MATRICES

Que son las matrices

Las matrices son un conjunto bidimensional de números o símbolos distribuidos de forma rectangular, en líneas verticales y horizontales, de manera que sus elementos se organizan en filas y columnas. Sirven para describir sistemas de ecuaciones lineales o diferenciales, así como para representar una aplicación lineal.

Matriz por su tamaño

Matriz fila

Una matriz fila es aquella que tiene una sola fila y varios elementos en esa fila. Se denota de la siguiente manera:

$$[A = [a_1 , a_2 , a_3 , \ldots , a_n]]$$

donde $a_1 , a_2 , a_3 , \ldots , a_n$ son los elementos de la matriz.

Matriz columna

Una matriz columna es aquella que tiene una sola columna y varios elementos en esa columna. Se denota de la siguiente manera:

$$[B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}]$$

donde $b_1 , b_2 , b_3 , \ldots , b_m$ son los elementos de la matriz.

Matriz cuadrada

Una matriz cuadrada es aquella que tiene el mismo número de filas que de columnas. Se denota de la siguiente manera:

$$[C = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}]$$

donde c_{ij} representa los elementos de la matriz.

Matriz por sus elementos

Matriz nula

Una matriz nula es aquella en la que todos sus elementos son cero. Se denota como O.

Matriz diagonal

Es un tipo de matriz cuadrada en la que los elementos que no se encuentran en la diagonal principal son iguales a cero:

$$[D = \begin{bmatrix} d_{11} & 0 & 0 \\ 0 & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix}]$$

Matriz triangular

Una matriz triangular es aquella en la que todos los elementos por encima o por debajo de la diagonal principal son cero. Dependiendo de la posición de los ceros, se distinguen entre matriz triangular superior y matriz triangular inferior.

Matriz triangular superior

se trata de una matriz cuadrada en la que al menos uno de los términos que está por encima de la diagonal principal es distinto a cero, y todos los que están situados por debajo a ella son iguales a cero.

$$[F = \begin{bmatrix} f_{11} & 0 & 0 \\ f_{21} & f_{22} & 0 \\ f_{31} & f_{32} & f_{33} \end{bmatrix}]$$

Matriz triangular inferior

A diferencia del tipo anterior, en este tipo de matriz al menosuno de los elementos que están debajo de la diagonal principal son diferentes a cero y todos los que están por encima de ella son iguales a cero.

$$[E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ 0 & e_{22} & e_{23} \\ 0 & 0 & e_{33} \end{bmatrix}]$$

Suma de Matrices

La suma de dos matrices del mismo tamaño se realiza sumando elemento por elemento. Si tenemos dos matrices A y B del mismo tamaño, la suma se denota de la siguiente manera:

[C = A + B]

donde C es la matriz resultante de la suma y $c_{ij} = a_{ij} + b_{ij}$ para cada elemento de la matriz.

Multiplicación por Escalar

La multiplicación de una matriz por un escalar implica multiplicar cada elemento de la matriz por dicho escalar. Si tenemos una matriz A y un escalar k, la multiplicación se denota de la siguiente manera:

[B = k \cdot A]

donde B es la matriz resultante y $b_{ij} = k \cdot a_{ij}$ para cada elemento de la matriz.

Matrices inversas

Se dice que una matriz cuadrada A es invertible si existe una matriz B con la propiedad de que

$$AB = BA = I$$

donde I es la matriz identidad . La matriz B es única, la llamamos la inversa de A y la denotamos por A^{-1}. Esto es,

$$AA^{-1} = A^{-1}A = I.$$

DETERMINANTES

Definición de determinante: El determinante es una función que asocia a cada matriz cuadrada un número. Se representa mediante barras verticales o con el símbolo det(). Para una matriz cuadrada A de tamaño nxn, el determinante se denota como |A| o det(A).

Determinantes de matrices 2x2: El determinante de una matriz 2x2 se calcula mediante la siguiente fórmula: |A| = ad - bc Donde A es una matriz de la forma: | a b | | c d |

Determinantes de matrices 3x3: El determinante de una matriz 3x3 se calcula mediante la regla de Sarrus o el método de Laplace. La regla de Sarrus es la siguiente: |A| = ($a_1b_2c_3$ + $b_1c_2a_3$ + $c_1a_2b_3$) - ($c_1b_2a_3$ + $a_1c_2b_3$ + $b_1a_2c_3$) Donde A es una matriz de la forma: | a_1 b_1 c_1 | | a_2 b_2 c_2 | | a_3 b_3 c_3 |

Determinantes de matrices nxn: Para calcular el determinante de una matriz nxn, se puede utilizar el método de Laplace, que consiste en expandir el determinante a lo largo de una fila o columna, multiplicando cada elemento por su cofactor y sumándolos. El cofactor de un elemento a_{ij} se calcula multiplicando (-1)^(i+j) por el determinante de la submatriz obtenida al eliminar la fila i y columna j.

Propiedades de los determinantes:

- Si una matriz tiene una fila o columna de ceros, su determinante es igual a cero.
- Si se intercambian dos filas o columnas de una matriz, el determinante cambia de signo.
- Si se multiplica una fila o columna por un escalar k, el determinante se multiplica por k.
- Si una matriz tiene dos filas o columnas iguales, su determinante es igual a cero.

- El determinante de una matriz triangular (superior o inferior) es igual al producto de los elementos de la diagonal principal.
- *Uso de los determinantes:**
- **Resolver sistemas de ecuaciones lineales:*** Un sistema de ecuaciones tiene una solución única si y solo si el determinante de la matriz de coeficientes es distinto de cero.
- **Calcular áreas y volúmenes:*** El valor absoluto del determinante de una matriz formada por los vectores que definen los lados de un paralelogramo en el plano o un paralelepípedo en el espacio es igual al área o volumen correspondiente.
- **Invertir matrices:*** Una matriz cuadrada A es invertible si y solo si su determinante es distinto de cero. Además, la matriz inversa se puede obtener dividiendo la adjunta de A por el determinante de A.
- **Determinar la independencia lineal:*** Un conjunto de vectores

```
import numpy as np

# Definir la matriz
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# Calcular el determinante
det_A = np.linalg.det(A)

# Imprimir el resultado
print("El determinante de la matriz A es:", det_A)
```

```
El determinante de la matriz A es: 0.0
```

Haz doble clic (o ingresa) para editar

Sistemas de ecuaciones de primer grado

Definición de sistema de ecuaciones de primer grado: Un sistema de ecuaciones de primer grado es un conjunto de ecuaciones lineales en las que todas las incógnitas tienen exponente 1 y no hay productos entre incógnitas. Por lo general, se representa de la siguiente manera:

$a_1x_1 + a_2x_2 + ... + a_nx_n = b_1$ $a_1'x_1 + a_2'x_2 + ... + a_n'x_n = b_1'$ ... $a_1"x_1 + a_2"x_2 + ... + a_n"x_n = b_1"$

Donde x_1, x_2, ..., x_n son las incógnitas, a_1, a_2, ..., a_n son los coeficientes de las incógnitas, y b_1, b_1', ..., $b_1"$ son los términos independientes.

Solución de un sistema de ecuaciones de primer grado: La solución de un sistema de ecuaciones de primer grado es el conjunto de valores para las incógnitas que satisfacen todas las ecuaciones simultáneamente. Puede haber tres tipos de soluciones:

Solución única: El sistema tiene una única solución, es decir, un conjunto de valores para las incógnitas que satisface todas las ecuaciones. En este caso, las ecuaciones son linealmente independientes y el sistema es consistente.

Infinitas soluciones: El sistema tiene infinitas soluciones, es decir, hay un conjunto de valores para las incógnitas que satisface todas las ecuaciones. En este caso, las ecuaciones son linealmente dependientes y el sistema es consistente.

Sin solución: El sistema no tiene solución, es decir, no hay ningún conjunto de valores para las incógnitas que satisfaga todas las ecuaciones. En este caso, las ecuaciones son inconsistentes.

Métodos de resolución de sistemas de ecuaciones de primer grado: Existen varios métodos para resolver sistemas de ecuaciones de primer grado, entre ellos:

- **Método de sustitución:** Consiste en despejar una variable en una de las ecuaciones y sustituirla en las demás ecuaciones. Repitiendo este proceso se obtiene la solución del sistema.
- **Método de igualación:** Se despeja una variable en una ecuación y se iguala a una variable despejada en otra ecuación. Se resuelve la ecuación resultante y se sustituye el valor encontrado en una de las ecuaciones originales para obtener el valor de la otra variable.
- **Método de eliminación:** Consiste en sumar o restar las ecuaciones del sistema de manera que se elimine una de las variables. Luego, se resuelve el sistema de ecuaciones resultante con una variable menos.
- **Método de matrices:** Se representa el sistema de ecuaciones en forma matricial y se utiliza la inversa de una matriz o la eliminación de Gauss-Jordan para obtener la solución.

```
import numpy as np

# Definimos los coeficientes del sistema de ecuaciones
A = np.array([[2, 1], [1, -3]])
B = np.array([9, -4])

# Resolvemos el sistema de ecuaciones utilizando el método de matrices
X = np.linalg.solve(A, B)

# Imprimimos la solución
print("La solución del sistema es:")
print("x =", X[0])
print("y =", X[1])
```

```
La solución del sistema es:
x = 3.2857142857142856
y = 2.4285714285714284
```

Haz doble clic (o ingresa) para editar

Sistemas de ecuaciones en matrices

Definición de sistema de ecuaciones: Un sistema de ecuaciones es un conjunto de ecuaciones lineales que se plantean simultáneamente. Por lo general, se busca encontrar los valores de las variables que satisfacen todas las ecuaciones del sistema.

Matriz de coeficientes: En un sistema de ecuaciones, los coeficientes de las variables se organizan en una matriz llamada matriz de coeficientes. Cada fila de la matriz representa una ecuación y cada columna corresponde a una variable.

Vector de términos independientes: En un sistema de ecuaciones, los términos independientes de cada ecuación se organizan en un vector llamado vector de términos independientes. Este vector tiene la misma cantidad de filas que la matriz de coeficientes.

Forma matricial del sistema de ecuaciones: Un sistema de ecuaciones puede expresarse en forma matricial utilizando la matriz de coeficientes y el vector de términos independientes. Se puede escribir como AX = B, donde A es la matriz de coeficientes, X es el vector de variables y B es el vector de términos independientes.

Matriz ampliada: Para resolver un sistema de ecuaciones utilizando operaciones matriciales, se forma una matriz ampliada que combina la matriz de coeficientes y el vector de términos independientes. Esta matriz tiene las mismas filas que la matriz de coeficientes y una columna adicional que representa al vector de términos independientes.

Operaciones elementales de filas: Las operaciones elementales de filas se utilizan para simplificar un sistema de ecuaciones. Estas operaciones incluyen intercambiar filas, multiplicar una fila por una constante no nula y sumar o restar una fila multiplicada por una constante a otra fila.

Método de eliminación de Gauss-Jordan: El método de eliminación de Gauss-Jordan es un procedimiento utilizado para resolver sistemas de ecuaciones en matrices. Consiste en aplicar operaciones elementales de filas a la matriz ampliada hasta obtener una forma escalonada reducida, donde se despejan las variables una a una.

Solución del sistema de ecuaciones: Un sistema de ecuaciones puede tener distintas soluciones. Puede ser compatible determinado si tiene una única solución, compatible indeterminado si tiene infinitas soluciones o incompatible si no tiene solución.

Matriz inversa: Si la matriz de coeficientes A es cuadrada y su determinante es diferente de cero, entonces existe la matriz inversa de A. La matriz inversa se denota como A^-1 y tiene la propiedad de que A * A^-1 = I, donde I es la matriz identidad.

Uso de matrices en aplicaciones: Los sistemas de ecuaciones en matrices tienen diversas aplicaciones en áreas como la física, la ingeniería, la economía y la computación. Por ejemplo,

se utilizan en el análisis de circuitos eléctricos, la resolución de problemas de optimización y la simulación de sistemas dinámicos.

```
import numpy as np

# Definición del sistema de ecuaciones
A = np.array([[2, 3], [1, -2]])  # Matriz de coeficientes
B = np.array([5, -3])            # Vector de términos independientes

# Resolución del sistema de ecuaciones
X = np.linalg.solve(A, B)

# Imprimir la solución
print("La solución del sistema de ecuaciones es:")
print("x =", X[0])
print("y =", X[1])
```

```
La solución del sistema de ecuaciones es:
x = 0.1428571428571428
y = 1.5714285714285714
```

MÉTODO CRAMMER

El método de Cramer es una técnica utilizada en álgebra lineal para resolver sistemas de ecuaciones lineales utilizando determinantes.

Supongamos que tenemos un sistema de ecuaciones lineales de la forma:

a11x1 + a12x2 + ... + a1nxn = b1 a21x1 + a22x2 + ... + a2nxn = b2 ..... am1x1 + am2x2 + ... + amnxn = bm

Donde aij son los coeficientes de las variables xj, y bi son los términos constantes. El objetivo es encontrar los valores de las variables x1, x2, ..., xn que satisfacen todas las ecuaciones simultáneamente.

El método de Cramer utiliza determinantes para resolver este sistema. En primer lugar, calculamos el determinante principal D del sistema, que se obtiene tomando los coeficientes de las variables en la matriz de coeficientes:

D = | a11 a12 ... a1n | | a21 a22 ... a2n | | . . ... . | | am1 am2 ... amn |

Si el determinante principal D es diferente de cero (D ≠ 0), entonces el sistema tiene una única solución y se puede utilizar el método de Cramer para encontrarla.

Luego, calculamos los determinantes Dx1, Dx2, ..., Dxj, ..., Dxk, ..., Dxn, donde cada determinante Dxj se obtiene al reemplazar la columna j de la matriz de coeficientes con el vector de términos constantes:

Dxj = | a11 a12 ... bi ... a1n | | a21 a22 ... b2 ... a2n | | . . .. | | am1 am2 ... bm ... amn |

A continuación, podemos encontrar el valor de cada variable xj dividiendo el determinante correspondiente Dxj por el determinante principal D:

xj = Dxj / D

Por lo tanto, el sistema de ecuaciones tiene la siguiente solución única:

x1 = Dx1 / D, x2 = Dx2 / D, ..., xn = Dxn / D

- Es importante tener en cuenta que el método de Cramer tiene ciertas limitaciones. En primer lugar, solo se puede aplicar a sistemas de ecuaciones lineales con el mismo número de ecuaciones y variables. Además, si el determinante principal D es igual a cero (D = 0), el método de Cramer no puede proporcionar una solución única y el sistema puede no tener solución o tener infinitas soluciones.

```
import numpy as np

def cramer_solver(coefficients, constants):
    # Verificar si el sistema tiene el mismo número de ecuaciones y variables
    n = len(coefficients)
    if len(coefficients) != n or len(coefficients[0]) != n or len(constants) != n:
        raise ValueError("El sistema de ecuaciones no es válido.")

    # Calcular el determinante principal D
    D = np.linalg.det(coefficients)

    # Verificar si el determinante principal es diferente de cero
    if D == 0:
        raise ValueError("El sistema de ecuaciones no tiene solución única.")

    # Calcular los valores de las variables utilizando el método de Cramer
    solutions = []
    for j in range(n):
        # Crear una copia de la matriz de coeficientes y reemplazar la columna j con el ve
        matrix_copy = coefficients.copy()
        matrix_copy[:, j] = constants

        # Calcular el determinante Dxj
        Dxj = np.linalg.det(matrix_copy)

        # Calcular el valor de la variable xj
        xj = Dxj / D
        solutions.append(xj)

    return solutions

# Ejemplo de sistema de ecuaciones lineales
coefficients = np.array([[2, 3, -1], [4, -2, 3], [1, 1, 1]])
constants = np.array([7, 3, 4])

# Resolver el sistema utilizando el método de Cramer
solutions = cramer_solver(coefficients, constants)
```

```
# Imprimir las soluciones
print("Soluciones:")
for i, solution in enumerate(solutions):
    print(f"x{i+1} = {solution}")
```

```
Soluciones:
x1 = 1.0
x2 = 2.0000000000000013
x3 = 1.0
```

MÉTODO LU

El método LU, también conocido como factorización LU, es una técnica utilizada en el álgebra lineal para descomponer una matriz en el producto de dos matrices triangulares. Esta descomposición se utiliza para resolver sistemas de ecuaciones lineales, calcular determinantes y realizar otros cálculos relacionados con matrices.

La factorización LU de una matriz A se puede expresar como A = LU, donde L es una matriz triangular inferior y U es una matriz triangular superior. La matriz L tiene unos en su diagonal principal y ceros por encima, mientras que la matriz U tiene ceros por debajo de su diagonal principal.

El proceso de factorización LU se realiza generalmente utilizando el método de eliminación de Gauss. Consiste en aplicar operaciones elementales de fila a la matriz original para obtener las matrices L y U. A continuación, se utilizan estas matrices para resolver sistemas de ecuaciones y realizar otros cálculos.

- El método LU tiene varias ventajas en comparación con otros métodos de resolución de sistemas de ecuaciones. Una vez que la matriz A se ha factorizado en L y U, es posible resolver sistemas de ecuaciones lineales simplemente sustituyendo y resolviendo los sistemas triangular inferior y superior. Esto es más eficiente computacionalmente, especialmente cuando se tienen varios sistemas de ecuaciones con la misma matriz A.
- Además, la factorización LU se utiliza para calcular el determinante de una matriz. El determinante de A es el producto de los elementos diagonales de U. Esto facilita el cálculo del determinante en comparación con otros métodos.

También es posible utilizar la factorización LU para calcular la matriz inversa de A. Una vez que se ha factorizado A en L y U, se pueden resolver sistemas de ecuaciones para encontrar las columnas de la matriz inversa.

Sin embargo, es importante destacar que la factorización LU solo es posible si la matriz A es no singular, es decir, si tiene un determinante distinto de cero. Si la matriz es singular, la factorización LU no se puede realizar.

```
import numpy as np
from scipy.linalg import lu

def factorizacion_lu(matriz):
    # Convertir la matriz a un arreglo NumPy
    arr = np.array(matriz, dtype=float)

    # Aplicar la factorización LU utilizando la función lu de SciPy
    P, L, U = lu(arr)

    return P, L, U

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

P, L, U = factorizacion_lu(matriz)

print("Matriz P:")
print(P)

print("Matriz L:")
print(L)

print("Matriz U:")
print(U)
```

```
Matriz P:
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
Matriz L:
[[1.         0.         0.        ]
 [0.14285714 1.         0.        ]
 [0.57142857 0.5        1.        ]]
Matriz U:
[[7.         8.         9.        ]
 [0.         0.85714286 1.71428571]
 [0.         0.         0.        ]]
```

SUBESPACIOS VECTORIALES

Definición de subespacio vectorial: Un subespacio vectorial V de un espacio vectorial W es un conjunto no vacío que cumple tres propiedades:

- El vector cero (0) siempre está en V.
- Para cualquier par de vectores u y v en V, la suma de u y v también está en V.
- Para cualquier vector u en V y cualquier escalar c, el producto escalar cu también está en V.

Propiedades de los subespacios vectoriales:

- Un subconjunto no vacío V de un espacio vectorial W es un subespacio vectorial si y solo si V es cerrado bajo la suma de vectores y la multiplicación por escalares.
- La intersección de dos o más subespacios vectoriales también es un subespacio vectorial.
- La suma de dos subespacios vectoriales V y W, denotada como V + W, es el conjunto de todas las sumas posibles de vectores de V y W.
- El subespacio generado por un conjunto de vectores es el conjunto de todas las combinaciones lineales de esos vectores.

Bases y dimensiones de subespacios:

- Una base de un subespacio vectorial es un conjunto de vectores linealmente independientes que generan el subespacio.
- La dimensión de un subespacio vectorial V, denotada como dim(V), es el número de vectores en cualquier base de V.
- La dimensión de un subespacio es siempre un número no negativo.
- El rango de una matriz es igual a la dimensión del espacio de columnas de la matriz, que es un subespacio vectorial de R^n.

Ejemplos de subespacios vectoriales:

- El espacio vectorial R^n, que consiste en todos los vectores n-dimensionales con componentes reales, es un subespacio vectorial de sí mismo.
- El espacio de soluciones de un sistema homogéneo de ecuaciones lineales es un subespacio vectorial.
- El conjunto de todas las matrices simétricas n × n forma un subespacio vectorial de las matrices n × n.
- El conjunto de todas las funciones continuas en un intervalo [a, b] forma un subespacio vectorial del espacio de todas las funciones definidas en [a, b].

```
import numpy as np

# Creamos un subespacio vectorial en R^3
# Definimos una base de dos vectores
v1 = np.array([1, 2, 3])
v2 = np.array([4, 5, 6])

# Generamos el subespacio vectorial V
V = np.array([v1, v2])

# Comprobamos si el vector cero está en V
zero_vector = np.zeros(3)
is_zero_in_V = np.all(V == zero_vector)
print("¿El vector cero está en V?", is_zero_in_V)

# Comprobamos si V es cerrado bajo la suma de vectores
u = np.array([7, 8, 9])
v = np.array([10, 11, 12])
```

```
sum_uv = u + v
is_sum_in_V = np.all(np.isin(sum_uv, V))
print("¿V es cerrado bajo la suma de vectores?", is_sum_in_V)

# Comprobamos si V es cerrado bajo la multiplicación por escalares
c = 3
mult_cv = c * v1
is_mult_in_V = np.all(np.isin(mult_cv, V))
print("¿V es cerrado bajo la multiplicación por escalares?", is_mult_in_V)
```

```
¿El vector cero está en V? False
¿V es cerrado bajo la suma de vectores? False
¿V es cerrado bajo la multiplicación por escalares? False
```

Ecuaciones de una recta

Las ecuaciones de una recta en álgebra lineal son una forma de representar geométricamente una línea recta en un plano o en el espacio tridimensional. Hay diferentes formas de expresar estas ecuaciones, pero las más comunes son la forma general, la forma punto-pendiente y la forma pendiente-intersección.

- ***Forma general:*** La ecuación de una recta en la forma general se expresa como: Ax + By + C = 0

Donde A, B y C son constantes y las variables x e y representan las coordenadas de los puntos en el plano. Esta forma general puede ser útil para realizar operaciones algebraicas con las rectas, como la determinación de intersecciones o la obtención de la pendiente.

- ***Forma punto-pendiente:*** La ecuación de una recta en la forma punto-pendiente se expresa como: $y - y_1 = m(x - x_1)$

Donde (x_1, y_1) son las coordenadas de un punto en la recta, y m es la pendiente de la recta. Esta forma es especialmente útil cuando se conoce un punto en la recta y su pendiente.

- ***Forma pendiente-intersección:*** La ecuación de una recta en la forma pendiente-intersección se expresa como: y = mx + b

Donde m es la pendiente de la recta y b es la ordenada al origen, es decir, el valor de y cuando x es igual a cero. Esta forma es útil cuando se conocen la pendiente y la intersección de la recta con el eje y.

```
def calcular_ecuacion_recta(x1, y1, pendiente):
    # Calcula la ecuación de una recta en la forma punto-pendiente
    # dados un punto (x1, y1) y la pendiente de la recta

    # y - y1 = m(x - x1)
    # y = mx - mx1 + y1

    ecuacion = f"y = {pendiente}x - {pendiente * x1} + {y1}"
```

```
    return ecuacion

# Ejemplo de uso
x1 = 2
y1 = 3
pendiente = 1.5

ecuacion_recta = calcular_ecuacion_recta(x1, y1, pendiente)
print(ecuacion_recta)
```

```
 y = 1.5x - 3.0 + 3
```

OPERACIONES ENTRE VECTORES

Suma de vectores: La suma de vectores se define componente a componente. Dados dos vectores del mismo tamaño, por ejemplo, u = (u1, u2, ..., un) y v = (v1, v2, ..., vn), la suma de estos vectores se obtiene sumando las componentes correspondientes: u + v = (u1 + v1, u2 + v2, ..., un + vn)

- La suma de vectores es conmutativa, es decir, u + v = v + u, y también es asociativa, es decir, (u + v) + w = u + (v + w), donde u, v y w son vectores del mismo tamaño.

Multiplicación por un escalar: La multiplicación de un vector por un escalar se realiza multiplicando cada componente del vector por el escalar. Si tienes un vector u = (u1, u2, ..., un) y un escalar k, la multiplicación se realiza de la siguiente manera: k * u = (k * u1, k * u2, ..., k * un)

Algunas propiedades importantes de la multiplicación por un escalar son:

- k * (u + v) = k * u + k * v, donde k es un escalar y u y v son vectores del mismo tamaño.
- (k1 * k2) * u = k1 * (k2 * u), donde k1 y k2 son escalares y u es un vector del mismo tamaño.

```
def sumar_vectores(u, v):
    # Verificar si los vectores tienen la misma longitud
    if len(u) != len(v):
        raise ValueError("Los vectores deben tener la misma longitud.")

    # Inicializar el vector resultado con ceros
    resultado = [0] * len(u)

    # Sumar las componentes correspondientes de los vectores
    for i in range(len(u)):
        resultado[i] = u[i] + v[i]

    return resultado

# Ejemplo de suma de vectores
u = [1, 2, 3]
v = [4, 5, 6]
```

```
suma = sumar_vectores(u, v)
print(suma)  # Salida: [5, 7, 9]
```

```
[5, 7, 9]
```

```
def multiplicar_por_escalar(u, k):
    # Multiplicar cada componente del vector por el escalar
    resultado = [k * componente for componente in u]
    return resultado

# Ejemplo de multiplicación por un escalar
u = [1, 2, 3]
k = 2
multiplicacion = multiplicar_por_escalar(u, k)
print(multiplicacion)  # Salida: [2, 4, 6]
```

```
[2, 4, 6]
```

Conjunto Linealmente dependiente e independiente

Los conceptos de conjunto linealmente dependiente y conjunto linealmente independiente son fundamentales para comprender la estructura y las propiedades de los vectores en un espacio vectorial.

Conjunto Linealmente Dependiente: Un conjunto de vectores se considera linealmente dependiente si existe una combinación lineal no trivial de esos vectores que suma cero. En otras palabras, un conjunto de vectores $\{v_1, v_2, ..., v_n\}$ es linealmente dependiente si existen coeficientes no todos iguales a cero $(a_1, a_2, ..., a_n)$ tal que:

$a_1v_1 + a_2v_2 + ... + a_nv_n = 0$

Para que un conjunto sea linealmente dependiente, basta con que exista una combinación lineal no trivial que de cero, es decir, que no todos los coeficientes sean cero.

Conjunto Linealmente Independiente: Un conjunto de vectores se considera linealmente independiente si no existe una combinación lineal no trivial de esos vectores que suma cero, excepto la combinación lineal trivial en la que todos los coeficientes son cero. En otras palabras, un conjunto de vectores $\{v_1, v_2, ..., v_n\}$ es linealmente independiente si la única forma de obtener una suma cero es cuando todos los coeficientes son cero, es decir:

$a_1v_1 + a_2v_2 + ... + a_nv_n = 0$

Solo tiene solución $a_1 = a_2 = ... = a_n = 0$.

En un conjunto linealmente independiente, no existe una dependencia lineal entre los vectores que lo componen.

Relación entre Conjunto Linealmente Dependiente e Independiente: Si un conjunto de vectores es linealmente dependiente, al menos uno de los vectores puede escribirse como combinación lineal de los demás. Si un conjunto de vectores es linealmente independiente, ninguno de los vectores puede expresarse como una combinación lineal de los demás.

Además, se cumple la siguiente propiedad:

- Si un conjunto de vectores es linealmente dependiente y se agrega un vector adicional a ese conjunto, el nuevo conjunto resultante también será linealmente dependiente. Por el contrario, si un conjunto de vectores es linealmente independiente y se agrega un vector adicional, el nuevo conjunto resultante será linealmente dependiente si y solo si el vector adicional es una combinación lineal de los vectores originales.

```
import numpy as np

# Conjunto linealmente dependiente
v1 = np.array([1, 2, 3])
v2 = np.array([2, 4, 6])
v3 = np.array([3, 6, 9])

# Comprobación de la dependencia lineal
combination = 2 * v1 - v2 + 0.5 * v3
if np.array_equal(combination, np.zeros(3)):
    print("El conjunto es linealmente dependiente")
else:
    print("El conjunto es linealmente independiente")
```

```
El conjunto es linealmente independiente
```

```
import numpy as np

# Conjunto linealmente independiente
v1 = np.array([1, 0, 0])
v2 = np.array([0, 1, 0])
v3 = np.array([0, 0, 1])

# Comprobación de la dependencia lineal
combination = 2 * v1 + 3 * v2 - 4 * v3
if np.array_equal(combination, np.zeros(3)):
    print("El conjunto es linealmente dependiente")
else:
    print("El conjunto es linealmente independiente")
```

```
El conjunto es linealmente independiente
```

CAMBIO DE BASE

El cambio de base es un concepto fundamental que permite expresar un vector o una matriz en términos de una nueva base. Esto resulta útil en diversos contextos, como la resolución de sistemas de ecuaciones lineales, la diagonalización de matrices y la representación de transformaciones lineales.

Aspectos clave sobre el cambio de base en álgebra lineal:

- **Definición de una base:** Una base es un conjunto de vectores linealmente independientes que generan todo el espacio vectorial. En un espacio vectorial de dimensión n, una base consiste en n vectores.
- **Coordenadas:** Cada vector en un espacio vectorial puede expresarse de manera única como una combinación lineal de los vectores de la base. Estas combinaciones lineales se representan mediante coordenadas. Por ejemplo, en un espacio vectorial tridimensional, un vector puede expresarse como una combinación lineal de los tres vectores de la base, y las coordenadas corresponden a los coeficientes de esta combinación lineal.
- **Matriz de cambio de base:** Si se tiene una base inicial y una base deseada, es posible establecer una relación entre las coordenadas de un vector en ambas bases mediante una matriz de cambio de base. Esta matriz se obtiene colocando como columnas los vectores de la base inicial expresados en términos de la base deseada.
- **Cambio de coordenadas de un vector:** Dado un vector expresado en términos de una base inicial, se puede obtener su representación en términos de una base deseada multiplicando la matriz de cambio de base por el vector original. Esta operación se realiza mediante una multiplicación matricial.
- **Cambio de coordenadas de una matriz:** Del mismo modo, es posible cambiar la base de una matriz aplicando el cambio de base a cada una de sus columnas. Cada columna se multiplica por la matriz de cambio de base correspondiente, obteniendo así la matriz expresada en términos de la nueva base.
- **Matriz de cambio de base inversa:** La matriz de cambio de base también puede invertirse para obtener el cambio de base inverso. Esto permite expresar un vector o una matriz en términos de la base original, si se conoce su representación en términos de la nueva base.
- **Transformaciones lineales:** El cambio de base es especialmente útil al estudiar transformaciones lineales. Al cambiar la base, es posible encontrar una matriz asociada a la transformación lineal que se exprese de manera más simple o que tenga una forma diagonal. Esto facilita el estudio de las propiedades de la transformación.

```
import numpy as np

# Definir la base inicial y las coordenadas del vector original
base_inicial = np.array([[1, 0], [0, 1]])  # Base canónica en R^2
coordenadas_originales = np.array([2, 3])

# Definir la base deseada
```

```
base_deseada = np.array([[1, 1], [1, -1]])  # Base de vectores propios de una matriz

# Calcular la matriz de cambio de base
matriz_cambio_base = np.linalg.inv(base_deseada) @ base_inicial

# Cambiar las coordenadas del vector a la nueva base
coordenadas_nuevas = matriz_cambio_base @ coordenadas_originales

print("Coordenadas originales:", coordenadas_originales)
print("Coordenadas en la nueva base:", coordenadas_nuevas)
```

```
Coordenadas originales: [2 3]
Coordenadas en la nueva base: [ 2.5 -0.5]
```

```
import numpy as np

# Definir la base inicial y la matriz original
base_inicial = np.array([[1, 0], [0, 1]])  # Base canónica en R^2
matriz_orig = np.array([[2, 1], [1, 3]])

# Definir la base deseada
base_deseada = np.array([[1, 1], [1, -1]])  # Base de vectores propios de una matriz

# Calcular la matriz de cambio de base
matriz_cambio_base = np.linalg.inv(base_deseada) @ base_inicial

# Cambiar la matriz a la nueva base
matriz_nueva = matriz_cambio_base @ matriz_orig @ np.linalg.inv(matriz_cambio_base)

print("Matriz original:\n", matriz_orig)
print("Matriz en la nueva base:\n", matriz_nueva)
```

```
Matriz original:
 [[2 1]
 [1 3]]
Matriz en la nueva base:
 [[ 3.5 -0.5]
 [-0.5  1.5]]
```

CÓNICAS

Elementos de las cónicas

Superficie: una superficie cónica de revolución está engendrada por la rotación de una recta alrededor de otra recta fija, llamada eje, a la que corta de modo oblicuo.

Generatriz: la generatriz es una cualquiera de las rectas oblicuas.

Vértice: el vértice es el punto central donde se cortan las generatrices.

Hojas: las hojas son las dos partes en las que el vértice divide a la superficie cónica de revolución.

Sección: se denomina sección cónica a la curva intersección de un cono con un plano que no pasa por su vértice. En función de la relación existente entre el ángulo de conicidad (\alpha) y la inclinación del plano respecto del eje del cono (\beta), pueden obtenerse diferentes secciones cónicas.

ELIPSE

La elipse es la sección producida en una superficie cónica de revolución por un plano oblicuo al eje, que no sea paralelo a la generatriz y que forme con el mismo un ángulo mayor que el que forman eje y generatriz.

La elipse es una curva cerrada.

CIRCUNFERENCIA

La circunferencia es la sección producida por un plano perpendicular al eje.

La circunferencia es un caso particular de elipse.

PARÁBOLA

La parábola es la sección producida en una superficie cónica de revolución por un plano oblicuo al eje, siendo paralelo a la generatriz.

La parábola es una curva abierta que se prolonga hasta el infinito.

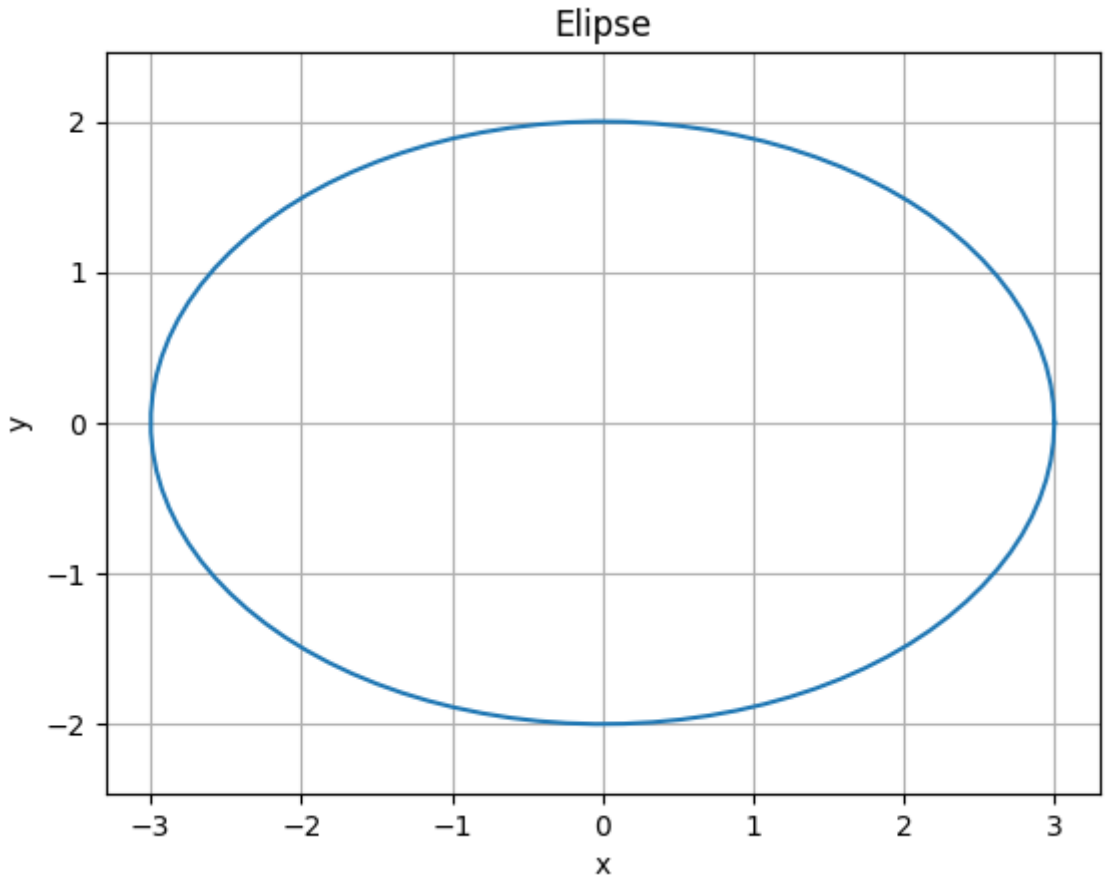
HIPÉRBOLA

La hipérbola es la sección producida en una superficie cónica de revolución por un plano oblicuo al eje, formando con él un ángulo menor al que forman eje y generatriz, por lo que incide en las dos hojas de la superficie cónica.

La hipérbola es una curva abierta que se prolonga indefinidamente y consta de dos ramas separadas.

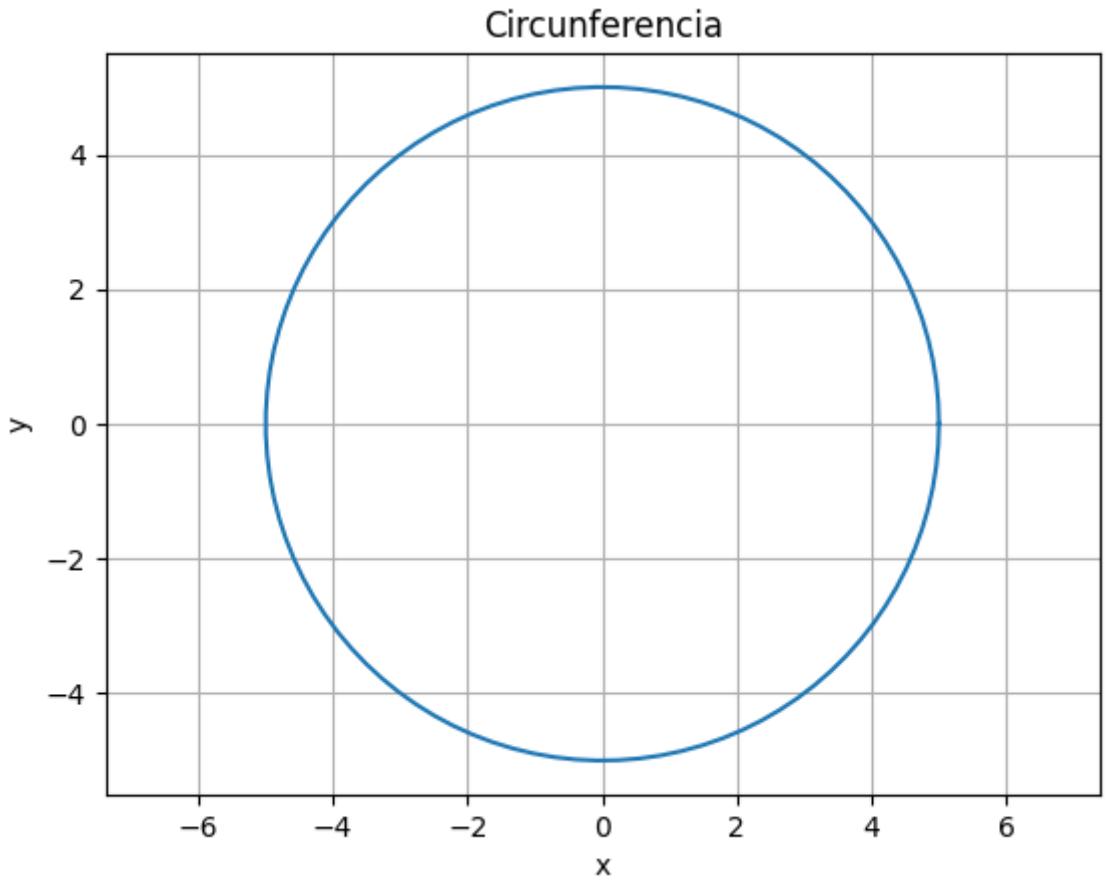
```
import numpy as np
import matplotlib.pyplot as plt

# Definir los parámetros de la elipse
a = 3  # Semieje mayor
b = 2  # Semieje menor

# Generar valores para el ángulo theta
theta = np.linspace(0, 2*np.pi, 100)

# Calcular los valores correspondientes en el eje x
x = a * np.cos(theta)

# Calcular los valores correspondientes en el eje y
y = b * np.sin(theta)
```

```
# Graficar la elipse
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Elipse')
plt.grid(True)
plt.axis('equal')
plt.show()
```

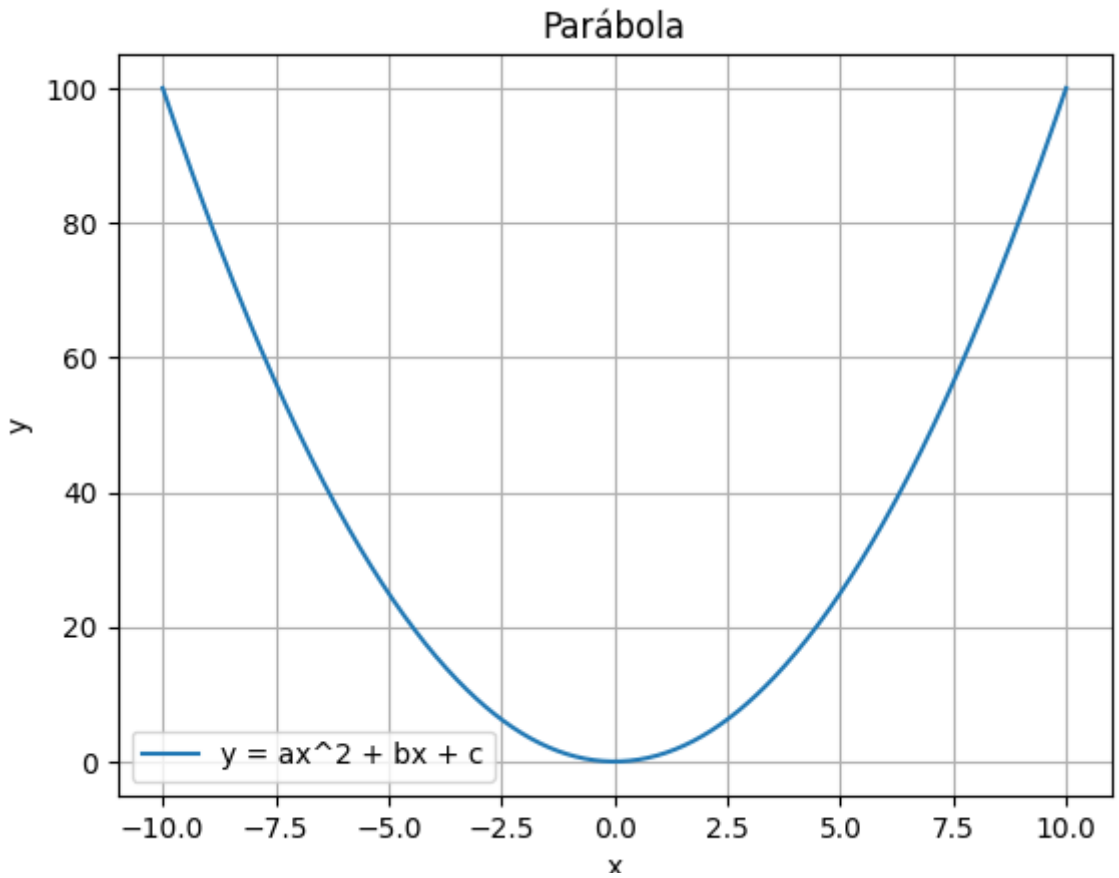



```
import numpy as np
import matplotlib.pyplot as plt

# Definir el radio de la circunferencia
r = 5

# Generar valores para el ángulo theta
theta = np.linspace(0, 2*np.pi, 100)

# Calcular los valores correspondientes en el eje x
x = r * np.cos(theta)

# Calcular los valores correspondientes en el eje y
y = r * np.sin(theta)

# Graficar la circunferencia
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
```

```
plt.title('Circunferencia')
plt.grid(True)
plt.axis('equal')
plt.show()
```

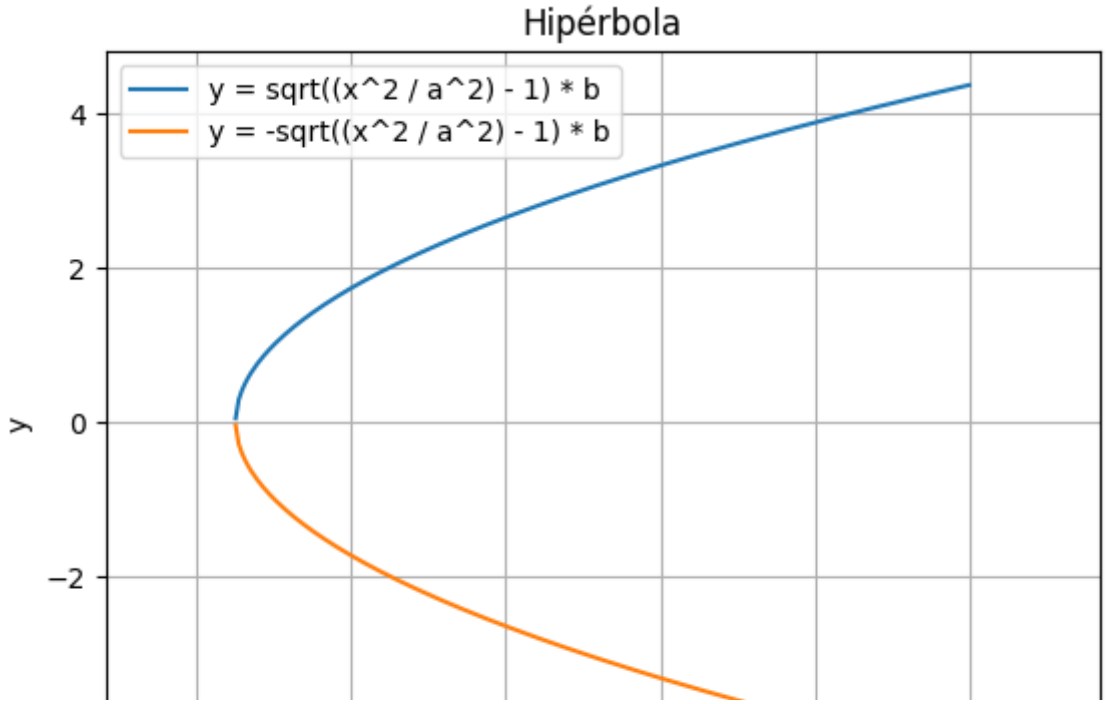



```
import numpy as np
import matplotlib.pyplot as plt

# Definir los parámetros de la parábola
a = 1  # Coeficiente cuadrático
b = 0  # Coeficiente lineal
c = 0  # Término independiente

# Generar valores para el eje x
x = np.linspace(-10, 10, 500)

# Calcular los valores correspondientes en el eje y
y = a * x**2 + b * x + c

# Graficar la parábola
plt.plot(x, y, label='y = ax^2 + bx + c')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Parábola')
plt.legend()
plt.grid(True)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

# Definir los parámetros de la hipérbola
a = 2  # Semieje mayor
b = 1  # Semieje menor

# Generar valores para el eje x
x = np.linspace(-10, 10, 500)

# Calcular los valores correspondientes en el eje y
y1 = np.sqrt((x * 2 / a * 2) - 1) * b
y2 = -np.sqrt((x * 2 / a * 2) - 1) * b

# Graficar la hipérbola
plt.plot(x, y1, label='y = sqrt((x^2 / a^2) - 1) * b')
plt.plot(x, y2, label='y = -sqrt((x^2 / a^2) - 1) * b')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Hipérbola')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```

```
<ipython-input-16-b2e66ab4a2c1>:12: RuntimeWarning: invalid value encountered in sqr
  y1 = np.sqrt((x * 2 / a * 2) - 1) * b
<ipython-input-16-b2e66ab4a2c1>:13: RuntimeWarning: invalid value encountered in sqr
  y2 = -np.sqrt((x * 2 / a * 2) - 1) * b
```



TRANSFORMACIONES LINEALES

Las transformaciones lineales son operaciones que se aplican a vectores. Estas transformaciones pueden modificar las propiedades geométricas y algebraicas de los vectores, como su escala, translación o rotación. Aquí una descripción de cada una de estas transformaciones lineales:

Escala (o dilatación): La escala es una transformación lineal que modifica la longitud de un vector. Puede aumentar o disminuir su tamaño en función de un factor de escala. Si el factor de escala es mayor que 1, el vector se expande, mientras que si es menor que 1, el vector se contrae.

- En general, la escala se puede realizar multiplicando cada componente del vector por el factor de escala correspondiente.

Translación: La translación es una transformación lineal que desplaza un vector en una dirección específica. Se suma un vector de translación constante a cada componente del vector original. El vector de translación define el desplazamiento en cada dirección.

- Por ejemplo, si se desea trasladar un vector en el plano xy, se sumaría un vector de translación (a, b) a las coordenadas (x, y) del vector original.

Rotación: La rotación es una transformación lineal que gira un vector alrededor de un punto fijo en un cierto ángulo. En dos dimensiones, una rotación se puede describir mediante una matriz de rotación. El ángulo de rotación determina la cantidad de giro.

- Por ejemplo, en el plano xy, una rotación en sentido antihorario se puede lograr mediante la multiplicación del vector original (x, y) por una matriz de rotación adecuada.

*Es importante destacar que todas estas transformaciones lineales son lineales porque preservan la estructura lineal de los vectores.**

Esto significa que cumplen con dos propiedades:

- Preservación de la suma y preservación de la multiplicación por un escalar. En otras palabras, la suma de dos vectores transformados es igual a la transformación de la suma de los vectores originales, y el producto de un vector transformado por un escalar es igual a la transformación del producto del vector original por ese escalar.

```
import numpy as np

# Vector original
vector = np.array([1, 2, 3])

# Factor de escala
factor_escala = 2

# Transformación de escala
vector_escala = vector * factor_escala

print("Vector original:", vector)
print("Vector escalado:", vector_escala)
```

```
Vector original: [1 2 3]
Vector escalado: [2 4 6]
```

```
import numpy as np

# Vector original
vector = np.array([1, 2, 3])

# Vector de translación
vector_translacion = np.array([2, 3, 1])

# Transformación de translación
vector_transladado = vector + vector_translacion

print("Vector original:", vector)
print("Vector trasladado:", vector_transladado)
```

```
Vector original: [1 2 3]
Vector trasladado: [3 5 4]
```

```
import numpy as np

# Vector original en el plano xy
```

```
vector = np.array([1, 0])

# Ángulo de rotación en radianes
angulo = np.pi / 4  # Rotación de 45 grados en sentido antihorario

# Matriz de rotación en el plano xy
matriz_rotacion = np.array([[np.cos(angulo), -np.sin(angulo)],
                            [np.sin(angulo), np.cos(angulo)]])

# Transformación de rotación
vector_rotado = np.dot(matriz_rotacion, vector)

print("Vector original:", vector)
print("Vector rotado:", vector_rotado)
```

```
Vector original: [1 0]
Vector rotado: [0.70710678 0.70710678]
```

VECTOR PROPIO

Los vectores propios son de gran importancia en la teoría de transformaciones lineales porque capturan direcciones especiales en las que una transformación lineal actúa simplemente por estiramiento o contracción. Son ampliamente utilizados en diversas áreas, como física, informática, estadísticas y ciencias de la ingeniería.

Aquí tienes algunos conceptos clave sobre los vectores propios:

- **Definición:** Dada una transformación lineal T, un vector no nulo v es un vector propio si existe un escalar λ tal que T(v) = λv.
- **Valor propio:** El escalar λ en la definición anterior se conoce como valor propio o autovalor asociado al vector propio v.
- **Espacio propio:** El conjunto de todos los vectores propios correspondientes a un mismo valor propio λ forma un subespacio vectorial llamado espacio propio asociado a λ.
- **Diagonalización:** Una matriz cuadrada A se dice diagonalizable si existe una matriz inversible P tal que P^(-1)AP es una matriz diagonal. En este caso, las columnas de P son los vectores propios de A, y los elementos diagonales de P^(-1)AP son los valores propios correspondientes.
- **Multiplicidad algebraica y geométrica:** Un valor propio λ tiene una multiplicidad algebraica igual al número de veces que aparece como raíz del polinomio característico de la matriz. La multiplicidad geométrica de λ es la dimensión del espacio propio asociado a ese valor propio.
- **Ortogonalidad:** Si dos vectores propios corresponden a diferentes valores propios, entonces son ortogonales entre sí. Esto significa que su producto escalar es cero.

- **Matriz diagonalizable:** Una matriz cuadrada A es diagonalizable si y solo si tiene un conjunto completo de vectores propios linealmente independientes.
- **Aplicaciones:** Los vectores propios y los valores propios tienen numerosas aplicaciones, como la reducción de dimensionalidad, la compresión de datos, la resolución de sistemas de ecuaciones diferenciales lineales, el análisis de redes y la clasificación de datos.

```
import numpy as np

# Ejemplo 1: Calcular vectores propios y valores propios de una matriz
A = np.array([[2, -1], [4, 3]])

# Usar la función eig de NumPy para calcular los valores propios (eigenvalues) y vectores
eigenvalues, eigenvectors = np.linalg.eig(A)

print("Valores propios:")
print(eigenvalues)

print("Vectores propios:")
print(eigenvectors)

# Ejemplo 2: Calcular vectores propios y valores propios de una matriz diagonalizable
B = np.array([[3, 0], [0, -2]])

eigenvalues, eigenvectors = np.linalg.eig(B)

print("Valores propios:")
print(eigenvalues)

print("Vectores propios:")
print(eigenvectors)
```

```
Valores propios:
[2.5+1.93649167j 2.5-1.93649167j]
Vectores propios:
[[-0.1118034 +0.4330127j -0.1118034 -0.4330127j]
 [ 0.89442719+0.j         0.89442719-0.j       ]]
Valores propios:
[ 3. -2.]
Vectores propios:
[[1. 0.]
 [0. 1.]]
```