U2A1. PROBLEMARIO. Operaciones matriciales y vectoriales

Actividad realizada por: Melissa Gómez Rentería

A 23 de septiembre de 2025.

Ejercicio 1: Suma de matrices

Dadas dos matrices A y B:

A = [[2, 4, 6], [1, 3, 5], [7, 9, 11]]
B = [[12, 10, 8], [6, 4, 2], [0, -2, -4]]

Realiza la suma de estas dos matrices y encuentra la matriz resultante *C = A + B*.

```
import numpy as np  #importamos la libreria numpy, la cual nos permite trabajar con ma

#creamos las matrices A y B con los valores dados
A=np.array([[2,4,6], [1,3,5], [7,9,11]])
B=np.array([[12,10,8], [6,4,2], [0,-2,-4]])

#realizamos la suma de las matrices en la variable
C= A+B

#mostramos la matriz resultante
print("Resultado de la suma de matrices: ")
print(C)
```

```
Resultado de la suma de matrices:
[[14 14 14]
 [ 7  7  7]
 [ 7  7  7]]
```

Ejercicio 2: Multiplicación de matrices

Dadas dos matrices A y B:

A = [[2, 1], [3, 4], [5, 6]]
B = [[7, 8], [9, 10]]

Realiza la multiplicación de estas dos matrices y encuentra la matriz resultante *C = A * B*.

```
#creamos las matrices A y B con los valores dados
A=np.array([[2,1], [3,4], [5,6]])
B=np.array([[7,8], [9,10]])

#realizamos la multiplicación de las matrices en la variable C
C=A@B #Se utiliza el operador @ para realizar la multiplicación de matrices
```

```
#mostramos la matriz resultante
print("Resultado de la multiplicación de matrices: ")
print(C)
```

```
Resultado de la multiplicación de matrices:
[[ 23  26]
 [ 57  64]
 [ 89 100]]
```

Ejercicio 3: Inversión de matriz

Dada la matriz cuadrada A:

A = [[4, 7, 2], [2, 6, 8], [3, 1, 9]]

Encuentra la matriz inversa de A, denotada como A^(-1).

```
#importamos la extension matrix de sympy para trabajar con matrices
from sympy import Matrix

#creamos la matriz A con los valores dados
A= Matrix([[4,7,2], [2,6,8], [3,1,9]])

#calculamos la inversa de la matriz A
A_inv= A.inv() #se utiliza el método inv() para calcular la inversa de la matriz

#mostramos la matriz inversa
print("Resultado de la inversa de la matriz: ")
print(A_inv)
```

```
Resultado de la inversa de la matriz:
Matrix([[23/97, -61/194, 22/97], [3/97, 15/97, -14/97], [-8/97, 17/194, 5/97]])
```

Ejercicio 4: Resolución de sistema de ecuaciones

Dado el sistema de ecuaciones lineales:

$2x + y + z = 8$

$3x + 5y + 2z = 21$

$x + 2y + 4z = 11$

Escribe este sistema en forma matricial *AX = B*, donde X es el vector de incógnitas [x, y, z] y B es el vector de términos constantes. Luego, resuelve el sistema para encontrar X.

```
from sympy import symbols, Eq, solve, Matrix

x, y, z = symbols('x y z')

# Definición del sistema de ecuaciones lineales
eq1 = Eq(2*x + y + z, 8)
eq2 = Eq(3*x + 5*y + 2*z, 21)
eq3 = Eq(x + 2*y + 4*z, 11)
```

```
# Representación matricial AX = B
# Matriz de coeficientes A
A = Matrix([[2, 1, 1], [3, 5, 2], [1, 2, 4]])

# Vector de incógnitas X
X = Matrix([x, y, z])

# Vector de términos constantes B
B = Matrix([[8], [21], [11]])

print("Representación matricial del sistema: AX = B")
print("Matriz A:")
print(A)
print("\nVector X:")
print(X)
print("\nVector B:")
print(B)

# Resolución del sistema para encontrar X usando la inversa de A
# Primero calculamos la inversa de A
A_inv = A.inv()

# Luego multiplicamos la inversa de A por B para encontrar X
sol_matrix = A_inv * B

print("\nSolución del sistema de ecuaciones usando matrices:")
print("Vector de incógnitas X:")
print(sol_matrix)

# También podemos verificar la solución usando solve de sympy
sol_sympy = solve((eq1, eq2, eq3), (x, y, z))
print("\nSolución del sistema de ecuaciones usando solve de sympy:")
print(sol_sympy)
```

```
Representación matricial del sistema: AX = B
Matriz A:
Matrix([[2, 1, 1], [3, 5, 2], [1, 2, 4]])

Vector X:
Matrix([[x], [y], [z]])

Vector B:
Matrix([[8], [21], [11]])

Solución del sistema de ecuaciones usando matrices:
Vector de incógnitas X:
Matrix([[53/23], [56/23], [22/23]])

Solución del sistema de ecuaciones usando solve de sympy:
{x: 53/23, y: 56/23, z: 22/23}
```

Ejercicio 5: Determinante

Dada la matriz cuadrada A:

A = [[3, -2, 1], [0, 5, 4], [2, 1, 7]]

Calcula el determinante de A.

```
#creamos la matriz A con los valores dados
A= Matrix([[3,-2,1], [0,5,4], [2,1,7]])

#calculamos el determinante de la matriz A
det= A.det()

#mostramos el determinante
print("Determinante de la matriz: ")
print(det)
```

```
Determinante de la matriz:
67
```

Ejercicio 6: Producto Cruz

Dados dos vectores A y B en el espacio tridimensional:

A = [2, 3, -1]
B = [1, -2, 4]

Calcula el producto cruz entre A y B, denotado como *A x B*.

```
#importamos la libreria numpy para trabajar con matrices y vectores
import numpy as np

#creamos los vectores A y B con los valores dados
A = np.array([2, 3, -1])
B = np.array([1, -2, 4])

#calculamos el producto cruz entre los vectores A y B
C = np.cross(A, B)

#mostramos el resultado
print("Resultado del producto cruz:")
print(C)
```

```
Resultado del producto cruz:
[10 -9 -7]
```

Ejercicio 7: Proyección ortogonal

Dado un vector V = [5, -3, 2] y un vector U = [2, 1, 2], encuentra la proyección ortogonal de V sobre U.

```
#creamos los vectores V y U con los valores dados
V=np.array([5,-3,2])
U=np.array([2,1,2])

#calculamos la proyección ortogonal de V sobre U
proyeccion = np.dot(V, U) / np.dot(U, U) * U
```

```
#mostramos el resultado
print("Resultado de la proyección ortogonal: ")
print(proyeccion)
```

```
Resultado de la proyección ortogonal:
[2.44444444 1.22222222 2.44444444]
```

Ejercicio 8: Producto escalar de proyecciones

Supongamos que tienes tres vectores V = [3, -1, 2], U = [2, 2, -1], y W = [1, 4, -2]\ Encuentra el producto escalar de la proyección de V sobre U con la proyección de V sobre W.

```
#creamos los vectores V, U y W con los valores dados
V=np.array([3,-1,2])
U=np.array([2,2,-1])
W=np.array([1,4,-2])

#calculamos la proyección de V sobre U y W
proyeccion1 = np.dot(V, U) / np.dot(U, U) * U
proyeccion2 = np.dot(V, W) / np.dot(W, W) * W

#calculamos el producto escalar de las proyecciones
producto_escalar = np.dot(proyeccion1, proyeccion2)

#mostramos el resultado
print("Resultado del producto escalar de proyecciones: ")
print(producto_escalar)
```

```
Resultado del producto escalar de proyecciones:
-0.6349206349206349
```

Ejercicio 9: Ortogonalización Gram-Schmidt

Dado un conjunto de vectores linealmente independientes:

v1 = [1, 1, 0]
v2 = [1, 2, 1]
v3 = [2, 1, 3]

Aplica el proceso de ortogonalización de Gram-Schmidt para obtener un conjunto ortogonal equivalente a los vectores originales.

```
import numpy as np

# Define la función para el proceso de Gram-Schmidt
def gram_schmidt(vectores):
    ortogonales = [] # Lista para almacenar los vectores ortogonales
    for v in vectores:
        v_ortho = v # Inicializa el vector ortogonal con el vector original
        # Resta la proyección de v_ortho sobre cada vector ortogonal ya encontrado
        for u in ortogonales:
            v_ortho = v_ortho - (np.dot(v_ortho, u) / np.dot(u, u)) * u
```

```
        ortogonales.append(v_ortho) # Añade el vector ortogonal a la lista
    return ortogonales

# Define los vectores de entrada
v1 = np.array([1, 1, 0], dtype=float)
v2 = np.array([1, 2, 1], dtype=float)
v3 = np.array([2, 1, 3], dtype=float)

# Aplica el proceso de Gram-Schmidt a los vectores
resultado = gram_schmidt([v1, v2, v3])

# Muestra los vectores ortogonales resultantes
print("Vectores ortogonales:")
for v in resultado:
    print(v)
```

```
Vectores ortogonales:
[1. 1. 0.]
[-0.5  0.5  1. ]
[ 1.33333333 -1.33333333  1.33333333]
```

Ejercicio 10: Espacio nulo

Dada una matriz A en forma escalonada reducida por filas:

A = [[1, 2, 0, 3], [0, 1, 0, 2], [0, 0, 1, 1]]

Encuentra una base para el espacio nulo de A, es decir, los vectores que satisfacen Av = 0, donde v es un vector columna.

```
# Definimos la matriz A en forma escalonada reducida por filas
A = Matrix([[1, 2, 0, 3], [0, 1, 0, 2], [0, 0, 1, 1]])

# Calculamos el espacio nulo de la matriz A
# El espacio nulo (kernel) de una matriz A es el conjunto de todos los vectores v
# para los cuales Av = 0.
nulo = A.nullspace()

# Mostramos una base para el espacio nulo
print("Espacio nulo de la matriz: ")
print(nulo)
```

```
Espacio nulo de la matriz:
[Matrix([
[ 1],
[-2],
[-1],
[ 1]])]
```

```
Comienza a programar o generar con IA.
```