Instrucciones para Resolver Sistemas Ax=b y By=b

Se te proporciona una serie de tablas en la cual la última columna corresponde a un elemento conocido como **etiqueta**, y el resto de columnas se conoce como **características**. Tu tarea es llevar a cabo los siguientes pasos **sin el uso de funciones definidas (`def`)**, para que te familiarices con lo que sucede detrás al usar herramientas como scikit-learn de Python:

1. Descomponer las tablas

Descompón las tablas en:

- **Matriz A:** que contiene todas las características.
- **Vector b:** que contiene las etiquetas.

2. Generar la Matriz Ampliada B

Genera una matriz ampliada **B** añadiendo una primera columna de $1's$ y el resto con los valores de la matriz **A**, de modo que:

$$B=\left[1 \vee A\right)$$

3. Cálculo del Determinante

En los casos en que sea posible, calcula el determinante de las matrices **A** y **B**:

- Determinante de **A**: $ \text{det}(A) $
- Determinante de **B**: $ \text{det}(B) $

4. Cálculo del Rango

Calcula el rango de las matrices:

- **Rango de A**: $ \text{rank}(A) $
- **Rango de B**: $ \text{rank}(B) $

También, calcula el rango de las matrices ampliadas **[A|b]** y **[B|b]**:

$$\text{rank}\left(\left[A \vee b\right)\right) \text{y rank}\left(\left[B \vee b\right)\right)$$

5. Clasificación de los Sistemas

Clasifica los sistemas:

- **Ax=b**
- **By=b**

según la clasificación de sistemas de ecuaciones lineales (determinado, indeterminado, o incompatible) basado en los rangos obtenidos.

6. Resolución de los Sistemas

Resuelve los sistemas utilizando:

- **Matriz inversa** si el sistema es determinado: $ x = A^{-1}b $
- **Pseudoinversa** en caso de ser un sistema indeterminado: $ x = A^{+}b $

Haz lo mismo para **B**:

$$y = B^{-1}b \text{ o } y = B^{+¿b¿}$$

7. Análisis de Estabilidad

Realiza un análisis de estabilidad de los sistemas a través de dilataciones y contracciones de las soluciones en factores del:

- 0.5%, 1%, 2%, 5%, y 10%.

Analiza los cambios porcentuales que se generan en las soluciones b' perturbadas en relación con la b real.

8. Intervalos de Confianza

Con el análisis anterior, establece una clase de intervalos de confianza en terminos de porcentaje para las soluciones obtenidas, considerando que:

- Las **dilataciones** podrían simular el redondeo o pérdida de precisión en los decimales de las soluciones.
- Las **contracciones** podrían simular errores numéricos pequeños o fluctuaciones en los datos originales.

9. Interpolantes

Las soluciones de los sistemas **Ax=b** y **By=b** pueden utilizarse para construir interpolantes:

- **Sin término independiente** para **Ax=b**.
- **Con término independiente** para **By=b**.

Construye dichos interpolantes.

10. Comparación de los Interpolantes

Analiza cuál de los interpolantes proporciona una solución más cercana a los valores de b, y argumenta si eso sería suficiente para justificar si es una mejor solución para el problema de interpolación.

```
import numpy as np
from numpy.linalg import det, matrix_rank, inv, pinv
```

```
M = lambda x: x[:, :-1]

aM = lambda x,y: np.hstack((x, y))

v = lambda x: x[:, -1].reshape(-1, 1)

det_M = lambda x : det(x) if x.shape[0] == x.shape[1] else None

rng_M = lambda x : matrix_rank(x)

class_M_det = lambda x : "SCD" if (det_M(x) != 0 and det_M(x) != None)
else "SCI o SI"

class_M_rng = lambda x, y : "SCD or SCI" if rng_M(x) == rng_M(y) else
"SI"

x_inv = lambda x, y : inv(x).dot(y)

x_pinv = lambda x, y : pinv(x).dot(y)

dilataciones = [1.005, 1.01, 1.02, 1.05, 1.10]
contracciones = [0.995, 0.99, 0.98, 0.95, 0.90]

x_perturbadas = lambda x, lista: [x * i for i in lista]

b_perturbadas = lambda M, lista : [M.dot(i) for i in lista]
```

Tabla 1

```
# tabla (6x6)
tabla = np.array([[0.32047, 0.82356, 0.25645, 0.49580, 0.67355,
0.61427],
                  [0.29187, 0.44708, 0.19897, 0.89325, 0.73233,
0.89347],
                  [0.13883, 0.64831, 0.02541, 0.84798, 0.57868,
0.37958],
                  [0.89019, 0.36956, 0.23375, 0.45624, 0.18749,
0.25037],
                  [0.54278, 0.18745, 0.96885, 0.25942, 0.65492,
0.12485],
                  [0.76956, 0.06538, 0.49874, 0.82947, 0.86217,
0.78432]])
```

1. Descomponer las tablas en A y b

```
A = M(tabla)
b = v(tabla)
```

2. Generar la Matrices ampliadas B, Ab y Bb

```
B = aM(np.ones((A.shape[0], 1)),A)
Ab = aM(A,b)
Bb = aM(B,b)
```

3. Cálculo de los Determinantes de A y B

```
dA = det_M(A)
print("det(A): ",dA)
dB = det_M(B)
print("det(B): ",dB)
```

```
det(A):  None
det(B):  -0.0016271442410992075
```

4. Cálculo del Rango de las Matrices A, B, Ab y Bb

```
rA = rng_M(A)
print("rng(A): ",rA)
rAb = rng_M(Ab)
print("rng([A|b]): ",rAb)


rB = rng_M(B)
print("rng(B): ",rB)
rBb = rng_M(Bb)
print("rng([B|b]): ",rBb)
```

```
rng(A):  5
rng([A|b]):  6
rng(B):  6
rng([B|b]):  6
```

5. Clasificación de los Sistemas

```
cAd = class_M_det(A)
cAr = class_M_rng(A,Ab)

print("clasificacion por determinante para A: ", cAd)
print("clasificacion por rango para A: ", cAr)


cBd = class_M_det(B)
cBr = class_M_rng(B,Bb)

print("clasificacion por determinante para B: ", cBd)
print("clasificacion por rango para B: ", cBr)
```

```
clasificacion por determinante para A:  SCI o SI
clasificacion por rango para A:  SI
```

```
clasificacion por determinante para B:  SCD
clasificacion por rango para B:  SCD or SCI
```

6. Resolución de los Sistemas

```
xB_base = x_inv(B,b)
bB_base = B.dot(xB_base)
```

7. Análisis de Estabilidad (dilataciones y contracciones)

```
xB_dil = x_perturbadas(xB_base, dilataciones)
xB_cont = x_perturbadas(xB_base, contracciones)

bB_dil = b_perturbadas(B,xB_dil)
bB_cont = b_perturbadas(B,xB_cont)

print("Dilataciones")
print("================================================")
print("valores en x")
print("================================================")
for i in range(len(dilataciones)):
    error_relativo = np.linalg.norm(xB_base - xB_dil[i]) /
np.linalg.norm(xB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de x_base con dilatación númer
{i} : {error_porcentual:.2f}%")
print("================================================")
print("valores en b")
print("================================================")
for i in range(len(dilataciones)):
    error_relativo = np.linalg.norm(bB_base - bB_dil[i]) /
np.linalg.norm(bB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de b_base con dilatación número
{i}: {error_porcentual:.2f}%")
print("================================================")
print("Contracciones")
print("================================================")
print("valores en x")
print("================================================")
for i in range(len(contracciones)):
    error_relativo = np.linalg.norm(xB_base - xB_cont[i]) /
np.linalg.norm(xB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de x_base con contraccion númer
{i} : {error_porcentual:.2f}%")
print("================================================")
print("valores en b")
print("================================================")
for i in range(len(contracciones)):
```

```
    error_relativo = np.linalg.norm(bB_base - bB_cont[i]) /
np.linalg.norm(bB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de b_base con contraccion número
{i}: {error_porcentual:.2f}%")
```

```
Dilataciones
================================================
valores en x
================================================
Error relativo porcentual de x_base con dilatación númer 0 : 0.50%
Error relativo porcentual de x_base con dilatación númer 1 : 1.00%
Error relativo porcentual de x_base con dilatación númer 2 : 2.00%
Error relativo porcentual de x_base con dilatación númer 3 : 5.00%
Error relativo porcentual de x_base con dilatación númer 4 : 10.00%
================================================
valores en b
================================================
Error relativo porcentual de b_base con dilatación número 0: 0.50%
Error relativo porcentual de b_base con dilatación número 1: 1.00%
Error relativo porcentual de b_base con dilatación número 2: 2.00%
Error relativo porcentual de b_base con dilatación número 3: 5.00%
Error relativo porcentual de b_base con dilatación número 4: 10.00%
================================================
Contracciones
================================================
valores en x
================================================
Error relativo porcentual de x_base con contraccion númer 0 : 0.50%
Error relativo porcentual de x_base con contraccion númer 1 : 1.00%
Error relativo porcentual de x_base con contraccion númer 2 : 2.00%
Error relativo porcentual de x_base con contraccion númer 3 : 5.00%
Error relativo porcentual de x_base con contraccion númer 4 : 10.00%
================================================
valores en b
================================================
Error relativo porcentual de b_base con contraccion número 0: 0.50%
Error relativo porcentual de b_base con contraccion número 1: 1.00%
Error relativo porcentual de b_base con contraccion número 2: 2.00%
Error relativo porcentual de b_base con contraccion número 3: 5.00%
Error relativo porcentual de b_base con contraccion número 4: 10.00%
```

8. Intervalos de Confianza

Las soluciones del sistema soportan dilataciones y contracciones del 10%

```
error_max_dilatacion = 0.10
error_max_contraccion = 0.10
```

9. Interpolantes

```
interpolante_B = lambda r : sum([r[i]*xB_base[i] for i in
range(len(r))])
```

10 Comparación de los Interpolantes

```
RMSE_B =  (sum([(interpolante_B(B[i,:]) - b[i])**2 for i in
range(len(b))])/len(b))**0.5
RMSE_B
```

```
array([5.00791724e-15])
```

Tabla 2

```
# tabla (6x6)
tabla = np.array([[0.42533, 0.30128, 0.67274, 0.56891, 0.84945,
0.61348],
                  [0.89747, 0.25390, 0.18011, 0.45375, 0.61618,
0.23765],
                  [0.53472, 0.74909, 0.10292, 0.92582, 0.42937,
0.97256],
                  [0.82625, 0.66803, 0.48776, 0.61463, 0.06765,
0.81539],
                  [0.16523, 0.81656, 0.41254, 0.72114, 0.82603,
0.15945],
                  [0.72064, 0.37514, 0.57535, 0.25643, 0.53927,
0.69474]])
```

1. Descomponer las tablas en A y b

```
A = M(tabla)
b = v(tabla)
```

2. Generar la Matrices ampliadas B, Ab y Bb

```
B = aM(np.ones((A.shape[0], 1)),A)
Ab = aM(A,b)
Bb = aM(B,b)
```

3. Cálculo de los Determinantes de A y B

```
dA = det_M(A)
print("det(A): ",dA)
dB = det_M(B)
print("det(B): ",dB)
```

```
det(A):  None
det(B):  0.004879188221007908
```

4. Cálculo del Rango de las Matrices A, B, Ab y Bb

```
rA = rng_M(A)
print("rng(A): ",rA)
rAb = rng_M(Ab)
print("rng([A|b]): ",rAb)


rB = rng_M(B)
print("rng(B): ",rB)
rBb = rng_M(Bb)
print("rng([B|b]): ",rBb)
```

```
rng(A):  5
rng([A|b]):  6
rng(B):  6
rng([B|b]):  6
```

5. Clasificación de los Sistemas

```
cAd = class_M_det(A)
cAr = class_M_rng(A,Ab)

print("clasificacion por determinante para A: ", cAd)
print("clasificacion por rango para A: ", cAr)


cBd = class_M_det(B)
cBr = class_M_rng(B,Bb)

print("clasificacion por determinante para B: ", cBd)
print("clasificacion por rango para B: ", cBr)
```

```
clasificacion por determinante para A:  SCI o SI
clasificacion por rango para A:  SI
clasificacion por determinante para B:  SCD
clasificacion por rango para B:  SCD or SCI
```

6. Resolución de los Sistemas

```
xB_base = x_inv(B,b)
bB_base = B.dot(xB_base)
```

7. Análisis de Estabilidad (dilataciones y contracciones)

```
xB_dil = x_perturbadas(xB_base, dilataciones)
xB_cont = x_perturbadas(xB_base, contracciones)

bB_dil = b_perturbadas(B,xB_dil)
bB_cont = b_perturbadas(B,xB_cont)
```

```
print("Dilataciones")
print("================================================")
print("valores en x")
print("================================================")
for i in range(len(dilataciones)):
    error_relativo = np.linalg.norm(xB_base - xB_dil[i]) /
np.linalg.norm(xB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de x_base con dilatación númer
{i} : {error_porcentual:.2f}%")
print("================================================")
print("valores en b")
print("================================================")
for i in range(len(dilataciones)):
    error_relativo = np.linalg.norm(bB_base - bB_dil[i]) /
np.linalg.norm(bB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de b_base con dilatación número
{i}: {error_porcentual:.2f}%")
print("================================================")
print("Contracciones")
print("================================================")
print("valores en x")
print("================================================")
for i in range(len(contracciones)):
    error_relativo = np.linalg.norm(xB_base - xB_cont[i]) /
np.linalg.norm(xB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de x_base con contraccion númer
{i} : {error_porcentual:.2f}%")
print("================================================")
print("valores en b")
print("================================================")
for i in range(len(contracciones)):
    error_relativo = np.linalg.norm(bB_base - bB_cont[i]) /
np.linalg.norm(bB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de b_base con contraccion número
{i}: {error_porcentual:.2f}%")
```

```
Dilataciones
================================================
valores en x
================================================
Error relativo porcentual de x_base con dilatación númer 0 : 0.50%
Error relativo porcentual de x_base con dilatación númer 1 : 1.00%
Error relativo porcentual de x_base con dilatación númer 2 : 2.00%
Error relativo porcentual de x_base con dilatación númer 3 : 5.00%
Error relativo porcentual de x_base con dilatación númer 4 : 10.00%
================================================
```

```
valores en b
================================================
Error relativo porcentual de b_base con dilatación número 0: 0.50%
Error relativo porcentual de b_base con dilatación número 1: 1.00%
Error relativo porcentual de b_base con dilatación número 2: 2.00%
Error relativo porcentual de b_base con dilatación número 3: 5.00%
Error relativo porcentual de b_base con dilatación número 4: 10.00%
================================================
Contracciones
================================================
valores en x
================================================
Error relativo porcentual de x_base con contraccion númer 0 : 0.50%
Error relativo porcentual de x_base con contraccion númer 1 : 1.00%
Error relativo porcentual de x_base con contraccion númer 2 : 2.00%
Error relativo porcentual de x_base con contraccion númer 3 : 5.00%
Error relativo porcentual de x_base con contraccion númer 4 : 10.00%
================================================
valores en b
================================================
Error relativo porcentual de b_base con contraccion número 0: 0.50%
Error relativo porcentual de b_base con contraccion número 1: 1.00%
Error relativo porcentual de b_base con contraccion número 2: 2.00%
Error relativo porcentual de b_base con contraccion número 3: 5.00%
Error relativo porcentual de b_base con contraccion número 4: 10.00%
```

8. Intervalos de Confianza

```
error_max_dilatacion = 0.10
error_max_contraccion = 0.10
```

9. Interpolantes

```
interpolante_B = lambda r : sum([r[i]*xB_base[i] for i in
range(len(r))])
```

10 Comparación de los Interpolantes

```
RMSE_B =  (sum([(interpolante_B(B[i,:]) - b[i])**2 for i in
range(len(b))])/len(b))**0.5
print("RMSE para el interpolante CON termino independiente: ", RMSE_B)

RMSE para el interpolante CON termino independiente:  [1.62103017e-15]
```

Tabla 3

```
# tabla (8x7)
tabla = np.array([[0.74898, 0.28122, 0.52920, 0.56900, 0.91599,
```

```
0.26114, 0.85767],
                   [0.87788, 0.57300, 0.21391, 0.18360, 0.27809,
0.47191, 0.52021],
                   [0.87446, 0.54117, 0.65085, 0.02731, 0.92548,
0.95857, 0.84126],
                   [0.44242, 0.31208, 0.45062, 0.48809, 0.24730,
0.08539, 0.49884],
                   [0.36721, 0.45338, 0.60793, 0.01837, 0.66224,
0.12615, 0.33194],
                   [0.54924, 0.69931, 0.05451, 0.81977, 0.56267,
0.36233, 0.99719],
                   [0.77000, 0.20419, 0.72319, 0.47703, 0.62884,
0.70215, 0.00228],
                   [0.59750, 0.39309, 0.90533, 0.88076, 0.34626,
0.56525, 0.59347]])
```

1. Descomponer las tablas en A y b

```
A = M(tabla)
b = v(tabla)
```

2. Generar la Matrices ampliadas B, Ab y Bb

```
B = aM(np.ones((A.shape[0], 1)),A)
Ab = aM(A,b)
Bb = aM(B,b)
```

3. Cálculo de los Determinantes de A y B

```
dA = det_M(A)
print("det(A): ",dA)
dB = det_M(B)
print("det(B): ",dB)
```

```
det(A):  None
det(B):  None
```

4. Cálculo del Rango de las Matrices A, B, Ab y Bb

```
rA = rng_M(A)
print("rng(A): ",rA)
rAb = rng_M(Ab)
print("rng([A|b]): ",rAb)


rB = rng_M(B)
print("rng(B): ",rB)
rBb = rng_M(Bb)
print("rng([B|b]): ",rBb)
```

```
rng(A):  6
rng([A|b]):  7
rng(B):  7
rng([B|b]):  8
```

5. Clasificación de los Sistemas

```
cAd = class_M_det(A)
cAr = class_M_rng(A,Ab)

print("clasificacion por determinante para A: ", cAd)
print("clasificacion por rango para A: ", cAr)


cBd = class_M_det(B)
cBr = class_M_rng(B,Bb)

print("clasificacion por determinante para B: ", cBd)
print("clasificacion por rango para B: ", cBr)
```

```
clasificacion por determinante para A:  SCI o SI
clasificacion por rango para A:  SI
clasificacion por determinante para B:  SCI o SI
clasificacion por rango para B:  SI
```

Tabla 4

```
# tabla (8x7)
tabla = np.array([[0.43217, 0.27185, 0.63925, 0.14753, 0.65845,
0.72451, 0.92756],
                  [0.85973, 0.55467, 0.10348, 0.38429, 0.97862,
0.30109, 0.36874],
                  [0.18459, 0.66988, 0.47842, 0.82746, 0.13367,
0.93471, 0.74512],
                  [0.04917, 0.76520, 0.56264, 0.05618, 0.31561,
0.90881, 0.09857],
                  [0.70551, 0.97730, 0.84481, 0.27559, 0.12154,
0.54392, 0.64783],
                  [0.96104, 0.86563, 0.05429, 0.34809, 0.25537,
0.21155, 0.28367],
                  [0.57384, 0.78641, 0.19932, 0.14379, 0.47252,
0.20055, 0.18792],
                  [0.31864, 0.96378, 0.48521, 0.11467, 0.89523,
0.69090, 0.45634]])
```

1. Descomponer las tablas en A y b

```
A = M(tabla)
b = v(tabla)
```

2. Generar la Matrices ampliadas B, Ab y Bb

```
B = aM(np.ones((A.shape[0], 1)),A)
Ab = aM(A,b)
Bb = aM(B,b)
```

3. Cálculo de los Determinantes de A y B

```
dA = det_M(A)
print("det(A): ",dA)
dB = det_M(B)
print("det(B): ",dB)
```

```
det(A):  None
det(B):  None
```

4. Cálculo del Rango de las Matrices A, B, Ab y Bb

```
rA = rng_M(A)
print("rng(A): ",rA)
rAb = rng_M(Ab)
print("rng([A|b]): ",rAb)


rB = rng_M(B)
print("rng(B): ",rB)
rBb = rng_M(Bb)
print("rng([B|b]): ",rBb)
```

```
rng(A):  6
rng([A|b]):  7
rng(B):  7
rng([B|b]):  8
```

5. Clasificación de los Sistemas

```
cAd = class_M_det(A)
cAr = class_M_rng(A,Ab)

print("clasificacion por determinante para A: ", cAd)
print("clasificacion por rango para A: ", cAr)


cBd = class_M_det(B)
cBr = class_M_rng(B,Bb)
```

```
print("clasificacion por determinante para B: ", cBd)
print("clasificacion por rango para B: ", cBr)
```

```
clasificacion por determinante para A:  SCI o SI
clasificacion por rango para A:  SI
clasificacion por determinante para B:  SCI o SI
clasificacion por rango para B:  SI
```

Tabla 5

```
# tabla (6x7)
tabla = np.array([[0.42074, 0.65412, 0.71950, 0.81775, 0.39471,
0.74543, 0.67315],
                  [0.12866, 0.89460, 0.56291, 0.21824, 0.99422,
0.40157, 0.58752],
                  [0.51278, 0.70214, 0.28451, 0.45349, 0.07861,
0.13209, 0.24194],
                  [0.89752, 0.67005, 0.46527, 0.84486, 0.87365,
0.26328, 0.45833],
                  [0.18930, 0.49567, 0.95384, 0.27422, 0.10112,
0.65198, 0.78461],
                  [0.36893, 0.76034, 0.01245, 0.21353, 0.07645,
0.84261, 0.03150]])
```

1. Descomponer las tablas en A y b

```
A = M(tabla)
b = v(tabla)
```

2. Generar la Matrices ampliadas B, Ab y Bb

```
B = aM(np.ones((A.shape[0], 1)),A)
Ab = aM(A,b)
Bb = aM(B,b)
```

3. Cálculo de los Determinantes de A y B

```
dA = det_M(A)
print("det(A): ",dA)
dB = det_M(B)
print("det(B): ",dB)
```

```
det(A):  -0.14396980921237948
det(B):  None
```

4. Cálculo del Rango de las Matrices A, B, Ab y Bb

```
rA = rng_M(A)
print("rng(A): ",rA)
rAb = rng_M(Ab)
print("rng([A|b]): ",rAb)


rB = rng_M(B)
print("rng(B): ",rB)
rBb = rng_M(Bb)
print("rng([B|b]): ",rBb)
```

```
rng(A):  6
rng([A|b]):  6
rng(B):  6
rng([B|b]):  6
```

5. Clasificación de los Sistemas

```
cAd = class_M_det(A)
cAr = class_M_rng(A,Ab)

print("clasificacion por determinante para A: ", cAd)
print("clasificacion por rango para A: ", cAr)


cBd = class_M_det(B)
cBr = class_M_rng(B,Bb)

print("clasificacion por determinante para B: ", cBd)
print("clasificacion por rango para B: ", cBr)
```

```
clasificacion por determinante para A:  SCD
clasificacion por rango para A:  SCD or SCI
clasificacion por determinante para B:  SCI o SI
clasificacion por rango para B:  SCD or SCI
```

6. Resolución de los Sistemas

```
xA_base = x_inv(A,b)
bA_base = A.dot(xA_base)

xB_base = x_pinv(B,b)
bB_base = B.dot(xB_base)
```

7. Análisis de Estabilidad (dilataciones y contracciones)

```
xA_dil = x_perturbadas(xA_base, dilataciones)
xA_cont = x_perturbadas(xA_base, contracciones)
```

```
bA_dil = b_perturbadas(A,xA_dil)
bA_cont = b_perturbadas(A,xA_cont)

xB_dil = x_perturbadas(xB_base, dilataciones)
xB_cont = x_perturbadas(xB_base, contracciones)

bB_dil = b_perturbadas(B,xB_dil)
bB_cont = b_perturbadas(B,xB_cont)
```

Matriz A

```
print("Dilataciones")
print("================================================")
print("valores en x")
print("================================================")
for i in range(len(dilataciones)):
    error_relativo = np.linalg.norm(xA_base - xA_dil[i]) /
np.linalg.norm(xA_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de x_base con dilatación númer
{i} : {error_porcentual:.2f}%")
print("================================================")
print("valores en b")
print("================================================")
for i in range(len(dilataciones)):
    error_relativo = np.linalg.norm(bA_base - bA_dil[i]) /
np.linalg.norm(bA_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de b_base con dilatación número
{i}: {error_porcentual:.2f}%")
print("================================================")
print("Contracciones")
print("================================================")
print("valores en x")
print("================================================")
for i in range(len(contracciones)):
    error_relativo = np.linalg.norm(xA_base - xA_cont[i]) /
np.linalg.norm(xA_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de x_base con contraccion númer
{i} : {error_porcentual:.2f}%")
print("================================================")
print("valores en b")
print("================================================")
for i in range(len(contracciones)):
    error_relativo = np.linalg.norm(bA_base - bA_cont[i]) /
np.linalg.norm(bA_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de b_base con contraccion número
{i}: {error_porcentual:.2f}%")
```

```
Dilataciones
================================================
valores en x
================================================
Error relativo porcentual de x_base con dilatación númer 0 : 0.50%
Error relativo porcentual de x_base con dilatación númer 1 : 1.00%
Error relativo porcentual de x_base con dilatación númer 2 : 2.00%
Error relativo porcentual de x_base con dilatación númer 3 : 5.00%
Error relativo porcentual de x_base con dilatación númer 4 : 10.00%
================================================
valores en b
================================================
Error relativo porcentual de b_base con dilatación número 0: 0.50%
Error relativo porcentual de b_base con dilatación número 1: 1.00%
Error relativo porcentual de b_base con dilatación número 2: 2.00%
Error relativo porcentual de b_base con dilatación número 3: 5.00%
Error relativo porcentual de b_base con dilatación número 4: 10.00%
================================================
Contracciones
================================================
valores en x
================================================
Error relativo porcentual de x_base con contraccion númer 0 : 0.50%
Error relativo porcentual de x_base con contraccion númer 1 : 1.00%
Error relativo porcentual de x_base con contraccion númer 2 : 2.00%
Error relativo porcentual de x_base con contraccion númer 3 : 5.00%
Error relativo porcentual de x_base con contraccion númer 4 : 10.00%
================================================
valores en b
================================================
Error relativo porcentual de b_base con contraccion número 0: 0.50%
Error relativo porcentual de b_base con contraccion número 1: 1.00%
Error relativo porcentual de b_base con contraccion número 2: 2.00%
Error relativo porcentual de b_base con contraccion número 3: 5.00%
Error relativo porcentual de b_base con contraccion número 4: 10.00%
```

Matriz B

```
print("Dilataciones")
print("================================================")
print("valores en x")
print("================================================")
for i in range(len(dilataciones)):
    error_relativo = np.linalg.norm(xB_base - xB_dil[i]) /
np.linalg.norm(xB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de x_base con dilatación númer
{i} : {error_porcentual:.2f}%")
print("================================================")
print("valores en b")
```

```
print("================================================")
for i in range(len(dilataciones)):
    error_relativo = np.linalg.norm(bB_base - bB_dil[i]) /
np.linalg.norm(bB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de b_base con dilatación número
{i}: {error_porcentual:.2f}%")
print("================================================")
print("Contracciones")
print("================================================")
print("valores en x")
print("================================================")
for i in range(len(contracciones)):
    error_relativo = np.linalg.norm(xB_base - xB_cont[i]) /
np.linalg.norm(xB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de x_base con contraccion númer
{i} : {error_porcentual:.2f}%")
print("================================================")
print("valores en b")
print("================================================")
for i in range(len(contracciones)):
    error_relativo = np.linalg.norm(bB_base - bB_cont[i]) /
np.linalg.norm(bB_base)
    error_porcentual = error_relativo * 100
    print(f"Error relativo porcentual de b_base con contraccion número
{i}: {error_porcentual:.2f}%")
```

```
Dilataciones
================================================
valores en x
================================================
Error relativo porcentual de x_base con dilatación númer 0 : 0.50%
Error relativo porcentual de x_base con dilatación númer 1 : 1.00%
Error relativo porcentual de x_base con dilatación númer 2 : 2.00%
Error relativo porcentual de x_base con dilatación númer 3 : 5.00%
Error relativo porcentual de x_base con dilatación númer 4 : 10.00%
================================================
valores en b
================================================
Error relativo porcentual de b_base con dilatación número 0: 0.50%
Error relativo porcentual de b_base con dilatación número 1: 1.00%
Error relativo porcentual de b_base con dilatación número 2: 2.00%
Error relativo porcentual de b_base con dilatación número 3: 5.00%
Error relativo porcentual de b_base con dilatación número 4: 10.00%
================================================
Contracciones
================================================
valores en x
================================================
```

```
Error relativo porcentual de x_base con contraccion númer 0 : 0.50%
Error relativo porcentual de x_base con contraccion númer 1 : 1.00%
Error relativo porcentual de x_base con contraccion númer 2 : 2.00%
Error relativo porcentual de x_base con contraccion númer 3 : 5.00%
Error relativo porcentual de x_base con contraccion númer 4 : 10.00%
==========================================
valores en b
==========================================
Error relativo porcentual de b_base con contraccion número 0: 0.50%
Error relativo porcentual de b_base con contraccion número 1: 1.00%
Error relativo porcentual de b_base con contraccion número 2: 2.00%
Error relativo porcentual de b_base con contraccion número 3: 5.00%
Error relativo porcentual de b_base con contraccion número 4: 10.00%
```

8. Intervalos de Confianza

```
# Matriz A

error_max_dilatacion = 0.10
error_max_contraccion = 0.10

# Matriz B

error_max_dilatacion = 0.10
error_max_contraccion = 0.10
```

9. Interpolantes

```
interpolante_A = lambda r : sum([r[i]*xA_base[i] for i in
range(len(r))])
interpolante_B = lambda r : sum([r[i]*xB_base[i] for i in
range(len(r))])
```

10 Comparación de los Interpolantes

```
RMSE_A =  (sum([(interpolante_A(A[i,:]) - b[i])**2 for i in
range(len(b))])/len(b))**0.5
print("RMSE para el interpolante SIN termino independiente: ", RMSE_A)

RMSE_B =  (sum([(interpolante_B(B[i,:]) - b[i])**2 for i in
range(len(b))])/len(b))**0.5
print("RMSE para el interpolante CON termino independiente: ", RMSE_B)

RMSE para el interpolante SIN termino independiente:  [5.77778381e-17]
RMSE para el interpolante CON termino independiente:  [1.54354651e-16]
```

Tabla 6

```
# tabla (8x7)
tabla = np.array([[0.74898, 0.28122, 0.52920, 0.56900, 0.91599,
0.26114, 0.85767],
                  [0.87788, 0.57300, 0.21391, 0.18360, 0.27809,
0.47191, 0.52021],
                  [0.87446, 0.54117, 0.65085, 0.02731, 0.92548,
0.95857, 0.84126],
                  [0.44242, 0.31208, 0.45062, 0.48809, 0.24730,
0.08539, 0.49884],
                  [0.36721, 0.45338, 0.60793, 0.01837, 0.66224,
0.12615, 0.33194],
                  [0.54924, 0.69931, 0.05451, 0.81977, 0.56267,
0.36233, 0.99719],
                  [0.77000, 0.20419, 0.72319, 0.47703, 0.62884,
0.70215, 0.00228],
                  [0.59750, 0.39309, 0.90533, 0.88076, 0.34626,
0.56525, 0.59347]])
```

1. Descomponer las tablas en A y b

```
A = M(tabla)
b = v(tabla)
```

2. Generar la Matrices ampliadas B, Ab y Bb

```
B = aM(np.ones((A.shape[0], 1)),A)
Ab = aM(A,b)
Bb = aM(B,b)
```

3. Cálculo de los Determinantes de A y B

```
dA = det_M(A)
print("det(A): ",dA)
dB = det_M(B)
print("det(B): ",dB)
```

```
det(A):  None
det(B):  None
```

4. Cálculo del Rango de las Matrices A, B, Ab y Bb

```
rA = rng_M(A)
print("rng(A): ",rA)
rAb = rng_M(Ab)
print("rng([A|b]): ",rAb)


rB = rng_M(B)
```

```
print("rng(B): ",rB)
rBb = rng_M(Bb)
print("rng([B|b]): ",rBb)
```

```
rng(A):  6
rng([A|b]):  7
rng(B):  7
rng([B|b]):  8
```

5. Clasificación de los Sistemas

```
cAd = class_M_det(A)
cAr = class_M_rng(A,Ab)

print("clasificacion por determinante para A: ", cAd)
print("clasificacion por rango para A: ", cAr)


cBd = class_M_det(B)
cBr = class_M_rng(B,Bb)

print("clasificacion por determinante para B: ", cBd)
print("clasificacion por rango para B: ", cBr)
```

```
clasificacion por determinante para A:  SCI o SI
clasificacion por rango para A:  SI
clasificacion por determinante para B:  SCI o SI
clasificacion por rango para B:  SI
```