Multiplicación de matrices

$$\begin{bmatrix} e & f \\ g & h \end{bmatrix} \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} ea+fc & eb+fd \\ ga+hc & gb+hd \end{bmatrix}$$

Nota: El producto de $A \in M_{mxq}$ y $B \in M_{pxn}$ no está definido si $q \neq p$

```
def MultMat(A, B):
    # Excluir los casos donde la multiplicación no esta definida
    if len(A[0]) != len(B):
        return "Error: Las dimensiones de las matrices no permiten su multiplicación."

    # Cantidad de renglones y columnas
    m, q, n = len(A), len(A[0]), len(B[0])
    C = [[0 for _ in range(n)] for _ in range(m)] # Matriz resultado

    # Realizar la multiplicación
    for i in range(m):
        for j in range(n):
            for k in range(q):  # q es el numero de columna de A y al mismo tiempo de renglones de B
                C[i][j] += A[i][k] * B[k][j]

    return C

# Ejemplo de uso
A = [[1, 2, 3],
     [4, 5, 6]]

B = [[7, 8],
     [9, 10],
     [11, 12]]

print( "Sea A = ")
for fila in A:
    print(fila)

print( "y B = ")
for fila in B:
    print(fila)

resultado = MultMat(A, B)
print("Tenemos que el resultado de AB es: " )
for fila in resultado:
    print(fila)
```

```
Sea A =
[1, 2, 3]
[4, 5, 6]
y B =
[7, 8]
[9, 10]
[11, 12]
Tenemos que el resultado de AB es:
[58, 64]
[139, 154]
```

Para realizar el producto de matrices A y B por bloques en necesario que:

- El numero de bloques columna de la matriz A, sea igual al numero de bloques fila de la matriz B.
- Los bloques correspondientes podrán multiplicase cuando coincidan el numero de columnas de la matriz A y el numero de filas de la matriz B.

Sean $\mathbf{A} = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$ y $\mathbf{B} = \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]$ $\mathbf{A} \cdot \mathbf{B} = \left[\begin{array}{c|c} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ \hline A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{array}\right]$

```
def MultMatBloques(A, B, bloque):
    # Excluir los casos donde la multiplicación no esta definida
    if len(A[0]) != len(B):
        return "Error: Las dimensiones de las matrices no permiten su multiplicación."

    n = len(A)
    C = [[0 for _ in range(n)] for _ in range(n)] #notese que se hace para una matriz cuadrada de nxn

    # Multiplicación por bloques
    for i in range(0, n, bloque):
        for j in range(0, n, bloque):
            for k in range(0, n, bloque):
                # Multiplicar bloques A[i:i+bloque, k:k+bloque] * B[k:k+bloque, j:j+bloque]
                for i1 in range(i, min(i + bloque, n)):
                    for j1 in range(j, min(j + bloque, n)):
                        for k1 in range(k, min(k + bloque, n)):
                            C[i1][j1] += A[i1][k1] * B[k1][j1]
    return C

# Ejemplo de uso
A = [[1, 2, 3, 4],
     [5, 6, 7, 8],
     [9, 10, 11, 12],
     [13, 14, 15, 16]]

B = [[16, 15, 14, 13],
     [12, 11, 10, 9],
     [8, 7, 6, 5],
     [4, 3, 2, 1]]

bloque = 2 # Tamaño del bloque

print( "Sea A = ")
for fila in A:
    print(fila)

print( "y B = ")
for fila in B:
    print(fila)
resultado = MultMatBloques(A, B, bloque)
print("E l resultado de la multiplicación por bloques es: ")
for fila in resultado:
    print(fila)
```

```
Sea A =
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 16]
y B =
[16, 15, 14, 13]
[12, 11, 10, 9]
[8, 7, 6, 5]
[4, 3, 2, 1]
E l resultado de la multiplicación por bloques es:
[80, 70, 60, 50]
[240, 214, 188, 162]
[400, 358, 316, 274]
[560, 502, 444, 386]
```

El producto punto (o producto escalar) de dos vectores en $\mathbb{R}^n$ puede descomponerse en bloques si los vectores se dividen en subvectores más pequeños.

Supongamos que tenemos dos vectores $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ y que los dividimos en k bloques de igual tamaño (asumiendo por simplicidad que n es divisible por k). Los vectores se pueden representar como:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_k \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_k \end{bmatrix}$$

donde $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^{n/k}$ son los subvectores (o bloques) de $\mathbf{x}$ y $\mathbf{y}$, respectivamente.

El producto punto de $\mathbf{x}$ y $\mathbf{y}$ se puede calcular como la suma de los productos punto de sus correspondientes bloques:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{k} \mathbf{x}_i \cdot \mathbf{y}_i = \sum_{i=1}^{k} \sum_{j=1}^{n/k} (x_{ij} \cdot y_{ij})$$

```
def ProductoPuntoBloques(v1, v2, bloque):
    # Verificar que los vectores tengan la misma dimensión
    if len(v1) != len(v2):
        return "Error: Los vectores no tienen la misma dimensión."

    n = len(v1)
    resultado = 0

    # Producto punto por bloques
    for i in range(0, n, bloque):
        bloqueSuma = 0
        for j in range(i, min(i + bloque, n)):
            bloqueSuma += v1[j] * v2[j]
        resultado += bloqueSuma

    return resultado

# Ejemplo de uso con vectores de dimensión 100
v1 = [i for i in range(100)]
v2 = [i for i in range(99, -1, -1)]

bloque = 10  # Tamaño del bloque

print("Sea v1 =", v1)
print("y v2 =", v2)
productoPunto = ProductoPuntoBloques(v1, v2, bloque)
print(f"El producto punto por bloques de dichos vectores es: {productoPunto}")
```

```
Sea v1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
y v2 = [99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68,
El producto punto por bloques de dichos vectores es: 161700
```

La transpuesta de una matriz A de dimensiones $m \times n$, denotada como A^T, es una nueva matriz de dimensiones $n \times m$ obtenida al intercambiar las filas por columnas de A.
Es decir, si $A = [a_{ij}]$ donde $1 \leq i \leq m$ y $1 \leq j \leq n$, entonces $A^T = [a'_{ij}]$ donde $a'_{ij} = a_{ji}$ y $1 \leq i \leq n, 1 \leq j \leq m$. En otras palabras, el elemento en la fila i, columna j de A^T es el elemento en la fila j, columna i de A.

```
def Transpuesta(A):
    # Cantidad de renglones y columnas
    n, m = len(A), len(A[0])
    # inicializacion de la matriz transpuesta de A
    AT = [[0 for _ in range(n)] for _ in range(m)]

    for i in range(n):
        for j in range(m):
            AT[j][i] = A[i][j]

    return AT

# Ejemplo de uso
A = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

print( "Sea A = ")
for fila in A:
    print(fila)

transpuesta_A = Transpuesta(A)
print("La matriz transpuesta de A es:")
for fila in transpuesta_A:
    print(fila)
```

```
Sea A =
[1, 2, 3]
[4, 5, 6]
```

```
[7, 8, 9]
La matriz transpuesta de A es:
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]
```

La potencia de una matriz cuadrada A de dimensiones $n \times n$, denotada como A^k donde k es un entero no negativo, se define como el producto de A por sí misma k veces. Las definiciones específicas dependen del valor de k:

- Si $k = 0$, A^0 se define como la matriz identidad I_n de dimensiones $n \times n$, donde I_n es una matriz con 1s en la diagonal principal y 0s en el resto de las posiciones.
- Si $k = 1$, $A^1 = A$.
- Para $k > 1$, $A^k = A \cdot A \cdot \ldots \cdot A$ (k veces).

Es decir, si $A = [a_{ij}]$ y $B = A^k = [b_{ij}]$, entonces cada elemento b_{ij} de B se calcula como:

$$b_{ij} = \sum_{r=1}^{n} a_{ir} \cdot a_{rj} \quad (\text{para } k = 2)$$

y de manera análoga para potencias mayores, extendiendo el producto a múltiples matrices A.

Nota: la potencia de una matriz solo está definida para matrices cuadradas, debido a la necesidad de que las dimensiones sean compatibles para la multiplicación.

```
def PotenciaMatriz(A, n):
    # Asegurar que n es un entero no negativo, para cumplir con la definición
    if n < 0:
        return "Error: El exponente debe ser un entero no negativo."

    # Caso base: la potencia 0 de cualquier matriz es la matriz identidad
    if n == 0:
        return [[1 if i == j else 0 for j in range(len(A))] for i in range(len(A))]

    # Caso base: la potencia 1 de la matriz A es A
    if n == 1:
        return A

    # Inicializar resultado como la matriz identidad
    resultado = [[1 if i == j else 0 for j in range(len(A))] for i in range(len(A))]

    # Copia de A para usar en las multiplicaciones
    B = A

    while n > 0:
        # si n es impar, multiplica el resultado acumulado hasta el momento (resultado) por la matriz B
        if n % 2 == 1:
            resultado = MultMat(resultado, B)

        B = MultMat(B, B) # Independientemente de si n es par o impar, B (que es A o el resultado de multiplicaciones previas de A por sí mi
        n //= 2 # el exponente n se divide por 2, , preparando n para la siguiente iteración del bucle. Ya que anteriormente lo elevamos al

    return resultado

# Ejemplo de uso
A = [[2, 0],
     [0, 2]]
n = 3

print( "Sea A = ")
for fila in A:
    print(fila)

potencia_A = PotenciaMatriz(A, n)
print(f"Resultado de A elevado a la potencia {n}:")
for fila in potencia_A:
    print(fila)
```

```
Sea A =
[2, 0]
[0, 2]
Resultado de A elevado a la potencia 3:
[8, 0]
[0, 8]
```

El producto cruz entre dos vectores tridimensionales $\mathbf{a}$ y $\mathbf{b}$, representados por $\mathbf{a} = (a_1, a_2, a_3)$ y $\mathbf{b} = (b_1, b_2, b_3)$ respectivamente, es un tercer vector $\mathbf{c}$ que es perpendicular a ambos $\mathbf{a}$ y $\mathbf{b}$ y cuya dirección está dada por la regla de la mano derecha.
Es decir, el producto cruz se define como:

$$\mathbf{c} = \mathbf{a} \times \mathbf{b} = (a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1)$$

```
def ProductoCruz(a, b):
    # a y b son vectores tridimensionales
    c = [0, 0, 0]  # inicializando el vector resultado a cero

    # Calculando cada componente del vector resultado
    c[0] = a[1]*b[2] - a[2]*b[1]
    c[1] = a[2]*b[0] - a[0]*b[2]
    c[2] = a[0]*b[1] - a[1]*b[0]

    return c

# Ejemplo de uso
a = [1, 2, 3]
b = [4, 5, 6]

print("Sea a =", a)
print("y b =", b)
resultado = ProductoCruz(a, b)
print("El producto cruz entre a y b es:", resultado)
```

```
Sea a = [1, 2, 3]
y b = [4, 5, 6]
El producto cruz entre a y b es: [-3, 6, -3]
```

Para rotar un vector bidimensional mediante una matriz de rotación, podemos utilizar la siguiente matriz de rotación $R(\theta)$, donde θ es el ángulo de rotación:

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

Dado un vector bidimensional $\mathbf{v} = (x, y)$, la rotación del vector por un ángulo θ se calcula multiplicando la matriz de rotación $R(\theta)$ por el vector $\mathbf{v}$:

$$\mathbf{v}' = R(\theta)\mathbf{v}$$

El vector resultante $\mathbf{v}'$ es la versión rotada de $\mathbf{v}$.

```
import math

def RotarVector(v, theta):
    # v es el vector bidimensional
    # theta es el ángulo de rotación en radianes

    # Matriz de rotación
    R = [[math.cos(theta), -math.sin(theta)],
         [math.sin(theta),  math.cos(theta)]]

    v_rotado = [0, 0]  # inicializando el vector resultado a cero

    # Aplicar la rotación
    v_rotado[0] = R[0][0]*v[0] + R[0][1]*v[1]
    v_rotado[1] = R[1][0]*v[0] + R[1][1]*v[1]

    return v_rotado

# Ejemplo de uso
v = [1, 0]
theta = math.pi / 4  # Ángulo de rotación de 45 grados (pi/4 radianes)

print("Sea v =", v)
v_rotado = RotarVector(v, theta)
print("El vector rotado es:", v_rotado)
```

```
Sea v = [1, 0]
Rotaremos el vector:  0.7853981633974483
El vector rotado es: [0.7071067811865476, 0.7071067811865475]
```

Una matriz de rotación es una matriz que se utiliza para realizar una rotación en el espacio Euclidiano. Para un espacio tridimensional, la rotación alrededor de un eje de rotación puede ser descrita por una matriz de rotación 3×3. Suponiendo que la rotación es alrededor de uno de los ejes coordenados, las matrices de rotación son las siguientes:

- Rotación alrededor del eje x por un ángulo θ:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

- Rotación alrededor del eje y por un ángulo θ:

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

- Rotación alrededor del eje z por un ángulo θ:

$$R_z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
import math

def RotarVector3D(v, theta, eje):
    # v es el vector tridimensional
    # theta es el ángulo de rotación en radianes
    # eje es el eje sobre el cual se hara la rotacion

    R = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]  # Matriz identidad como base

    #Definiremos las matrices de rotacion
    if eje == 'z':
        R[0][0] = math.cos(theta)
        R[0][1] = -math.sin(theta)
        R[1][0] = math.sin(theta)
        R[1][1] = math.cos(theta)
    elif eje == 'y':
        R[0][0] = math.cos(theta)
        R[0][2] = math.sin(theta)
        R[2][0] = -math.sin(theta)
        R[2][2] = math.cos(theta)
    elif eje == 'x':
        R[1][1] = math.cos(theta)
        R[1][2] = -math.sin(theta)
        R[2][1] = math.sin(theta)
        R[2][2] = math.cos(theta)
    else:
        return "Eje no válido" # solo haremos las rotaciones alrededor de los ejes primarios

    # Aplicar la rotación
    v_rotado = [0, 0, 0]
    for i in range(3):
        for j in range(3):
            v_rotado[i] += R[i][j] * v[j]

    return v_rotado

# Ejemplo de uso
v = [1, 0, 0]
theta = math.pi / 4  # Ángulo de rotación de 45 grados (pi/4 radianes)

print("Sea v =", v)

# Rotación alrededor del eje z
v_rotado_z = RotarVector3D(v, theta, 'z')
print("Vector rotado alrededor del eje z:", v_rotado_z)

# Rotación alrededor del eje y
v_rotado_y = RotarVector3D(v, theta, 'y')
print("Vector rotado alrededor del eje y:", v_rotado_y)

# Rotación alrededor del eje x
v_rotado_x = RotarVector3D(v, theta, 'x')
print("Vector rotado alrededor del eje x:", v_rotado_x)
```

```
Sea v = [1, 0, 0]
Vector rotado alrededor del eje z: [0.7071067811865476, 0.7071067811865475, 0]
Vector rotado alrededor del eje y: [0.7071067811865476, 0, -0.7071067811865475]
Vector rotado alrededor del eje x: [1, 0.0, 0.0]
```

Calcule la norma 1,

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$

Es decir, la norma 1 de una matriz A, se define como el máximo de las sumas absolutas de las columnas de la matriz.

```
def Norma1(A):
    # Cantidad de renglones y columnas de A
    m, n = len(A), len(A[0])
    max_suma = 0 # inicializamos

    for j in range(n):  # Itera sobre cada columna
        suma_columna = 0
        for i in range(m):  # suma los valores absolutos de los elementos en la columna j
            suma_columna += abs(A[i][j])
        max_suma = max(max_suma, suma_columna)  # actualiza el máximo

    return max_suma

# Ejemplo de uso
A = [[1, -2, 3],
     [-4, 5, -6],
     [7, -8, 9]]
```