Clase 020 — NumPy: álgebra lineal con numpy.linalg

Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 2 § 2.9 Structured Arrays (referencia) · Numerical Linear Algebra (Trefethen & Bau). · Duración estimada: 90 min.

Clase 020 — NumPy: álgebra lineal con numpy.linalg

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 2 § 2.9 Structured Arrays (referencia) · Numerical Linear Algebra (Trefethen & Bau). · Duración estimada: 90 min.*

Objetivo

Que el alumno opere con vectores y matrices al nivel necesario para entender ML: producto punto, multiplicación matricial, inversa, sistema de ecuaciones (solve), descomposiciones (SVD, eigen). Saber cuándo no usar la inversa (lentitud + inestabilidad numérica).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Multiplicar vectores y matrices con @ (operador moderno) y np.dot.
2. Resolver sistemas Ax = b con np.linalg.solve (NO con inv(A) @ b).
3. Calcular norma, determinante, rango, traza.
4. Computar SVD con np.linalg.svd y entender qué retorna.
5. Calcular eigenvalores/eigenvectores con np.linalg.eig / eigh (simétrica).

Temas

#	Tema	Por qué importa
1	@ operador (PEP 465): multiplicación matri	Reemplaza np.matmul.
2	Producto punto vs producto matricial	Vector·vector vs matriz·matriz.
3	Resolver sistemas: solve vs inv	Por qué NUNCA usar inv.
4	Norma, det, rank, trace	Diagnóstico estructural de matrices.
5	SVD — la factorización universal	Base de PCA, regresión lineal, recomendado
6	Eigen	Base de PCA conceptual.

Definiciones y características

Operador @

: Multiplicación matricial (PEP 465). A @ B ≡ np.matmul(A, B) ≡ A.dot(B). NO confundir con * (elementwise). Disponible Python 3.5+.

Producto punto vs producto matricial

: Punto (vector·vector → escalar): a @ b = sum(a*b). Matricial (matriz @ matriz → matriz): regla "fila por columna". Shapes: (m,n) @ (n,p) → (m,p).

np.linalg.solve(A, b)

: Resuelve Ax = b usando descomposición LU. Siempre preferible a inv(A) @ b: más rápido (no construye inversa), más estable numéricamente, menos memoria.

SVD (Singular Value Decomposition)

: Descomposición universal: $M = U \cdot \Sigma \cdot V^T$. U y V son ortogonales; Σ diagonal con valores singulares decrecientes ≥ 0. Base de PCA, recomendadores (matrix factorization), compresión, pseudo-inversa.

Eigen (eigenvalues/eigenvectors)

: Para A cuadrada, $A\,v = \lambda\,v$. eig general; eigh para matrices simétricas (más rápido, garantiza eigenvalores reales). Base de PCA conceptual.

Número de condición (cond)

: Ratio entre el valor singular más grande y el más pequeño. Mide sensibilidad de la solución a perturbaciones. cond > 1e10 matriz mal condicionada, solve perderá precisión.

Dataset / recursos

Sintético: matrices y vectores para los ejercicios. Sin descarga.

Ejercicios

1. Producto punto. Dados dos vectores 100-dim aleatorios, calcula np.dot(a, b) y verifica que coincide con sum(a*b).

2. Multiplicación matricial. (50, 30) @ (30, 20) → (50, 20). Verifica shapes y un elemento manualmente.

3. Resuelve sistema. Genera A = (5,5) aleatoria, b = (5,), resuelve Ax = b con solve. Verifica A @ x ≈ b.

4. Inv vs solve benchmark. Para A (1000,1000) y b (1000,), mide tiempo de inv(A) @ b vs solve(A, b). Reporta speedup.

5. SVD de matriz baja rank. Crea M = u @ v.T (rank 1). Calcula SVD y observa que solo el primer valor singular es no-cero.

Homework verificable

Notebook que: (a) compara inv(A) @ b vs solve(A, b) en tiempo Y precisión (np.allclose); (b) implementa regresión lineal cerrada $\beta = (X^TX)^{-1} X^Ty$ y luego con solve; (c) calcula SVD de una matriz y verifica M = U @ diag(s) @ Vt; (d) eigen de matriz de covarianza.

Criterio de aceptación: solve más rápido y más preciso que inv. SVD reconstruye la matriz dentro de tolerancia.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
A @ B falla con ValueError: shapes ... not	Las dimensiones internas no coinciden: (m,
Implementé inv(A) @ b y los resultados son	inv es inestable numéricamente para matric
np.linalg.solve lanza LinAlgError: Singula	det(A) ≈ 0 — el sistema no tiene solución
A * B da resultado raro y esperaba A @ B	Operador * es elementwise (Hadamard produc
SVD devuelve Vt no V	Por convención NumPy devuelve V^T (transpu

Preguntas frecuentes

¿@, np.matmul, o np.dot?

Para matriz×matriz son equivalentes — @ es el más legible. np.dot tiene comportamiento distinto para arrays >2D (no broadcasting); @ y matmul sí. Usa @ siempre que puedas.

¿eig o eigh?

eigh si la matriz es simétrica (covarianza, kernel matrices, métrica de distancias). Más rápido, garantiza eigenvalores reales. eig para matrices generales (no simétricas) — puede dar valores complejos.

¿Por qué np.linalg.det para chequear singularidad es mala idea?

El determinante es 0 o no-0 sin gradiente útil — para matrices grandes, det puede ser tan chico/grande que cause underflow/overflow numérico. Mejor: np.linalg.cond(A) — si > 1e10, problemática.

¿Cuándo necesito BLAS/LAPACK?

NumPy ya los usa por debajo (vía OpenBLAS o MKL). Si tu np.linalg.solve parece lento, instala MKL (pip install mkl) o usa la build de conda con mkl.

¿GPU para álgebra lineal?

CuPy (drop-in replacement de NumPy con CUDA), PyTorch tensors (.to('cuda')), o JAX. Para matrices >1000×1000 la GPU vale la pena.

Referencias

- VanderPlas cap. 2 (overview NumPy).
- numpy.linalg reference
- PEP 465 — @ operator
- Trefethen & Bau, Numerical Linear Algebra (1997) — fondo matemático.

Siguiente clase

Clase 021 — NumPy: aleatoriedad y semillas

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import numpy as np
import time
rng = np.random.default_rng(42)
```

Archivos complementarios

- notebook.ipynb