Matemáticas para la Inteligencia Artificial Agroambiental

Vectores, Matrices y Tensores con aplicaciones en Python

Autor:
John Jairo Leal Gómez

Universidad Nacional de Colombia
30 de diciembre de 2025

Índice general

Capítulo 1

El Motor de la IA: De Escalares a Tensores

En este capítulo se presentan los conceptos preliminares de álgebra lineal necesarios para el desarrollo de modelos de inteligencia artificial en contextos agro-ambientales, industriales y administrativos.

1.1. Estructuras de datos: escalares, vectores, matrices y tensores

1.1.1. Escalar (Rango 0)

Un **escalar** es un número real que representa una cantidad simple, sin dirección ni estructura interna. En ciencia de datos, los escalares suelen ser hiperparámetros, métricas de desempeño o mediciones puntuales.

Ejemplos:

- La tasa de aprendizaje en un modelo de predicción de rendimiento: $\eta = 0{,}01$,
- El error cuadrático medio (MSE) de un modelo de predicción de peso animal: MSE $= 4{,}3$,
- La humedad del suelo en un punto específico: $28{,}5\,\%$,
- La concentración horaria de $PM_{2.5}$ en una estación ambiental: $32{,}7\ \mu g/m^3$,
- El presupuesto total asignado a un programa de sostenibilidad: 1,25 millones de pesos.

1.1.2. Vector (Rango 1)

Un **vector** es una lista ordenada de números que describe un objeto o fenómeno mediante múltiples atributos medibles. Cada componente corresponde a una variable relevante en el contexto de análisis.

Ejemplos:

- **Agronómico:** $\mathbf{x} = (120, 50, 3, 6{,}2)^\top \rightarrow$ (N, P, K en kg/ha; pH del suelo)[1],
- **Zootécnico:** $\mathbf{x} = (650, 3{,}8, 42, 18)^\top \rightarrow$ (peso en kg, producción lechera en L/día, % grasa, edad en meses),
- **Agroindustrial:** $\mathbf{x} = (85, 120, 0{,}45)^\top \rightarrow$ (temperatura del horno en °C, tiempo de cocción en min, humedad final del producto),

[1] $\mathbf{x} = (120, 50, 3, 6{,}2)^\top$ Significa que el vector es columna, y lo hemos escrito como fila

- **Ambiental:** $\mathbf{x} = (45, 28, 32, 65, 1{,}2)^\top \rightarrow$ (PM_{10}, $PM_{2.5}$, NO_2, O_3, CO en $\mu g/m^3$),
- **Administrativo:** $\mathbf{x} = (320, 95, 210, 75)^\top \rightarrow$ (presupuesto en millones de pesos para: infraestructura, capacitación, operación, monitoreo ambiental).

Los vectores son la unidad básica de representación en modelos de aprendizaje automático, ya que permiten tratar cada observación como un punto en un espacio multidimensional.

1.1.3. Matriz (Rango 2)

Una **matriz** es un arreglo rectangular de números reales ordenados en filas y columnas. Si bien visualmente se asemeja a una tabla, en álgebra lineal aplicada y ciencia de datos posee una **dualidad fundamental**:

1. **Como estructura de datos (Estática):** Es un contenedor donde las filas suelen representar observaciones (ej. pacientes, plantas, transacciones) y las columnas variables (ej. edad, altura, costo).
2. **Como operador (Dinámica):** Representa una *transformación lineal* que actúa sobre vectores, capaz de rotar, escalar o proyectar datos en el espacio geométrico.

Definición formal Una matriz $\mathbf{A}$ de dimensiones $m \times n$ es un elemento del espacio vectorial $\mathbb{R}^{m \times n}$. Se denota explícitamente como:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}_{m \times n} \in \mathbb{R}^{m \times n}.$$

Donde:

- m es el número de **filas** (eje 0 en librerías como NumPy o PyTorch).
- n es el número de **columnas** (eje 1).
- $a_{ij} \in \mathbb{R}$ es la entrada escalar ubicada en la fila i y columna j.

Descomposición en Vectores: Filas vs. Columnas

Para entender las operaciones matriciales en IA, es crucial no ver la matriz como un bloque sólido, sino como una colección de vectores.

1. Visión por Columnas (Espacio de Características) Podemos ver a $\mathbf{A}$ como una colección de n vectores columna verticales. Esta visión es útil en álgebra lineal para entender conceptos como *independencia lineal* o *bases*.

$$\mathbf{A} = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}, \quad \text{donde } \mathbf{c}_j \in \mathbb{R}^m.$$

2. Visión por Filas (Espacio de Muestras) Podemos ver a $\mathbf{A}$ como una pila de m vectores fila horizontales. Esta es la visión estándar en **Data Science**, donde cada fila es un objeto de estudio.

$$\mathbf{A} = \begin{pmatrix} — & \mathbf{r}_1 & — \\ — & \mathbf{r}_2 & — \\ & \vdots & \\ — & \mathbf{r}_m & — \end{pmatrix}, \quad \text{donde } \mathbf{r}_i \in \mathbb{R}^{1\times n}.$$

La Matriz como Dataset: Ejemplo Agronómico

Considere un estudio de suelos con 100 muestras ($m = 100$) y 5 variables químicas ($n = 5$). La matriz de datos $\mathbf{X} \in \mathbb{R}^{100\times 5}$ se estructura de la siguiente forma, donde cada fila es una"foto química" de una parcela distinta:

$\mathbf{X} =$

	N	P	K	pH	M.O.
Muestra 1	110	45	2,8	6,1	3,5
Muestra 2	130	55	3,1	6,3	4,1
⋮	⋮	⋮	⋮	⋮	⋮
Muestra 100	100	50	2,9	5,9	3,2

En este contexto, el álgebra lineal nos permite operar sobre todo el conjunto de datos simultáneamente (por ejemplo, normalizar la columna del pH para todas las muestras a la vez) mediante una técnica computacional conocida como **vectorización**, evitando el uso de bucles lentos.

1.1.4. Tensor: Rango k, Generalización Multidimensional

Un **tensor** es la estructura de datos fundamental en inteligencia artificial. Matemáticamente, representa una generalización de los conceptos de escalar, vector y matriz a un espacio de K dimensiones (denominadas *modos* o *ejes*).

El **rango** (u *orden*) de un tensor es el número de índices necesarios para localizar un elemento de forma unívoca. En términos computacionales, esto coincide con el número de dimensiones del arreglo (propiedad `.ndim` en Python).

Jerarquía de Tensores por su Rango:

- **Rango 0 (Escalar):** Un único número $s \in \mathbb{R}$. Representa una magnitud puntual (ej. la temperatura de un invernadero).
- **Rango 1 (Vector):** Una lista ordenada $\mathbf{v} \in \mathbb{R}^{n_1}$. Representa una línea de datos o un perfil de atributos (ej. parámetros de suelo N, P, K).
- **Rango 2 (Matriz):** Una rejilla bidimensional $\mathbf{M} \in \mathbb{R}^{n_1\times n_2}$. Estructura estándar para bases de datos tabulares (filas por columnas).
- **Rango 3 (Tensor de 3er orden):** Un "cubo" de números $\mathcal{T} \in \mathbb{R}^{n_1\times n_2\times n_3}$. Común en imágenes a color (Alto × Ancho × Canales RGB) como en la figura **??**.
- **Rango 4 (Tensor de 4to orden):** Un conjunto de cubos $\mathcal{T} \in \mathbb{R}^{n_1\times n_2\times n_3\times n_4}$. Estructura típica para *batches* (lotes) de imágenes en redes neuronales o series temporales de mapas satelitales.

Definición formal. Un tensor de orden K se define como un elemento de un espacio producto de K dimensiones:

$$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_K}$$

donde cada d_i representa la cardinalidad (tamaño) del i-ésimo eje. Para acceder a una entrada específica, se requiere una tupla de K índices:

$$t_{i_1,i_2,\ldots,i_K} = \mathcal{T}(i_1, i_2, \ldots, i_K)$$

Ingeniería Ambiental y Teledetección

El uso de tensores de rango 4 permite modelar la **dispersión espaciotemporal** de contaminantes. Los ejes representan: (1) Tiempo, (2) Latitud, (3) Longitud y (4) Tipo de contaminante ($PM_{2.5}, O_3$, etc.). Esta estructura es la entrada para redes neuronales recurrentes-convolucionales (ConvLSTM).

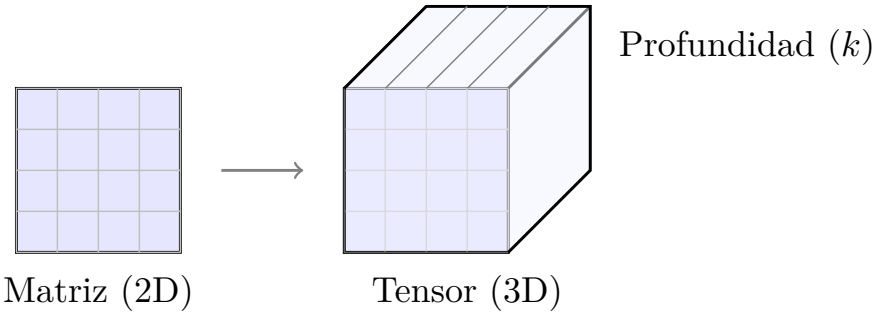


Figura 1.1: Visualización del Tensor

Ejemplo explícito: tensor de rango 4 en teledetección agrícola

Considere un estudio de monitoreo de cultivos en una región agrícola mediante imágenes multiespectrales tomadas por un dron a lo largo de una temporada de crecimiento. Los datos se organizan naturalmente en un **tensor de rango 4**:

$$\mathcal{T} \in \mathbb{R}^{30 \times 64 \times 64 \times 5}.$$

Cada dimensión (modo) del tensor representa una característica esencial del conjunto de datos:

- **Modo 1 (tiempo):** 30 fechas de vuelo distribuidas a lo largo de la temporada (una imagen cada 3–4 días).
- **Modo 2 (altura):** 64 filas de píxeles en cada imagen (resolución espacial de 64×64 píxeles por parcela).
- **Modo 3 (ancho):** 64 columnas de píxeles.
- **Modo 4 (bandas espectrales):** 5 bandas capturadas por el sensor multiespectral:
  1. Azul (450–515 nm),
  2. Verde (515–595 nm),
  3. Rojo (600–680 nm),

4. Infrarrojo cercano – NIR (770–890 nm),
5. Borde rojo – Red Edge (690–750 nm).

Un elemento genérico del tensor se denota como:

$$\mathcal{T}(t, i, j, b) = \text{valor de reflectancia en la fecha } t,\ \text{píxel } (i, j),\ \text{y banda } b,$$

donde:

$$\begin{aligned} t &\in \{1, 2, \ldots, 30\} \\ i &\in \{1, 2, \ldots, 64\} \\ j &\in \{1, 2, \ldots, 64\} \\ b &\in \{1, 2, 3, 4, 5\} \end{aligned}$$

Por ejemplo, el valor $\mathcal{T}(15, 32, 45, 4) = 0.82$ indica que, en la decimoquinta fecha de muestreo, el píxel ubicado en la fila 32 y columna 45 presentó una reflectancia relativa del 82 % en la banda NIR. Este valor alto es típico de vegetación sana y se usa para calcular índices como el NDVI.

Este tipo de tensor es la entrada estándar para arquitecturas de redes neuronales convolucionales 3D (3D-CNN) o modelos basados en transformers espaciotemporales, que predicen variables como rendimiento, estrés hídrico o presencia de enfermedades a partir de la dinámica espectral y espacial del cultivo.

Otros ejemplos prácticos con dimensiones explícitas:

- **Serie temporal de imágenes satelitales (rango 4):** $\mathcal{T} \in \mathbb{R}^{36\times512\times512\times6}$
  - Eje 1 (36): días de observación (una imagen cada 5 días durante 180 días),
  - Ejes 2–3 (512×512): resolución espacial de la parcela,
  - Eje 4 (6): bandas espectrales disponibles en el satélite (e.g., Sentinel-2).

  Este tensor es ideal para redes neuronales recurrentes o 3D-CNN que predicen rendimiento, fenología o presencia de plagas a partir de la dinámica del cultivo.

- **Monitoreo de un hato lechero (rango 3):** $\mathcal{T} \in \mathbb{R}^{90\times150\times4}$
  - Eje 1 (90): días de seguimiento (3 meses),
  - Eje 2 (150): número de vacas en el hato,
  - Eje 3 (4): variables fisiológicas: temperatura corporal, actividad (pasos), rumia (min/día), producción de leche (L/día).

  Este tensor permite detectar brotes de enfermedad (e.g., mastitis) mediante análisis de patrones anómalos en múltiples variables y animales simultáneamente.

- **Ensayo factorial en invernadero (rango 4):** $\mathcal{T} \in \mathbb{R}^{5\times4\times10\times8}$
  - Eje 1 (5): niveles de riego,
  - Eje 2 (4): tipos de fertilizante,
  - Eje 3 (10): repeticiones experimentales (macetas),
  - Eje 4 (8): variables de respuesta: altura, número de hojas, biomasa seca, contenido de nitrógeno, etc.

  Este tensor estructura un diseño experimental complejo y facilita el análisis multivariado de interacciones entre factores.

- **Secuencia de video para navegación robótica (rango 4):** $\mathcal{T} \in \mathbb{R}^{30\times224\times224\times4}$
  - Eje 1 (30): fotogramas temporales (1 segundo de video a 30 fps),
  - Ejes 2–3 (224×224): resolución espacial de la cámara del robot (redimensionada para una CNN),
  - Eje 4 (4): canales de información sensorial (R, G, B y *Depth*/Profundidad).

  Este tensor es la entrada típica para sistemas de *Visual Servoing* o SLAM (Localización y Mapeo Simultáneos), permitiendo al robot distinguir objetos reales de sombras y calcular trayectorias libres de colisiones en entornos dinámicos.

  Teledetección y Medio Ambiente

  Un tensor $\mathcal{T} \in \mathbb{R}^{30\times64\times64\times5}$ permite a una red neuronal convolucional (CNN) detectar estrés hídrico analizando la evolución temporal de la reflectancia en la banda NIR (Infrarrojo Cercano).

En la práctica de la inteligencia artificial moderna —dominada por librerías como `PyTorch` o `TensorFlow`—, el tensor evoluciona de una abstracción matemática a un **objeto computacional** de alto rendimiento. Se implementa como un *arreglo multidimensional* optimizado para ejecutarse en aceleradores de hardware (GPU/TPU) y con soporte nativo para la *diferenciación automática*. Esta infraestructura es la base común que permite entrenar modelos complejos en cualquier dominio: desde la visión computacional en agricultura y la navegación robótica, hasta la proyección de escenarios financieros en administración.

1.2. Manejo de los datos en Python

A continuación, se presentan las formas para definir los datos en sus diferentes presentaciones:

```python
import numpy as np
import torch

print("--- BLOQUE 1: CIENCIA DE DATOS (NumPy) ---")

# 1. Escalar (Rango 0)
s = 28.5
print(f"Escalar (Humedad): {s} | Tipo: {type(s)}")

# 2. Vector (Rango 1): Perfil de suelo [N, P, K, pH]
v = np.array([120, 50, 3, 6.2])
print(f"Vector: {v} | Forma: {v.shape}")

# 3. Matriz (Rango 2): Ensayo con 3 parcelas y 2 variables (Rendimiento, pH)
M = np.array([[110, 6.1],
              [130, 6.3],
              [100, 5.9]])
print(f"Matriz (Parcelas x Vars):\n{M}")

# Operación de Slicing (Acceso a datos):
# "Deme el pH (columna 1) de la segunda parcela (fila 1)"
ph_parcela_2 = M[1, 1]
print(f"--> pH de la parcela 2: {ph_parcela_2}")

```

```python

print("\n--- BLOQUE 2: INTELIGENCIA ARTIFICIAL (PyTorch) ---")

# 4. Tensor Rango 3: Imagen individual para un robot (Canales, Alto, Ancho)
# PyTorch prefiere el formato (C, H, W) para procesamiento
img_robot = torch.rand(3, 128, 128)
print(f"Imagen Robot (C,H,W): {img_robot.shape} | Rango: {img_robot.ndim}")

# 5. Tensor Rango 4: Serie de Tiempo Satelital (Multidimensional)
# Dimensiones: (Tiempo/Batch, Canales, Alto, Ancho)
# Ejemplo: 10 fechas, 5 bandas espectrales, resolución 64x64
serie_satelital = torch.randn(10, 5, 64, 64)

print(f"Serie Satelital: {serie_satelital.shape}")

# Acceso complejo:
# "Valor del pixel central (32,32) en la Banda Roja (índice 0) de la
# última fecha (índice -1)"
pixel_val = serie_satelital[-1, 0, 32, 32]
print(f"--> Valor pixel específico: {pixel_val:.4f}")
```

Listing 1.1: Implementación y Exploración de Estructuras (NumPy y PyTorch)

Salida

>_ Resultados

```
--- BLOQUE 1: CIENCIA DE DATOS (NumPy) ---
Escalar (Humedad): 28.5 | Tipo: <class 'float'>

Vector: [120.  50.   3.   6.2] | Forma: (4,)

Matriz (Parcelas x Vars):
[[110.    6.1]
 [130.    6.3]
 [100.    5.9]]
--> pH de la parcela 2: 6.3

--- BLOQUE 2: INTELIGENCIA ARTIFICIAL (PyTorch) ---
Imagen Robot (C,H,W): torch.Size([3, 128, 128]) | Rango: 3

Serie Satelital: torch.Size([10, 5, 64, 64])

--> Valor pixel específico: -0.4281
```

1.3. Conceptos fundamentales de vectores en $\mathbb{R}^n$

Antes de introducir las operaciones entre vectores, es esencial comprender su estructura interna y sus propiedades geométricas. En ciencia de datos agro-ambiental, los vectores en Rn representan observaciones multivariadas, y su análisis requiere comprender cómo se comparan, cómo se mide su tamaño y cómo se orientan en el espacio.

1.3.1. Transpuesta de vectores y matrices

La **transposición** es una operación fundamental que reorganiza la estructura de los datos, invirtiendo sus dimensiones: transforma un arreglo de tamaño $m \times n$ en uno de $n \times m$. Geométrica y algebraicamente, esto equivale a reflejar los elementos de la matriz respecto a su diagonal principal.

En el contexto de la ciencia de datos y la modelación, la transposición no es solo un cambio de formato, sino una herramienta indispensable para la **alineación dimensional**. Es el paso previo necesario para realizar operaciones críticas como el cálculo del producto punto, la proyección de vectores y la construcción de la **matriz de covarianza**, permitiendo así relacionar variables y observaciones de manera coherente.

Definición Dado un vector columna $\mathbf{x} \in \mathbb{R}^n$ (o $\mathbb{R}^{n\times 1}$), definido explícitamente como:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Su **transpuesta**, denotada como $\mathbf{x}^\top$, es el vector fila asociado en $\mathbb{R}^{1\times n}$:

$$\mathbf{x}^\top = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}$$

Definición Dada una matriz $\mathbf{A} \in \mathbb{R}^{m\times n}$, expresada explícitamente como:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}_{m\times n}$$

Su **transpuesta** $\mathbf{A}^\top \in \mathbb{R}^{n\times m}$ se construye convirtiendo la fila i de $\mathbf{A}$ en la columna i de $\mathbf{A}^\top$:

$$\mathbf{A}^\top = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}_{n\times m}$$

Formalmente, se define por la propiedad de sus entradas:

$$(\mathbf{A}^\top)_{ij} = a_{ji}, \quad \text{para todo } i = 1, \ldots, n,\ j = 1, \ldots, m.$$

Propiedades clave:

$$\begin{aligned} (\mathbf{A}^\top)^\top &= \mathbf{A} \\ (\mathbf{A} + \mathbf{B})^\top &= \mathbf{A}^\top + \mathbf{B}^\top \\ (\alpha\mathbf{A})^\top &= \alpha\mathbf{A}^\top \quad (\text{para todo } \alpha \in \mathbb{R}) \\ (\mathbf{AB})^\top &= \mathbf{B}^\top\mathbf{A}^\top \quad (\text{el orden se invierte}) \end{aligned}$$

Relevancia en ciencia de datos

- **Producto punto:** El producto escalar de dos vectores columna $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ se escribe como $\mathbf{x}^\top \mathbf{y}$, que resulta en un escalar. Esta notación es la base de la similitud coseno y las proyecciones.
- **Matriz de covarianza:** Si $\mathbf{X} \in \mathbb{R}^{m \times n}$ es una matriz de datos (filas = observaciones, columnas = variables), la matriz de covarianza se calcula como $\boldsymbol{\Sigma} = \frac{1}{m-1}\mathbf{X}^\top \mathbf{X}$. La transposición permite alinear correctamente las variables para el cálculo de covarianzas.
- **Ajuste de modelos:** En regresión lineal, el modelo $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ requiere que $\mathbf{X}^\top \mathbf{X}$ sea invertible para estimar $\boldsymbol{\beta}$, lo que depende directamente de la transposición.

Ejemplo numérico. Considere una matriz de datos de un ensayo agronómico con 2 parcelas y 3 variables (N, P, K):

$$\mathbf{X} = \begin{pmatrix} 120 & 50 & 3 \\ 100 & 60 & 4 \end{pmatrix}.$$

Su transpuesta es:

$$\mathbf{X}^\top = \begin{pmatrix} 120 & 100 \\ 50 & 60 \\ 3 & 4 \end{pmatrix}.$$

1.3.2. Implementación Computacional y Matices Prácticos

Aunque la definición matemática es estricta, en librerías como `NumPy` o `PyTorch` existe una distinción técnica importante entre un arreglo unidimensional (plano) y un vector columna formal.

- **Arreglo 1D (Rank-1):** Tiene forma `(n,)`. Su transpuesta `.T` no altera nada (sigue siendo plano). Es eficiente en memoria pero peligroso en álgebra lineal estricta.
- **Vector Columna (Rank-2):** Tiene forma `(n, 1)`. Su transpuesta cambia la forma a `(1, n)`, comportándose exactamente como la teoría matemática.

A continuación, implementamos estos conceptos y verificamos la propiedad crítica de la inversión del producto $(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top$.

```
import torch

print("--- 1. EL 'ENGAÑO' DE LOS VECTORES 1D ---")
# Vector plano (común en programación básica)
v_flat = torch.tensor([1, 2, 3])
print(f"Vector plano: {v_flat.shape}")
print(f"Transpuesta v.T: {v_flat.T.shape} (¡No cambia!)")

# Vector Matemático (Columna explícita)
# Usamos .unsqueeze(1) o definimos los corchetes dobles [[...]]
v_col = v_flat.unsqueeze(1) # Transforma (3) -> (3, 1)
print(f"Vector Columna:\n{v_col.shape}")
print(f"Vector Fila (v_col.T):\n{v_col.T.shape}")

print("\n--- 2. MATRICES Y PROPIEDAD (AB)^T ---")
# A: Matriz de datos (2 muestras, 3 variables)
A = torch.tensor([[1., 2., 3.],
                  [4., 5., 6.]]) # Shape (2, 3)
```

```python

# B: Matriz de transformación (3 entradas, 2 salidas)
B = torch.tensor([[0.1, 0.2],
                  [0.3, 0.4],
                  [0.5, 0.6]])   # Shape (3, 2)

# Operación: Transpuesta del producto
lhs = torch.matmul(A, B).T   # (AB)^T

# Verificación de la propiedad teórica
# INCORRECTO: A.T @ B.T (Error de dimensiones o resultado erróneo)
# CORRECTO: B.T @ A.T (Invirtiendo el orden)
rhs = torch.matmul(B.T, A.T)

print(f"Forma de (AB).T: {lhs.shape}")
print(f"¿Es igual a B.T @ A.T?: {torch.allclose(lhs, rhs)}")
```

Listing 1.2: Transposición y Verificación de Propiedades en PyTorch

>_ Resultados

```
--- 1. EL 'ENGAÑO' DE LOS VECTORES 1D ---
Vector plano: torch.Size([3])
Transpuesta v.T: torch.Size([3]) (¡No cambia!)
Vector Columna:
torch.Size([3, 1])
Vector Fila (v\_col.T):  % <--- AQUÍ ESTABA EL ERROR (agregué \)
torch.Size([1, 3])

--- 2. MATRICES Y PROPIEDAD (AB)\^T ---
Forma de (AB).T: torch.Size([2, 2])
¿Es igual a B.T @ A.T?: True
```

1.3.3. Transposición en Tensores (Permutación)

Para tensores de rango $N \geq 3$, la noción de transposición se generaliza a la **permutación de ejes**. No existe una única "transpuesta", sino múltiples reordenamientos posibles de las dimensiones.

Dado un tensor $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$, con entradas t_{ijk} donde i es el índice del primer eje, j del segundo y k del tercero.

Una permutación común (por ejemplo, invertir el orden de los ejes) genera un nuevo tensor $\mathcal{T}' \in \mathbb{R}^{d_3 \times d_2 \times d_1}$ tal que:

$$\mathcal{T}'_{kji} = \mathcal{T}_{ijk}$$

En la práctica de visión computacional, la operación más frecuente es intercambiar el eje de *canales* (C) para moverlo del principio al final:

Definición práctica (Cambio de Formato): Sea un tensor de imagen $\mathbf{I} \in \mathbb{R}^{C \times H \times W}$ (formato PyTorch). Su versión permutada $\mathbf{I}_{perm} \in \mathbb{R}^{H \times W \times C}$ (formato Matplotlib/OpenCV) se define reordenando los índices $(c, h, w) \to (h, w, c)$:

$$\mathbf{I} = [\text{Canales}, \text{Alto}, \text{Ancho}] \xrightarrow{\text{permute}(1,2,0)} \mathbf{I}_{perm} = [\text{Alto}, \text{Ancho}, \text{Canales}]$$

```python
import torch

```

```python
# Tensor 3D: Una imagen RGB simulada (3 canales, 4 alto, 4 ancho)
# Formato PyTorch: (C, H, W)
imagen_torch = torch.rand(3, 4, 4)
print(f"Forma original (C, H, W): {imagen_torch.shape}")

# PROBLEMA: Las librerías de visualización (matplotlib) esperan (H, W, C)
# SOLUCIÓN: Permutar los ejes.
# Índice 0->2 (Canales al final)
# Índice 1->0 (Alto al principio)
# Índice 2->1 (Ancho al medio)
imagen_plot = imagen_torch.permute(1, 2, 0)

print(f"Forma permutada (H, W, C): {imagen_plot.shape}")

# Cuidado: .T (transpuesta simple) en PyTorch no siempre funciona
    intuitivamente en tensores > 2D
# Es preferible ser explícito con .permute()
```

Listing 1.3: Permutación de Ejes en PyTorch

```
>_ Resultados
Forma original (C, H, W): torch.Size([3, 4, 4])
Forma permutada (H, W, C): torch.Size([4, 4, 3])
```

1.3.4. Igualdad de vectores

Dos vectores $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ se consideran **iguales** si satisfacen simultáneamente dos condiciones:

1. Pertenecen al mismo espacio vectorial (tienen la misma dimensión n).
2. Sus componentes correspondientes son idénticas en valor y posición.

Formalmente:

$$\mathbf{x} = \mathbf{y} \quad \iff \quad x_i = y_i \quad \text{para todo } i = 1, \ldots, n.$$

Interpretación Agronómica La igualdad vectorial implica una réplica exacta de condiciones. Por ejemplo, dos lotes de cultivo tienen un manejo nutricional idéntico solo si sus vectores de fertilización $\mathbf{f}_A = (N, P, K, \text{Micro})$ y $\mathbf{f}_B$ son iguales componente a componente. Si $N_A = N_B$ pero $K_A \neq K_B$, entonces $\mathbf{f}_A \neq \mathbf{f}_B$, lo que significa que los tratamientos son agronómicamente distintos.

Administración y Finanzas: Conciliación

La igualdad vectorial es la base de la **auditoría**. Si definimos un vector de presupuesto planificado $\mathbf{p} \in \mathbb{R}^3$ y un vector de ejecución real $\mathbf{e} \in \mathbb{R}^3$ para tres departamentos (Ventas, I+D, Operaciones):

$$\mathbf{p} = \begin{pmatrix} 100 \\ 50 \\ 80 \end{pmatrix}, \quad \mathbf{e} = \begin{pmatrix} 100 \\ 50 \\ 80 \end{pmatrix}$$

La condición $\mathbf{p} = \mathbf{e}$ indica un cumplimiento presupuestario perfecto (varianza cero). Si $\mathbf{p} \neq \mathbf{e}$, la diferencia $\mathbf{d} = \mathbf{p} - \mathbf{e}$ generará un vector de desviaciones no nulo que debe ser justificado.

1.3.5. Igualdad de matrices

Dos matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ se consideran **iguales** si y solo si satisfacen simultáneamente dos condiciones:

1. Tienen las mismas dimensiones (igual número de filas m y columnas n).
2. Sus entradas correspondientes son idénticas en valor y posición.

Formalmente:

$$\mathbf{A} = \mathbf{B} \quad \Longleftrightarrow \quad A_{ij} = B_{ij} \quad \text{para todo } i = 1, \ldots, m;\ j = 1, \ldots, n.$$

Interpretación Ambiental: Detección de Cambios En monitoreo satelital, una imagen espectral se representa como una matriz numérica donde cada entrada corresponde a un píxel. Sea $\mathbf{M}_{2020}$ la matriz de índices de vegetación (NDVI) de una reserva forestal en 2020 y $\mathbf{M}_{2024}$ la del año actual. La igualdad $\mathbf{M}_{2020} = \mathbf{M}_{2024}$ indicaría una conservación absoluta del ecosistema. En la práctica, los científicos buscan la matriz diferencia $\mathbf{D} = \mathbf{M}_{2024} - \mathbf{M}_{2020}$; si una entrada D_{ij} es significativamente distinta de cero (negativa), alerta sobre una posible deforestación en esa coordenada específica.

Mecatrónica: Control de Robots

En robótica, la posición y orientación de un brazo manipulador se describen mediante matrices de transformación homogénea de 4×4. Sea $\mathbf{T}_{\text{obj}}$ la matriz que representa la pose deseada (target) del efector final (la pinza) y $\mathbf{T}_{\text{act}}$ la pose actual leída por los sensores:

$$\mathbf{T}_{\text{obj}} = \begin{pmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 20 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{T}_{\text{act}} = \begin{pmatrix} 1 & 0 & 0 & 9.8 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 20 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

El sistema de control verifica la igualdad. Como T_{14} (posición en x) es 10 en la deseada y 9.8 en la actual, $\mathbf{T}_{\text{obj}} \neq \mathbf{T}_{\text{act}}$. Esto genera una señal de error que activa los motores para corregir esa diferencia de 0.2 unidades.

Generalización a Tensores El concepto de igualdad se extiende naturalmente a arreglos multidimensionales de orden superior, conocidos como **tensores**. Sean $\mathcal{A}$ y $\mathcal{B}$ dos tensores de orden 3 (por ejemplo, de dimensiones $m \times n \times p$). Estos se consideran iguales solo si coinciden en cada voxel o celda cúbica:

$$\mathcal{A} = \mathbf{B} \quad \Longleftrightarrow \quad A_{ijk} = B_{ijk} \quad \forall i, j, k.$$

Ejemplo: Imágenes RGB Una imagen digital a color se representa como un tensor de Alto $\times$ Ancho $\times$ 3 (Canales: Rojo, Verde, Azul). Si tenemos una imagen original $\mathcal{I}_{\text{orig}}$ y una copia transmitida por internet $\mathcal{I}_{\text{copia}}$, la igualdad $\mathcal{I}_{\text{orig}} = \mathcal{I}_{\text{copia}}$ asegura la integridad de los datos. Basta con que un solo píxel cambie levemente su tono en el canal azul para que $\mathcal{I}_{\text{orig}} \neq \mathcal{I}_{\text{copia}}$, lo cual en criptografía o esteganografía podría indicar que la imagen fue alterada.

1.3.6. Igualdad Computacional: El Desafío del Punto Flotante

En la práctica profesional de la Ciencia de Datos, rara vez verificamos la igualdad matemática estricta ($\mathbf{A} = \mathbf{B}$). Esto se debe a que las computadoras utilizan el estándar **IEEE 754** para representar números decimales (punto flotante), donde operaciones simples introducen errores infinitesimales de redondeo en los bits menos significativos.

Para abordar la comparación de tensores, es fundamental distinguir dos enfoques y una solución técnica:

1. **Comparación Element-wise (Elemento a elemento):** Genera una matriz de valores booleanos del mismo tamaño que las originales. Es útil para crear "máscaras" y detectar *dónde* difieren los datos.
2. **Comparación Estricta:** Verifica si la estructura es idéntica en su totalidad. Falla frecuentemente con números decimales (floats).
3. **Comparación con Tolerancia (La Solución):** Sustituye la igualdad estricta por la proximidad dentro de un umbral ϵ.

Matemáticamente, definimos la igualdad computacional como:

$$\mathbf{A} \approx \mathbf{B} \iff |a_{ij} - b_{ij}| < \epsilon, \quad \forall i, j$$

Donde ϵ (epsilon) suele ser un valor muy pequeño (ej. 10^{-5} o 10^{-8}).

```python
import torch

print("--- 1. ELEMENT-WISE VS ESTRUCTURA ---")
A = torch.tensor([1.0, 2.0])
B = torch.tensor([1.0, 5.0]) # El segundo elemento difiere

# Comparación elemento a elemento (Genera máscara)
print(f"Máscara: {A == B}")
# Salida esperada: [True, False]

print("\n--- 2. EL PROBLEMA DEL PUNTO FLOTANTE ---")
# Matemáticamente: (Raíz de 2) al cuadrado = 2
raiz = torch.sqrt(torch.tensor(2.0))
calculado = raiz * raiz
teorico   = torch.tensor(2.0)

# Mostramos con 10 decimales para revelar el "error fantasma"
print(f"Valor Teórico:   {teorico.item():.10f}")
print(f"Valor Calculado: {calculado.item():.10f}")

# Intento 1: Igualdad Estricta (==)
# Falla porque 2.0000000000 != 2.0000002384
print(f"¿Igualdad Estricta? {calculado == teorico}")

print("\n--- 3. SOLUCIÓN: TOLERANCIA (ALLCLOSE) ---")
# Verificamos si la diferencia es despreciable
# atol = tolerancia absoluta
es_cercano = torch.allclose(calculado, teorico, atol=1e-05)
print(f"¿Igualdad con tolerancia (allclose)? {es_cercano}")
```

Listing 1.4: El peligro de la igualdad estricta y la solución con allclose

>_ Resultados

```
--- 1. ELEMENT-WISE VS ESTRUCTURA ---
Máscara: tensor([ True, False])

--- 2. EL PROBLEMA DEL PUNTO FLOTANTE ---
Valor Teórico:   2.0000000000
Valor Calculado: 2.0000002384
¿Igualdad Estricta? tensor(False)

--- 3. SOLUCIÓN: TOLERANCIA (ALLCLOSE) ---
¿Igualdad con tolerancia (allclose)? True
```

1.4. Norma euclidiana (magnitud) y dirección

La **norma euclidiana** (también llamada longitud o módulo) de un vector $\mathbf{x} = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$ se define como la raíz cuadrada de la suma de sus componentes al cuadrado:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

Esta medida generaliza el concepto de distancia desde el origen hasta el punto $\mathbf{x}$. En ciencia de datos, es fundamental para normalizar variables (escalar datos), calcular el error cuadrático medio o evaluar la ”fuerza”de una señal.

Interpretación en el plano ($\mathbb{R}^2$)

En dos dimensiones, un vector $\mathbf{x} = (x_1, x_2)^\top$ se representa geométricamente como una flecha. Sus propiedades fundamentales son:

1. **Magnitud:** La longitud de la flecha, dada por Pitágoras: $\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2}$.
2. **Dirección:** El ángulo θ respecto al eje horizontal, calculado como $\theta = \arctan(x_2/x_1)$.

La Figura 1.2 ilustra cómo las componentes definen tanto la posición final como la orientación del vector.

Ejemplo Administrativo: Presupuesto Vectorial

Considere una propuesta presupuestaria $\mathbf{p} \in \mathbb{R}^2$ (en millones de pesos) asignada a Infraestructura y Capacitación:

$$\mathbf{p} = \begin{pmatrix} 4 \\ 3 \end{pmatrix}$$

- **Magnitud (Esfuerzo Total):** $\|\mathbf{p}\| = \sqrt{4^2 + 3^2} = \sqrt{25} = 5$. El tamaño total de la inversión es 5 millones.
- **Dirección (Prioridad Estratégica):** El ángulo respecto a infraestructura es $\theta = \arctan(3/4) \approx 36{,}9°$.
  - Si $\theta \to 0°$, la prioridad es 100 % Infraestructura.
  - Si $\theta \to 90°$, la prioridad es 100 % Capacitación.
  - Con 36,9°, existe un balance inclinado hacia la infraestructura.

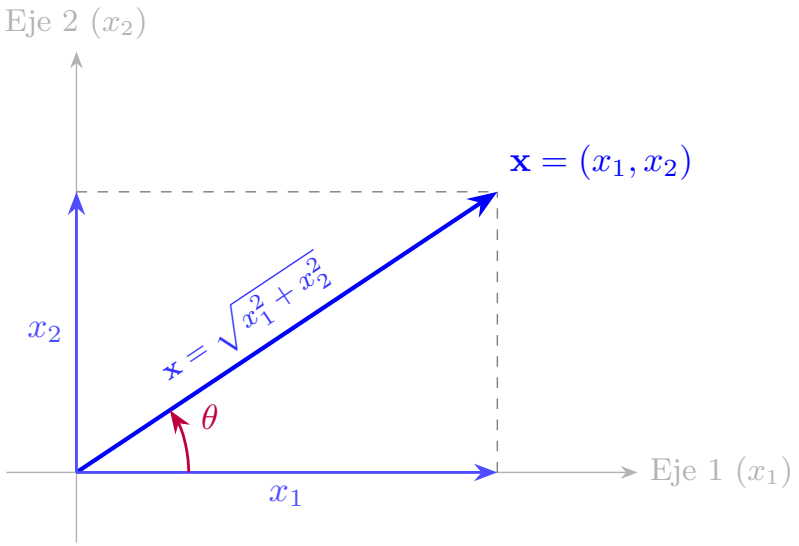


Figura 1.2: Representación geométrica: la norma es la longitud de la hipotenusa y θ determina la orientación.

Orientación en el espacio ($\mathbb{R}^3$): Cosenos directores

En tres dimensiones, un solo ángulo no basta para definir la dirección. Para un vector $\mathbf{x} = (x_1, x_2, x_3)^\top$, utilizamos los **cosenos directores**, que son los cosenos de los ángulos (α, β, γ) que forma el vector con cada uno de los ejes coordenados (x, y, z respectivamente):

$$\cos(\alpha) = \frac{x_1}{\|\mathbf{x}\|}, \quad \cos(\beta) = \frac{x_2}{\|\mathbf{x}\|}, \quad \cos(\gamma) = \frac{x_3}{\|\mathbf{x}\|}.$$

Estos valores oscilan entre -1 y 1. Un valor cercano a 1 indica que el vector está muy alineado con ese eje específico, dominando el comportamiento de la variable.

Ejemplo Ambiental: Calidad del Aire

Analicemos el vector de contaminantes (en μg/m^3) de una estación urbana:

$$\mathbf{a} = \begin{pmatrix} \text{PM}_{10} \\ \text{NO}_2 \\ \text{O}_3 \end{pmatrix} = \begin{pmatrix} 40 \\ 30 \\ 50 \end{pmatrix}$$

1. **Magnitud (Intensidad de Contaminación):**

$$\|\mathbf{a}\| = \sqrt{40^2 + 30^2 + 50^2} = \sqrt{5000} \approx 70{,}71.$$

2. **Análisis de Dominancia (Cosenos Directores):**

$$\cos(\gamma)_{\text{O}_3} = \frac{50}{70{,}71} \approx \mathbf{0{,}707}, \quad \cos(\alpha)_{\text{PM}_{10}} \approx 0{,}566.$$

La componente de Ozono (O_3) tiene el coseno director más alto (0,707), lo que indica que es el contaminante dominante en este perfil. Esto sugiere un problema de tipo fotoquímico (reacción a la luz solar) más que de emisiones primarias de combustión (NO_2), permitiendo a las autoridades decidir si restringen el tráfico o la industria.

1.4.1. Implementación Computacional: Normas y Normalización

En el ecosistema de Python (NumPy y PyTorch), el cálculo de la magnitud y la dirección está altamente optimizado. No es necesario iterar manualmente sobre los elementos sumando cuadrados; las librerías utilizan rutinas de bajo nivel (BLAS/LAPACK) para hacerlo instantáneamente.

Cálculo de la Norma (L_2)

Aunque la fórmula es $\sqrt{\sum x_i^2}$, en código utilizamos funciones dedicadas como `torch.norm` o `numpy.linalg.norm`. Esto previene errores de desbordamiento numérico (overflow) cuando los números son muy grandes.

```python
import torch

# Ejemplo Administrativo: Presupuesto [Infra, Capa]
p = torch.tensor([4.0, 3.0])

# FORMA 1: Manual (Solo con fines educativos)
# Paso a paso: Cuadrado -> Suma -> Raíz
norma_manual = torch.sqrt(torch.sum(p**2))

# FORMA 2: Profesional (La que usarás siempre)
# Es más rápida y numéricamente estable
norma_pro = torch.norm(p)

print(f"Vector p: {p}")
print(f"Norma Manual: {norma_manual.item()}") # 5.0
print(f"Norma Pro:    {norma_pro.item()}")    # 5.0
```

Listing 1.5: Cálculo de la Magnitud (Norma Euclidiana)

Dirección y Normalización

Aquí distinguimos entre 2D y N-Dimensiones:

1. **En 2D (Ángulos):** No usamos $\arctan(y/x)$ porque falla si $x = 0$. Usamos la función especial `atan2(y, x)`, que maneja todos los cuadrantes y la división por cero automáticamente.
2. **En ND (Cosenos Directores = Normalización):** Calcular los cosenos directores equivale a convertir el vector en un **Vector Unitario** (longitud 1). Esta operación se llama **Normalización L2** y es vital en redes neuronales.

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|} = (\cos\alpha, \cos\beta, \cos\gamma)^\top$$

```python
import torch
import math

print("--- 1. DIRECCIÓN EN 2D (PRESUPUESTO) ---")
p = torch.tensor([4.0, 3.0]) # x=4, y=3

# Usamos atan2(y, x). Nota: El orden es (y, x)
theta_rad = torch.atan2(p[1], p[0])

```

```python
# Convertimos radianes a grados para humanos
theta_deg = torch.rad2deg(theta_rad)

print(f"Ángulo (theta): {theta_deg.item():.1f} grados")


print("\n--- 2. DIRECCIÓN EN 3D (CALIDAD AIRE) ---")
# Vector: [PM10, NO2, O3]
a = torch.tensor([40.0, 30.0, 50.0])

# Calculamos la norma
magnitud = torch.norm(a)

# Cosenos Directores: Dividimos el vector por su magnitud
# Esto crea un 'Vector Unitario' (Unit Vector)
cosenos_directores = a / magnitud

print(f"Magnitud total: {magnitud:.2f}")
print(f"Cosenos Directores (Dirección):\n{cosenos_directores}")

# Verificación: La norma de los cosenos directores siempre es 1
print(f"Comprobación (Norma del unitario): {torch.norm(cosenos_directores)
    :.1f}")
```

Listing 1.6: De la Orientación a la Normalización

>_ Resultados

```
--- 1. DIRECCIÓN EN 2D (PRESUPUESTO) ---
Ángulo (theta): 36.9 grados

--- 2. DIRECCIÓN EN 3D (CALIDAD AIRE) ---
Magnitud total: 70.71
Cosenos Directores (Dirección):
tensor([0.5657, 0.4243, 0.7071])
Comprobación (Norma del unitario): 1.0
```

1.5. Ejercicios propuestos

1. **Clasificación de Estructuras (Rangos):** Identifique el rango (0, 1, 2 o ≥ 3) y la dimensión matemática aproximada (ej. $\mathbb{R}^n$) de los siguientes objetos de datos agroambientales:

   a) La concentración de nitratos en una muestra de suelo (un solo valor numérico).

   b) El perfil de temperatura de un silo medido a 10 alturas diferentes.

   c) Una fotografía aérea de un cultivo en escala de grises de 1024×768 píxeles.

   d) Un conjunto de datos multitemporal que contiene 5 bandas espectrales, para una imagen de 500×500 píxeles, tomada durante 12 meses consecutivos.

2. **Interpretación de la Transpuesta:** Sea $\mathbf{D} \in \mathbb{R}^{100\times5}$ una matriz de datos donde las filas ($i = 1 \dots 100$) representan plantas de maíz individuales y las columnas ($j = 1 \dots 5$) representan variables medidas (Altura, Grosor de tallo, Número de hojas, Clorofila, Rendimiento).

   a) ¿Cuáles son las dimensiones de la matriz transpuesta $\mathbf{D}^\top$?

b) En la matriz $\mathbf{D}^\top$, ¿qué representa ahora una fila? ¿Y una columna?

c) ¿Por qué podría ser útil calcular la media de las filas de $\mathbf{D}^\top$ para un análisis estadístico?

3. **Verificación de Igualdad:** Dadas las matrices:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} 1 & 2 \\ 3 & 4.00001 \end{pmatrix}$$

a) ¿Es $\mathbf{A} = \mathbf{B}$? Justifique su respuesta basándose en las dimensiones.

b) ¿Es $\mathbf{A} = \mathbf{C}$ en términos matemáticos estrictos?

c) Si $\mathbf{C}$ proviene de un sensor digital ruidoso, ¿consideraría estas matrices iguales en un contexto aplicado?

4. **Laboratorio de Python (NumPy):** Escriba un script en Python que realice lo siguiente:

- Cree un vector $\mathbf{v}$ con los valores $[0, 10, 20, 30]$ (representando lecturas ideales de un sensor).
- Simule un vector de lectura real $\mathbf{v}_{ruido}$ sumando un pequeño valor aleatorio (ej. 0.005) a cada componente.
- Intente verificar la igualdad con el operador standard `==` y comente el resultado.
- Verifique la igualdad correctamente utilizando `np.allclose` con una tolerancia adecuada.

5. **Agronómico (Análisis de Suelos - Transpuesta):** Un laboratorio entrega los resultados de análisis de suelo de 3 lotes diferentes en una matriz $\mathbf{S} \in \mathbb{R}^{3\times4}$, donde las filas son los Lotes (A, B, C) y las columnas son los parámetros (pH, M.O., P, K).

$$\mathbf{S} = \begin{pmatrix} 5.5 & 2.1 & 15 & 0.4 \\ 6.2 & 3.5 & 20 & 0.6 \\ 5.8 & 2.8 & 12 & 0.5 \end{pmatrix}$$

a) Escriba explícitamente la matriz transpuesta $\mathbf{S}^\top$.

b) Interprete el significado de la **segunda fila** de la matriz transpuesta. ¿Qué información agrupa?

6. **Agrícola (Monitoreo de Cosecha - Igualdad):** Se tiene un vector de rendimiento estimado $\mathbf{r}_{est} = [4.5, 5.0, 4.2]^\top$ (ton/ha) para tres variedades de maíz. Al finalizar la cosecha, el vector real fue $\mathbf{r}_{real} = [4.48, 5.01, 3.8]^\top$.

a) Si definimos una tolerancia de error de ± 0.05 ton/ha, ¿para cuáles variedades se cumple que $\mathbf{r}_{est} \approx \mathbf{r}_{real}$?

b) ¿Cómo expresaría esta comparación utilizando la resta vectorial $\mathbf{d} = \mathbf{r}_{est} - \mathbf{r}_{real}$?

7. **Agroindustrial (Control de Calidad - Matrices):** Una planta procesadora de jugos produce lotes de 3 sabores (Naranja, Mango, Mora). La matriz $\mathbf{M}_{std}$ define la formulación estándar (kg de fruta, litros de agua, kg de azúcar) y $\mathbf{M}_{lote}$ es la mezcla actual.

$$\mathbf{M}_{std} = \begin{pmatrix} 100 & 50 & 5 \\ 120 & 40 & 6 \\ 90 & 60 & 4 \end{pmatrix}$$

Si el sistema de control detecta que $\mathbf{M}_{lote} \neq \mathbf{M}_{std}$, explique qué consecuencias físicas tiene esto para el producto final si la diferencia ocurre en la posición $(2, 3)$ (Fila 2, Columna 3).

8. **Ambiental (Datos Climáticos - Tensores):** Una estación meteorológica registra datos utilizando un tensor $\mathcal{C}$ de dimensiones $365 \times 24 \times 3$.

   - Dimensión 1: Días del año $(1 \dots 365)$.
   - Dimensión 2: Horas del día $(0 \dots 23)$.
   - Dimensión 3: Variables (Temperatura, Humedad, Radiación Solar).

   a) ¿Qué representa una rebanada"(slice) del tensor si fijamos la primera dimensión (ej. día 100) y tomamos todos los datos restantes $(100, :, :)$?

   b) Si extraemos el vector $\mathbf{v} = \mathcal{C}(:, 12, 0)$, ¿qué serie de tiempo estamos analizando?

9. **Administrativo (Inventarios - Dimensión):** Una cooperativa gestiona 2 bodegas. La Bodega Norte tiene un inventario representado por el vector $\mathbf{b}_N \in \mathbb{R}^5$ (5 tipos de insumos). La Bodega Sur maneja 6 tipos de insumos, representada por $\mathbf{b}_S \in \mathbb{R}^6$.

   a) ¿Es matemáticamente posible realizar la operación de comparación $\mathbf{b}_N = \mathbf{b}_S$? ¿Por qué?

   b) Desde el punto de vista administrativo, ¿qué paso previo (padding o relleno) debería realizarse para poder consolidar ambos inventarios en una sola matriz?

10. **Mecatrónica (Visión Artificial para Pick-and-Place):** Un brazo robótico utiliza una cámara de baja resolución para identificar piezas defectuosas en una banda transportadora. La imagen ideal de una pieza correcta está representada por la matriz binaria $\mathbf{P}_{ideal}$ (donde 1 es metal y 0 es fondo). La cámara captura una imagen $\mathbf{P}_{cam}$ de la pieza que pasa actualmente.

$$\mathbf{P}_{ideal} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad , \quad \mathbf{P}_{cam} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

   a) **Pre-procesamiento (Transposición):** El ingeniero nota que la cámara fue montada girada 90° respecto al modelo ideal, o la pieza llegó rotada. Calcule $\mathbf{P}_{ideal}^{\top}$. ¿Se cumple que $\mathbf{P}_{ideal}^{\top} = \mathbf{P}_{cam}$?

   b) **Detección de Defectos (Matrices Diferencia):** Suponga ahora que llega una nueva pieza $\mathbf{P}_{nueva}$. Para encontrar defectos, el robot calcula la "matriz de error"$\mathbf{E} = \mathbf{P}_{ideal} - \mathbf{P}_{nueva}$. Si el resultado es:

$$\mathbf{E} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

   Interprete el resultado: ¿Le falta material a la pieza o le sobra? ¿En qué coordenada específica (i, j) está el defecto físico?

   c) **Lógica de Control:** En un script de Python para el controlador del robot, ¿por qué la siguiente condición lógica es peligrosa para la producción?

```
if imagen_camara != imagen_ideal:
    detener_linea_produccion()
```

   Pista: Piense en el ruido eléctrico de los sensores o cambios de iluminación.

Ejercicios de Integración: Magnitud vs. Dirección Estos ejercicios están diseñados para distinguir cuándo un cambio en los datos es de *escala* (magnitud) y cuándo es de *comportamiento* (dirección).

1. **Agronomía: Diagnóstico Nutricional (Colinealidad)** Se analizan dos muestras de suelo de lotes vecinos. Los vectores de nutrientes (Nitrógeno, Fósforo) en ppm son:

$$\mathbf{x}_A = \begin{pmatrix} 20 \\ 10 \end{pmatrix}, \quad \mathbf{x}_B = \begin{pmatrix} 60 \\ 30 \end{pmatrix}$$

   a) Calcule la magnitud (riqueza total nutricional) de cada lote: $\|\mathbf{x}_A\|$ y $\|\mathbf{x}_B\|$.

   b) Calcule la dirección (ángulo θ) de cada vector respecto al eje de Nitrógeno.

   c) **Interpretación:** ¿Puede afirmar que el Lote B tiene un "balance químico"diferente al Lote A, o simplemente está "más concentrado¿ Justifique usando el ángulo.

2. **Finanzas: Perfiles de Riesgo** Dos fondos de inversión distribuyen su capital en (Bonos del Estado, Acciones Tecnológicas). Los vectores en millones de USD son:

$$\mathbf{f}_1 = \begin{pmatrix} 8 \\ 2 \end{pmatrix}, \quad \mathbf{f}_2 = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

   a) ¿Qué fondo maneja mayor capital total? (Compare normas).

   b) Calcule el ángulo de cada fondo respecto al eje de "Bonos del Estado"(x_1).

   c) Si definimos un perfil Çonservadorçomo aquel con $\theta < 30°$ y .ªgresivoçomo $\theta > 60°$, clasifique a cada fondo.

3. **Ingeniería/Mecatrónica: Fuerza Resultante en 3D** Un dron agrícola está sometido a tres fuerzas de viento representadas por el vector $\mathbf{v} = (3, 4, 12)$ m/s (componentes en x, y, z).

   a) Calcule la velocidad total del viento (la rapidez) hallando $\|\mathbf{v}\|$. *(Pista:* $3^2 + 4^2 = 25$ *y* $12^2 = 144$*).*

   b) Calcule los cosenos directores. ¿Cuál es la componente dominante?

   c) Si el dron solo puede tolerar vientos verticales (z) que representen menos del 50 % de la fuerza total del viento (es decir, $\cos(\gamma) < 0.5$), ¿es seguro volar?

4. **Desafío de Python: Normalización de Datos** En Inteligencia Artificial, a menudo necesitamos que los vectores tengan una longitud estándar de 1 (vectores unitarios) para compararlos justamente. Escriba un pequeño script o pseudocódigo que:

   - Defina el vector $\mathbf{v} = [10, 20, 20]$.
   - Calcule su norma $L = \|\mathbf{v}\|$.
   - Cree un nuevo vector $\mathbf{u} = \mathbf{v}/L$.
   - Verifique computacionalmente que la norma del nuevo vector $\mathbf{u}$ es igual a 1.

Capítulo 2

Poniendo el Motor en Marcha: Aritmética Tensorial

2.1. Suma de vectores y Aplicaciones

La suma combina dos vectores componente a componente. En términos de datos, esta operación representa la acumulación o superposición de efectos provenientes de distintas fuentes.

Definición formal. Sean $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. La suma $\mathbf{x} + \mathbf{y}$ se define como:

$$\mathbf{x} + \mathbf{y} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{pmatrix}.$$

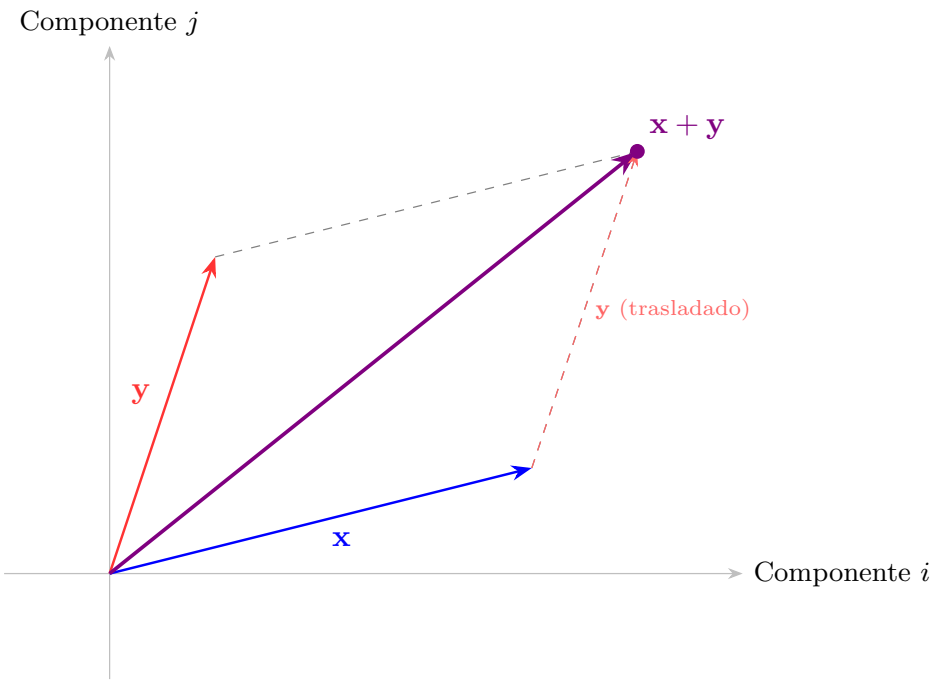


Figura 2.1: Representación geométrica de la suma vectorial (Ley del Paralelogramo). El vector resultante conecta el origen con la esquina opuesta formada por la proyección de los vectores $\mathbf{x}$ y $\mathbf{y}$.

Geométricamente, esto sigue la **ley del paralelogramo** 2.1: si colocamos el inicio de $\mathbf{y}$ en la punta final de $\mathbf{x}$, el vector resultante va desde el origen de $\mathbf{x}$ hasta la punta final de $\mathbf{y}$.

Ejemplo Ambiental: Inventario de Emisiones Supongamos que en una zona industrial existen dos fuentes principales de contaminación: una Termoeléctrica ($\mathbf{e}_T$) y una Fábrica de Cemento ($\mathbf{e}_C$). Los vectores representan la emisión diaria (en toneladas) de tres contaminantes distintos: $[\mathrm{CO}_2, \mathrm{NO}_x, \text{Material Particulado}]$.

$$\mathbf{e}_T = \begin{pmatrix} 100 \\ 5 \\ 0.5 \end{pmatrix}, \quad \mathbf{e}_C = \begin{pmatrix} 40 \\ 2 \\ 3.5 \end{pmatrix}$$

Para conocer la **carga total** que recibe la atmósfera en esa zona, realizamos la suma vectorial:

$$\mathbf{e}_{total} = \mathbf{e}_T + \mathbf{e}_C = \begin{pmatrix} 100 + 40 \\ 5 + 2 \\ 0.5 + 3.5 \end{pmatrix} = \begin{pmatrix} 140 \\ 7 \\ 4.0 \end{pmatrix}$$

Interpretación: La suma vectorial garantiza la integridad de las variables: sumamos dióxido de carbono solo con dióxido de carbono (140 ton), y partículas solo con partículas (4.0 ton). Mezclar componentes (sumar el CO_2 de una fábrica con el polvo de otra) carecería de sentido físico.

Aplicaciones sectoriales: Agro y Mecatrónica

- **Fusión de sensores (Agro):** Un nodo IoT registra variables climáticas de dos sensores para redundancia. Si el sensor A entrega el vector de estado $\mathbf{s}_A$ y el sensor B entrega $\mathbf{s}_B$, la suma (o promedio vectorial) permite reducir el ruido aleatorio y obtener una lectura más robusta del ambiente.
- **Acumulación temporal (Ambiental):** El vector de emisiones diarias de un biodigestor se suma a lo largo de una semana ($t = 1 \dots 7$) para obtener el impacto total acumulado:

$$\mathbf{e}_{\text{semanal}} = \mathbf{e}_{\text{lun}} + \mathbf{e}_{\text{mar}} + \cdots + \mathbf{e}_{\text{dom}}$$

  Esto permite reportar totales de CH_4 y CO_2 sin perder la distinción entre gases.
- **Composición de insumos (Agroindustrial):** En la formulación de alimento balanceado, el vector nutricional final (proteína, energía, fibra) se construye sumando los aportes vectoriales de cada ingrediente (maíz, soya, núcleo vitamínico), lo que permite verificar si la mezcla final cumple con los requerimientos dietarios.
- **Corrección de Trayectoria (Mecatrónica/Robótica):** Un dron de fumigación agrícola debe volar en una ruta ideal representada por el vector de velocidad $\mathbf{v}_{ideal}$. Sin embargo, enfrenta un viento lateral representado por el vector $\mathbf{v}_{viento}$. Para mantener el curso, el sistema de control o la IA debe calcular la velocidad real resultante mediante la suma vectorial:

$$\mathbf{v}_{real} = \mathbf{v}_{ideal} + \mathbf{v}_{viento}$$

  Si el resultado desvía al dron del cultivo, la IA debe generar un vector de compensación opuesto para anular la perturbación.

En todos estos casos, la suma vectorial permite construir representaciones compuestas que capturan la complejidad de los sistemas físicos, preparándolas para su análisis mediante algoritmos de control o modelos de inteligencia artificial.

2.2. Multiplicación por un escalar

Esta operación modifica la magnitud del vector sin alterar su línea de acción. Matemáticamente, escala todas las componentes por un mismo factor α.

Definición formal. Dado un escalar $\alpha \in \mathbb{R}$ y un vector $\mathbf{x} \in \mathbb{R}^n$, el producto $\alpha\mathbf{x}$ se define como:

$$\alpha\mathbf{x} = \alpha \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \alpha x_1 \\ \vdots \\ \alpha x_n \end{pmatrix}.$$

Interpretación Geométrica: El efecto del escalar α sobre el vector original $\mathbf{x}$ depende de su valor:

- Si $|\alpha| > 1$, el vector se **alarga** (dilatación).
- Si $|\alpha| < 1$, el vector se **contrae** (compresión).
- Si $\alpha < 0$, el vector **invierte su sentido** (180°), aunque mantiene la misma línea de dirección.

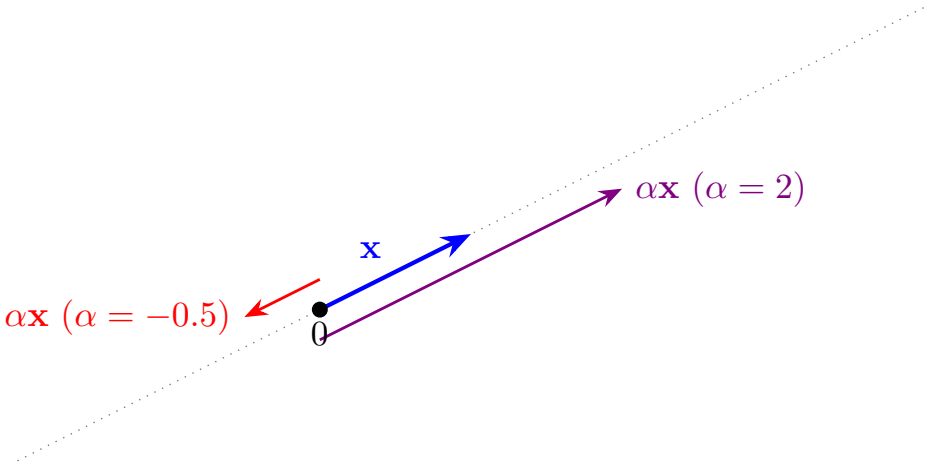


Figura 2.2: Efecto geométrico de la multiplicación escalar. Nótese cómo α=2 duplica la longitud, mientras que α=-0.5 reduce la longitud a la mitad e invierte el sentido.

Aplicaciones: Escalamiento y Control

1. **Agro (Proyección de Insumos):** Suponga que el vector $\mathbf{d}$ representa la dosis de fertilizantes para **1 hectárea**: $\mathbf{d} = [100, 50, 30]^\top$ (kg de N, P, K). Si un agricultor desea fertilizar un lote de 15 hectáreas, el requerimiento total es simplemente el vector escalado por la superficie:

$$\textbf{Total} = 15 \cdot \mathbf{d} = \begin{pmatrix} 1500 \\ 750 \\ 450 \end{pmatrix} \text{ kg.}$$

2. **Mecatrónica (Ganancia de Control):** En un sistema de control de un robot, el "error" de posición es un vector $\mathbf{e}$ (diferencia entre dónde está y dónde debería estar). El controlador aplica una corrección proporcional multiplicando ese error por una ganancia K_p (un escalar).

$$\mathbf{u} = K_p \cdot \mathbf{e}$$

Si K_p es muy grande, el robot reacciona violentamente (gran vector de fuerza); si es pequeño, reacciona suavemente.

2.3. Producto punto (producto escalar)

El **producto punto** es la operación fundamental para conectar la geometría (ángulos, longitudes) con el álgebra. Mide el grado de alineación entre dos vectores: si apuntan en la misma dirección, el valor es grande y positivo; si son perpendiculares, es cero; si apuntan en sentidos opuestos, es negativo.

Definición formal.

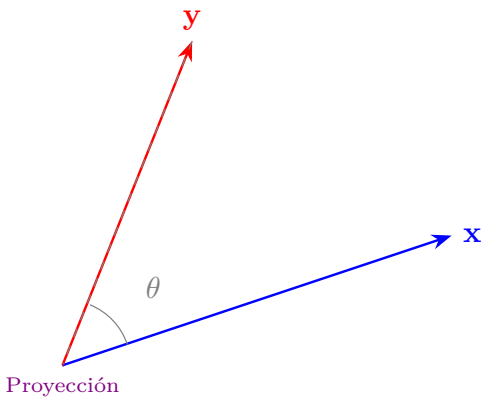
$$\cdot : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}, \quad (\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i.$$

Interpretación Geométrica y Similitud

Además de la operación algebraica componente a componente, el producto punto satisface una identidad geométrica fundamental, ilustrada en la Figura 2.3:

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos(\theta),$$

donde θ es el ángulo entre los vectores. Esta relación permite aislar el término del coseno para definir la **Similitud Coseno**, una métrica esencial en Inteligencia Artificial para medir qué tan similares son dos vectores independientemente de su magnitud:



Interpretación:

$\mathbf{x} \cdot \mathbf{y} > 0 \rightarrow$ Ángulo agudo ($\theta < 90°$)

$\mathbf{x} \cdot \mathbf{y} = 0 \rightarrow$ Ortogonales ($\theta = 90°$)

$\mathbf{x} \cdot \mathbf{y} < 0 \rightarrow$ Ángulo obtuso ($\theta > 90°$)

Figura 2.3: Relación entre magnitudes y ángulo. El producto punto conecta la longitud de los vectores con el coseno del ángulo θ que forman entre sí.

$$\cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}.$$

Como se observa en la Figura 2.4, geométricamente esto equivale a evaluar la proyección o ”sombra”de un vector sobre el otro. Si el ángulo es cero (vectores alineados), la similitud es máxima (1); si son ortogonales (90°), es nula (0).

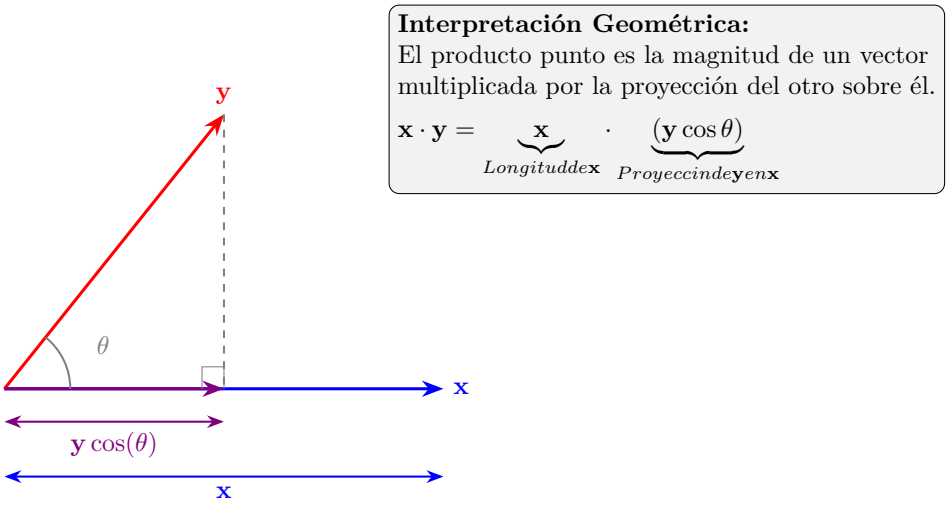


Figura 2.4: Interpretación de la proyección. La línea violeta muestra la componente de y que está “alineada” con x. El producto punto es el resultado de multiplicar esta proyección por la longitud total de x.

Aplicaciones en Ingeniería y Agro

1. **Agro (Firmas Espectrales):** En teledetección, una planta sana tiene una “firma” o vector ideal $\mathbf{v}_{sana}$ (valores de reflectancia en distintas bandas). Si el dron mide un vector actual $\mathbf{v}_{medido}$, calculamos el producto punto (normalizado) entre ambos.
   - Si $\cos(\theta) \approx 1$, la firma es casi idéntica $\rightarrow$ Planta Sana.
   - Si $\cos(\theta) \ll 1$, la alineación es baja $\rightarrow$ Posible estrés hídrico o plaga.
2. **Mecatrónica (Cálculo de Trabajo y Potencia):** Para un robot móvil, el trabajo mecánico W realizado al mover una carga es el producto punto entre el vector de fuerza aplicada $\mathbf{F}$ y el vector de desplazamiento $\mathbf{d}$:

   $$W = \mathbf{F} \cdot \mathbf{d} = \|\mathbf{F}\|\|\mathbf{d}\| \cos(\theta)$$

   Si el robot aplica fuerza perpendicular al movimiento ($\theta = 90°$), el producto punto es 0 y no se realiza trabajo útil (energía desperdiciada).
3. **Administrativo (Sistemas de Puntuación):** Sea $\mathbf{w}$ un vector de “pesos” o importancia para tres criterios (Costo, Calidad, Tiempo) y $\mathbf{x}$ el vector de puntajes de un proveedor. El puntaje total es simplemente:

   $$\text{Score} = \mathbf{w} \cdot \mathbf{x} = w_1x_1 + w_2x_2 + w_3x_3$$

   Esta es la base de las redes neuronales: una neurona realiza un producto punto entre los datos de entrada y sus pesos sinápticos.

Interpretación geométrica del Producto Punto

El producto punto ($\mathbf{x} \cdot \mathbf{y}$) es mucho más que una suma de productos de componentes: es la operación geométrica fundamental que cuantifica cuánto dos vectores apuntan en la misma dirección. Esta noción de "alineación direccional" es la base de técnicas esenciales en inteligencia artificial, como la similitud coseno, la proyección ortogonal y el Análisis de Componentes Principales (PCA).

La definición geométrica del producto punto para dos vectores $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ es:

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos(\theta),$$

donde $\|\mathbf{x}\|$ y $\|\mathbf{y}\|$ son las longitudes euclidianas de los vectores, y $\theta \in [0, \pi]$ es el ángulo entre ellos.

Esta fórmula revela tres casos clave:

- Si $\theta = 0°$ (vectores **paralelos y en la misma dirección**), $\cos(\theta) = 1$ y el producto punto es **máximo**.
- Si $\theta = 90°$ (vectores **ortogonales**), $\cos(\theta) = 0$ y el producto punto es **cero**.
- Si $\theta = 180°$ (vectores **opuestos**), $\cos(\theta) = -1$ y el producto punto es **mínimo** (negativo).

Conexión con aplicaciones en IA y agro-ambiente. En el análisis de datos, el producto punto permite:

- Comparar perfiles de fertilización entre parcelas: si θ es pequeño, las prácticas son similares.
- Detectar animales atípicos en un hato: un vector fisiológico con θ grande respecto a la media indica una posible anomalía.
- Calcular la similitud coseno (normalizando las magnitudes): $\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \cos(\theta)$, que mide *solo* la alineación, ignorando la escala.

Esta interpretación geométrica es el primer paso hacia la comprensión de cómo los algoritmos de IA "ven" y comparan datos en espacios multidimensionales.

Dependencia de la Proyección El concepto más fundamental es la relación con la proyección ortogonal, como se ilustra en la figura 2.4. El término $\|\mathbf{y}\| \cos(\theta)$ representa precisamente la longitud de la *componente* del vector $\mathbf{y}$ que está alineada con $\mathbf{x}$ (la proyección, $\text{proj}_{\mathbf{x}}\mathbf{y}$).

Por lo tanto, el producto punto puede reescribirse como:

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \cdot (\|\mathbf{y}\| \cos(\theta)) = \|\mathbf{x}\| \cdot \|\text{proj}_{\mathbf{x}}\mathbf{y}\|.$$

El producto punto es, en esencia, la magnitud de $\mathbf{x}$ multiplicada por la longitud de la "sombra" que $\mathbf{y}$ proyecta sobre $\mathbf{x}$. El signo de $\mathbf{x} \cdot \mathbf{y}$ depende únicamente del $\cos(\theta)$.

Casos Críticos El valor del producto punto está dominado por el coseno del ángulo θ, asumiendo que las magnitudes de los vectores son positivas:

1. **Alineación Perfecta ($\theta = 0°$):** Si $\mathbf{x}$ y $\mathbf{y}$ apuntan exactamente en la misma dirección, $\cos(0°) = 1$.

   $$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\|.$$

   El producto punto es *máximo y positivo*. En IA, esto indica máxima similitud o máxima compatibilidad direccional.

2. **Ortogonalidad ($\theta = 90°$):** Si $\mathbf{x}$ y $\mathbf{y}$ son perpendiculares (ortogonales), $\cos(90°) = 0$.

$$\mathbf{x} \cdot \mathbf{y} = 0.$$

El producto punto es *cero*, indicando que los vectores no tienen ninguna componente en común y son linealmente independientes, un principio clave en la decorrelación de datos (ej. PCA).

3. **Alineación Opuesta ($\theta = 180°$):** Si $\mathbf{x}$ y $\mathbf{y}$ apuntan en direcciones opuestas, $\cos(180°) = -1$.

$$\mathbf{x} \cdot \mathbf{y} = -\|\mathbf{x}\|\|\mathbf{y}\|.$$

El producto punto es *mínimo y negativo*, indicando la máxima disimilitud o incompatibilidad.

Síntesis y Aplicaciones Contextuales del Producto Punto

La utilidad del producto punto ($\mathbf{x}\cdot\mathbf{y}$) radica en su capacidad para actuar como una medida de *alineación multidimensional* o *similitud* entre vectores. Su valor es grande cuando las magnitudes y la alineación son altas, reflejando perfiles compatibles. Este concepto es la base de la **similitud coseno** ($\cos(\theta) = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$), ampliamente usada en motores de recomendación y clustering.

Ejemplos en Contextos de Modelado Híbrido (Agronómico y Zootécnico)

- **Ejemplo Agronómico.** Considere dos parcelas de maíz caracterizadas por el vector de insumos aplicados:

$$\mathbf{x} = \begin{pmatrix} 120 \\ 50 \\ 3 \end{pmatrix} \text{ (kg/ha de N, P, K)}, \quad \mathbf{y} = \begin{pmatrix} 100 \\ 60 \\ 4 \end{pmatrix}.$$

El producto punto $\mathbf{x} \cdot \mathbf{y} = 12\,000 + 3\,000 + 12 = 15\,012$ es alto, lo que sugiere un perfil de fertilización similar. Esta medida agrupa parcelas con prácticas comparables.

- **Ejemplo Zootécnico.** Considere dos vacas lecheras descritas por su historial productivo en una ventana de control:

$$\mathbf{x} = \begin{pmatrix} 28 \\ 4{,}2 \\ 38 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 30 \\ 4{,}0 \\ 39 \end{pmatrix},$$

donde las componentes representan días en lactancia, producción diaria (L/día) y contenido de grasa (%). El producto punto:

$$\mathbf{x} \cdot \mathbf{y} = (28)(30) + (4{,}2)(4{,}0) + (38)(39) = 2338{,}8.$$

Este valor alto refleja perfiles productivos similares: producción alta, lactancia avanzada y elevado contenido de grasa. Esta medida es útil en sistemas de segmentación de hatos.

Otros Contextos Aplicados

- **Ambiental:** Dos estaciones de monitoreo registran concentraciones medias (en $\mu g/m^3$) de $PM_{2.5}$, NO_2 y O_3: **a** y **b**. Un producto punto alto sugiere un patrón similar de contaminación, útil para agrupar zonas con fuentes emisoras comunes.
- **Administrativo:** Dos propuestas presupuestales **p** y **q** (para rubros como infraestructura, capacitación, operación) tienen un producto punto que cuantifica la *alineación de prioridades presupuestales*.
- **Diseño (Industrial/Agrícola):** Dos prototipos de invernadero $\mathbf{d}_1$ y $\mathbf{d}_2$ (descritos por área, sensores, consumo energético). Un producto punto elevado indica diseños estructuralmente similares, facilitando la clasificación de alternativas.

En resumen, el producto punto sirve como un indicador fundamental de cuán coherentes o compatibles son dos conjuntos de mediciones o características multidimensionales.

2.4. Laboratorio de Programación: Aritmética Tensorial en la Práctica

En esta sección, trasladamos las operaciones de suma, escalamiento y producto punto al código. En IA, estas operaciones no se realizan mediante bucles (*for loops*), sino a través de **operaciones vectorizadas**, las cuales aprovechan el paralelismo del procesador.

2.4.1. Implementación de Suma y Escalamiento

La suma de vectores requiere que ambos tengan la misma dimensión (*shape*). La multiplicación por un escalar, en cambio, utiliza un mecanismo llamado **Broadcasting**, donde el escalar se "difunde" sobre todos los elementos del vector.

```
import numpy as np
import torch

# 1. Suma de vectores (Agro: Integración de dosis de fertilizante)
dosis_neta = np.array([50, 20, 10])  # N, P, K inicial
suplemento = np.array([10,  5,  5])  # Refuerzo aplicado
dosis_total = dosis_neta + suplemento
print(f''Dosis Total (Vector): {dosis_total}'')

# 2. Multiplicación por Escalar (Mecatrónica: Control de Ganancia)
# Escalar una señal de sensor de torque
torque_raw = torch.tensor([1.2, 0.8, 1.5])
ganancia = 2.5
torque_ajustado = ganancia * torque_raw
print(f''Torque ajustado: {torque_ajustado}'')
```

Listing 2.1: Suma de vectores y multiplicación por escalar

2.4.2. El Producto Punto: Cuantificando la Afinidad

El producto punto es la operación más importante en IA. En Python, podemos ejecutarlo usando el operador `@` (recomendado en versiones modernas) o las funciones específicas de las librerías.

```
# Vectores de ejemplo (Administración: Gastos vs Presupuesto)
unidades = np.array([10, 5, 20])   # Cantidad de productos comprados
precios = np.array([1.5, 10.0, 0.5]) # Precio unitario por categoría

```

```
# Producto punto: Suma de (unidades[i] * precios[i])
gasto_total = np.dot(unidades, precios)
# Forma alternativa (estándar en álgebra lineal de Python):
gasto_total_alt = unidades @ precios

print(f''Gasto total calculado via producto punto: ${gasto_total}'')
```

Listing 2.2: Cálculo del Producto Punto

2.4.3. Aplicaciones Sectoriales en Código

Mecatrónica: Resultante de Fuerzas En un brazo robótico, si dos motores ejercen fuerzas representadas por los vectores $\mathbf{f}_1$ y $\mathbf{f}_2$, la fuerza resultante sobre el efector final es simplemente $f_res = f1 + f2$. La magnitud de esta fuerza se obtiene con `np.linalg.norm(`f_res`)`.

Agroindustrial: Mezclas y Diluciones Si un vector $\mathbf{v}$ representa la concentración de azúcares y acidez de un lote de jugo, y queremos diluirlo al 50 %, aplicamos $nuevo_lote = 0.5 * v$. El producto punto se usa para calcular el costo total de la mezcla si tenemos un vector de precios por litro de cada componente.

Resumen de Funciones Clave

Operación	**Sintaxis NumPy**	**Sintaxis PyTorch**
Suma	`a + b`	`a + b`
Escalamiento	`k * a`	`k * a`
Producto Punto	`np.dot(a, b)`	`torch.dot(a, b)`
Producto de Matriz	`a @ b`	`torch.matmul(a, b)`

Cuadro 2.1: Comparativa de funciones para aritmética tensorial.

2.5. Multiplicación Matriz-Vector: El Motor de las Redes Neuronales

Antes de entrar en el formalismo, entendamos la intuición: si un vector representa un dato (ej. una parcela, un animal, una propuesta), multiplicar ese vector por una matriz es equivalente a procesar ese dato a través de un banco de filtros. La matriz toma la información cruda, mezcla sus componentes según ciertas reglas (pesos) y produce una nueva representación más útil para la toma de decisiones.

Esta operación es el corazón de las redes neuronales profundas (Deep Learning).

Definición formal Sea $\mathbf{W} \in \mathbb{R}^{m \times n}$ una matriz y $\mathbf{x} \in \mathbb{R}^n$ un vector de entrada. Su producto es un nuevo vector $\mathbf{z} \in \mathbb{R}^m$ definido por:

$$\mathbf{z} = \mathbf{W}\mathbf{x} = \begin{pmatrix} \mathbf{fila}_1(\mathbf{W}) \cdot \mathbf{x} \\ \vdots \\ \mathbf{fila}_m(\mathbf{W}) \cdot \mathbf{x} \end{pmatrix}.$$

Cada entrada z_i es el **producto punto** entre la fila i-ésima de la matriz y el vector $\mathbf{x}$. Para que la operación sea válida, el número de columnas de la matriz (n) debe coincidir con la dimensión de entrada del vector.

Ejemplo Teórico: Transformación de Insumos a Nutrientes Imaginemos un sistema inteligente para la formulación de raciones en ganado lechero. Queremos calcular el aporte nutricional total a partir de una mezcla de ingredientes.

1. El Vector de Entrada (x): Representa la cantidad de materia prima (en kg) que vamos a utilizar en la mezcla.

$$\mathbf{x} = \begin{pmatrix} 10 \\ 5 \end{pmatrix} \begin{matrix} \text{(kg de Maíz)} \\ \text{(kg de Soya)} \end{matrix}$$

Aquí, $n = 2$ (tenemos 2 ingredientes).

2. La Matriz de Pesos (W): Representa el contenido nutricional por cada kg de ingrediente.

- La **Fila 1** corresponde a la **Proteína Cruda** (en kg/kg).
- La **Fila 2** corresponde a la **Energía Neta** (en Mcal/kg).

$$\mathbf{W} = \begin{pmatrix} 0{,}08 & 0{,}45 \\ 3{,}20 & 2{,}80 \end{pmatrix} \begin{matrix} \leftarrow \text{Perfil de Proteína} \\ \leftarrow \text{Perfil de Energía} \end{matrix}$$

Aquí, $m = 2$ (tenemos 2 métricas de salida). Note que las columnas ($n = 2$) coinciden con los ingredientes (Maíz y Soya).

3. El Cálculo ($\mathbf{z} = \mathbf{Wx}$): Aplicamos la definición del producto punto fila por columna:

$$\mathbf{z} = \begin{pmatrix} (\text{Fila}_1 \cdot \mathbf{x}) \\ (\text{Fila}_2 \cdot \mathbf{x}) \end{pmatrix} = \begin{pmatrix} (0{,}08)(10) + (0{,}45)(5) \\ (3{,}20)(10) + (2{,}80)(5) \end{pmatrix}$$

$$\mathbf{z} = \begin{pmatrix} 0{,}8 + 2{,}25 \\ 32{,}0 + 14{,}0 \end{pmatrix} = \begin{pmatrix} 3{,}05 \\ 46{,}0 \end{pmatrix} \begin{matrix} \text{ kg de Proteína Total} \\ \text{ Mcal de Energía Total} \end{matrix}$$

Interpretación en Inteligencia Artificial: En este ejemplo, la matriz **W** codifica el "conocimiento" del sistema sobre los alimentos.

- En una Red Neuronal, **x** serían los datos de entrada, **W** serían los *pesos sinápticos* aprendidos durante el entrenamiento, y **z** sería la activación resultante.
- La operación **Wx** transforma el espacio de "kilos de comida" al espacio de "requerimientos nutricionales".

Enfoque en Redes Neuronales: La Capa Densa En el contexto de la inteligencia artificial, esta operación no es solo álgebra; es la definición de una **capa densa** (fully connected layer). La ecuación fundamental que ejecuta una neurona artificial (antes de la activación no lineal) es:

$$\mathbf{z} = \mathbf{Wx} + \mathbf{b},$$

donde cada elemento cumple un rol biológico-computacional preciso:

- **x** (Entradas): Son las señales recibidas (ej. variables del cultivo o píxeles de una imagen).
- **W** (Matriz de Pesos): Es la "memoria" del modelo. Cada fila de **W** representa una neurona, y sus valores indican qué tanto importa cada entrada para esa neurona específica.
- **z** (Activaciones): Es la respuesta de las neuronas ante el estímulo **x**.

Ejemplo aplicado: Neuronas "Expertas" en Evaluación Administrativa Imagine que una IA administrativa debe evaluar automáticamente propuestas de proyectos. El vector de entrada $\mathbf{x}$ contiene $n = 4$ variables presupuestales: (infraestructura, capacitación, operación, monitoreo).

La red neuronal tiene una capa con $m = 2$ neuronas, donde cada una se ha especializado (aprendido) para detectar un criterio diferente:

1. **Neurona 1:** Evalúa el "Equilibrio Estratégico".
2. **Neurona 2:** Evalúa la "Sostenibilidad Ambiental".

La matriz de pesos $\mathbf{W} \in \mathbb{R}^{2\times 4}$ codifica estas prioridades:

$$\mathbf{W} = \begin{pmatrix} 0{,}3 & 0{,}3 & 0{,}3 & 0{,}1 \\ 0{,}1 & 0{,}2 & 0{,}4 & 0{,}3 \end{pmatrix}.$$

Si llega una propuesta con el siguiente perfil de inversión (en millones):

$$\mathbf{x} = (300, 80, 220, 75)^\top.$$

El paso hacia adelante (*forward pass*) de la red calcula:

$$\mathbf{z} = \mathbf{W}\mathbf{x} = \begin{pmatrix} (0{,}3)(300) + (0{,}3)(80) + (0{,}3)(220) + (0{,}1)(75) \\ (0{,}1)(300) + (0{,}2)(80) + (0{,}4)(220) + (0{,}3)(75) \end{pmatrix} = \begin{pmatrix} 187{,}5 \\ 156{,}5 \end{pmatrix}.$$

Interpretación del resultado: El vector de salida $\mathbf{z}$ nos dice que esta propuesta tiene una puntuación alta en equilibrio (187,5) y moderada en sostenibilidad (156,5).

Por qué esto es fundamental Este mecanismo permite que una red neuronal transforme datos brutos en conceptos abstractos. En este ejemplo, pasamos de "dinero en rubros" (espacio de entrada $\mathbb{R}^4$) a "calidad estratégica" (espacio de características $\mathbb{R}^2$). Durante el entrenamiento, el algoritmo de *backpropagation* ajusta los valores de la matriz $\mathbf{W}$ para minimizar el error en la evaluación, "aprendiendo" así los pesos ideales para clasificar proyectos correctamente.

2.6. Producto de Matrices

Las matrices constituyen la estructura algebraica fundamental para representar datos tabulares, transformaciones lineales y relaciones entre variables en el contexto de la Inteligencia Artificial. A continuación, se detallan las operaciones y propiedades esenciales para el modelado de datos.

La multiplicación de matrices actúa como el motor computacional del aprendizaje profundo. Esta operación no solo generaliza el producto punto, sino que permite ejecutar múltiples operaciones simultáneamente (lo que conocemos como procesamiento en lote o *batch*) y realizar la composición de transformaciones lineales.

Regla de Dimensiones

Para que el producto matricial $\mathbf{AB}$ esté definido, las dimensiones internas deben coincidir (es decir, el número de columnas de la primera debe igualar al número de filas de la segunda):

$$\underbrace{\mathbf{A}}_{m\times \mathbf{n}} \times \underbrace{\mathbf{B}}_{\mathbf{n}\times p} = \underbrace{\mathbf{C}}_{m\times p}$$

Definición Formal Dadas $\mathbf{A} \in \mathbb{R}^{m \times n}$ y $\mathbf{B} \in \mathbb{R}^{n \times p}$, su producto $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$ se define entrada por entrada. Como se ilustra conceptualmente en la Figura 2.5, el valor c_{ij} se obtiene mediante el producto punto entre la fila i de $\mathbf{A}$ y la columna j de $\mathbf{B}$:

$$c_{ij} = \mathbf{fila}_i(\mathbf{A}) \cdot \mathbf{columna}_j(\mathbf{B}) = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

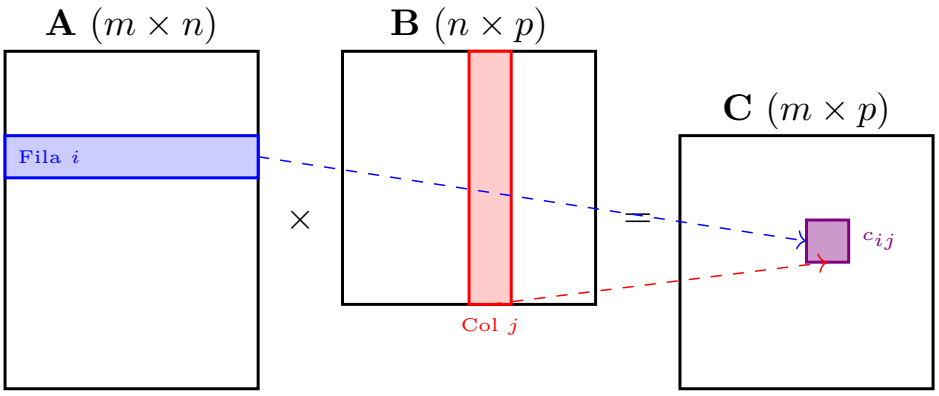


Figura 2.5: Visualización del producto matricial. El elemento resultante c_{ij} captura la interacción total entre la fila i de la matriz izquierda y la columna j de la matriz derecha.

Propiedades clave:

- **No Conmutatividad:** En general, $\mathbf{AB} \neq \mathbf{BA}$. El orden importa: rotar y luego trasladar no es lo mismo que trasladar y luego rotar.
- **Asociatividad:** $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$. Esto es vital en Deep Learning para optimizar el cómputo en capas profundas.

  Ejemplo Práctico: Lotes × Proveedores Imaginemos que queremos calcular costos para diferentes escenarios.

  - **A** (Requerimientos): 2 Lotes (filas) necesitan cantidades de 3 insumos (columnas: N, P, K).
  - **B** (Precios): Esos 3 insumos tienen precios diferentes en 2 Proveedores distintos (columnas).

$$\mathbf{A}_{(2\times3)} = \begin{pmatrix} 10 & 20 & 5 \\ 15 & 10 & 2 \end{pmatrix}, \quad \mathbf{B}_{(3\times2)} = \begin{pmatrix} 2 & 3 \\ 4 & 4 \\ 10 & 8 \end{pmatrix}$$

  El producto $\mathbf{C} = \mathbf{AB}$ nos dará una matriz de 2×2 donde cada elemento c_{ij} es el **costo total** del Lote i comprando al Proveedor j.

  Realizamos los cálculos para cada celda:

$$\begin{aligned} c_{11} &= (10 \cdot 2) + (20 \cdot 4) + (5 \cdot 10) = 20 + 80 + 50 = 150 \\ c_{12} &= (10 \cdot 3) + (20 \cdot 4) + (5 \cdot 8) = 30 + 80 + 40 = 150 \\ c_{21} &= (15 \cdot 2) + (10 \cdot 4) + (2 \cdot 10) = 30 + 40 + 20 = 90 \\ c_{22} &= (15 \cdot 3) + (10 \cdot 4) + (2 \cdot 8) = 45 + 40 + 16 = 101 \end{aligned}$$

El resultado final es:

$$\mathbf{C} = \begin{pmatrix} 150 & 150 \\ 90 & 101 \end{pmatrix}$$

Interpretación para toma de decisiones:

- Para el **Lote 1** (Fila 1), ambos proveedores resultan en el mismo costo total ($150), aunque los precios unitarios sean distintos.
- Para el **Lote 2** (Fila 2), es más económico comprar al **Proveedor 1** ($90) que al Proveedor 2 ($101).

Esta operación permite evaluar múltiples escenarios económicos de forma simultánea.

2.7. Escalares asociados: Traza, Determinante y Rango

Antes de analizar propiedades más complejas, definimos tres escalares que resumen la estructura de una matriz cuadrada $\mathbf{A} \in \mathbb{R}^{n \times n}$:

- **Traza (**tr**):** La suma de los elementos de la diagonal principal. En matrices de covarianza, representa la *varianza total* del sistema.
- **Determinante (**det**):** Una medida del cambio de volumen que produce la matriz como transformación lineal. Si $\det(\mathbf{A}) = 0$, la matriz "aplasta" el espacio y destruye información.
- **Rango:** El número máximo de filas o columnas linealmente independientes. Indica la cantidad de información no redundante en los datos.

El Determinante El **determinante** es una función escalar que asigna a cada matriz cuadrada $\mathbf{A} \in \mathbb{R}^{n \times n}$ un número real, denotado como $\det(\mathbf{A})$ o $|\mathbf{A}|$. Este valor condensa información crítica sobre la naturaleza geométrica y algebraica de la matriz.

Interpretación geométrica En el contexto del análisis de datos, el determinante representa el **factor de escala** del volumen (o área en 2D) cuando la matriz $\mathbf{A}$ actúa como una transformación lineal.

- Si $|\det(\mathbf{A})| > 1$, la transformación expande el espacio.
- Si $0 < |\det(\mathbf{A})| < 1$, la transformación contrae el espacio.
- Si $\det(\mathbf{A}) = 0$, la transformación "aplasta" el volumen hasta convertirlo en una superficie, línea o punto (pérdida de dimensionalidad).

Cálculo en 2×2 Para una matriz de $\mathbb{R}^{2 \times 2}$, la fórmula es:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Singularidad e Invertibilidad La propiedad más importante para la inteligencia artificial es su relación con la inversión de matrices:

$$\mathbf{A} \text{ es invertible } \iff \det(\mathbf{A}) \neq 0.$$

Una matriz con determinante cero se llama **singular**. En términos de datos, esto implica que las filas (o columnas) son linealmente dependientes, es decir, existe redundancia perfecta en la información (colinealidad).

Ejemplo Agro-Ambiental: Detección de Redundancia Suponga que intentamos modelar el crecimiento de un cultivo usando dos variables que creemos distintas: x_1 (agua de riego en L) y x_2 (tiempo de riego en minutos). Sin embargo, si el sistema de riego tiene un flujo constante, x_1 es exactamente proporcional a x_2.

La matriz de correlación o covarianza de estos datos tendría la forma:

$$\mathbf{C} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Calculando el determinante:

$$\det(\mathbf{C}) = (1)(1) - (1)(1) = 0.$$

El determinante nulo nos alerta matemáticamente de que no tenemos dos dimensiones reales de información, sino solo una. Intentar invertir esta matriz para un modelo de regresión lineal generará un error computacional, indicando que debemos eliminar una de las variables redundantes antes de entrenar el modelo.

2.8. Inversa de una matriz

La **inversa** de una matriz cuadrada $\mathbf{A} \in \mathbb{R}^{n\times n}$, denotada $\mathbf{A}^{-1}$, es la matriz única que satisface:

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n,$$

donde $\mathbf{I}_n$ es la matriz identidad (con 1s en la diagonal y 0s fuera).

Condición de existencia Una matriz $\mathbf{A}$ es invertible (o no singular) si y solo si cumple cualquiera de estas condiciones equivalentes:

- $\det(\mathbf{A}) \neq 0$,
- Su rango es completo ($\text{rango}(\mathbf{A}) = n$),
- Sus columnas son linealmente independientes (no hay redundancia perfecta entre variables).

Relevancia en IA

- **Regresión Lineal:** Los coeficientes óptimos se estiman como $\boldsymbol{\beta} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$.
- **Distancia de Mahalanobis:** $\sqrt{(\mathbf{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$. Usada para detectar outliers multivariados (ej. animales enfermos con patrones fisiológicos atípicos).

Ejemplo agronómico. Para predecir rendimiento (y) a partir de N y P ($\mathbf{X}$), necesitamos calcular $(\mathbf{X}^\top\mathbf{X})^{-1}$. Si

$$\mathbf{X}^\top\mathbf{X} = \begin{pmatrix} 5 & 2 \\ 2 & 2 \end{pmatrix} \implies \det = 10 - 4 = 6 \neq 0.$$

Como el determinante es no nulo, la inversa existe y el modelo tiene solución única:

$$(\mathbf{X}^\top\mathbf{X})^{-1} = \frac{1}{6}\begin{pmatrix} 2 & -2 \\ -2 & 5 \end{pmatrix}.$$

2.9. Implementación en Python: Operaciones Matriciales

En el ecosistema de Python científico (NumPy), las operaciones matriciales están altamente optimizadas. A diferencia de otros lenguajes donde se requieren bucles, aquí operamos directamente sobre las estructuras de datos.

2.9.1. El Operador Producto (@)

Desde Python 3.5, el estándar para la multiplicación de matrices es el operador arroba (`@`). Este operador verifica automáticamente la consistencia de las dimensiones internas.

2.9.2. Álgebra Lineal con `numpy.linalg`

Para operaciones más avanzadas como determinantes, trazas e inversas, utilizamos el submódulo de álgebra lineal. A continuación, se presenta una implementación completa.

```python
import numpy as np

# --- 1. PRODUCTO MATRICIAL (@) ---
# Matriz A (2x3) y B (3x2)
A = np.array([[10, 20, 5],
              [15, 10, 2]])
B = np.array([[2, 3],
              [4, 4],
              [10, 8]])

# Producto punto generalizado
C = A @ B
print(f"Producto C (2x2):\n{C}\n")

# --- 2. PROPIEDADES ---
# Ejemplo A: Matriz 2x2
M_2x2 = np.array([[4, 1],
                  [2, 3]])

# Traza (suma diagonal) y Determinante
traza_2 = np.trace(M_2x2)
det_2   = np.linalg.det(M_2x2)

print(f"Matriz 2x2 -> Traza: {traza_2}, Det: {det_2:.2f}")

# --- 3. INVERSA Y VERIFICACIÓN ---
M_3x3 = np.array([[1, 0, 2], [0, 3, 1], [2, 1, 0]])
det_3 = np.linalg.det(M_3x3)

if det_3 != 0:
    M_inv = np.linalg.inv(M_3x3)

    # VERIFICACIÓN: A @ A_inv = Identidad
    # Usamos allclose por precisión flotante
    identidad_calc = M_3x3 @ M_inv
    es_identidad = np.allclose(identidad_calc, np.eye(3))

    print(f"¿Es Identidad? {es_identidad}")
else:
    print("Matriz singular.")
```

Listing 2.3: Operaciones matriciales en NumPy

△ Atención: Nota sobre Punto Flotante

Al calcular la inversa, es común obtener números como `0.9999999` en lugar de `1.0` debido a la precisión finita de las computadoras. Por eso, en lugar de comparar con `==`, utilizamos `np.allclose()` para verificar si el resultado es matemáticamente correcto dentro de una tolerancia aceptable.

Errores comunes y buenas prácticas

1. **El operador correcto:** No confunda el operador `@` con el asterisco `*`.
   - `A @ B`: Producto matricial (fila por columna).
   - `A * B`: Producto elemento a elemento (requiere mismas dimensiones exactas o broadcasting).
2. **Gestión de dimensiones:** Si intenta multiplicar matrices incompatibles, NumPy arrojará un error. Es vital verificar siempre `.shape`.

```
# Intentar multiplicar A por sí misma: (2x3) @ (2x3)
try:
    Error = A @ A
except ValueError as e:
    print(''Error de dimensión:'', e)

# Salida:
# ValueError: matmul: Input operand 1 has a mismatch in its core
   dimension 0...
# (size 3 is different from 2)
```

Nota de Ingeniería

En el código de redes neuronales (como TensorFlow o PyTorch), la operación `capa_oculta @ pesos` ocurre millones de veces por segundo. La eficiencia de esta operación es la razón por la que usamos GPUs (Tarjetas Gráficas), ya que están diseñadas por hardware para realizar multiplicaciones de matrices en paralelo masivo.

2.10. Operaciones con Tensores en Bioingeniería

A diferencia de las matrices, donde el producto punto es la estrella, en los tensores operamos frecuentemente con **filtrado (convolución)**, **operaciones elemento a elemento** (Hadamard) y **re-dimensionamiento** (Reshaping).

Caso de Estudio: Detección de Tumores en MRI Una Resonancia Magnética (MRI) del cerebro no es una foto plana; es una volumetría. Podemos representarla como un tensor $\mathcal{X}$ de dimensiones $256 \times 256 \times 120$:

- 256×256: Resolución de cada "rebanada" (slice) de imagen (alto $\times$ ancho).
- 120: El número de rebanadas tomadas desde la base del cráneo hasta la coronilla (profundidad).

Operación 1: Aplicación de una Máscara (Producto Hadamard) Para aislar el cerebro y eliminar el cráneo o el fondo, multiplicamos el tensor de la imagen $\mathcal{X}$ por un tensor binario "máscara" $\mathcal{M}$ (donde 1 es tejido cerebral y 0 es hueso/fondo).

$$\mathcal{Y} = \mathcal{X} \odot \mathcal{M} \quad \Longrightarrow \quad y_{ijk} = x_{ijk} \cdot m_{ijk}$$

Esta operación se realiza simultáneamente en los 7.8 millones de vóxeles (píxeles 3D).

Operación 2: Aplanado (Flattening) para Diagnóstico Una red neuronal clásica no puede "tragar" un cubo. Debemos convertir el tensor 3D en un vector largo 1D para clasificarlo (ej. ¿Hay tumor? Sí/No).

$$\text{Flatten}(\mathbb{R}^{256\times256\times120}) \to \mathbb{R}^{7,864,320}$$

Este proceso de reestructurar los datos sin perder información es la base de la arquitectura de redes convolucionales (CNN).

Implementación: Manipulación de Vóxeles en Python

En Python, bibliotecas como `NumPy`, `TensorFlow` o `PyTorch` tratan estas estructuras de forma nativa. Nótese que aquí usamos el término `shape` (forma) para describir las dimensiones del tensor.

```python
import numpy as np

# 1. Simular una MRI cerebral (Tensor 3D)
# Dimensiones: (Alto, Ancho, Profundidad)
# Valores aleatorios simulando intensidad de señal
mri_tensor = np.random.rand(256, 256, 120)

print(f''Forma original del tensor: {mri_tensor.shape}'')
# Salida: (256, 256, 120)

# 2. Operación de Slicing (Rebanado)
# El médico quiere ver solo la rebanada central (corte axial)
corte_central = mri_tensor[:, :, 60]

print(f''Forma del corte 2D: {corte_central.shape}'')
# Salida: (256, 256) -> Ahora es una matriz clásica

# 3. Operación de Máscara (Thresholding)
# Queremos resaltar solo tejidos con alta intensidad (posibles anomalías)
# Creamos una máscara booleana (Tensor de True/False)
mascara_tejido = mri_tensor > 0.8

# Aplicamos la máscara (Hadamard product implícito)
tejido_resaltado = mri_tensor * mascara_tejido

# 4. Flattening (Preparar para IA)
input_vector = mri_tensor.flatten()

print(f''Vector de entrada para la Red Neuronal: {input_vector.shape}'')
# Salida: (7864320,) -> Un vector gigante
```

Listing 2.4: Procesamiento de un tensor volumétrico (MRI Simulado)

Advertencia de Memoria

El peligro de los tensores es la explosión combinatoria. Un tensor 3D pequeño (256^3) consume pocos MB, pero añadir una dimensión más (ej. tiempo en un video 4K) puede desbordar la memoria RAM de cualquier computadora estándar. Por eso, en IA, el diseño eficiente de la **shape** del tensor es crítico.

2.11. Ejercicios y Proyectos Computacionales

En esta sección, saldremos de la teoría abstracta para resolver problemas reales de ingeniería agroambiental. El objetivo es modelar situaciones de campo utilizando vectores y matrices, y resolverlas mediante la implementación computacional en Python.

2.11.1. Ejercicio 1: Planificación de Fertilizantes (Combinación Lineal)

♠ Nutrición de Suelos

Un ingeniero agrónomo dispone de tres tipos de fertilizantes comerciales con distintas concentraciones de Nitrógeno (N), Fósforo (P) y Potasio (K). Necesitamos calcular el perfil nutricional final al mezclar distintas cantidades de estos productos.

Datos del problema: Los fertilizantes tienen las siguientes composiciones (vectores en $\mathbb{R}^3$):

- **Fertilizante A (Urea+):** $\mathbf{v}_A = [46, 0, 0]$
- **Fertilizante B (DAP):** $\mathbf{v}_B = [18, 46, 0]$
- **Fertilizante C (Potasa):** $\mathbf{v}_C = [0, 0, 60]$

Tarea Computacional: Escriba un script en Python que:

1. Defina estos vectores como arreglos de NumPy.
2. Calcule el vector de nutrientes totales $\mathbf{v}_{total}$ si se aplican: 10 kg del A, 5 kg del B y 8 kg del C.
3. Imprima el resultado interpretado (Total de N, P y K).

```python
import numpy as np

# Definicion de vectores
v_A = np.array([46, 0, 0])
v_B = np.array([18, 46, 0])
v_C = np.array([0, 0, 60])

# Coeficientes (escalares en kg)
c_A, c_B, c_C = 10, 5, 8

# Combinacion Lineal
v_total = (c_A * v_A) + (c_B * v_B) + (c_C * v_C)

print(f"Perfil Nutricional Total (N-P-K): {v_total}")
```

2.11.2. Ejercicio 2: Eficiencia Energética Solar (Producto Punto)

♠ Automatización de Invernaderos

Para maximizar la fotosíntesis en un invernadero automatizado o la energía en paneles solares, el ángulo de incidencia de la luz es vital. La eficiencia es máxima cuando los rayos golpean perpendicularmente la superficie. Matemáticamente, esto se relaciona con el coseno del ángulo entre el vector normal de la superficie y el vector de la luz.

Tarea Computacional: Dados dos vectores en 3D que representan la dirección de los rayos solares ($\mathbf{s}$) y la orientación normal del panel ($\mathbf{p}$):

$$\mathbf{s} = [2, 3, -5] \quad \text{y} \quad \mathbf{p} = [1, 2, 1]$$

1. Calcule la norma (magnitud) de ambos vectores. 2. Calcule el producto punto $\mathbf{s} \cdot \mathbf{p}$. 3. Encuentre el ángulo θ entre ellos usando la fórmula:

$$\cos(\theta) = \frac{\mathbf{s} \cdot \mathbf{p}}{\|\mathbf{s}\|\|\mathbf{p}\|}$$

4. Determine si la orientación es eficiente (consideraremos eficiente si $\theta < 45°$ o $> 135°$, dependiendo de la dirección de referencia).

2.11.3. Ejercicio 3: Predicción de Cosecha (Multiplicación Matriz-Vector)

♠ Modelado Predictivo

Las redes neuronales simples a menudo comienzan como una multiplicación matricial. Supongamos que queremos predecir el rendimiento (toneladas/ha) de 3 lotes diferentes basándonos en 2 variables: Humedad del suelo y Horas de sol.

Modelo Matemático:

$$\mathbf{Y} = \mathbf{X} \cdot \mathbf{w} + b$$

Donde:

- $\mathbf{X}$ es la matriz de datos (3 lotes $\times$ 2 variables).
- $\mathbf{w}$ es el vector de "pesos.º importancia ($\mathbf{w} \in \mathbb{R}^2$).
- b es el sesgo (bias), un ajuste base.

Datos:

$$\mathbf{X} = \begin{bmatrix} 0.8 & 120 \\ 0.6 & 150 \\ 0.9 & 100 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} 5 \\ 0.02 \end{bmatrix}, \quad b = 1.5$$

Tarea Computacional: Implemente esta operación usando el operador '@' de Python o 'torch.matmul'. ¿Cuál es el rendimiento predicho para el segundo lote?

2.11.4. Ejercicio 4: El Problema de la Mezcla Inversa (Sistemas Lineales)

♠ Ingeniería Inversa

A menudo sabemos qué resultado queremos (ej. un suelo con 50ppm de Nitrógeno y 30ppm de Fósforo) y necesitamos saber cuánto fertilizante comprar. Esto requiere invertir la matriz de composición.

Se tiene el sistema lineal $\mathbf{Ax} = \mathbf{b}$, donde:

- **A**: Matriz de composición de 2 fertilizantes (columnas) para 2 nutrientes (filas).
- **b**: Vector de requerimientos del suelo (lo que queremos lograr).
- **x**: Cantidad desconocida de cada fertilizante a aplicar.

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.2 \\ 0.1 & 0.4 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 10 \\ 5 \end{bmatrix}$$

Tarea: 1. Verifique si la matriz **A** es invertible calculando su determinante (‘np.linalg.det‘). 2. Si es distinto de cero, calcule **x** usando la inversa: $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. 3. Verifique su resultado recalculando $\mathbf{A} \cdot \mathbf{x}$ y comparándolo con **b**.

Capítulo 3

Matrices Especiales y sus Propiedades

En el mundo del Álgebra Lineal Computacional, no todas las matrices son iguales. Existen ciertas estructuras "privilegiadas" que hacen que los algoritmos de Inteligencia Artificial funcionen miles de veces más rápido o garanticen que un entrenamiento converja a una solución estable. En este capítulo, exploraremos estas joyas matemáticas.

3.1. Matrices Simétricas: El Espejo de los Datos

Antes de entrar en definiciones rigurosas, pensemos en una red de sensores en un cultivo. La distancia entre el Sensor A y el Sensor B es la misma que entre el B y el A. Esta reciprocidad es la esencia de la simetría.

3.1.1. Definición Formal

Una matriz cuadrada $\mathbf{A} \in \mathbb{R}^{n \times n}$ se dice *simétrica* si es igual a su transpuesta:

$$\mathbf{A} = \mathbf{A}^\top$$

Esto implica que sus elementos reflejan esa igualdad respecto a la diagonal principal: $a_{ij} = a_{ji}$ para todo i, j.

♠ Redes de Sensores y Adyacencia

Imagine que monitoreamos una plantación con 3 torres de control comunicadas entre sí. Queremos representar la distancia (o la calidad de la señal) entre ellas.

$$\mathbf{D} = \begin{bmatrix} 0 & 50 & 120 \\ 50 & 0 & 80 \\ 120 & 80 & 0 \end{bmatrix}$$

Note que $d_{12} = 50$ (Torre 1 a Torre 2) es igual a $d_{21} = 50$. Esta matriz es simétrica. En IA, las matrices de correlación y covarianza (que veremos más adelante) son siempre simétricas.

3.1.2. Propiedad Espectral (Teorema Espectral)

La razón por la que las matrices simétricas son tan importantes en IA, no es solo estética. Tienen una propiedad computacional vital: Sus autovalores[1] son siempre números reales (no

[1] Los autovalores son valores λ que satisfacen $|A - \lambda I| = 0$

complejos) y sus autovectores son ortogonales entre sí. Esto garantiza estabilidad numérica.

3.2. Matrices Definidas Positivas: La Energía del Sistema

Este es quizás el concepto más importante para entender la *optimización* (cómo aprenden las redes neuronales).

Imagine una función de costo (error) que tiene forma de "cuenco" o "tazón". Si soltamos una canica (el algoritmo), esta caerá inevitablemente al fondo (el mínimo error). Una matriz Definida Positiva garantiza que esa superficie tenga esa forma de cuenco perfecto, sin puntos de silla ni caídas al infinito.

3.2.1. Definición Formal

Una matriz simétrica $\mathbf{A}$ es **Definida Positiva (DP)** si para cualquier vector no nulo $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} > 0$$

El término $\mathbf{x}^\top \mathbf{A}\mathbf{x}$ se conoce como *Forma Cuadrática*. Si el resultado es ≥ 0, se llama **Semidefinida Positiva (SDP)**.

♠ Función de Costo en Riego Automatizado

Queremos minimizar el gasto de energía (J) de una bomba de agua. Este costo depende de dos variables de control: la velocidad del motor (v) y la presión de salida (p).

$$\mathbf{x} = \begin{bmatrix} v \\ p \end{bmatrix}$$

El modelo de costo energético se define como una forma cuadrática:

$$J(\mathbf{x}) = \mathbf{x}^\top \mathbf{Q}\mathbf{x}$$

Supongamos un punto de operación $\mathbf{x} = [2, 4]^\top$ (2000 RPM y 4 Bar) y una matriz de coeficientes del sistema $\mathbf{Q}$:

$$\mathbf{Q} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

(Nota: $\mathbf{Q}$ es simétrica y sus autovalores son positivos, por lo tanto es Definida Positiva).
Cálculo paso a paso:
1. Primero multiplicamos la matriz $\mathbf{Q}$ por el vector $\mathbf{x}$:

$$\mathbf{Q}\mathbf{x} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} (2)(2) + (1)(4) \\ (1)(2) + (3)(4) \end{bmatrix} = \begin{bmatrix} 8 \\ 14 \end{bmatrix}$$

2. Luego multiplicamos el vector fila $\mathbf{x}^\top$ por el resultado anterior:

$$J = \begin{bmatrix} 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 14 \end{bmatrix}$$

3. Realizamos el producto punto final:

$$J = (2)(8) + (4)(14) = 16 + 56 = \mathbf{72} \text{ unidades de energía}$$

Conclusión: Dado que el resultado (72) es positivo, confirmamos la propiedad de la matriz. Al ser $\mathbf{Q}$ definida positiva, la superficie de costo tiene forma de "tazón", garantizando que el algoritmo de optimización (como el Descenso de Gradiente) siempre podrá descender hacia un mínimo estable sin oscilaciones infinitas.

3.2.2. Verificación Computacional

¿Cómo sabemos si una matriz es DP en Python? Verificamos que todos sus autovalores sean positivos.

```python
import numpy as np

# Matriz Simetrica
A = np.array([[2, -1, 0],
              [-1, 2, -1],
              [0, -1, 2]])

# Verificamos Simetria
es_simetrica = np.allclose(A, A.T)
print(f"Es simetrica: {es_simetrica}")

# Verificamos si es Definida Positiva (Autovalores > 0)
autovalores = np.linalg.eigvals(A)
es_def_positiva = np.all(autovalores > 0)

print(f"Autovalores: {autovalores}")
print(f"Es Definida Positiva: {es_def_positiva}")
```

3.3. Matrices Diagonales e Identidad

Son las matrices más ”limpias”. Toda la información está en la diagonal principal; fuera de ella, todo es ruido (ceros).

3.3.1. Definición y Ventajas

Una matriz diagonal $\mathbf{D}$ tiene $d_{ij} = 0$ si $i \neq j$.

$$\mathbf{D} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

Ventaja Computacional:

- Multiplicar por $\mathbf{D}$ es solo escalar cada elemento (barato computacionalmente).
- La inversa $\mathbf{D}^{-1}$ es trivial: solo invertimos los elementos de la diagonal $(1/\lambda_i)$.

♠ Sistemas de Cultivos Independientes

Si gestionamos 3 invernaderos aislados donde el clima de uno NO afecta al otro, la matriz del sistema es diagonal.

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}_{nuevo} = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.95 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}_{actual}$$

Aquí, el invernadero 1 retiene el 90 % del calor, el 2 el 80 %, etc. No hay ”mezcla”de términos.

3.4. Matrices Ortogonales: Rotaciones Perfectas

Las matrices ortogonales son fundamentales en robótica (mecatrónica) y en el preprocesamiento de imágenes (PCA). Representan transformaciones rígidas: rotan los datos pero no los estiran ni los deforman.

3.4.1. Definición Formal

Una matriz cuadrada $\mathbf{Q}$ es ortogonal si su transpuesta es igual a su inversa:

$$\mathbf{Q}^\top = \mathbf{Q}^{-1} \implies \mathbf{Q}^\top \mathbf{Q} = \mathbf{Q}\mathbf{Q}^\top = \mathbf{I}$$

Esto implica que las columnas de $\mathbf{Q}$ son vectores unitarios y perpendiculares entre sí.

♠ Mecatrónica: Rotación de un Brazo Robótico

Un brazo robótico recolector de frutas necesita rotar su pinza sin cambiar el tamaño de la fruta que ve. La matriz de rotación en 2D es el ejemplo clásico de matriz ortogonal:

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Si aplicamos esta matriz a un vector posición $\mathbf{v}$, la nueva posición $\mathbf{v}' = \mathbf{R}\mathbf{v}$ tendrá exactamente la misma longitud (norma) que $\mathbf{v}$. $||\mathbf{R}\mathbf{v}|| = ||\mathbf{v}||$.

3.4.2. Implementación en Python

Podemos verificar la ortogonalidad comprobando si $Q@Q.T$ es la identidad.

```
theta = np.radians(45) # Rotacion de 45 grados
c, s = np.cos(theta), np.sin(theta)

Q = np.array([[c, -s],
              [s, c]])

# Verificacion: Q @ Q.T debe ser Identidad
identidad_aprox = Q @ Q.T
print("Q @ Q.T (deberia ser Identidad):")
print(identidad_aprox)
```

3.5. Conexión Integradora: La Matriz de Covarianza

Cerramos este capítulo conectando todo. Cuando recolectamos datos del mundo real (Big Data Agroambiental), construimos la **Matriz de Covarianza** ($\mathbf{\Sigma}$).

Esta matriz es mágica porque cumple todo lo que vimos: 1. Es **Simétrica** (la covarianza de humedad vs temperatura es igual a temperatura vs humedad). 2. Es **Semidefinida Positiva** (la varianza siempre es no negativa). 3. Sus autovectores son **Ortogonales** (apuntan a las direcciones principales de variación de los datos).

Esta matriz será la protagonista en el siguiente capítulo sobre Autovalores y Análisis de Componentes Principales (PCA).

Propiedades y conexiones

- Toda matriz de covarianza $\mathbf{\Sigma}$ es **simétrica y SDP**.
- Si $\mathbf{A}$ es SDP, todos sus autovalores son no negativos ($\lambda_i \geq 0$).
- Si $\mathbf{A}$ es DP, entonces $\det(\mathbf{A}) > 0$ y es **invertible**.

Interpretación en ciencia de datos La condición SDP garantiza que la varianza calculada en cualquier dirección proyectada sea no negativa, lo cual es una **consistencia estadística fundamental**. En PCA, los autovalores de $\mathbf{\Sigma}$ representan varianzas explicadas; si fueran negativos, el modelo físico estaría roto.

Ejemplo. La matriz $\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ es simétrica y DP, ya que su determinante es $3 > 0$ y su traza es $4 > 0$ (criterio rápido para matrices 2×2).

Matrices Definidas Positivas El concepto de una matriz **definida positiva** es análogo a la idea de un número real positivo ($a > 0$), pero extendido al álgebra matricial. En ingeniería, estas matrices son fundamentales porque garantizan la estabilidad de los sistemas y la existencia de mínimos únicos en problemas de optimización (costos, energía, error).

Definición paso a paso Sea $\mathbf{A} \in \mathbb{R}^{n \times n}$ una matriz simétrica. Decimos que $\mathbf{A}$ es definida positiva si satisface la siguiente condición energética:

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} > 0, \quad \text{para todo vector } \mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq \mathbf{0}.$$

El término escalar $E = \mathbf{x}^\top \mathbf{A}\mathbf{x}$ se conoce como *forma cuadrática*. Geométricamente, si $\mathbf{A}$ es definida positiva, la gráfica de esta función cuadrática tiene forma de "tazón" o "cuenco" curvado hacia arriba, lo que implica que tiene un fondo (un mínimo global).

Criterios de identificación Para verificar si una matriz es definida positiva sin probar infinitos vectores, utilizamos dos criterios prácticos:

1. **Autovalores (Eigenvalues):** Todos los autovalores λ_i de $\mathbf{A}$ deben ser estrictamente positivos ($\lambda_i > 0$).
2. **Criterio de Sylvester:** Todos los determinantes de los sub-bloques principales superiores (los menores principales) deben ser positivos.

Aplicación en Ingeniería Agrícola: Minimización de Costos En la optimización de procesos agroindustriales, buscamos minimizar funciones de costo. La condición matemática para asegurar que hemos encontrado un **costo mínimo** (y no un máximo o un punto de silla) es que la matriz de segundas derivadas (la Matriz Hessiana) sea definida positiva.

Ejemplo práctico: Un ingeniero agrícola desea minimizar el costo operativo C de un sistema de fertirriego, el cual depende de dos variables:

- w: Cantidad de agua (m^3/ha).
- f: Cantidad de fertilizante (kg/ha).

Supongamos que el modelo de costos se aproxima localmente mediante una función cuadrática:

$$C(w, f) = 2w^2 + 2wf + 4f^2 - 100w - 200f + 5000.$$

Para verificar si este sistema tiene un costo mínimo estable, analizamos la curvatura de la función mediante su matriz Hessiana $\mathbf{H}$ (la matriz de coeficientes cuadráticos):

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 C}{\partial w^2} & \frac{\partial^2 C}{\partial w \partial f} \\ \frac{\partial^2 C}{\partial f \partial w} & \frac{\partial^2 C}{\partial f^2} \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ 2 & 8 \end{pmatrix}.$$

Verificación paso a paso:

1. **Simetría:** La matriz es simétrica ($H_{12} = H_{21} = 2$).

2. **Criterio de Autovalores:** Calculamos $\det(\mathbf{H}-\lambda\mathbf{I}) = (4-\lambda)(8-\lambda)-4 = \lambda^2-12\lambda+28 = 0$. Resolviendo, obtenemos $\lambda_1 \approx 9.4$ y $\lambda_2 \approx 2.6$.

3. **Conclusión:** Como $\lambda_1 > 0$ y $\lambda_2 > 0$, la matriz $\mathbf{H}$ es **definida positiva**.

Interpretación Ingenieril: Dado que la matriz es definida positiva, la superficie de costos es convexa (tiene forma de tazón). Esto garantiza al ingeniero que existe una única combinación óptima de agua y fertilizante que minimiza los costos operativos, permitiendo el uso de algoritmos de optimización (como el Descenso de Gradiente) con total seguridad de convergencia.

3.6. Conexión integradora: la matriz de covarianza

En ciencia de datos agro-ambiental, la matriz más omnipresente es la matriz de covarianza muestral:

$$\mathbf{\Sigma} = \frac{1}{m-1}\mathbf{X}_c^\top \mathbf{X}_c,$$

donde $\mathbf{X}_c$ es la matriz de datos centrada (media cero). Esta estructura unifica todos los conceptos anteriores:

- Es **cuadrada y simétrica** (propiedad de $\mathbf{A}^\top\mathbf{A}$).
- Es **semidefinida positiva** (reflejando la naturaleza no negativa de la dispersión de datos).
- **Invertibilidad y Colinealidad:** Si dos variables son colineales (ej. "kg de fertilizante" y "g de nitrógeno aportado"), las columnas de $\mathbf{X}$ son dependientes, el determinante de $\mathbf{\Sigma}$ cae a 0, y la matriz no se puede invertir. Esto alerta al científico de datos sobre redundancia en el modelo.

Así, el producto, la transpuesta, el determinante y la inversa no son conceptos aislados, sino herramientas coordinadas que permiten modelar, transformar y diagnosticar la calidad de los datos agro-ambientales.

3.7. Intuición Geométrica y Notación Matricial

Un sistema de ecuaciones lineales puede interpretarse geométricamente como un conjunto de planos (en $\mathbb{R}^3$) o, en general, hiperplanos (en $\mathbb{R}^n$). Resolver el sistema equivale a encontrar el punto —o conjunto de puntos— donde todos estos objetos geométricos se intersecan simultáneamente.

Según la configuración relativa, pueden darse tres situaciones fundamentales (ver Figura 3.1):

- *Solución única:* Los planos se intersecan en un único punto (sistema compatible determinado).
- *Infinitas soluciones:* Los planos comparten una recta o un eje común, como las páginas de un libro abierto (sistema compatible indeterminado).
- *Sin solución:* No existe un punto común a todos; por ejemplo, planos paralelos como los pisos de un edificio o formando un prisma triangular (sistema incompatible).

En la figura 3.2 se ilustra la interpretación en 3D, donde cada ecuación representa un plano en el espacio tridimensional. La solución del sistema corresponde al punto (o conjunto de puntos) donde estos planos se intersectan. Si la solución es única, los planos se cruzan en un solo punto; si hay infinitas soluciones, los planos se intersectan a lo largo de una línea o plano común; y si no hay solución, los planos no se intersectan en absoluto.

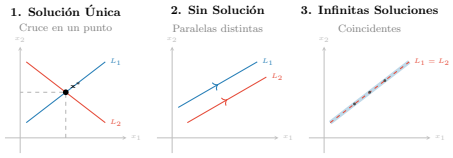


Figura 3.1: Casos en 2D (Rectas).

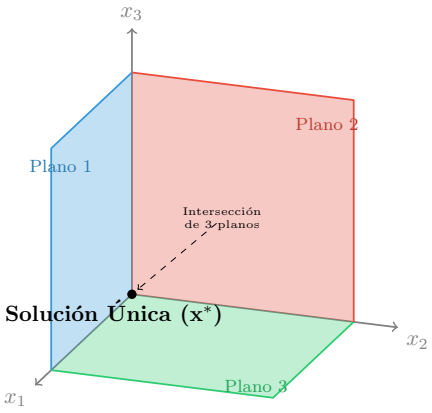


Figura 3.2: Intersección en 3D (Planos).

El concepto de Hiperplano y la Ceguera Dimensional

Mientras que en 2D visualizamos rectas y en 3D planos, los problemas reales en agroambiental (como datos satelitales con 12 bandas espectrales) o mecatrónica (un robot con 7 grados de libertad) habitan en espacios de dimensión superior.

Aquí surge el concepto de **hiperplano**: una generalización matemática que representa un subespacio "plano" de dimensión $n-1$ en un espacio de dimensión n. Aunque nuestra intuición biológica está limitada a tres dimensiones, el álgebra lineal no sufre esta restricción. La ecuación:

$$2x_1 + 5x_2 - x_3 + 8x_4 = 10$$

describe un hiperplano en $\mathbb{R}^4$. No podemos dibujarlo, pero podemos operar con él algebraicamente con la misma facilidad que con una recta. El álgebra se convierte así en nuestros "ojos" que nos permite ver y manipular estructuras en dimensiones que nuestro cerebro no puede concebir.

3.7.1. 1. Eliminación Gaussiana (El Algoritmo Paso a Paso)

Este es el algoritmo fundamental. Su objetivo es transformar un problema complejo (sistema acoplado) en uno sencillo (sistema triangular) mediante operaciones que no alteran la solución.

El proceso tiene dos fases:

1. **Eliminación hacia adelante:** Convertir la matriz original A en una matriz triangular superior U (hacer ceros debajo de la diagonal).
2. **Sustitución hacia atrás:** Despejar las incógnitas empezando por la última ecuación.

Ejemplo Práctico 3x3

Consideremos el siguiente sistema. Para manipularlo numéricamente, formamos la **matriz aumentada** $[A|\mathbf{b}]$, que incluye los términos independientes:

$$\begin{cases} 2x_1 + x_2 + x_3 = 4 \\ 4x_1 - 6x_2 = -2 \\ -2x_1 + 7x_2 + 2x_3 = 7 \end{cases} \implies \left[\begin{array}{ccc|c} \mathbf{2} & 1 & 1 & 4 \\ 4 & -6 & 0 & -2 \\ -2 & 7 & 2 & 7 \end{array}\right]$$

Paso 1: Primer Pivote (Columna 1)
Nuestro objetivo es eliminar los números debajo del primer elemento de la diagonal (el **2**, llamado pivote).

- Para eliminar el 4 (Fila 2): Hacemos $F_2 \leftarrow F_2 - 2F_1$.
- Para eliminar el -2 (Fila 3): Hacemos $F_3 \leftarrow F_3 + F_1$.

$$\left[\begin{array}{ccc|c} 2 & 1 & 1 & 4 \\ \mathbf{0} & -8 & -2 & -10 \\ \mathbf{0} & 8 & 3 & 11 \end{array}\right]$$

Paso 2: Segundo Pivote (Columna 2)
Ahora nos enfocamos en la sub-matriz restante. El nuevo pivote es el -8. Necesitamos eliminar el 8 que está debajo de él.

- Operación: $F_3 \leftarrow F_3 + F_2$.

$$\left[\begin{array}{ccc|c} 2 & 1 & 1 & 4 \\ 0 & -8 & -2 & -10 \\ 0 & \mathbf{0} & 1 & 1 \end{array}\right]$$

¡El sistema ya está triangulado! Observa que hemos obtenido ceros en el "triánguloïnferior izquierdo.

Paso 3: Sustitución hacia atrás
Reescribimos el sistema equivalente que nos ha quedado, que ahora es trivial de resolver de abajo hacia arriba:

1. **Tercera ecuación:**
$$1x_3 = 1 \implies \mathbf{x_3 = 1}$$
2. **Segunda ecuación** (sustituyendo x_3):
$$-8x_2 - 2(1) = -10 \implies -8x_2 = -8 \implies \mathbf{x_2 = 1}$$
3. **Primera ecuación** (sustituyendo x_2, x_3):
$$2x_1 + 1 + 1 = 4 \implies 2x_1 = 2 \implies \mathbf{x_1 = 1}$$

La solución del sistema es el vector $\mathbf{x} = [1, 1, 1]^\top$. Computacionalmente, este proceso tiene una complejidad de $O(n^3)$, lo que significa que si duplicamos el número de variables, el tiempo de cálculo se multiplica aproximadamente por 8.

♠ Aplicación: Balanceo de Cargas en Drones

En el diseño del chasis de un dron fumigador, las fuerzas estáticas se resuelven mediante Gauss. Como la estructura del dron no cambia, el método es directo y exacto para asegurar que los brazos soporten el tanque de líquido.

3.7.2. 2. Método de la Matriz Inversa

Teóricamente, si A es cuadrada y su determinante es no nulo ($\det(A) \neq 0$), existe una matriz A^{-1} tal que:

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Aunque matemáticamente elegante, computacionalmente es costoso. Calcular la inversa requiere muchas más operaciones que la eliminación gaussiana.

△ Atención: Advertencia Computacional

En sistemas grandes (ej. análisis de genoma vegetal o simulación de fluidos), **nunca** se calcula la inversa explícita. Es numéricamente inestable y lenta. Se prefieren métodos de descomposición.

3.7.3. 3. Descomposición LU (Lower-Upper)

Este es el método rey.[en] la ingeniería aplicada. Consiste en factorizar la matriz A en el producto de dos matrices triangulares: una inferior (L) y una superior (U).

$$A = L \cdot U$$

Esto permite resolver el sistema en dos pasos rápidos y baratos computacionalmente. Es ideal cuando tenemos una misma matriz A (ej. un robot) pero múltiples vectores $\mathbf{b}$ (diferentes posiciones objetivo).

3.8. Conexión Agro-Mecatrónica: Sensores Espectrales

♣ Calibración de Sensores Multiespectrales

Imagina un sensor que mide la salud de una planta. El sensor tiene 3 fotodiodos, pero cada uno tiene una ligera çontaminación"de otras longitudes de onda (crosstalk).

- Lectura Diodo Rojo $= 1.0 \cdot \text{RojoReal} + 0.1 \cdot \text{VerdeReal}$
- Lectura Diodo Verde $= 0.2 \cdot \text{RojoReal} + 0.9 \cdot \text{VerdeReal}$

Para recuperar los valores reales de luz (RojoReal, VerdeReal) a partir de las lecturas sucias del sensor, debemos resolver el sistema usando la matriz de calibración inversa del fabricante.

3.9. Implementación en Python (NumPy)

A diferencia del capítulo anterior donde usamos optimización (TensorFlow), aquí usaremos álgebra lineal exacta con **NumPy**, la librería base de la ciencia de datos.

```python
import numpy as np

# 1. Definir el sistema (Ejemplo de calibración de sensores)
# Matriz de coeficientes (Crosstalk del sensor)
A = np.array([
    [1.0, 0.1, 0.05], # Diodo 1 sensible a Banda 1, 2 y 3
    [0.2, 0.9, 0.1],  # Diodo 2
    [0.1, 0.2, 0.8]   # Diodo 3
])

# Vector b (Lecturas crudas del sensor)
b = np.array([500, 800, 300]) # Valores en milivoltios

# 2. Método 1: Resolución directa (Usa LU internamente - RECOMENDADO)
# Es el metodo mas rapido y estable numéricamente
x_solve = np.linalg.solve(A, b)

```

```
# 3. Método 2: Calculando la Inversa explícita (NO RECOMENDADO para N
    grande)
A_inv = np.linalg.inv(A)
x_inv = np.dot(A_inv, b)

# 4. Verificación
print("--- Resultados de Calibración ---")
print(f"Valores reales de luz: {x_solve}")

# Comprobamos si Ax = b
check = np.dot(A, x_solve)
print(f"Reconstrucción de lecturas (Check): {check}")
print(f"Error numérico: {np.allclose(check, b)}")
```

Listing 3.1: Resolución exacta de sistemas lineales con NumPy

Capítulo 4

Ajuste de Modelos: Cuando el Sistema no es Perfecto

En el capítulo anterior, asumimos que nuestros sensores eran perfectos y que podíamos encontrar una solución exacta para $A\mathbf{x} = \mathbf{b}$. Sin embargo, en un campo de cultivo real, dos plantas con la misma cantidad de agua y nutrientes pueden crecer diferente debido a factores aleatorios (genética, viento, plagas).

Aquí entramos en el terreno de la **Ciencia de Datos**: rara vez buscamos una solución exacta (que no existe); buscamos la **mejor solución aproximada**.

4.1. Intuición: El Problema del Sistema Sobredeterminado

Imagina que quieres predecir el rendimiento de maíz (y) basándote en la cantidad de Nitrógeno aplicado (x). Tomas 100 muestras en el campo. Esto genera un sistema de ecuaciones con 100 ecuaciones (una por muestra) pero solo 2 incógnitas (la pendiente y la intersección de la recta: $y = mx + c$).

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_{100} & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{100} \end{bmatrix}$$

Matemáticamente, esto es una matriz A alta y delgada (100×2). Es un sistema **sobredeterminado**. No existe una línea recta que pase exactamente por los 100 puntos a la vez. El vector $\mathbf{b}$ no vive en el espacio columna de A.

4.2. Formalización: Ecuaciones Normales

Ya que no podemos hacer que el error sea cero, tratamos de que sea lo más pequeño posible. Definimos el error (residuo) como la distancia entre lo que predice nuestro modelo ($A\hat{\mathbf{x}}$) y la realidad ($\mathbf{b}$):

$$\mathbf{e} = \mathbf{b} - A\hat{\mathbf{x}}$$

Para minimizar la longitud de este vector de error ($||\mathbf{e}||^2$), utilizamos cálculo o geometría proyectiva para llegar a las famosas **Ecuaciones Normales**:

♣ La Ecuación Maestra del Machine Learning Clásico

La mejor aproximación $\hat{\mathbf{x}}$ se encuentra resolviendo:

$$A^T A\hat{\mathbf{x}} = A^T\mathbf{b}$$

Nota que $A^T A$ es una matriz cuadrada y simétrica, lo que (casi siempre) nos permite resolver el sistema.

4.3. Conexión Agro-Mecatrónica

♠ Calibración de Sensores de Humedad Capacitivos

Los sensores de humedad de suelo baratos devuelven un voltaje analógico. Para convertir ese voltaje a "Porcentaje de Humedad Volumétrica", necesitamos calibrarlos. 1. Tomamos muestras de suelo con humedades conocidas (10 %, 20 %, ..., 50 %). 2. Medimos el voltaje en cada muestra. 3. Usamos Mínimos Cuadrados para encontrar la ecuación $Humedad = m \cdot Voltaje + c$ que minimice el error de lectura.

4.4. Implementación: Predicción de Rendimiento

Vamos a usar datos simulados para encontrar la relación entre agua de riego y producción de biomasa usando las Ecuaciones Normales (sin librerías de caja negra primero) y luego validando con Scikit-Learn.

```
import numpy as np

# 1. DATOS DE CAMPO (Simulados)
# X: Litros de agua/semana, y: Kg de biomasa
X_raw = np.array([10, 15, 20, 25, 30, 35])
y = np.array([1.2, 1.8, 2.5, 3.1, 3.4, 4.0])

# 2. CONSTRUCCIÓN DE LA MATRIZ DE DISEÑO A
# Necesitamos agregar una columna de 1s para el término independiente (
    intercepto)
# A tendrá forma (6 filas, 2 columnas)
ones = np.ones(len(X_raw))
A = np.column_stack((X_raw, ones))

print("Matriz A (Diseño):")
print(A[:3]) # Mostramos solo las primeras 3 filas

# 3. ECUACIONES NORMALES: (A^T * A) * x = (A^T * b)
# Paso A: Calcular A transpuesta por A
At_A = np.dot(A.T, A)

# Paso B: Calcular A transpuesta por y
At_y = np.dot(A.T, y)

# Paso C: Resolver el sistema lineal cuadrado resultante
# Usamos np.linalg.solve (que usa Descomposición LU internamente)
theta = np.linalg.solve(At_A, At_y)

pendiente = theta[0]
intercepto = theta[1]
```

```
30
31 # 4. PREDICCIÓN
32 agua_nueva = 40 # Litros
33 prediccion = pendiente * agua_nueva + intercepto
```

Listing 4.1: Cálculo de Regresión Lineal 'desde cero' usando Álgebra Lineal

>_ Resultados de la Regresión

```
Matriz A (Diseño):
[[10.  1.]
 [15.  1.]
 [20.  1.]]

Modelo encontrado:
Biomasa = 0.110 * Agua + 0.205

Predicción para 40L de agua:
4.62 Kg de biomasa
```

△ Atención: Atención: Correlación no implica Causalidad

Un R^2 alto en nuestro modelo matemático no significa que el agua sea la única causa del crecimiento. Si ignoramos plagas o temperatura, el modelo fallará en producción real. El álgebra lineal no tiene sentido común biológico.