

Escuela Politécnica Nacional

Nombre: Francisco Ulloa

Fecha: Quito, 28 de enero de 2026

Tema: Gauss-Jacobi y Gauss-Seidel

Repositorio:

https://github.com/Fu5CHAR/Metodos_numericos_2025B_Ulloa-Francisco/tree/main

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

a. $3x_1 - x_2 + x_3 = 1,$
 $3x_1 + 6x_2 + 2x_3 = 0,$
 $3x_1 + 3x_2 + 7x_3 = 4.$

b. $10x_1 - x_2 = 9,$
 $-x_1 + 10x_2 - 2x_3 = 7,$
 $-2x_2 + 10x_3 = 6.$

c. $10x_1 + 5x_2 = 6,$
 $5x_1 + 10x_2 - 4x_3 = 25,$
 $-4x_2 + 8x_3 - x_4 = -11,$
 $-x_3 + 5x_4 = -11.$

d. $4x_1 + x_2 + x_3 + x_5 = 6,$
 $-x_1 - 3x_2 + x_3 + x_4 = 6,$
 $2x_1 + x_2 + 5x_3 - x_4 - x_5 = 6,$
 $-x_1 - x_2 - x_3 + 4x_4 = 6,$
 $2x_2 - x_3 + x_4 + 4x_5 = 6.$

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

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

4. Utilice el método de Gauss-Seidel para resolver los sistemas lineales en el ejercicio 1, con $TOL = 10^{-3}$.

```
In [1]: import numpy as np

TOL = 1e-2 # Tolerancia
MAX_ITER = 100 # Límite de iteraciones

def jacobi(A, b, x0=None, tol=TOL, max_iter=MAX_ITER):
    n = len(b)
    x = np.zeros(n) if x0 is None else x0.copy()
    x_new = np.zeros(n)

    for k in range(max_iter):
        for i in range(n):
            s = sum(A[i][j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - s) / A[i][i]

        if k < 2:
            print(f"Jacobi - Iteración {k+1}: {x_new}")

        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            return x_new

        x[:] = x_new

    return x

def gauss_seidel(A, b, x0=None, tol=TOL, max_iter=MAX_ITER):
    n = len(b)
    x = np.zeros(n) if x0 is None else x0.copy()

    for k in range(max_iter):
```

```

x_old = x.copy()
for i in range(n):
    s1 = sum(A[i][j] * x[j] for j in range(i))
    s2 = sum(A[i][j] * x_old[j] for j in range(i + 1, n))
    x[i] = (b[i] - s1 - s2) / A[i][i]

if k < 2:
    print(f"Gauss-Seidel - Iteración {k+1}: {x}")

if np.linalg.norm(x - x_old, ord=np.inf) < tol:
    return x

return x

```

Sistema 1

```

In [3]: A = np.array([[3, -1, 1],
                    [3, 6, 2],
                    [3, 3, 7]], dtype=float)

b = np.array([1, 0, 4], dtype=float)

print("Resultado con Jacobi:")
print('Jacobi:', jacobi(A, b))

print("\nResultado con Gauss-Seidel:")
print('Seidel:', gauss_seidel(A, b))

```

Resultado con Jacobi:

```

Jacobi - Iteración 1: [0.33333333 0.          0.57142857]
Jacobi - Iteración 2: [ 0.14285714 -0.35714286  0.42857143]
Jacobi: [ 0.03490444 -0.23975543  0.6547619 ]

```

Resultado con Gauss-Seidel:

```

Gauss-Seidel - Iteración 1: [ 0.33333333 -0.16666667  0.5          ]
Gauss-Seidel - Iteración 2: [ 0.11111111 -0.22222222  0.61904762]
Seidel: [ 0.0361492  -0.23660752  0.65733928]

```

Sistema 2

```

In [4]: A = np.array([[10, -1, 0],
                    [-1, 10, -2],
                    [0, -2, 10]], dtype=float)

b = np.array([9, 7, 6], dtype=float)

print("\nSistema 1b:")
print("Jacobi:", jacobi(A, b))
print("Seidel:", gauss_seidel(A, b))

```

Sistema 1b:

```

Jacobi - Iteración 1: [0.9 0.7 0.6]
Jacobi - Iteración 2: [0.97 0.91 0.74]
Jacobi: [0.99555 0.95725 0.7911 ]
Gauss-Seidel - Iteración 1: [0.9  0.79  0.758]
Gauss-Seidel - Iteración 2: [0.979  0.9495 0.7899]
Seidel: [0.9957475  0.95787375 0.79157475]

```

Sistema 3

```
In [5]: A = np.array([[10, 5, 0, 0],
                    [5, 10, -4, 0],
                    [0, -4, 8, -1],
                    [0, 0, -1, 5]], dtype=float)
b = np.array([6, 25, -11, -11], dtype=float)

print("\nSistema 1c:")
print("Jacobi:", jacobi(A, b))
print("Seidel:", gauss_seidel(A, b))
```

Sistema 1c:

```
Jacobi - Iteración 1: [ 0.6   2.5  -1.375 -2.2 ]
Jacobi - Iteración 2: [-0.65  1.65  -0.4  -2.475]
Jacobi: [-0.79213889  2.79403779 -0.2646472 -2.25205255]
Gauss-Seidel - Iteración 1: [ 0.6   2.2  -0.275 -2.255]
Gauss-Seidel - Iteración 2: [-0.5   2.64  -0.336875 -2.267375]
Seidel: [-0.79019461  2.78841617 -0.26245949 -2.2524919 ]
```

Sistema 4

```
In [7]: A = np.array([[4, 1, 1, 0, 1],
                    [-1, -3, 1, 1, 0],
                    [2, 1, 5, -1, -1],
                    [-1, -1, -1, 4, 0],
                    [0, 2, -1, 1, 4]], dtype=float)
b = np.array([6, 6, 6, 6, 6], dtype=float)

print("\nSistema 1d:")
print("Jacobi:", jacobi(A, b))
print("Seidel:", gauss_seidel(A, b))
```

Sistema 1d:

```
Jacobi - Iteración 1: [ 1.5 -2.   1.2  1.5  1.5]
Jacobi - Iteración 2: [ 1.325 -1.6   1.6   1.675  2.425]
Jacobi: [ 0.7850751 -0.99873844  1.8646296  1.91522095  1.98538479]
Gauss-Seidel - Iteración 1: [ 1.5  -2.5   1.1   1.525  2.64375]
Gauss-Seidel - Iteración 2: [ 1.1890625 -1.52135417  1.86239583  1.88252604  2.2
5564453]
Seidel: [ 0.78616258 -1.00240703  1.86606999  1.91245638  1.98960692]
```

5. El sistema lineal

$$\begin{aligned} 2x_1 - x_2 + x_3 &= -1, \\ 2x_1 + 2x_2 + 2x_3 &= 4, \\ -x_1 - x_2 + 2x_3 &= -5, \end{aligned}$$

tiene la solución $(1, 2, -1)$.

a) Muestre que el método de Jacobi con $\mathbf{x}_{(0)} = \mathbf{0}$ falla al proporcionar una buena aproximación después de 25 iteraciones.

b) Utilice el método de Gauss-Seidel con $\mathbf{x}_{(0)} = \mathbf{0}$ para aproximar la solución para el sistema lineal dentro de 10^{-5} .

```
In [8]: import numpy as np

def jacobi(A, b, x0, max_iter=25):
    n = len(b)
    x = x0.copy()
    x_new = np.zeros_like(x)
    history = []
```

```

for k in range(max_iter):
    for i in range(n):
        s = np.dot(A[i,:], x) - A[i,i]*x[i]
        x_new[i] = (b[i] - s) / A[i,i]

    x = x_new.copy()
    history.append(x.copy())

return x, np.array(history)

# Sistema dado
A = np.array([[2, -1, 1],
              [2, 2, 2],
              [-1, -1, 2]])
b = np.array([-1, 4, -5])
x0 = np.zeros(3)

# Aplicar Jacobi
x_jacobi, hist_jacobi = jacobi(A, b, x0)
print("Resultado después de 25 iteraciones (Jacobi):", x_jacobi)

#Sistema
A = np.array([[2, -1, 1],
              [2, 2, 2],
              [-1, -1, 2]])
b = np.array([-1, 4, -5])
x0 = np.zeros(3)

# Aplicar Jacobi con solo 25 iteraciones
x_jacobi, hist_jacobi = jacobi(A, b, x0)
print("Resultado después de 25 iteraciones (Jacobi):", x_jacobi)

```

Resultado después de 25 iteraciones (Jacobi): [-20.82787284 2. -22.82787284]

Resultado después de 25 iteraciones (Jacobi): [-20.82787284 2. -22.82787284]

Con Gauss-Jacobi el sistema no llega a una solución adecuada.

```

In [9]: def gauss_seidel(A, b, x0, tol=1e-5, max_iter=100):
        n = len(b)
        x = x0.copy()
        history = []

        for k in range(max_iter):
            x_old = x.copy()
            for i in range(n):
                s1 = np.dot(A[i,:i], x[:i])
                s2 = np.dot(A[i,i+1:], x_old[i+1:])
                x[i] = (b[i] - s1 - s2) / A[i,i]

            history.append(x.copy())
            if np.linalg.norm(x - x_old) < tol:
                break

        return x, np.array(history)

#Sistema
A = np.array([[2, -1, 1],

```

```

        [2, 2, 2],
        [-1, -1, 2]])
b = np.array([-1, 4, -5])
x0 = np.zeros(3)

# Aplicar Gauss-Seidel
x_gs, hist_gs = gauss_seidel(A, b, x0)
print("Resultado Gauss-Seidel:", x_gs)
print("Iteraciones realizadas:", len(hist_gs))

```

Resultado Gauss-Seidel: [1.00000226 1.9999975 -1.00000012]
Iteraciones realizadas: 23

```

In [15]: import numpy as np
import matplotlib.pyplot as plt

sol_exacta = [1, 2, -1]
# Crear gráfico
plt.figure(figsize=(12, 8))
variables = ['x1', 'x2', 'x3']
colors = ['#1f77b4', '#ff7f0e', '#2ca02c']
styles = {'Jacobi': '--', 'Gauss-Seidel': '-'}

for i in range(3):
    # Jacobi
    plt.plot(range(25), hist_jacobi[:, i], styles['Jacobi'],
             color=colors[i], label=f'Jacobi {variables[i]}', alpha=0.7)

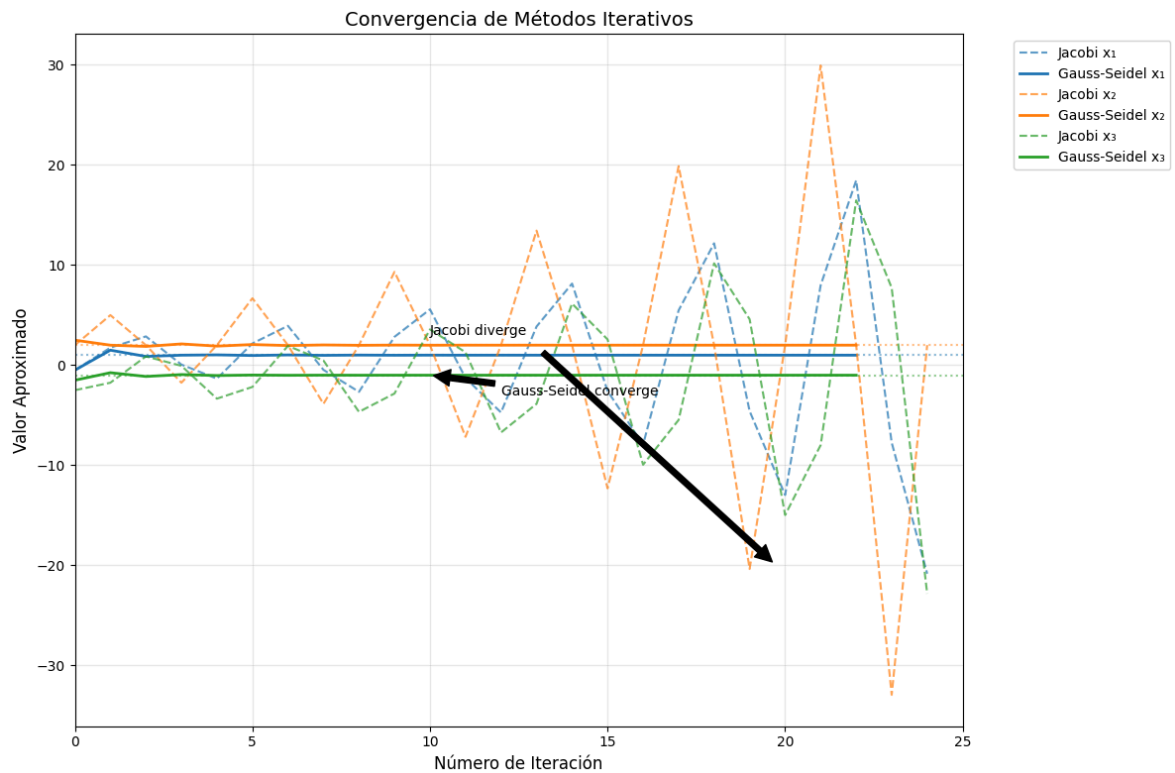
    # Gauss-Seidel
    gs_iter = len(hist_gs)
    plt.plot(range(gs_iter), hist_gs[:, i], styles['Gauss-Seidel'],
             color=colors[i], label=f'Gauss-Seidel {variables[i]}', linewidth=2)

    # Solución exacta
    plt.axhline(y=sol_exacta[i], color=colors[i], linestyle=':', alpha=0.5)

plt.title('Convergencia de Métodos Iterativos', fontsize=14)
plt.xlabel('Número de Iteración', fontsize=12)
plt.ylabel('Valor Aproximado', fontsize=12)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, alpha=0.3)
plt.xlim(0, max(25, len(hist_gs)+1))
plt.tight_layout()

# Añadir anotaciones
plt.annotate('Jacobi diverge', xy=(20, hist_jacobi[-1, 0]), xytext=(10, 3),
            arrowprops=dict(facecolor='black', shrink=0.05))
plt.annotate('Gauss-Seidel converge', xy=(10, hist_gs[10, 2]), xytext=(12, -3),
            arrowprops=dict(facecolor='black', shrink=0.05))
plt.show()

```



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?

b) Utilice el método iterativo de Gauss-Seidel para aproximar la solución para el sistema lineal con una tolerancia de 10^{-22} y un máximo de 300 iteraciones.

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

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

7. Repita el ejercicio 11 usando el método de Jacobi.

a) La matriz no es diagonal dominante.

```
In [16]: import numpy as np

def gauss_seidel_mejorado(A, b, tol=1e-22, max_iter=300):
    n = len(b)
    x = np.zeros(n)

    for k in range(max_iter):
        x_old = x.copy()

        for i in range(n):
            # Evitar división por cero
            if A[i,i] == 0:
                # Buscar intercambio de filas
                for j in range(i+1, n):
```

```

        if A[j,i] != 0:
            A[[i,j]] = A[[j,i]] # Intercambiar filas
            b[[i,j]] = b[[j,i]]
            break
    if A[i,i] == 0:
        raise ValueError("Sistema singular - no se puede resolver")

    s1 = np.dot(A[i, :i], x[:i])
    s2 = np.dot(A[i, i+1:], x_old[i+1:])
    x[i] = (b[i] - s1 - s2) / A[i,i]

    if np.linalg.norm(x - x_old) < tol:
        print(f'Convergencia en {k+1} iteraciones')
        return x

print("Advertencia: Máximo de iteraciones alcanzado")
return x

A_corr = np.array([
    [1.0, 0.0, -1.0],
    [0.5, 0.1, -0.25],
    [1.0, -0.5, 1.0]
])

b_corr = np.array([0.2, -1.425, 2.0])

# Solución del sistema corregido
sol = gauss_seidel_mejorado(A_corr, b_corr)
print("Solución del sistema corregido:", sol)

# Verificación
print("\nVerificación:")
print("A·x - b =", np.dot(A_corr, sol) - b_corr)

```

Advertencia: Máximo de iteraciones alcanzado

Solución del sistema corregido: [-3.59737311e+105 8.99343277e+105 8.09408950e+105]

Verificación:

A·x - b = [-1.16914626e+106 -2.92286565e+105 -2.00000000e+000]

```

In [19]: def gauss_seidel(A, b, x0, tol=1e-5, max_iter=100):
    n = len(b)
    x = x0.copy()
    history = []

    for k in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            s1 = np.dot(A[i,:i], x[:i])
            s2 = np.dot(A[i,i+1:], x_old[i+1:])
            x[i] = (b[i] - s1 - s2) / A[i,i]

        history.append(x.copy())
        if np.linalg.norm(x - x_old) < tol:
            break

    return x, np.array(history)

#Sistema

```

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

# Aplicar Gauss-Seidel
x_gs, hist_gs = gauss_seidel(A, b, x0)
print("Resultado Gauss-Seidel:", x_gs)
print("Iteraciones realizadas:", len(hist_gs))
A_mod = np.array([
    [1.0, 0.0, -2.0],
    [-0.5, 1.0, -0.25],
    [1.0, -0.5, 1.0]
])

b = np.array([0.2, -1.425, 2.0])

hist = gauss_seidel(A_mod, b, x0)
solucion = hist[-1]
print("Solución con sistema modificado", solucion)
```

Resultado Gauss-Seidel: [1.00000226 1.9999975 -1.0000012]

Iteraciones realizadas: 23

Solución con sistema modificado [[2.00000000e-01 -1.32500000e+00 1.13750000e+00]

0]
[2.47500000e+00 9.68750000e-02 -4.26562500e-01]
[-6.53125000e-01 -1.85820313e+00 1.72402344e+00]
[3.64804688e+00 8.30029297e-01 -1.23303223e+00]
[-2.26606445e+00 -2.86629028e+00 2.83291931e+00]
[5.86583862e+00 2.21614914e+00 -2.75776405e+00]
[-5.31552811e+00 -4.77220507e+00 4.92942557e+00]
[1.00588511e+01 4.83678197e+00 -5.64046016e+00]
[-1.10809203e+01 -8.37557520e+00 8.89313272e+00]
[1.79862654e+01 9.79141591e+00 -1.10905575e+01]
[-2.19811150e+01 -1.51881969e+01 1.63870166e+01]
[3.29740331e+01 1.91587707e+01 -2.13946478e+01]
[-4.25892955e+01 -2.80683097e+01 3.05551407e+01]
[6.13102814e+01 3.68689258e+01 -4.08758184e+01]
[-8.15516369e+01 -5.24197730e+01 5.73417503e+01]
[1.14883501e+02 7.03521879e+01 -7.77074067e+01]
[-1.55214813e+02 -9.84592584e+01 1.07985184e+02]
[2.16170368e+02 1.33656480e+02 -1.47342128e+02]
[-2.94484257e+02 -1.85502660e+02 2.03732926e+02]
[4.07665853e+02 2.53341158e+02 -2.78995274e+02]
[-5.57790548e+02 -3.50069092e+02 3.84756002e+02]
[7.69712003e+02 4.79620002e+02 -5.27902002e+02]
[-1.05560400e+03 -6.61202503e+02 7.27002753e+02]
[1.45420551e+03 9.07428441e+02 -9.98491285e+02]
[-1.99678257e+03 -1.24943911e+03 1.37406302e+03]
[2.74832603e+03 1.71625377e+03 -1.88819915e+03]
[-3.77619830e+03 -2.36157394e+03 2.59741133e+03]
[5.19502266e+03 3.24543916e+03 -3.57030308e+03]
[-7.14040616e+03 -4.46420385e+03 4.91030423e+03]
[9.82080847e+03 6.13655529e+03 -6.75053082e+03]
[-1.35008616e+04 -8.43948853e+03 9.28311738e+03]
[1.85664348e+04 1.16025717e+04 -1.27631489e+04]
[-2.55260978e+04 -1.59552611e+04 1.75504672e+04]
[3.51011345e+04 2.19367590e+04 -2.41307549e+04]
[-4.82613099e+04 -3.01647687e+04 3.31809255e+04]
[6.63620511e+04 4.14748319e+04 -4.56226351e+04]
[-9.12450702e+04 -5.70296189e+04 6.27322608e+04]
[1.25464722e+05 7.84140010e+04 -8.62557211e+04]
[-1.72511242e+05 -1.07820976e+05 1.18602754e+05]
[2.37205708e+05 1.48252118e+05 -1.63077649e+05]
[-3.26155099e+05 -2.03848387e+05 2.24232905e+05]
[4.48466010e+05 2.80289807e+05 -3.08319107e+05]
[-6.16638014e+05 -3.85400209e+05 4.23939910e+05]
[8.47880020e+05 5.29923562e+05 -5.82916239e+05]
[-1.16583228e+06 -7.28646623e+05 8.01510966e+05]
[1.60302213e+06 1.00188738e+06 -1.10207644e+06]
[-2.20415268e+06 -1.37759688e+06 1.51535624e+06]
[3.03071269e+06 1.89419398e+06 -2.08361370e+06]
[-4.16722719e+06 -2.60451845e+06 2.86496997e+06]
[5.72994014e+06 3.58121114e+06 -3.93933257e+06]
[-7.87866494e+06 -4.92416704e+06 5.41658342e+06]
[1.08331670e+07 6.77072795e+06 -7.44780107e+06]
[-1.48956019e+07 -9.30975266e+06 1.02407276e+07]
[2.04814554e+07 1.28009082e+07 -1.40809993e+07]
[-2.81619984e+07 -1.76012505e+07 1.93613752e+07]
[3.87227506e+07 2.42017177e+07 -2.66218898e+07]
[-5.32437793e+07 -3.32773635e+07 3.66050996e+07]

```
[ 7.32101993e+07  4.57563731e+07 -5.03320108e+07]
[-1.00664021e+08 -6.29150148e+07  6.92065159e+07]
[ 1.38413032e+08  8.65081436e+07 -9.51589583e+07]
[-1.90317916e+08 -1.18948699e+08  1.30843569e+08]
[ 2.61687138e+08  1.63554460e+08 -1.79909906e+08]
[-3.59819812e+08 -2.24887384e+08  2.47376122e+08]
[ 4.94752244e+08  3.09220151e+08 -3.40142166e+08]
[-6.80284333e+08 -4.25177709e+08  4.67695480e+08]
[ 9.35390960e+08  5.84619349e+08 -6.43081284e+08]
[-1.28616257e+09 -8.03851606e+08  8.84236766e+08]
[ 1.76847353e+09  1.10529596e+09 -1.21582555e+09]
[-2.43165110e+09 -1.51978194e+09  1.67176014e+09]
[ 3.34352027e+09  2.08970017e+09 -2.29867019e+09]
[-4.59734037e+09 -2.87333773e+09  3.16067151e+09]
[ 6.32134301e+09  3.95083938e+09 -4.34592332e+09]
[-8.69184664e+09 -5.43240415e+09  5.97564457e+09]
[ 1.19512891e+10  7.46955571e+09 -8.21651128e+09]
[-1.64330226e+10 -1.02706391e+10  1.12977030e+10]
[ 2.25954060e+10  1.41221288e+10 -1.55343416e+10]
[-3.10686833e+10 -1.94179270e+10  2.13597197e+10]
[ 4.27194395e+10  2.66996497e+10 -2.93696147e+10]
[-5.87392293e+10 -3.67120183e+10  4.03832201e+10]
[ 8.07664403e+10  5.04790252e+10 -5.55269277e+10]
[-1.11053855e+11 -6.94086596e+10  7.63495256e+10]
[ 1.52699051e+11  9.54369070e+10 -1.04980598e+11]
[-2.09961195e+11 -1.31225747e+11  1.44348322e+11]
[ 2.88696644e+11  1.80435402e+11 -1.98478942e+11]
[-3.96957885e+11 -2.48098678e+11  2.72908546e+11]
[ 5.45817092e+11  3.41135682e+11 -3.75249251e+11]
[-7.50498501e+11 -4.69061563e+11  5.15967720e+11]
[ 1.03193544e+12  6.44959650e+11 -7.09455615e+11]
[-1.41891123e+12 -8.86819518e+11  9.75501470e+11]
[ 1.95100294e+12  1.21937684e+12 -1.34131452e+12]
[-2.68262904e+12 -1.67664315e+12  1.84430747e+12]
[ 3.68861493e+12  2.30538433e+12 -2.53592277e+12]
[-5.07184553e+12 -3.16990346e+12  3.48689380e+12]
[ 6.97378761e+12  4.35861726e+12 -4.79447898e+12]
[-9.58895796e+12 -5.99309873e+12  6.59240860e+12]
[ 1.31848172e+13  8.24051075e+12 -9.06456182e+12]
[-1.81291236e+13 -1.13307023e+13  1.24637725e+13]
[ 2.49275450e+13  1.55797156e+13 -1.71376872e+13]
[-3.42753744e+13 -2.14221090e+13  2.35643199e+13]
[ 4.71286398e+13  2.94553999e+13 -3.24009399e+13]]
```

Responda el ejercicio 11 usando el método de Gauss.

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.

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

- ¿La matriz es estrictamente diagonalmente dominante?
- Resuelva el sistema lineal usando el método de Jacobi con $\mathbf{x}_0 = \mathbf{0}$ y $TOL = 10^{-2}$.
- Repita la parte b) mediante el método de Gauss-Siedel.

a) La matriz es diagonal dominante.

```
In [24]: import numpy as np
import numpy as np

TOL = 1e-2 # Tolerancia
MAX_ITER = 100 # Límite de iteraciones

def jacobi(A, b, x0=None, tol=TOL, max_iter=MAX_ITER):
    n = len(b)
    x = np.zeros(n) if x0 is None else x0.copy()
    x_new = np.zeros(n)

    for k in range(max_iter):
        for i in range(n):
            s = sum(A[i][j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - s) / A[i][i]

        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            return x_new

        x[:] = x_new

    return x

def gauss_seidel(A, b, x0=None, tol=TOL, max_iter=MAX_ITER):
    n = len(b)
    x = np.zeros(n) if x0 is None else x0.copy()

    for k in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            s1 = sum(A[i][j] * x[j] for j in range(i))
            s2 = sum(A[i][j] * x_old[j] for j in range(i + 1, n))
            x[i] = (b[i] - s1 - s2) / A[i][i]
```

```

        if np.linalg.norm(x - x_old, ord=np.inf) < tol:
            return x

    return x

x0 = [0,0,0,0,0,0,0,0,0,0,0,0,]
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,  0, -1,  0,  0,  0,  0,  0],
    [ 0,  0,  0, -1,  0,  4,  0, -1,  0,  0,  0,  0],
    [ 0,  0,  0,  0, -1,  0,  4,  0, -1,  0,  0,  0],
    [ 0,  0,  0,  0,  0, -1,  0,  4,  0,  0,  0, -1],
    [ 0,  0,  0,  0,  0,  0, -1,  0,  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, -1,  0,  0, -1,  4]
], dtype=float)

b = np.array([
    220, 110, 110, 220,
    110, 110, 110, 110,
    220, 110, 110, 220
], dtype=float)

print("Jacobi:", jacobi(A, b), '\n')
print("Gauss-Seidel:", gauss_seidel(A, b))
```

```
Jacobi: [87.9910484  65.99104807 65.99104807 87.9910484  65.99104807 65.99104807
 65.99104807 65.99104807 87.9910484  65.99104807 65.99104807 87.9910484 ]
```

```
Gauss-Seidel: [87.9986895  65.99931189 65.99963128 87.99980331 65.99931189 65.999
89703
 65.99963128 65.99994736 87.99980331 65.99989703 65.99994736 87.99997368]
```