# Computación científica LFIS-214

Profesor: Julio C. Marín

Departamento de Meteorología, Universidad de Valparaíso

Primer Semestre

# Temas de la clase

- Calcular matriz inversa
- Vectores y valores propios
- Ecuaciones no-lineales. Método gráfico

# Calcular la inversa de una matriz $\mathbf{A}^{-1}$

- El siguiente problema
$$AX = v$$
- pudiera resolverse multiplicando ambos lados de ecuación por $A^{-1}$ aunque sería método más lento que EG y factorización LU
- Suponiendo que queremos calcular la matriz inversa $A^{-1}$
- Algebra Lineal: $A^{-1}$ se calcula con matriz de cofactores dividida por el determinante.
- PERO este método es bastante lento y puede provocar errores de redondeos grandes.
- Mejor aproximación: convertir el problema a uno donde resolvemos sist. de ecs. lineales con métodos conocidos!!

# Calcular la inversa de una matriz $\mathbf{A}^{-1}$

- Consideremos el problema:

$$AX = V$$

- Mismo problema visto en clases PERO ahora $X$ y $V$ no son vectores, sino matrices $N \times N$
- Podemos resolver problema usando eliminación Gaussiana o factorización LU y sustitución inversa.
- Reducimos matriz $A$ y encontramos elementos de $X$ para cada columna!

# Calcular la inversa de una matriz $\mathbf{A}^{-1}$

$$AX = V$$

- Si escogemos $V$ como la matriz identidad:

$$A \begin{bmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Entonces $X = A^{-1}$

Solo tenemos que resolver $AX = I$ usando EG o factorización LU.

# Calcular la inversa de una matriz $\mathbf{A}^{-1}$ en Python

- from numpy.linalg import inv
- $X = inv(A)$

# Calcular la inversa de una matriz $\mathbf{A}^{-1}$ en Python

- from numpy.linalg import inv
- $X = inv(A)$

Opción alternativa:

- from numpy.linalg import solve
- $X = solve(A, I)$

# Ejercicios

- $A = \begin{bmatrix} 1 & 2 & 3 \\ 22 & 32 & 42 \\ 55 & 66 & 100 \end{bmatrix}$ y $b = [1, 2, 3]$

1. Resolver la ecuación $Ax = b$
2. Probar que la solución encontrada es exacta

# Ejercicios

- $A = \begin{bmatrix} 1 & 2 & 3 \\ 22 & 32 & 42 \\ 55 & 66 & 100 \end{bmatrix}$ y $b = [1, 2, 3]$

1. Resolver la ecuación $Ax = b$
2. Probar que la solución encontrada es exacta

- from numpy.linalg import solve
- $X = solve(A, b)$
- print ('x = ',$X$)
- print ('Residual = ',dot($A, x$) $- b$)
- Residual = [4.44....e-16 0.00000000e+0 -3.5527...e-15]

# Ejercicios

1. Encontrar la matriz inversa de $A = \begin{bmatrix} 4 & -2 & 1 \\ 3 & 6 & -4 \\ 2 & 1 & 8 \end{bmatrix}$
    1. Compruebe que la soluc. encontrada es correcta y analice el No. de cifras decimales para la que se cumple la identidad.

# Ejercicios

1. Encontrar la matriz inversa de $A = \begin{bmatrix} 4 & -2 & 1 \\ 3 & 6 & -4 \\ 2 & 1 & 8 \end{bmatrix}$
    1. Compruebe que la soluc. encontrada es correcta y analice el No. de cifras decimales para la que se cumple la identidad.

- from numpy.linalg import inv
- $Ai = inv(A)$
- print ('Id1 = ',$dot(inv(A), A)$)
- print ('Id2 = ',$dot(A, inv(A))$)

Esto nos da una idea de la precisión de nuestros cálculos!!

# Ejemplo solución ecs. lineales usando función inv

$$\begin{pmatrix} 4 & -1 & -1 & -1 \\ -1 & 3 & 0 & -1 \\ -1 & 0 & 3 & -1 \\ -1 & -1 & -1 & 4 \end{pmatrix} \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \\ 5 \\ 0 \end{pmatrix}$$

$$AX = v$$

- import numpy as np
- A = np.array([[4, -1, -1, -1], [-1, 3, 0, -1], [-1, 0, 3, -1], [-1, -1, -1, 4]])
- B = np.array([5, 0, 5, 0])
- C = np.linalg.solve(A, B) # Método 1
- D = np.linalg.inv(A).dot(B) # Método 2

Soluciones: $V_1 = 3$, $V_2 = 1.66666$, $V_3 = 3.33333$, $V_4 = 2$

# Vectores y valores propios

- Para una matriz simétrica **A**, un vector propio **v** es el que satisface la ecuación:

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}$$

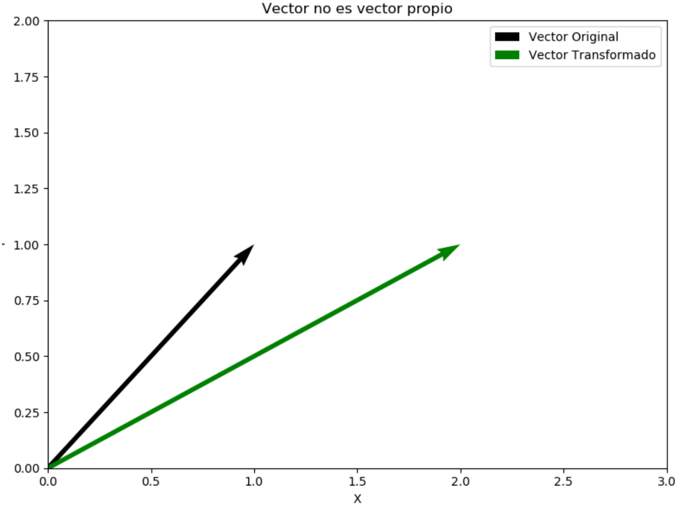
donde $\lambda$ es el correspondiente valor propio.

- Vector $v$ puede transformarse en nuevo vector con multiplicación **Av**
- Transformación afecta la escala y/o rotación de vector
- En algunos vectores, la transformación solo afecta su escala (estiramiento, compresión)

Los vectores propios son los vectores que tienen esta propiedad y los valores propios son los factores de escala.

- ### Transformación pero vector NO es vector propio
- import numpy as np
- import matplotlib.pyplot as plt
- def plot_vect(x, b, xlim, ylim):
  - plt.clf()
  - plt.quiver(0,0,x[0],x[1], color='k', angles='xy',
  - scale_units='xy',scale=1, label='Vector Original')
  - plt.quiver(0,0,b[0],b[1], color='g',angles='xy',
  - scale_units='xy',scale=1, label ='Vector Transformado')
  - plt.title('Vector no es vector propio')
  - plt.xlim(xlim); plt.ylim(ylim)
  - plt.xlabel('X'); plt.ylabel('Y'); plt.legend()
- A = np.array([[2, 0],[0, 1]]); x = np.array([[1],[1]])
- b = np.dot(A, x)
- plot_vect(x, b,(0,3),(0,2))

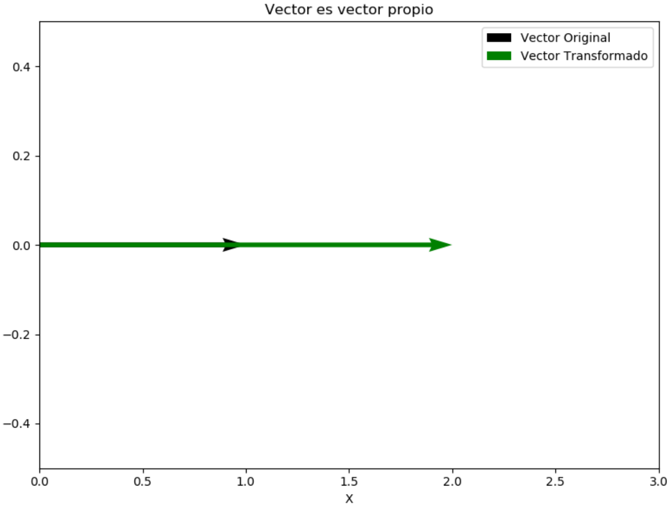
# Vectores y valores propios



# Vectores y valores propios

- ### Transformación y vector ES vector propio
- import numpy as np
- import matplotlib.pyplot as plt
- def plot_vect(x, b, xlim, ylim):
    - plt.clf()
    - plt.quiver(0,0,x[0],x[1], color='k', angles='xy',
    - scale_units='xy',scale=1, label='Vector Original')
    - plt.quiver(0,0,b[0],b[1], color='g',angles='xy',
    - plt.title('Vector es vector propio')
    - scale_units='xy',scale=1, label ='Vector Transformado')
    - plt.xlim(xlim); plt.ylim(ylim)
    - plt.xlabel('X'); plt.ylabel('Y'); plt.legend()
- A = np.array([[2, 0],[0, 1]]); x = np.array([[1],[0]])
- b = np.dot(A, x)
- plot_vect(x, b,(0,3),(-0.5, 0.5))

# Vectores y valores propios



# Vectores y valores propios

- Para una matriz $N \times N$, existen $N$ vectores propios $v_1, v_2, \ldots, v_N$ con valores propios $\lambda_1, \lambda_2, \ldots, \lambda_N$
- Los autovectores son ortogonales entre ellos.
- Vectores propios pueden ser también columnas de una matriz $V$ $(N \times N)$ donde se combinen las ecuaciones $Av_i = \lambda_i v_i$ en:

$$AV = VD$$

$D$: matriz diagonal con autovalores $\lambda_i$ en la diagonal.

# Vectores y valores propios

$$AV = VD$$

$D$: matriz diagonal con autovalores $\lambda_i$ en la diagonal.

- Matriz $V$ es ortogonal $\Longrightarrow V^T = V^{-1} \Longrightarrow V^T V = VV^T = I$
- Método $QR$: más usado para calcular vectores y valores propios
- Método $QR$ calcula las matrices $V$ y $D$ en ecuación.
- Método $QR$: matriz $A$ se descompone en matriz ortogonal $Q$ y matriz triangular superior $R$.

# Vectores y valores propios en Python

Función 'eigh' del módulo numpy.linalg calcula valores y vectores propios para matrices simétricas o hermíticas.

- import numpy as np
- A = np.array([[1,2],[2,1]], float)
- x, V = np.linalg.eigh(A)
- print('Los autovalores son ', x)
- print('Los autovectores son ', V)
- #############
- x: Arreglo 1-d con autovalores
- V: matriz cuyas columnas contienen autovectores

Función 'eigvalsh' calcula solo el valor propio lo cual puede ahorrar tiempo al resolver un problema.

# Vectores y valores propios en Python

- Funciones eigh y eigvalsh se utilizan para calcular valores y vectores propios en matriz simétrica o Hermítica
- Qué pasa si proporcionamos matriz no-simétrica como argumento a funciones?

$$\begin{pmatrix} 4 & 1 & -1 \\ 3 & 2 & 1 \\ 5 & 4 & 0 \end{pmatrix}$$

- from numpy.linalg import eigh, eigvalsh
- A = np.array([[4,1,-1],[3,2,1],[5,4,0]], float)
- x, V = eigh(A)
- print("Los autovalores de A = ", x)
- print("Los autovectores de A = ", V)

# Vectores y valores propios en Python

- Funciones ignoran elementos encima de diagonal.
- Asumen que matriz es simétrica, como si copiaran elementos trángulo inferior en triángulo superior
- Si matriz es compleja, funciones eigh y eigvalsh asumen que matriz es Hermítica

**Ejercicio:** Comparar cálculo anterior de vectores y valores propios con las siguientes matrices:

$$\begin{pmatrix} 4 & 3 & 5 \\ 3 & 2 & 4 \\ 5 & 4 & 0 \end{pmatrix} \qquad \begin{pmatrix} 4 & 1 & -1 \\ 1 & 2 & 1 \\ -1 & 1 & 0 \end{pmatrix}$$

# Vectores y valores propios en Python

- from numpy.linalg import eigh, eigvalsh
- A = np.array([[4,1,-1],[3,2,1],[5,4,0]], float)
- A2 = np.array([[4,3,5],[3,2,1],[5,4,0]], float)
- A3 = np.array([[4,1,-1],[1,2,1],[-1,1,0]], float)
- x, V = eigh(A)
- print("Los autovalores de A = ", x)
- print("Los autovectores de A = ", V)
- x2, V2 = eigh(A2)
- print(); print("Los autovalores de A = ", x2)
- print("Los autovectores de A = ", V2)
- x3, V3 = eigh(A3)
- print(); print("Los autovalores de A = ", x3)
- print("Los autovectores de A = ", V3)

# Vectores y valores propios en Python

- Módulo linalg ofrece funciones 'eig' y 'eigvals' para calcular valores y vectores propios en matrices que no son simétricas.
- Sin embargo, problemas de valores propios no simétricos son raros en Física

- from numpy.linalg import eig, eigvals
- A = np.array([[4,1,-1],[3,2,1],[5,4,0]], float)
- x, V = eig(A)
- print("Los autovalores de A = ", x)
- print("Los autovectores de A = ", V)

Ejercicio: Dados los vectores: $\overrightarrow{a} = (1, 2, 3, 4)$ y $\overrightarrow{b} = (2, 4, 6, 8)$. Calcule el producto escalar de estos dos vectores en Python usando dos métodos distintos.

# Ejercicios

Ejercicio: Dados los vectores: $\overrightarrow{a} = (1, 2, 3, 4)$ y $\overrightarrow{b} = (2, 4, 6, 8)$. Calcule el producto escalar de estos dos vectores en Python usando dos métodos distintos.

- a = np.arange(1,5)
- b = np.arange(2,9,2)
- print("El producto escalar a*b = ", np.dot(a,b))
- print("El producto escalar a*b = ", sum(a*b))

# Ejercicios

Ejercicio: Dada la siguiente matriz cuyas columnas son vectores:

$$A = \begin{pmatrix} 3 & 5 & 2 & 4 & -4 & -3 \\ 4 & 1 & 0 & -3 & 2 & 0 \\ 2 & 3 & 1 & 0 & 1 & 5 \end{pmatrix}$$

Encuentre e imprima los vectores que son ortogonales entre sí.

Ejercicio: Dada la siguiente matriz cuyas columnas son vectores:

$$A = \begin{pmatrix} 3 & 5 & 2 & 4 & -4 & -3 \\ 4 & 1 & 0 & -3 & 2 & 0 \\ 2 & 3 & 1 & 0 & 1 & 5 \end{pmatrix}$$

Encuentre e imprima los vectores que son ortogonales entre sí.

- import numpy as np
- A = np.array([[3,5,2,4,-4,-3],[4,1,0,-3,2,0],[2,3,1,0,1,5]])
- for ii in range(A.shape[1]):
-         a1 = A[:,ii]
-         for jj in range(ii+1, A.shape[1]):
-                 if np.dot(a1, A[:,jj]) == 0:
-                         print("Los vectores ",a1, " y ", A[:,jj], " son ortogonales")