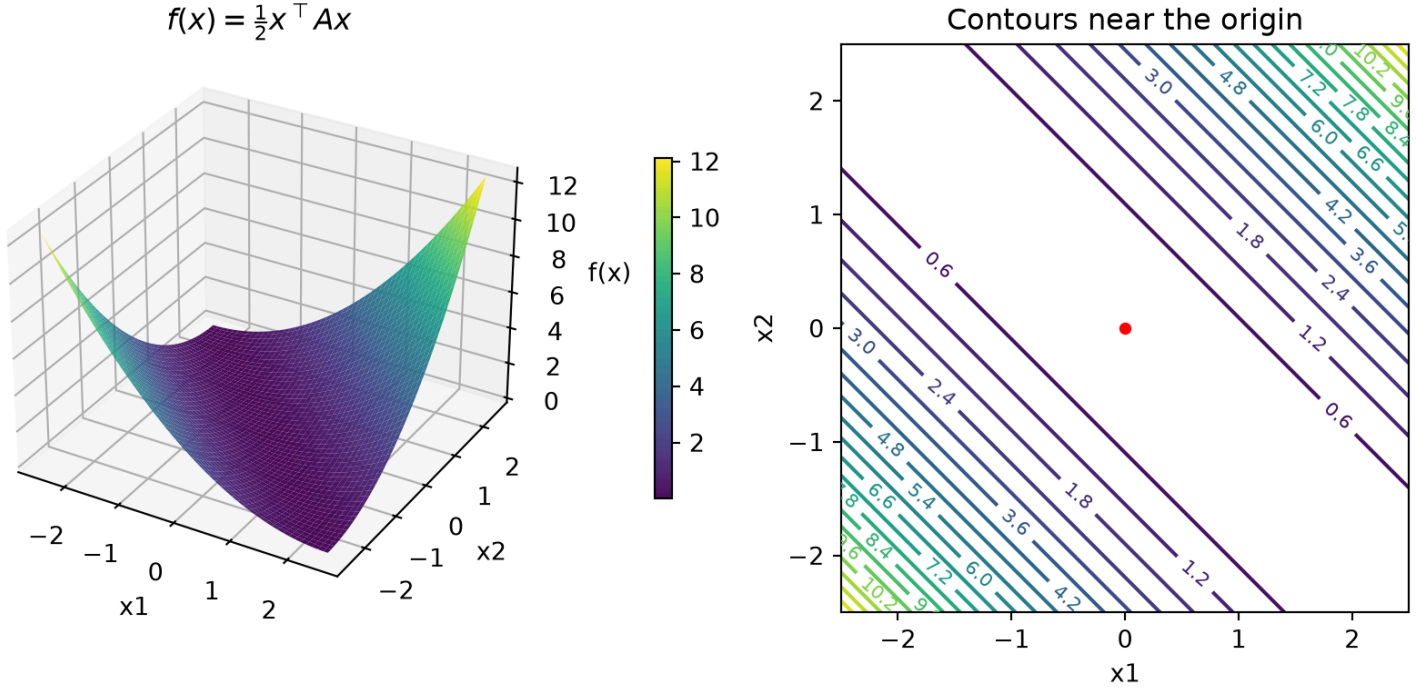
Regularization



Recall a Positive Semi-Definite Function

- Flip the sign of the negative semi-definite shape from earlier in this lecture: $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ is positive semi-definite with multiple minima along a line!

```
eigenvalues of A: [0. 2.]
```



Fudge the Diagonal?

- Replace with $A = A + \lambda I$ for λ very small (e.g., 10^{-5})
- Now a unique minimum at $(0, 0)$

```
eigenvalues of A: [1.00000e-05 2.00001e+00]
```



Motivating this Fudge

- Previously solved $\min_x \left\{\frac{1}{2}x^\top Ax\right\}$, which only has a unique solution if A is positive definite.
- Replace with

$$\min_x \left\{\frac{1}{2}x^\top Ax + \frac{\lambda}{2}||x||_2^2\right\}$$

  → i.e., penalize solutions by the euclidean length of x, called a “ridge” term

  → Could instead penalize by different norms, e.g. $||x||_1$ is called LASSO

- What are the first order conditions? Let’s look at least squares

Ridge Regression

- More generally, for OLS think of the following

$$\min_{\beta} \left\{ ||y - X\beta||_2^2 + \lambda ||\beta||_2^2 \right\}$$

- Take the FOCs and rearrange to get

$$(X^\top X + \lambda I)\beta = X^\top y$$

→ Note: if $X^\top X$ is not full rank (i.e., has a zero eigenvalue) then the addition of the λ term helps make things strictly positive definite

→ Sometimes you need to do this to overcome nearly collinear data or numerical approximations, even when it should be technically positive definite

QUESTION What Did Ridge Actually Fix?

Two columns of X are identical. With $\lambda > 0$, we estimate

$$\hat{\beta}_\lambda = \arg\min_\beta \left\{ \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right\}.$$

Which statement is correct?

A. A unique penalized estimate, but the data still cannot distinguish the columns' separate effects

B. The data now identify both coefficients separately

C. Ridge guarantees zero training residuals

D. Ridge guarantees better predictions on new data

Ridgeless Regression (Optional)

- The min-norm solution `lstsq` returns for a fat X, $\min_\beta \|\beta\|_2^2 \text{ s.t. } X\beta = y$, is the limit of ridge regression (i.e., "ridgeless")

$$\lim_{\lambda \to 0} \arg\min_\beta \left\{ \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right\}$$

Regularization in ML (Optional)

- In ML, with rich data sources there are often many possible ways to explain the data
- Economists often avoid this like the plague, and make assumptions to ensure perfect identification
  - → Identification arguments ensure positive definiteness of OLS, etc.
- As data becomes richer, it becomes hard to write down models with only a single explanation
  - → Regularization lets you bias your solution towards ones with certain properties
- There are Bayesian interpretations of all of these approaches



Review **Questions**



QUESTION Minimum, Maximum, or Saddle?

$f(x) = x_1^2 + x_2^2 + 4x_1x_2$ has $\nabla f(x) = \mathbf{0}$ only at $x^* = \mathbf{0}$.

```
H = np.array([[2, 4],
              [4, 2]])  # Hessian of f
print(eigvalsh(H))  # prints [-2.  6.]
```

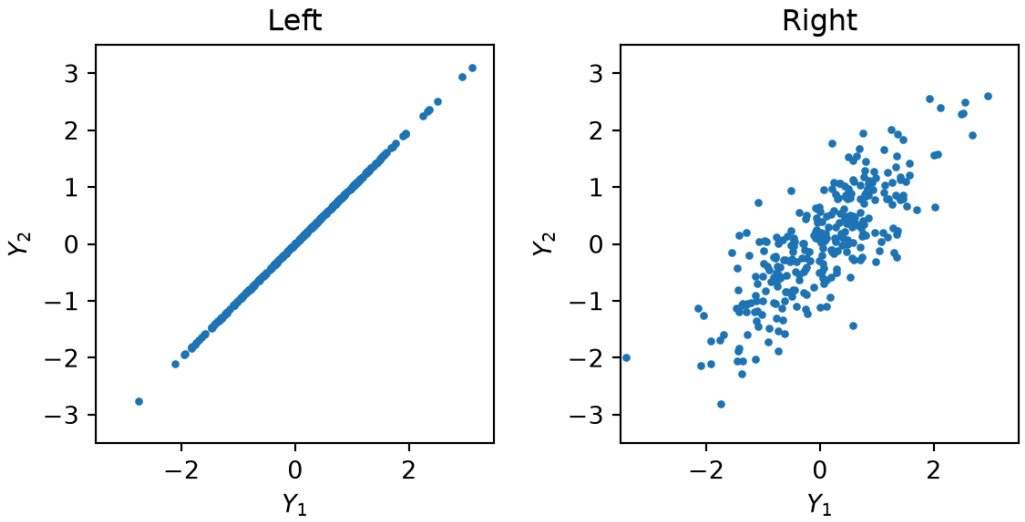
What kind of point is x^*?

A. A unique minimum
B. A unique maximum
C. A saddle point
D. A minimum with a flat direction



QUESTION Which Covariance Matrix?

Draws from $\mathcal{N}(\mathbf{0}, \Sigma_i)$ for $i = 1, 2$



$$\Sigma_1 = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

A. Left Σ_1, Right Σ_2
B. Left Σ_2, Right Σ_1
C. Both Σ_1, different seeds
D. Neither can be Σ_2

QUESTION Rank-Deficient Normal Equations

```
X = np.array([[1, 2, 2],
              [2, 5, 4],
              [3, 7, 6]])  # last column is twice the first
print(matrix_rank(X))       # prints 2
```

Which is true of $X^\top X$?

A. Positive definite, so `solve(X.T @ X, X.T @ y, assume_a="pos")` works

B. Positive semi-definite and singular; `lstsq` still returns the minimum-norm β

C. Indefinite

D. Negative semi-definite

QUESTION Ridge Shifts the Eigenvalues

```
X = np.array([[1, 2, 2],
              [2, 5, 4],
              [3, 7, 6]])  # last column is twice the first
print(eigvalsh(X.T @ X))   # approx [0.00, 0.10, 147.90]
print(eigvalsh(X.T @ X + 0.1 * np.eye(3)))
```

What are the new eigenvalues (rounded to two decimals), and is $(X^\top X + 0.1I)\beta = X^\top y$ uniquely solvable?

A. `[0.10, 0.20, 148.00]`, positive definite

B. `[0.00, 0.20, 148.00]`, still singular

C. `[0.10, 0.10, 147.90]`, positive definite

D. Unchanged: adding λI only changes the eigenvectors

QUESTION Perfect Fit and Identification

```
X = np.array([[1, 2, 3],
              [0, 5, 7]])
y = np.array([5, 10])
beta_hat, *_ = scipy.linalg.lstsq(X, y)
print(np.allclose(X @ beta_hat, y))  # prints True
```

The fit is exact. What can we conclude?

A. `beta_hat` is the unique OLS estimate: a perfect fit identifies it

B. A line of β fits exactly; `lstsq` returned the one with the smallest norm

C. $X^\top X$ is positive definite, since the fit is perfect

D. Adding more regressors would improve identification