```python
from IPython.core.interactiveshell import InteractiveShell
import pandas as pd

# Mostrar solo la última salida en cada celda para evitar muchas líneas de output
InteractiveShell.ast_node_interactivity = "last_expr"

# Configurar Pandas para evitar DataFrames gigantes
pd.set_option('display.width', 80)  # Limitar ancho de salida de texto

print("Configuración aplicada: Salidas ajustadas para exportar a PDF sin sobresalir.")
```

```
Configuración aplicada: Salidas ajustadas para exportar a PDF sin sobresalir.
```

ESCUELA POLITÉCNICA NACIONAL	Tarea No.
Metodos Numericos – Computación NOMBRE: Ivonne Carolina Ayala	**11**

[Tarea 11] Ejercicios Unidad 04-D | Gauss-Jacobi y Gauss-Seidel

Resuelva los ejercicios adjuntos.

Encuentre las primeras dos iteraciones del método de Jacobi para los siguientes sistemas lineales, por medio de $\mathbf{x}^{(0)} = 0$:

```python
import numpy as np

def jacobi_method_tolerance(A, b, x0, iteraciones, tolerancia):
    D = np.diag(np.diag(A))
    R = A - D
    x = x0

    for i in range(iteraciones):
        x_new = np.dot(np.linalg.inv(D), b - np.dot(R, x))
        error = np.linalg.norm(x_new - x, ord=np.inf)
        print(f"Iteración {i+1}: x = {x_new}, Error = {error}")
```

```
        if error < tolerancia:
            print(f"Convergencia alcanzada en la iteración {i+1} con error {error:.4e}.\n")
            print(f"Solución final: x = {x_new}\n")
            return x_new
        x = x_new

    print(f"No se alcanzó la tolerancia después de {iteraciones} iteraciones.")
    print(f"Solución aproximada: x = {x}")

tolerancia = 1e-6
```

a.

$$
\begin{aligned}
3x_1 - x_2 + x_3 &= 1, \\
3x_1 + 6x_2 + 2x_3 &= 0, \\
3x_1 + 3x_2 + 7x_3 &= 4.
\end{aligned}
$$

```
A = np.array([
    [3, -1, 1],
    [3, 6, 2],
    [3, 3, 7]
])
b = np.array([1, 0, 4])
x0 = np.zeros(3)
jacobi_method_tolerance(A, b, x0, 2, tolerancia)
```

```
Iteración 1: x = [0.33333333 0.         0.57142857], Error = 0.5714285714285714
Iteración 2: x = [ 0.14285714 -0.35714286  0.42857143], Error = 0.3571428571428571
No se alcanzó la tolerancia después de 2 iteraciones.
Solución aproximada: x = [ 0.14285714 -0.35714286  0.42857143]
```

b.

$$
\begin{aligned}
10x_1 - x_2 &= 9, \\
-x_1 + 10x_2 - 2x_3 &= 7, \\
-2x_2 + 10x_3 &= 6.
\end{aligned}
$$

```
A = np.array([
    [10, -1, 0],
    [-1, 10, -2],
    [0, -2, 10]
])
```

```python
b = np.array([9, 7, 6])
x0 = np.zeros(3)
jacobi_method_tolerance(A, b, x0, 2, tolerancia)
```

```
Iteración 1: x = [0.9 0.7 0.6], Error = 0.9
Iteración 2: x = [0.97 0.91 0.74], Error = 0.20999999999999996
No se alcanzó la tolerancia después de 2 iteraciones.
Solución aproximada: x = [0.97 0.91 0.74]
```

c.

$$
\begin{aligned}
10x_1 + 5x_2 &= 6, \\
5x_1 + 10x_2 - 4x_3 &= 25, \\
-4x_2 + 8x_3 - x_4 &= -11, \\
-x_3 + 5x_4 &= -11.
\end{aligned}
$$

```python
A = np.array([
    [10, 5, 0, 0],
    [5, 10, -4, 0],
    [0, -4, 8, -1],
    [0, 0, -1, 5]
])
b = np.array([6, 25, -11, -11])
x0 = np.zeros(4)
jacobi_method_tolerance(A, b, x0, 2, tolerancia)
```

```
Iteración 1: x = [ 0.6    2.5   -1.375 -2.2  ], Error = 2.5
Iteración 2: x = [-0.65   1.65  -0.4   -2.475], Error = 1.25
No se alcanzó la tolerancia después de 2 iteraciones.
Solución aproximada: x = [-0.65   1.65  -0.4   -2.475]
```

d.

$$
\begin{aligned}
4x_1 + x_2 + x_3 + x_5 &= 6, \\
-x_1 - 3x_2 + x_3 + x_4 &= 6, \\
2x_1 + x_2 + 5x_3 - x_4 - x_5 &= 6, \\
-x_1 - x_2 - x_3 + 4x_4 &= 6, \\
2x_2 - x_3 + x_4 + 4x_5 &= 6.
\end{aligned}
$$

```
A = np.array([
    [4, 1, 1, 1, 1],
    [-1, -3, 1, 1, 0],
    [2, 1, 5, -1, -1],
    [-1, -1, 3, 4, 0],
    [2, 2, 1, 0, 4]
])
b = np.array([6, 6, 6, 6, 6])
x0 = np.zeros(5)
jacobi_method_tolerance(A, b, x0, 2, tolerancia)
```

```
Iteración 1: x = [ 1.5 -2.   1.2  1.5  1.5], Error = 2.0
Iteración 2: x = [ 0.95  -1.6    1.6    0.475  1.45 ], Error = 1.0250000000000001
No se alcanzó la tolerancia después de 2 iteraciones.
Solución aproximada: x = [ 0.95  -1.6    1.6    0.475  1.45 ]
```

2. Repita el ejercicio 1 usando el método de Gauss-Siedel.

```
def gauss_seidel_method(A, b, x0, iteraciones, tolerancia):
    n = len(b)
    x = x0.copy()

    for k in range(iteraciones):
        x_new = x.copy()
        for i in range(n):
            suma = sum(A[i, j] * x_new[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - suma) / A[i, i]

        error = np.linalg.norm(x_new - x, ord=np.inf)
        print(f"Iteración {k+1}: x = {x_new}, Error = {error}")

        if error < tolerancia:
            print(f"Convergencia alcanzada en la iteración {k+1} con error {error:.4e}.\n")
            print(f"Solución final: x = {x_new}\n")
            return x_new
        x = x_new

    print(f"No se alcanzó la tolerancia después de {iteraciones} iteraciones.")
    print(f"Solución aproximada: x = {x}")
```

```
tolerancia = 1e-6
```

a.

$$
\begin{aligned}
3x_1 - x_2 + x_3 &= 1,\\
3x_1 + 6x_2 + 2x_3 &= 0,\\
3x_1 + 3x_2 + 7x_3 &= 4.
\end{aligned}
$$

```
A = np.array([
    [3, -1, 1],
    [3, 6, 2],
    [3, 3, 7]
])
b = np.array([1, 0, 4])
x0 = np.zeros(3)
gauss_seidel_method(A, b, x0, 2, tolerancia)
```

```
Iteración 1: x = [ 0.33333333 -0.16666667  0.5       ], Error = 0.5
Iteración 2: x = [ 0.11111111 -0.22222222  0.61904762], Error = 0.2222222222222222
No se alcanzó la tolerancia después de 2 iteraciones.
Solución aproximada: x = [ 0.11111111 -0.22222222  0.61904762]
```

b.

$$
\begin{aligned}
10x_1 - x_2 &= 9,\\
-x_1 + 10x_2 - 2x_3 &= 7,\\
-2x_2 + 10x_3 &= 6.
\end{aligned}
$$

```
A = np.array([
    [10, -1, 0],
    [-1, 10, -2],
    [0, -2, 10]
])
b = np.array([9, 7, 6])
x0 = np.zeros(3)
gauss_seidel_method(A, b, x0, 2, tolerancia)
```

```
Iteración 1: x = [0.9   0.79  0.758], Error = 0.9
Iteración 2: x = [0.979  0.9495 0.7899], Error = 0.1595000000000001
No se alcanzó la tolerancia después de 2 iteraciones.
Solución aproximada: x = [0.979  0.9495 0.7899]
```

c.

$$\begin{aligned} 10x_1 + 5x_2 &= 6, \\ 5x_1 + 10x_2 - 4x_3 &= 25, \\ -4x_2 + 8x_3 - x_4 &= -11, \\ -x_3 + 5x_4 &= -11. \end{aligned}$$

```
A = np.array([
    [10, 5, 0, 0],
    [5, 10, -4, 0],
    [0, -4, 8, -1],
    [0, 0, -1, 5]
])
b = np.array([6, 25, -11, -11])
x0 = np.zeros(4)
gauss_seidel_method(A, b, x0, 2, tolerancia)
```

```
Iteración 1: x = [ 0.6    2.2   -0.275 -2.255], Error = 2.255
Iteración 2: x = [-0.5       2.64     -0.336875 -2.267375], Error = 1.1
No se alcanzó la tolerancia después de 2 iteraciones.
Solución aproximada: x = [-0.5       2.64     -0.336875 -2.267375]
```

d.

$$\begin{aligned} 4x_1 + x_2 + x_3 + x_5 &= 6, \\ -x_1 - 3x_2 + x_3 + x_4 &= 6, \\ 2x_1 + x_2 + 5x_3 - x_4 - x_5 &= 6, \\ -x_1 - x_2 - x_3 + 4x_4 &= 6, \\ 2x_2 - x_3 + x_4 + 4x_5 &= 6. \end{aligned}$$

```
A = np.array([
    [4, 1, 1, 1, 1],
    [-1, -3, 1, 1, 0],
    [2, 1, 5, -1, -1],
    [-1, -1, 3, 4, 0],
    [2, 2, 1, 0, 4]
])
b = np.array([6, 6, 6, 6, 6])
x0 = np.zeros(5)
gauss_seidel_method(A, b, x0, 2, tolerancia)
```

```
Iteración 1: x = [ 1.5    -2.5     1.1     0.425   1.725], Error = 2.5
Iteración 2: x = [ 1.3125     -1.92916667  1.49083333  0.22770833  1.435625  ], Error = 0.570
No se alcanzó la tolerancia después de 2 iteraciones.
Solución aproximada: x = [ 1.3125     -1.92916667  1.49083333  0.22770833  1.435625  ]
```

3. Utilice el método de Jacobi para resolver los sistemas lineales en el ejercicio 1, con TOL = 10-3.

```
tolerancia = 1e-3
```

a.

$$
\begin{aligned}
3x_1 - x_2 + x_3 &= 1,\\
3x_1 + 6x_2 + 2x_3 &= 0,\\
3x_1 + 3x_2 + 7x_3 &= 4.
\end{aligned}
$$

```
A = np.array([
    [3, -1, 1],
    [3, 6, 2],
    [3, 3, 7]
])
b = np.array([1, 0, 4])
x0 = np.zeros(3)
_ =jacobi_method_tolerance(A, b, x0, 50, tolerancia)
```

```
Iteración 1: x = [0.33333333 0.         0.57142857], Error = 0.5714285714285714
Iteración 2: x = [ 0.14285714 -0.35714286  0.42857143], Error = 0.3571428571428571
Iteración 3: x = [ 0.07142857 -0.21428571  0.66326531], Error = 0.23469387755102028
Iteración 4: x = [ 0.04081633 -0.25680272  0.63265306], Error = 0.04251700680272108
Iteración 5: x = [ 0.03684807 -0.23129252  0.66399417], Error = 0.031341107871720064
Iteración 6: x = [ 0.03490444 -0.23975543  0.6547619 ], Error = 0.00923226433430513
Iteración 7: x = [ 0.03516089 -0.23570619  0.65922185], Error = 0.0044599472442037325
Iteración 8: x = [ 0.03502399 -0.23732106  0.65737656], Error = 0.0018452959415058423
Iteración 9: x = [ 0.03510079 -0.23663751  0.65812732], Error = 0.0007507619179839553
Convergencia alcanzada en la iteración 9 con error 7.5076e-04.

Solución final: x = [ 0.03510079 -0.23663751  0.65812732]
```

b.

$$
\begin{aligned}
10x_1 - x_2 &= 9,\\
-x_1 + 10x_2 - 2x_3 &= 7,\\
-2x_2 + 10x_3 &= 6.
\end{aligned}
$$

```python
A = np.array([
    [10, -1, 0],
    [-1, 10, -2],
    [0, -2, 10]
])
b = np.array([9, 7, 6])
x0 = np.zeros(3)
_ =jacobi_method_tolerance(A, b, x0, 50, tolerancia)
```

```
Iteración 1: x = [0.9 0.7 0.6], Error = 0.9
Iteración 2: x = [0.97 0.91 0.74], Error = 0.20999999999999996
Iteración 3: x = [0.991 0.945 0.782], Error = 0.041999999999999926
Iteración 4: x = [0.9945 0.9555 0.789 ], Error = 0.010500000000000065
Iteración 5: x = [0.99555 0.95725 0.7911 ], Error = 0.0020999999999999908
Iteración 6: x = [0.995725 0.957775 0.79145 ], Error = 0.0005249999999999977
Convergencia alcanzada en la iteración 6 con error 5.2500e-04.

Solución final: x = [0.995725 0.957775 0.79145 ]
```

c.

$$
\begin{aligned}
10x_1 + 5x_2 &= 6, \\
5x_1 + 10x_2 - 4x_3 &= 25, \\
-4x_2 + 8x_3 - x_4 &= -11, \\
-x_3 + 5x_4 &= -11.
\end{aligned}
$$

```python
A = np.array([
    [10, 5, 0, 0],
    [5, 10, -4, 0],
    [0, -4, 8, -1],
    [0, 0, -1, 5]
])
b = np.array([6, 25, -11, -11])
x0 = np.zeros(4)
_ =jacobi_method_tolerance(A, b, x0, 50, tolerancia)
```

```
Iteración 1: x = [ 0.6    2.5   -1.375 -2.2  ], Error = 2.5
Iteración 2: x = [-0.65   1.65  -0.4   -2.475], Error = 1.25
Iteración 3: x = [-0.225     2.665    -0.859375 -2.28    ], Error = 1.015
Iteración 4: x = [-0.7325    2.26875  -0.3275   -2.371875], Error = 0.5318749999999999
Iteración 5: x = [-0.534375    2.73525    -0.53710937 -2.2655    ], Error = 0.46649999999999
```

```
Iteración 6: x = [-0.767625    2.55234375 -0.2905625  -2.30742188], Error = 0.246546874999999
Iteración 7: x = [-0.67617188  2.7675875  -0.38725586 -2.2581125 ], Error = 0.215243749999999
Iteración 8: x = [-0.78379375  2.68318359 -0.27347031 -2.27745117], Error = 0.113785546875000
Iteración 9: x = [-0.7415918   2.78250875 -0.3180896  -2.25469406], Error = 0.099325156249999
Iteración 10: x = [-0.79125438  2.74356006 -0.26558238 -2.26361792], Error = 0.052507216796868
Iteración 11: x = [-0.77178003  2.78939423 -0.28617221 -2.25311648], Error = 0.045834175781125
Iteración 12: x = [-0.79469712  2.77142113 -0.26194244 -2.25723444], Error = 0.024229768310544
Iteración 13: x = [-0.78571057  2.79257158 -0.27144374 -2.25238849], Error = 0.021150451269533
Iteración 14: x = [-0.79628579  2.78427779 -0.26026277 -2.25428875], Error = 0.011180969842522
Iteración 15: x = [-0.79213889  2.79403779 -0.2646472  -2.25205255], Error = 0.009760000754399
Iteración 16: x = [-0.79701889  2.79021057 -0.25948768 -2.25292944], Error = 0.005159524623266
Iteración 17: x = [-0.79510528  2.79471438 -0.2615109  -2.25189754], Error = 0.004503810037900
Iteración 18: x = [-0.79735719  2.79294828 -0.25913    -2.25230218], Error = 0.002380893134533
Iteración 19: x = [-0.79647414  2.79502659 -0.26006363 -2.251826  ], Error = 0.002078309763288
Iteración 20: x = [-0.7975133   2.79421162 -0.25896495 -2.25201273], Error = 0.001098677210000
Iteración 21: x = [-0.79710581  2.79517067 -0.25939578 -2.25179299], Error = 0.000959048324822
Convergencia alcanzada en la iteración 21 con error 9.5905e-04.

Solución final: x = [-0.79710581  2.79517067 -0.25939578 -2.25179299]
```

d.

$$
\begin{aligned}
4x_1 + x_2 + x_3 + x_5 &= 6, \\
-x_1 - 3x_2 + x_3 + x_4 &= 6, \\
2x_1 + x_2 + 5x_3 - x_4 - x_5 &= 6, \\
-x_1 - x_2 - x_3 + 4x_4 &= 6, \\
2x_2 - x_3 + x_4 + 4x_5 &= 6.
\end{aligned}
$$

```python
A = np.array([
    [4, 1, 1, 1, 1],
    [-1, -3, 1, 1, 0],
    [2, 1, 5, -1, -1],
    [-1, -1, 3, 4, 0],
    [2, 2, 1, 0, 4]
])
b = np.array([6, 6, 6, 6, 6])
x0 = np.zeros(5)
_ =jacobi_method_tolerance(A, b, x0, 50, tolerancia)
```

```
Iteración 1: x = [ 1.5 -2.   1.2  1.5  1.5], Error = 2.0
Iteración 2: x = [ 0.95  -1.6    1.6    0.475  1.45 ], Error = 1.0250000000000001
Iteración 3: x = [ 1.01875 -1.625    1.525    0.1375   1.425  ], Error = 0.3374999999999999
```

```
Iteración 4: x = [ 1.134375   -1.78541667  1.43        0.2046875   1.421875  ], Error = 0.16
Iteración 5: x = [ 1.18221354 -1.83322917  1.42864583  0.26473958  1.46802083], Error = 0.06
Iteración 6: x = [ 1.16795573 -1.82960937  1.4403125   0.26576172  1.46834635], Error = 0.01
Iteración 7: x = [ 1.1637972  -1.82062717  1.4455612   0.25435221  1.4707487 ], Error = 0.01
Iteración 8: x = [ 1.16249127 -1.8212946   1.44362674  0.25162161  1.46702469], Error = 0.00
Iteración 9: x = [ 1.16475539 -1.82241431  1.44299167  0.25257912  1.46849498], Error = 0.00
Iteración 10: x = [ 1.16458713 -1.82306153  1.44279552  0.25334152  1.46808154], Error = 0.0
Convergencia alcanzada en la iteración 10 con error 7.6240e-04.

Solución final: x = [ 1.16458713 -1.82306153  1.44279552  0.25334152  1.46808154]
```

4. Utilice el método de Gauss-Siedel para resolver los sistemas lineales en el ejercicio 1, con TOL = 10-3.

a.

$$\begin{aligned} 3x_1 - x_2 + x_3 &= 1, \\ 3x_1 + 6x_2 + 2x_3 &= 0, \\ 3x_1 + 3x_2 + 7x_3 &= 4. \end{aligned}$$

```python
A = np.array([
    [3, -1, 1],
    [3, 6, 2],
    [3, 3, 7]
])
b = np.array([1, 0, 4])
x0 = np.zeros(3)
_ =gauss_seidel_method(A, b, x0, 50, tolerancia)
```

```
Iteración 1: x = [ 0.33333333 -0.16666667  0.5       ], Error = 0.5
Iteración 2: x = [ 0.11111111 -0.22222222  0.61904762], Error = 0.2222222222222222
Iteración 3: x = [ 0.05291005 -0.23280423  0.64852608], Error = 0.05820105820105818
Iteración 4: x = [ 0.03955656 -0.23595364  0.65559875], Error = 0.013353489543965757
Iteración 5: x = [ 0.0361492  -0.23660752  0.65733928], Error = 0.003407359416429702
Iteración 6: x = [ 0.03535107 -0.23678863  0.65775895], Error = 0.0007981356647807844
Convergencia alcanzada en la iteración 6 con error 7.9814e-04.

Solución final: x = [ 0.03535107 -0.23678863  0.65775895]
```

b.

$$\begin{aligned} 10x_1 - x_2 &= 9, \\ -x_1 + 10x_2 - 2x_3 &= 7, \\ -2x_2 + 10x_3 &= 6. \end{aligned}$$

```
A = np.array([
    [10, -1, 0],
    [-1, 10, -2],
    [0, -2, 10]
])
b = np.array([9, 7, 6])
x0 = np.zeros(3)
_ =gauss_seidel_method(A, b, x0, 50, tolerancia)
```

```
Iteración 1: x = [0.9    0.79   0.758], Error = 0.9
Iteración 2: x = [0.979  0.9495 0.7899], Error = 0.1595000000000001
Iteración 3: x = [0.99495  0.957475 0.791495], Error = 0.01595000000000013
Iteración 4: x = [0.9957475  0.95787375 0.79157475], Error = 0.0007975000000000065
Convergencia alcanzada en la iteración 4 con error 7.9750e-04.

Solución final: x = [0.9957475  0.95787375 0.79157475]
```

c.

$$
\begin{aligned}
10x_1 + 5x_2 &= 6, \\
5x_1 + 10x_2 - 4x_3 &= 25, \\
-4x_2 + 8x_3 - x_4 &= -11, \\
-x_3 + 5x_4 &= -11.
\end{aligned}
$$

```
A = np.array([
    [10, 5, 0, 0],
    [5, 10, -4, 0],
    [0, -4, 8, -1],
    [0, 0, -1, 5]
])
b = np.array([6, 25, -11, -11])
x0 = np.zeros(4)
_ =gauss_seidel_method(A, b, x0, 50, tolerancia)
```

```
Iteración 1: x = [ 0.6    2.2    -0.275 -2.255], Error = 2.255
Iteración 2: x = [-0.5       2.64      -0.336875 -2.267375], Error = 1.1
Iteración 3: x = [-0.72        2.72525    -0.29579687 -2.25915938], Error = 0.219999999999999
Iteración 4: x = [-0.762625    2.76299375 -0.27589805 -2.25517961], Error = 0.042624999999999
Iteración 5: x = [-0.78149687  2.78038922 -0.26670284 -2.25334057], Error = 0.018871875000000
Iteración 6: x = [-0.79019461  2.78841617 -0.26245949 -2.2524919 ], Error = 0.008697734374999
Iteración 7: x = [-0.79420808  2.79212025 -0.26050136 -2.25210027], Error = 0.004013474609375
```

```
Iteración 8: x = [-0.79606012  2.79382952 -0.25959778 -2.25191956], Error = 0.00185203959960
Iteración 9: x = [-0.79691476  2.79461827 -0.25918081 -2.25183616], Error = 0.00085463459350
Convergencia alcanzada en la iteración 9 con error 8.5463e-04.

Solución final: x = [-0.79691476  2.79461827 -0.25918081 -2.25183616]
```

d.

$$
\begin{aligned}
4x_1 + x_2 + x_3 + x_5 &= 6,\\
-x_1 - 3x_2 + x_3 + x_4 &= 6,\\
2x_1 + x_2 + 5x_3 - x_4 - x_5 &= 6,\\
-x_1 - x_2 - x_3 + 4x_4 &= 6,\\
2x_2 - x_3 + x_4 + 4x_5 &= 6.
\end{aligned}
$$

```python
A = np.array([
    [4, 1, 1, 1, 1],
    [-1, -3, 1, 1, 0],
    [2, 1, 5, -1, -1],
    [-1, -1, 3, 4, 0],
    [2, 2, 1, 0, 4]
])
b = np.array([6, 6, 6, 6, 6])
x0 = np.zeros(5)
_ =gauss_seidel_method(A, b, x0, 50, tolerancia)
```

```
Iteración 1: x = [ 1.5   -2.5    1.1    0.425  1.725], Error = 2.5
Iteración 2: x = [ 1.3125     -1.92916667  1.49083333  0.22770833  1.435625  ], Error = 0.57
Iteración 3: x = [ 1.19375    -1.82506944  1.42018056  0.27703472  1.46061458], Error = 0.11
Iteración 4: x = [ 1.1668099  -1.82319821  1.44544554  0.25181876  1.46683277], Error = 0.02
Iteración 5: x = [ 1.16477528 -1.82250366  1.44232093  0.25382721  1.46828396], Error = 0.00
Iteración 6: x = [ 1.16451789 -1.82278992  1.44317306  0.2530522   1.46834275], Error = 0.00
Convergencia alcanzada en la iteración 6 con error 8.5214e-04.

Solución final: x = [ 1.16451789 -1.82278992  1.44317306  0.2530522   1.46834275]
```

6. El sistema lineal

$$
\begin{aligned}
x_1 \qquad - x_3 &= 0.2,\\
-\frac{1}{2}x_1 + x_2 - \frac{1}{4}x_3 &= -1.425,\\
x_1 - \frac{1}{2}x_2 + x_3 &= 2,
\end{aligned}
$$

tiene la solución $(0.9, -0.8, 0.7)$:

a) ¿La matriz de coeficientes

$$A = \begin{bmatrix} 1 & 0 & -1 \\ -\frac{1}{2} & 1 & -\frac{1}{4} \\ 1 & -\frac{1}{2} & 1 \end{bmatrix}$$

tiene diagonal estrictamente dominante?

```python
A = np.array([
    [1, 0, -1],
    [-0.5, 1, -0.25],
    [1, -0.5, 1]
])

def verificar_diagonal_dominante(A):
    n = A.shape[0]
    for i in range(n):
        diagonal = abs(A[i, i])
        suma_fila = sum(abs(A[i, j]) for j in range(n) if j != i)
        if diagonal <= suma_fila:
            return False
    return True

es_dominante = verificar_diagonal_dominante(A)
print(f"La matriz A tiene diagonal estrictamente dominante: {es_dominante}")
```

```
La matriz A tiene diagonal estrictamente dominante: False
```

b) Utilice el método iterativo de Gauss-Siedel para aproximar la solución para el sistema lineal con una tolerancia de 10^{-2} y un máximo de 300 iteraciones.

```python
b = np.array([0.2, -1.425, 2])
x0 = np.zeros(len(b))

_ = gauss_seidel_method(A, b, x0, 300, 1e-2)
```

```
Iteración 1: x = [ 0.2    -1.325   1.1375], Error = 1.325
Iteración 2: x = [ 1.3375    -0.471875   0.4265625], Error = 1.1375
Iteración 3: x = [ 0.6265625  -1.00507812  0.87089844], Error = 0.7109374999999998
Iteración 4: x = [ 1.07089844 -0.67182617  0.59318848], Error = 0.4443359374999998
Iteración 5: x = [ 0.79318848 -0.88010864  0.7667572 ], Error = 0.2777099609374998
```

```
Iteración 6: x = [ 0.9667572  -0.7499321   0.65827675], Error = 0.17356872558593728
Iteración 7: x = [ 0.85827675 -0.83129244  0.72607703], Error = 0.10848045349121072
Iteración 8: x = [ 0.92607703 -0.78044223  0.68370185], Error = 0.06780028343200661
Iteración 9: x = [ 0.88370185 -0.81222361  0.71018634], Error = 0.04237517714500405
Iteración 10: x = [ 0.91018634 -0.79236024  0.69363354], Error = 0.026484485715627448
Iteración 11: x = [ 0.89363354 -0.80477485  0.70397904], Error = 0.016552803572267072
Iteración 12: x = [ 0.90397904 -0.79701572  0.6975131 ], Error = 0.010345502232666837
Iteración 13: x = [ 0.8975131  -0.80186517  0.70155431], Error = 0.006465938895541669
Convergencia alcanzada en la iteración 13 con error 6.4659e-03.

Solución final: x = [ 0.8975131  -0.80186517  0.70155431]
```

c) ¿Qué pasa en la parte b) cuando el sistema cambia por el siguiente?

```python
A = np.array([
    [1, 0, -2],
    [-0.5, 1, -0.25],
    [1, -0.5, 1]
])
b = np.array([0.2, -1.425, 2])

x0 = np.zeros(len(b))

_ = gauss_seidel_method(A, b, x0, 20, 1e-22)
```

```
Iteración 1: x = [ 0.2    -1.325   1.1375], Error = 1.325
Iteración 2: x = [ 2.475      0.096875  -0.4265625], Error = 2.275
Iteración 3: x = [-0.653125   -1.85820313  1.72402344], Error = 3.1281250000000007
Iteración 4: x = [ 3.64804688  0.8300293  -1.23303223], Error = 4.301171875000001
Iteración 5: x = [-2.26606445 -2.86629028  2.83291931], Error = 5.914111328125001
Iteración 6: x = [ 5.86583862  2.21614914 -2.75776405], Error = 8.131903076171877
Iteración 7: x = [-5.31552811 -4.77220507  4.92942557], Error = 11.181366729736332
Iteración 8: x = [10.05885115  4.83678197 -5.64046016], Error = 15.374379253387456
Iteración 9: x = [-11.08092033  -8.3755752    8.89313272], Error = 21.13977147340775
Iteración 10: x = [ 17.98626545   9.79141591 -11.0905575 ], Error = 29.06718577593566
Iteración 11: x = [-21.98111499 -15.18819687  16.38701656], Error = 39.96738044191153
Iteración 12: x = [ 32.97403311  19.1587707   -21.39464777], Error = 54.95514810762836
Iteración 13: x = [-42.58929553 -28.06830971  30.55514068], Error = 75.56332864798898
Iteración 14: x = [ 61.31028136  36.86892585 -40.87581843], Error = 103.89957689098486
Iteración 15: x = [-81.55163687 -52.41977304  57.34175035], Error = 142.8619182251042
Iteración 16: x = [114.88350069  70.35218793 -77.70740673], Error = 196.43513755951827
Iteración 17: x = [-155.21481345  -98.45925841  107.98518425], Error = 270.0983141443376
```

```
Iteración 18: x = [ 216.1703685   133.65648031 -147.34212834], Error = 371.38518194846426
Iteración 19: x = [-294.48425668 -185.50266043  203.73292647], Error = 510.6546251791383
Iteración 20: x = [ 407.66585294  253.34115809 -278.9952739 ], Error = 702.1501096213151
No se alcanzó la tolerancia después de 20 iteraciones.
Solución aproximada: x = [ 407.66585294  253.34115809 -278.9952739 ]
```

A pesar de poner mas iteraciones, no converge.

7. Repita el ejercicio 11 usando el método de Jacobi

b) Utilice el método iterativo de Gauss-Jacobi para aproximar la solución para el sistema lineal con una tolerancia de 10^{-2} y un máximo de 300 iteraciones.

```
b = np.array([0.2, -1.425, 2])
x0 = np.zeros(len(b))

_ = jacobi_method_tolerance(A, b, x0, 25, 1e-2)
```

```
Iteración 1: x = [ 0.2   -1.425  2.   ], Error = 2.0
Iteración 2: x = [ 4.2    -0.825   1.0875], Error = 4.0
Iteración 3: x = [ 2.375     0.946875 -2.6125  ], Error = 3.6999999999999997
Iteración 4: x = [-5.025     -0.890625   0.0984375], Error = 7.3999999999999995
Iteración 5: x = [ 0.396875   -3.91289062  6.5796875 ], Error = 6.481249999999999
Iteración 6: x = [13.359375    0.41835938 -0.35332031], Error = 12.962499999999999
Iteración 7: x = [ -0.50664063   5.16635742 -11.15019531], Error = 13.866015625
Iteración 8: x = [-22.10039062  -4.46586914   5.08981934], Error = 21.593749999999996
Iteración 9: x = [ 10.37963867 -11.20274048  21.86745605], Error = 32.480029296874996
Iteración 10: x = [ 43.93491211   9.23168335 -13.98100891], Error = 35.848464965820305
Iteración 11: x = [-27.76201782  17.04720383 -37.31907043], Error = 71.69692993164061
Iteración 12: x = [-74.43814087 -24.63577652  38.28561974], Error = 75.60469017028808
Iteración 13: x = [ 76.77123947 -29.0726655   64.12025261], Error = 151.20938034057616
Iteración 14: x = [128.44050522  52.99068289 -89.30757222], Error = 153.42782483100888
Iteración 15: x = [-178.41514444   40.46835955  -99.94516377], Error = 306.85564966201775
Iteración 16: x = [-199.69032755 -115.61886317  200.64932422], Error = 300.5944879949092
Iteración 17: x = [401.49864844 -51.10783272 143.88089597], Error = 601.1889759898183
Iteración 18: x = [ 287.96179193  235.29454821 -425.0525648 ], Error = 568.9334607668219
Iteración 19: x = [-849.9051296    36.29275477 -168.31451783], Error = 1137.8669215336438
Iteración 20: x = [-336.42903565 -468.45619426  870.05150698], Error = 1038.3660248107271
Iteración 21: x = [1740.30301397   47.87335892  104.20093852], Error = 2076.7320496214543
Iteración 22: x = [  208.60187705   894.77674162 -1714.36633451], Error = 1818.5672730331419
Iteración 23: x = [-3428.53266902  -325.7156451    240.78649376], Error = 3637.1345460662837
```

```
Iteración 24: x = [  481.77298752 -1655.49471107  3267.67484647], Error = 3910.3056565340603
Iteración 25: x = [ 6535.54969293  1056.38020537 -1307.52034305], Error = 6053.776705414727
No se alcanzó la tolerancia después de 25 iteraciones.
Solución aproximada: x = [ 6535.54969293  1056.38020537 -1307.52034305]
```

A pesar de poner mas iteraciones, no converge.

c) ¿Qué pasa en la parte b) cuando el sistema cambia por el siguiente?

```python
A = np.array([
    [1, 0, -2],
    [-0.5, 1, -0.25],
    [1, -0.5, 1]
])
b = np.array([0.2, -1.425, 2])

x0 = np.zeros(len(b))

_ = jacobi_method_tolerance(A, b, x0, 25, 1e-22)
```

```
Iteración 1: x = [ 0.2   -1.425  2.   ], Error = 2.0
Iteración 2: x = [ 4.2    -0.825   1.0875], Error = 4.0
Iteración 3: x = [ 2.375     0.946875 -2.6125  ], Error = 3.6999999999999997
Iteración 4: x = [-5.025     -0.890625   0.0984375], Error = 7.3999999999999995
Iteración 5: x = [ 0.396875   -3.91289062  6.5796875 ], Error = 6.481249999999999
Iteración 6: x = [13.359375    0.41835938 -0.35332031], Error = 12.962499999999999
Iteración 7: x = [ -0.50664063   5.16635742 -11.15019531], Error = 13.866015625
Iteración 8: x = [-22.10039062  -4.46586914   5.08981934], Error = 21.593749999999996
Iteración 9: x = [ 10.37963867 -11.20274048  21.86745605], Error = 32.480029296874996
Iteración 10: x = [ 43.93491211   9.23168335 -13.98100891], Error = 35.848464965820305
Iteración 11: x = [-27.76201782  17.04720383 -37.31907043], Error = 71.69692993164061
Iteración 12: x = [-74.43814087 -24.63577652  38.28561974], Error = 75.60469017028808
Iteración 13: x = [ 76.77123947 -29.0726655   64.12025261], Error = 151.20938034057616
Iteración 14: x = [128.44050522  52.99068289 -89.30757222], Error = 153.42782483100888
Iteración 15: x = [-178.41514444   40.46835955  -99.94516377], Error = 306.85564966201775
Iteración 16: x = [-199.69032755 -115.61886317  200.64932422], Error = 300.5944879949092
Iteración 17: x = [401.49864844 -51.10783272 143.88089597], Error = 601.1889759898183
Iteración 18: x = [ 287.96179193  235.29454821 -425.0525648 ], Error = 568.9334607668219
Iteración 19: x = [-849.9051296    36.29275477 -168.31451783], Error = 1137.8669215336438
Iteración 20: x = [-336.42903565 -468.45619426  870.05150698], Error = 1038.3660248107271
Iteración 21: x = [1740.30301397   47.87335892  104.20093852], Error = 2076.7320496214543
Iteración 22: x = [  208.60187705   894.77674162 -1714.36633451], Error = 1818.5672730331419
```

```
Iteración 23: x = [-3428.53266902  -325.7156451    240.78649376], Error = 3637.1345460662837
Iteración 24: x = [  481.77298752 -1655.49471107  3267.67484647], Error = 3910.3056565340603
Iteración 25: x = [ 6535.54969293  1056.38020537 -1307.52034305], Error = 6053.776705414727
No se alcanzó la tolerancia después de 25 iteraciones.
Solución aproximada: x = [ 6535.54969293  1056.38020537 -1307.52034305]
```

A pesar de poner mas iteraciones, no converge.

8. Un cable coaxial está formado por un conductor interno de 0.1 pulgadas cuadradas y un conductor externo de 0.5 pulgadas cuadradas. El potencial en un punto en la sección transversal del cable se describe mediante la ecuación de Laplace.

Suponga que el conductor interno se mantiene en 0 volts y el conductor externo se mantiene en 110 volts. Aproximar el potencial entre los dos conductores requiere resolver el siguiente sistema lineal.

$$
\begin{bmatrix}
4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 4 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4
\end{bmatrix}
\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \\ w_{10} \\ w_{11} \\ w_{12} \end{bmatrix}
=
\begin{bmatrix} 220 \\ 110 \\ 110 \\ 220 \\ 110 \\ 110 \\ 110 \\ 110 \\ 110 \\ 110 \\ 110 \\ 220 \end{bmatrix}.
$$

a) ¿La matriz es estrictamente diagonalmente dominante?

```python
A = np.array([
    [4, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0],
    [-1, 4, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, -1, 4, -1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, -1, 4, 0, -1, 0, 0, 0, 0, 0, 0],
    [-1, 0, 0, 0, 4, -1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, -1, -1, 4, -1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, -1, 4, -1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, -1, 4, 0, -1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 4, -1, 0, -1],
```

```python
    [0, 0, 0, 0, 0, 0, 0, -1, -1, 4, -1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 4, -1],
    [0, 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, 4]
])

es_dominante = verificar_diagonal_dominante(A)
print(f"La matriz A tiene diagonal estrictamente dominante: {es_dominante}")
```

```
La matriz A tiene diagonal estrictamente dominante: True
```

b) Resuelva el sistema lineal usando el método de Jacobi con $\mathbf{x}^{(0)} = 0$ y $\mathrm{TOL} = 10^{-2}$.

```python
print("Método de Jacobi:")
b = np.array([220, 110, 110, 220, 110, 110, 110, 110, 220, 110, 110, 220])
x0 = np.zeros(len(b))

_ = jacobi_method_tolerance(A, b, x0, 50, 1e-2)
```

```
Método de Jacobi:
Iteración 1: x = [55.  27.5 27.5 55.  27.5 27.5 27.5 27.5 55.  27.5 27.5 55. ], Error = 55.0
Iteración 2: x = [68.75  48.125 48.125 68.75  48.125 55.    41.25  41.25  75.625 55.
 48.125 75.625], Error = 27.5
Iteración 3: x = [79.0625  56.71875 56.71875 80.78125 58.4375  67.03125 51.5625  51.5625
 87.65625 68.75    60.15625 85.9375 ], Error = 13.75
Iteración 4: x = [83.7890625 61.4453125 61.875     85.9375    64.0234375 75.1953125
 57.1484375 57.578125  93.671875  77.34375   66.171875  91.953125 ], Error = 8.59375
Iteración 5: x = [86.3671875  63.91601562 64.34570312 89.26757812 67.24609375 79.27734375
 60.69335938 61.12304688 97.32421875 81.85546875 69.82421875 94.9609375 ], Error = 4.5117187
Iteración 6: x = [87.79052734 65.17822266 65.79589844 90.90576172 68.91113281 81.80175781
 62.60009766 63.13720703 99.20410156 84.56787109 71.70410156 96.78710938], Error = 2.7124023
Iteración 7: x = [ 88.52233887  65.89660645  66.52099609  91.89941406  69.89807129
  83.10424805  63.73474121  64.29199219 100.33874512  86.01135254
  72.83874512  97.72705078], Error = 1.4434814453125
Iteración 8: x = [ 88.94866943  66.26083374  66.94900513  92.40631104  70.40664673
  83.88305664  64.34906006  64.93652344 100.93460083  86.86737061
  73.43460083  98.29437256], Error = 0.85601806640625
Iteración 9: x = [ 89.16687012  66.47441864  67.16678619  92.70801544  70.70793152
  84.29050446  64.70489502  65.30410767 101.29043579  87.32643127
  73.79043579  98.59230042], Error = 0.4590606689453125
Iteración 10: x = [ 89.29558754  66.58341408  67.29560852  92.86432266  70.86434364
  84.53021049  64.89865303  65.50783157 101.47968292  87.59624481
```

```
  73.97968292  98.7702179 ], Error = 0.26981353759765625
Iteración 11: x = [ 89.36193943  66.64779902  67.36193419  92.95645475  70.95644951
  84.65682983  65.00951052  65.62372446 101.59161568  87.74179935
  74.09161568  98.86484146], Error = 0.1455545425415039
Iteración 12: x = [ 89.40106213  66.6809684   67.40106344  93.004691    71.00469232
  84.73060369  65.07013857  65.68782747 101.6516602   87.82673895
  74.1516602   98.92080784], Error = 0.08493959903717041
Iteración 13: x = [ 89.42141518  66.70053139  67.42141485  93.03291678  71.03291646
  84.76988047  65.10460779  65.72421938 101.6868867   87.87278697
  74.1868867   98.9508301 ], Error = 0.04604801535606384
Iteración 14: x = [ 89.43336196  66.71070751  67.43336204  93.04782383  71.04782391
  84.79261026  65.12352496  65.74434869 101.70590427  87.89949819
  74.20590427  98.96844335], Error = 0.026711225509643555
Iteración 15: x = [ 89.43963286  66.716681    67.43963283  93.05649308  71.05649306
  84.80479318  65.13423974  65.75575579 101.71698539  87.91403931
  74.21698539  98.97795213], Error = 0.014541111886501312
Iteración 16: x = [ 89.44329351  66.71981642  67.44329352  93.0611065   71.06110651
  84.81180647  65.14013724  65.76206976 101.72299786  87.92243164
  74.22299786  98.98349269], Error = 0.008392333984375
Convergencia alcanzada en la iteración 16 con error 8.3923e-03.

Solución final: x = [ 89.44329351  66.71981642  67.44329352  93.0611065   71.06110651
  84.81180647  65.14013724  65.76206976 101.72299786  87.92243164
  74.22299786  98.98349269]
```

c) Repita la parte **b)** mediante el método de Gauss-Siedel.

```python
b = np.array([220, 110, 110, 220, 110, 110, 110, 110, 220, 110, 110, 220])
x0 = np.zeros(len(b))

print("Método de Siedel:")
_ = gauss_seidel_method(A, b, x0, 50, 1e-2)
```

```
Método de Siedel:
Iteración 1: x = [55.         41.25       37.8125     64.453125   41.25       53.92578125
 40.98144531 37.74536133 55.         50.68634033 40.17158508 78.79289627], Error = 78.7928962
Iteración 2: x = [75.625      55.859375   57.578125   82.87597656 59.88769531 73.4362793
 55.29541016 53.99543762 87.36980915 72.88420796 65.41927606 93.1972713 ], Error = 32.3698091
Iteración 3: x = [83.93676758 62.87872314 63.93867493 89.34373856 66.84326172 80.37060261
 61.09151006 60.99392951 96.52036982 83.23339385 71.60766629 97.03200903], Error = 10.3491858
Iteración 4: x = [ 87.43049622  65.34229279  66.17150784  91.63552761  69.45027471
  83.04432809  63.5095644   64.18573956 100.06635072  86.46493914
```

```
  73.37423704  98.36014694], Error = 3.545980901180883
Iteración 5: x = [ 88.69814187  66.21741243  66.96323501  92.50189078  70.43561749
  84.11176817  64.57437693  65.25982902 101.20627152  87.4600844
  73.95505783  98.79033234], Error = 1.2676456570625305
Iteración 6: x = [ 89.16325748  66.53162312  67.25837847  92.84253666  70.81875641
  84.5589175   64.95468663  65.60369276 101.56260418  87.78033869
  74.14266776  98.92631799], Error = 0.46511560678482056
Iteración 7: x = [ 89.33759488  66.64899334  67.3728825   92.98295     70.9741281
  84.72794118  65.08290848  65.71581179 101.67666417  87.88378593
  74.20252598  98.96979754], Error = 0.17433740380511153
Iteración 8: x = [ 89.40578036  66.69466571  67.41940393  93.03683628  71.03343039
  84.78829379  65.1260264   65.75245308 101.71339587  87.91709373
  74.22172282  98.98377967], Error = 0.06818547542934539
Iteración 9: x = [ 89.43202402  66.71285699  67.43742332  93.05642928  71.05507945
  84.80938378  65.14045922  65.76438824 101.72521835  87.92783235
  74.22790301  98.98828034], Error = 0.026243666054341475
Iteración 10: x = [ 89.44198411  66.71985186  67.44407028  93.06336352  71.06284197
  84.81666618  65.1452636   65.76827399 101.72902817  87.93130129
  74.22989541  98.9897309 ], Error = 0.009960085383056594
Convergencia alcanzada en la iteración 10 con error 9.9601e-03.

Solución final: x = [ 89.44198411  66.71985186  67.44407028  93.06336352  71.06284197
  84.81666618  65.1452636   65.76827399 101.72902817  87.93130129
  74.22989541  98.9897309 ]
```

Declaración de uso de IA

En la preparación de este contenido, se utilizó ChatGPT para generar las instrucciones de los ejercicios, comprender el código, realizar correcciones y asistir en la presentación de las gráficas, con el objetivo de optimizar el proceso de elaboración y mantener la responsabilidad del producto final en el criterio del autor.

Link del repositorio https://github.com/carol230/MetodosNumericos