ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA EN SISTEMAS

METODOS NUMERICOS ICCD412

JUAN FRANCISCO PINTO ANDRANGO

Tarea 11 Ejercicios Unidad 04-D Gauss-Jacobi y Gauss-Seidel

GR1CC

FECHA DE ENTREGA 28 DE ENERO DEL 2026

Indicaciones

Puede realizar los cálculos a mano, o utilizar cualquier librería o implementar su propia función.

En caso de usar código, subir la resolución de los ejercicios en un repositorio público en Github e incluir enlace de su repositorio.

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

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

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

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

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

In [20]:
```python
import numpy as np
def jacobi_metodo(A, b):

    # calcula 2 iteracion con metodo de Jacobi con x^(0) = 0

    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)

    n = len(b)
```

```
    x0 = np.zeros(n)
    x1 = np.zeros(n)
    x2 = np.zeros(n)

    # iteracion 1
    for i in range(n):
        suma = sum(A[i, j] * x0[j] for j in range(n) if j != i)
        x1[i] = (b[i] - suma) / A[i, i]

    # iteracion 2
    for i in range(n):
        suma = sum(A[i, j] * x1[j] for j in range(n) if j != i)
        x2[i] = (b[i] - suma) / A[i, i]

    return x0, x1, x2
```

In [25]:

```
sistemas = [
    ("Sistema 1",
     [[3, -1, 1], [3,  6, 2],[3,  3, 7]],
     [1, 0, 4]),

    ("Sistema 2",
     [[10,  1,  0],[-1, 10, -2], [0,  -2, 10]],
     [9, 7, 6]),

    ("Sistema 3",
     [[10,  5,  0,  0], [5,  10, -4,  0],[0,  -4,  8, -1],[0,   0, -1,  5]],
     [6, 25, -11, -11]),

    ("Sistema 4",
     [[4,  1,  1,  0,  1], [-1, -3,  1,  1,  0],[2,  1,  5, -1, -1],[-1, -1,
     [6, 6, 6, 6, 6])
]

for nombre, A, b in sistemas:
    x0, x1, x2 = jacobi_metodo(A, b)

    print("=" * 50)
    print(nombre)
    print("con valores de  0 =", x0)
    print("iteracion 1 =", x1)
    print("iteracion 2 =", x2)
```

```
==================================================
Sistema 1
con valores de  0 = [0. 0. 0.]
iteracion 1 = [0.33333333 0.         0.57142857]
iteracion 2 = [ 0.14285714 -0.35714286  0.42857143]
==================================================
Sistema 2
con valores de  0 = [0. 0. 0.]
iteracion 1 = [0.9 0.7 0.6]
iteracion 2 = [0.83 0.91 0.74]
==================================================
Sistema 3
con valores de  0 = [0. 0. 0. 0.]
iteracion 1 = [ 0.6    2.5   -1.375 -2.2  ]
iteracion 2 = [-0.65   1.65  -0.4   -2.475]
==================================================
Sistema 4
con valores de  0 = [0. 0. 0. 0. 0.]
iteracion 1 = [ 1.5 -2.   1.2  1.5  1.5]
iteracion 2 = [ 1.325 -1.6    1.6    1.675  2.425]
```

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

In [26]:

```python
def gauss_seidel(A, b):

    #calculamos con el metodo de Gauss-Seidel con x^(0) = 0

    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)

    n = len(b)

    x0 = np.zeros(n)
    x1 = x0.copy()
    x2 = x1.copy()

    # iteracion 1
    for i in range(n):
        suma = 0
        for j in range(n):
            if j < i:
                suma += A[i, j] * x1[j]   # valores nuevos
            elif j > i:
                suma += A[i, j] * x0[j]   # valores anteriores

        x1[i] = (b[i] - suma) / A[i, i]

    # iteracion 2
    for i in range(n):
        suma = 0
        for j in range(n):
            if j < i:
                suma += A[i, j] * x2[j]
            elif j > i:
                suma += A[i, j] * x1[j]
```

```
        x2[i] = (b[i] - suma) / A[i, i]

    return x0, x1, x2
```

In [27]:
```
sistemas = [
    ("Sistema 1",
     [[3, -1, 1],[3,  6, 2],[3,  3, 7]],
     [1, 0, 4]),

    ("Sistema 2",
     [[10,  1,  0],[-1, 10, -2],[0,  -2, 10]],
     [9, 7, 6]),

    ("Sistema 3",
     [[10,  5,  0,  0],[5,  10, -4,  0],[0,  -4,  8, -1],[0,   0, -1,  5]],
     [6, 25, -11, -11]),

    ("Sistema 4",
     [[4,  1,  1,  0,  1],[-1, -3,  1,  1,  0],[2,  1,  5, -1, -1],[-1, -1,
     [6, 6, 6, 6, 6])
]

for nombre, A, b in sistemas:
    x0, x1, x2 = gauss_seidel(A, b)

    print("=" * 50)
    print(nombre)
    print("con paratemtro 0 x^(0) =", x0)
    print("iteracion 1 x^(1) =", x1)
    print("iteracion 2 x^(2) =", x2)
```

```
==================================================
Sistema 1
con paratemtro 0 x^(0) = [0. 0. 0.]
iteracion 1 x^(1) = [ 0.33333333 -0.16666667  0.5       ]
iteracion 2 x^(2) = [ 0.11111111 -0.22222222  0.61904762]
==================================================
Sistema 2
con paratemtro 0 x^(0) = [0. 0. 0.]
iteracion 1 x^(1) = [0.9   0.79  0.758]
iteracion 2 x^(2) = [0.821   0.9337  0.78674]
==================================================
Sistema 3
con paratemtro 0 x^(0) = [0. 0. 0. 0.]
iteracion 1 x^(1) = [ 0.6    2.2   -0.275 -2.255]
iteracion 2 x^(2) = [-0.5       2.64     -0.336875 -2.267375]
==================================================
Sistema 4
con paratemtro 0 x^(0) = [0. 0. 0. 0. 0.]
iteracion 1 x^(1) = [ 1.5     -2.5      1.1      1.525    2.64375]
iteracion 2 x^(2) = [ 1.1890625  -1.52135417  1.86239583  1.88252604  2.2556
4453]
```

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

In [35]:
```python
import numpy as np

def jacobi_metd(A, b):

    ### metodo de Jacobi con x^(0) = 0

    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)

    n = len(b)
    x_old = np.zeros(n)

    resultados = [x_old.copy()]  # x^(0)

    for k in range(1, 11):  # 10 iteraciones
        x_new = np.zeros(n)

        for i in range(n):
            suma = 0
            for j in range(n):
                if j != i:
                    suma += A[i, j] * x_old[j]

            x_new[i] = (b[i] - suma) / A[i, i]

        resultados.append(x_new.copy())
        x_old = x_new.copy()

    return resultados
```

In [ ]:
```python
sistemas = [
    ("Sistema 1",
     [[3, -1, 1],[3,  6, 2], [3,  3, 7]],
     [1, 0, 4]),

    ("Sistema 2",
     [[10,  1,  0],[-1, 10, -2],[0,  -2, 10]],
     [9, 7, 6]),

    ("Sistema 3",
     [[10,  5,  0,  0],[5,  10, -4,  0],[0,  -4,  8, -1],[0,   0, -1,  5]],
     [6, 25, -11, -11]),

    ("Sistema 4",
     [[4,  1,  1,  0,  1],[-1, -3,  1,  1,  0],[2,  1,  5, -1, -1],[-1, -1,
     [6, 6, 6, 6, 6])
]

for nombre, A, b in sistemas:
    print("=" * 60)
    print(nombre)
```

```
    resultados = jacobi_metd(A, b)

    for i, x in enumerate(resultados):
        print(f"x^({i}) = {x}")
```

```
============================================================
Sistema 1
x^(0) = [0. 0. 0.]
x^(1) = [0.33333333 0.         0.57142857]
x^(2) = [ 0.14285714 -0.35714286  0.42857143]
x^(3) = [ 0.07142857 -0.21428571  0.66326531]
x^(4) = [ 0.04081633 -0.25680272  0.63265306]
x^(5) = [ 0.03684807 -0.23129252  0.66399417]
x^(6) = [ 0.03490444 -0.23975543  0.6547619 ]
x^(7) = [ 0.03516089 -0.23570619  0.65922185]
x^(8) = [ 0.03502399 -0.23732106  0.65737656]
x^(9) = [ 0.03510079 -0.23663751  0.65812732]
x^(10) = [ 0.03507839 -0.23692617  0.65780145]
============================================================
Sistema 2
x^(0) = [0. 0. 0.]
x^(1) = [0.9 0.7 0.6]
x^(2) = [0.83 0.91 0.74]
x^(3) = [0.809 0.931 0.782]
x^(4) = [0.8069 0.9373 0.7862]
x^(5) = [0.80627 0.93793 0.78746]
x^(6) = [0.806207 0.938119 0.787586]
x^(7) = [0.8061881 0.9381379 0.7876238]
x^(8) = [0.80618621 0.93814357 0.78762758]
x^(9) = [0.80618564 0.93814414 0.78762871]
x^(10) = [0.80618559 0.93814431 0.78762883]
============================================================
Sistema 3
x^(0) = [0. 0. 0. 0.]
x^(1) = [ 0.6    2.5   -1.375 -2.2  ]
x^(2) = [-0.65   1.65  -0.4   -2.475]
x^(3) = [-0.225     2.665    -0.859375 -2.28    ]
x^(4) = [-0.7325    2.26875  -0.3275   -2.371875]
x^(5) = [-0.534375    2.73525    -0.53710938 -2.2655    ]
x^(6) = [-0.767625    2.55234375 -0.2905625  -2.30742188]
x^(7) = [-0.67617187  2.7675875  -0.38725586 -2.2581125 ]
x^(8) = [-0.78379375  2.68318359 -0.27347031 -2.27745117]
x^(9) = [-0.7415918   2.78250875 -0.3180896  -2.25469406]
x^(10) = [-0.79125437  2.74356006 -0.26558238 -2.26361792]
============================================================
Sistema 4
x^(0) = [0. 0. 0. 0. 0.]
x^(1) = [ 1.5 -2.   1.2  1.5  1.5]
x^(2) = [ 1.325 -1.6    1.6    1.675  2.425]
x^(3) = [ 0.89375 -1.35     1.81     1.83125  2.28125]
x^(4) = [ 0.8146875  -1.08416667  1.935       1.8384375   2.1696875 ]
x^(5) = [ 0.74486979 -1.01375     1.89258333  1.91638021  2.06622396]
x^(6) = [ 0.76373568 -0.97863542  1.90132292  1.90592578  2.00092578]
x^(7) = [ 0.76909668 -0.98549566  1.87160313  1.92160579  1.98816699]
x^(8) = [ 0.78143139 -0.99196259  1.87141502  1.91380104  1.98024716]
x^(9) = [ 0.7850751  -0.99873844  1.8646296   1.91522095  1.98538479]
x^(10) = [ 0.78718101 -1.00174151  1.8658388   1.91274157  1.98672138]
```

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

In [37]:
```python
import numpy as np

def gauss_seidel(A, b, tol=1e-3, max_iter=100):

    ### metodo de Gauss-Seidel para sistemas lineales

    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)

    n = len(b)
    x = np.zeros(n)              # x^(0) = 0
    resultados = [x.copy()]      # guardar x^(0)

    for k in range(1, max_iter + 1):
        x_new = x.copy()

        for i in range(n):
            suma1 = np.dot(A[i, :i], x_new[:i])   # valores nuevos
            suma2 = np.dot(A[i, i+1:], x[i+1:])   # valores viejos
            x_new[i] = (b[i] - suma1 - suma2) / A[i, i]

        resultados.append(x_new.copy())

        # criterio de parada (norma infinito)
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            break

        x = x_new

    return resultados
```

In [ ]:
```python
sistemas = [
    ("Sistema 1",
     [[3, -1, 1],[3,  6, 2], [3,  3, 7]],
     [1, 0, 4]),

    ("Sistema 2",
     [[10,  1,  0],[-1, 10, -2],[0,  -2, 10]],
     [9, 7, 6]),

    ("Sistema 3",
     [[10,  5,  0,  0],[5,  10, -4,  0],[0,  -4,  8, -1],[0,   0, -1,  5]],
     [6, 25, -11, -11]),

    ("Sistema 4",
     [[4,  1,  1,  0,  1],[-1, -3,  1,  1,  0],[2,  1,  5, -1, -1],[-1, -1,
     [6, 6, 6, 6, 6])
]

TOL = 1e-3

for nombre, A, b in sistemas:
    print("=" * 60)
    print(nombre, " metodo  de Gauss-Seidel TOL = 10⁻³)")
```

```
    resultados = gauss_seidel(A, b, tol=TOL)

    for i, x in enumerate(resultados):
        print(f"x^({i}) = {x}")
```

```
============================================================
Sistema 1  metodo  de Gauss-Seidel TOL = 10⁻³)
x^(0) = [0. 0. 0.]
x^(1) = [ 0.33333333 -0.16666667  0.5       ]
x^(2) = [ 0.11111111 -0.22222222  0.61904762]
x^(3) = [ 0.05291005 -0.23280423  0.64852608]
x^(4) = [ 0.03955656 -0.23595364  0.65559875]
x^(5) = [ 0.0361492  -0.23660752  0.65733928]
x^(6) = [ 0.03535107 -0.23678863  0.65775895]
============================================================
Sistema 2  metodo  de Gauss-Seidel TOL = 10⁻³)
x^(0) = [0. 0. 0.]
x^(1) = [0.9   0.79  0.758]
x^(2) = [0.821   0.9337  0.78674]
x^(3) = [0.80663   0.938011  0.7876022]
x^(4) = [0.8061989  0.93814033 0.78762807]
============================================================
Sistema 3  metodo  de Gauss-Seidel TOL = 10⁻³)
x^(0) = [0. 0. 0. 0.]
x^(1) = [ 0.6    2.2   -0.275 -2.255]
x^(2) = [-0.5       2.64     -0.336875 -2.267375]
x^(3) = [-0.72        2.72525    -0.29579688 -2.25915938]
x^(4) = [-0.762625    2.76299375 -0.27589805 -2.25517961]
x^(5) = [-0.78149687  2.78038922 -0.26670284 -2.25334057]
x^(6) = [-0.79019461  2.78841617 -0.26245949 -2.2524919 ]
x^(7) = [-0.79420808  2.79212025 -0.26050136 -2.25210027]
x^(8) = [-0.79606012  2.79382952 -0.25959778 -2.25191956]
x^(9) = [-0.79691476  2.79461827 -0.25918081 -2.25183616]
============================================================
Sistema 4  metodo  de Gauss-Seidel TOL = 10⁻³)
x^(0) = [0. 0. 0. 0. 0.]
x^(1) = [ 1.5     -2.5      1.1      1.525    2.64375]
x^(2) = [ 1.1890625  -1.52135417  1.86239583  1.88252604  2.25564453]
x^(3) = [ 0.85082845 -1.03530219  1.89436317  1.92747236  2.0093738 ]
x^(4) = [ 0.7828913  -0.98701859  1.87161643  1.91687229  1.98219533]
x^(5) = [ 0.78330171 -0.998271    1.86614704  1.91279444  1.98747365]
x^(6) = [ 0.78616258 -1.00240703  1.86606999  1.91245638  1.98960692]
x^(7) = [ 0.78668253 -1.00271872  1.86628339  1.9125618   1.98978976]
```

5 El sistema lineal

$$2x_1 - x_2 + x_3 = -1,$$

$$2x_1 + 2x_2 + 2x_3 = 4,$$

$$-x_1 - x_2 + 2x_3 = -5,$$

tiene lan solución (1,2,-1)

a) Muestre que el método de Jacobi con x(0) = 0 falla al proporcionar una buena aproximación después de 25 iteraciones.

b) Utilice el método de Gauss-Siedel con x(0) = 0:para aproximar la solución para el sistema lineal dentro de 10

In [39]:
```
###a) Muestre que el método de Jacobi con x(0) = 0 falla al proporcionar una
A = np.array([[2, -1,  1],
              [2,  2,  2],
              [-1, -1,  2]])

b = np.array([-1, 4, -5])

x0 = np.zeros(len(b))

max_iter = 25

def jacobi_method_with_iterations(A, b, x0, max_iter):
    n = len(b)
    x = x0.copy()
    results = [x0.copy()]

    for _ in range(max_iter):
        x_new = np.zeros_like(x)
        for i in range(n):
            sum1 = np.dot(A[i, :i], x[:i])
            sum2 = np.dot(A[i, i + 1:], x[i + 1:])
            x_new[i] = (b[i] - sum1 - sum2) / A[i, i]

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

    return results

jacobi_results = jacobi_method_with_iterations(A, b, x0, max_iter)

print("\nResultados del método de Jacobi con 25 iteraciones:")
for i, res in enumerate(jacobi_results):
    print(f"Iteración {i}: {res}")
```

```
Resultados del método de Jacobi con 25 iteraciones:
Iteración 0: [0. 0. 0.]
Iteración 1: [-0.5  2.  -2.5]
Iteración 2: [ 1.75  5.   -1.75]
Iteración 3: [2.875 2.    0.875]
Iteración 4: [ 0.0625 -1.75   -0.0625]
Iteración 5: [-1.34375  2.      -3.34375]
Iteración 6: [ 2.171875  6.6875   -2.171875]
Iteración 7: [3.9296875 2.        1.9296875]
Iteración 8: [-0.46484375 -3.859375    0.46484375]
Iteración 9: [-2.66210938  2.         -4.66210938]
Iteración 10: [ 2.83105469  9.32421875 -2.83105469]
Iteración 11: [5.57763672 2.         3.57763672]
Iteración 12: [-1.28881836 -7.15527344  1.28881836]
Iteración 13: [-4.7220459  2.        -6.7220459]
Iteración 14: [ 3.86102295 13.4440918  -3.86102295]
Iteración 15: [8.15255737 2.         6.15255737]
Iteración 16: [ -2.57627869 -12.30511475   2.57627869]
Iteración 17: [-7.94069672  2.         -9.94069672]
Iteración 18: [ 5.47034836 19.88139343 -5.47034836]
Iteración 19: [12.1758709  2.        10.1758709]
Iteración 20: [ -4.58793545 -20.35174179   4.58793545]
Iteración 21: [-12.96983862   2.         -14.96983862]
Iteración 22: [ 7.98491931 29.93967724 -7.98491931]
Iteración 23: [18.46229827  2.         16.46229827]
Iteración 24: [ -7.73114914 -32.92459655   7.73114914]
Iteración 25: [-20.82787284   2.         -22.82787284]
```

In [41]:

```python
### b) Utilice el método de Gauss-Siedel con x(0) = 0:para aproximar la solu

def gauss_seidel_with_tol(A, b, x0, tol):

    n = len(b)
    x = x0.copy()
    results = [x0.copy()]

    while True:
        x_new = x.copy()
        for i in range(n):
            sum1 = np.dot(A[i, :i], x_new[:i])
            sum2 = np.dot(A[i, i+1:], x[i+1:])
            x_new[i] = (b[i] - sum1 - sum2) / A[i, i]

        results.append(x_new.copy())

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

        x = x_new

    return results

A = np.array([[2, -1, 1],
              [2,  2, 2],
              [-1, -1, 2]], dtype=float)
b = np.array([-1, 4, -5], dtype=float)
```

```python
x0 = np.zeros(3)
tol = 1e-5
gauss_seidel_results = gauss_seidel_with_tol(A, b, x0, tol)

print("\nResultados del método de Gauss-Seidel con tolerancia 10^(-5):")
for i, res in enumerate(gauss_seidel_results):
    print(f"Iteración {i}: {res}")
```

```
Resultados del método de Gauss-Seidel con tolerancia 10^(-5):
Iteración 0: [0. 0. 0.]
Iteración 1: [-0.5  2.5 -1.5]
Iteración 2: [ 1.5   2.   -0.75]
Iteración 3: [ 0.875  1.875 -1.125]
Iteración 4: [ 1.      2.125  -0.9375]
Iteración 5: [ 1.03125  1.90625 -1.03125]
Iteración 6: [ 0.96875   2.0625   -0.984375]
Iteración 7: [ 1.0234375  1.9609375 -1.0078125]
Iteración 8: [ 0.984375    2.0234375  -0.99609375]
Iteración 9: [ 1.00976562  1.98632812 -1.00195312]
Iteración 10: [ 0.99414062  2.0078125  -0.99902344]
Iteración 11: [ 1.00341797  1.99560547 -1.00048828]
Iteración 12: [ 0.99804688  2.00244141 -0.99975586]
Iteración 13: [ 1.00109863  1.99865723 -1.00012207]
Iteración 14: [ 0.99938965  2.00073242 -0.99993896]
Iteración 15: [ 1.00033569  1.99960327 -1.00003052]
Iteración 16: [ 0.99981689  2.00021362 -0.99998474]
Iteración 17: [ 1.00009918  1.99988556 -1.00000763]
Iteración 18: [ 0.99994659  2.00006104 -0.99999619]
Iteración 19: [ 1.00002861  1.99996758 -1.00000191]
Iteración 20: [ 0.99998474  2.00001717 -0.99999905]
Iteración 21: [ 1.00000811  1.99999094 -1.00000048]
Iteración 22: [ 0.99999571  2.00000477 -0.99999976]
Iteración 23: [ 1.00000226  1.9999975  -1.00000012]
```

6 El sistema lineal

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

tiene 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

In [42]:
```python
def es_diagonal_estrictamente_dominante(A):
    n = A.shape[0]
    for i in range(n):
        suma_otros = sum(abs(A[i, j]) for j in range(n) if j != i)
        diagonal = abs(A[i, i])
        if diagonal <= suma_otros:
            return False
    return True

A = np.array([
    [1, 0, -1],
    [-1/2, 1, -1/4],
    [1, -1/2, 1]
])

if es_diagonal_estrictamente_dominante(A):
    print("La matriz es diagonal estrictamente dominante.")
else:
    print("La matriz NO es diagonal estrictamente dominante.")
```

```
La matriz NO es diagonal estrictamente dominante.
```

In [43]:
```python
### #### b) Utilice el método iterativo de Gauss-Siedel para aproximar la so

def gauss_seidel(*, A: np.array, b: np.array, x0: np.array, tol: float, max_

    if not isinstance(A, np.ndarray):
        A = np.array(A, dtype=float)
    assert A.shape[0] == A.shape[1], "La matriz A debe ser de tamaño n-by-(r

    if not isinstance(b, np.ndarray):
        b = np.array(b, dtype=float)
    assert b.shape[0] == A.shape[0], "El vector b debe ser de tamaño n."

    if not isinstance(x0, np.ndarray):
        x0 = np.array(x0, dtype=float)
    assert x0.shape[0] == A.shape[0], "El vector x0 debe ser de tamaño n."

    n = A.shape[0]
    x = x0.copy()
    print(f"i= {0} x: {x.T}")
    for k in range(1, max_iter + 1):
```

```
        for i in range(n):
            suma = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - suma) / A[i, i]
        print(f"i= {k} x: {x.T}")
        if np.linalg.norm(x - x0) < tol:
            print(f"Convergencia alcanzada en {k} iteraciones.")
            break
        x0 = x.copy()
    else:
        print(f"No se alcanzó la convergencia en {max_iter} iteraciones.")
    return x

tol = 1e-2
max_iter = 300

A = np.array([[1, 0, -1], [-0.5, 1, -0.25], [1, -0.5, 1]], dtype=float)
b = np.array([0.2, -1.425, 2], dtype=float)
x0 = np.zeros((3,), dtype=float)

solution = gauss_seidel(A=A, b=b, x0=x0, tol=tol, max_iter=max_iter)
print("Solución del sistema lineal método de Gauss-Seidel:\n", solution)
```

```
i= 0 x: [0. 0. 0.]
i= 1 x: [ 0.2    -1.325   1.1375]
i= 2 x: [ 1.3375    -0.471875   0.4265625]
i= 3 x: [ 0.6265625  -1.00507812  0.87089844]
i= 4 x: [ 1.07089844 -0.67182617  0.59318848]
i= 5 x: [ 0.79318848 -0.88010864  0.7667572 ]
i= 6 x: [ 0.9667572  -0.7499321   0.65827675]
i= 7 x: [ 0.85827675 -0.83129244  0.72607703]
i= 8 x: [ 0.92607703 -0.78044223  0.68370185]
i= 9 x: [ 0.88370185 -0.81222361  0.71018634]
i= 10 x: [ 0.91018634 -0.79236024  0.69363354]
i= 11 x: [ 0.89363354 -0.80477485  0.70397904]
i= 12 x: [ 0.90397904 -0.79701572  0.6975131 ]
i= 13 x: [ 0.8975131  -0.80186517  0.70155431]
Convergencia alcanzada en 13 iteraciones.
Solución del sistema lineal método de Gauss-Seidel:
 [ 0.8975131  -0.80186517  0.70155431]
```

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

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

In [44]:
```python
def gauss_seidel(*, A: np.array, b: np.array, x0: np.array, tol: float, max_

    if not isinstance(A, np.ndarray):
        A = np.array(A, dtype=float)
    assert A.shape[0] == A.shape[1], "La matriz A debe ser de tamaño n-by-(r

    if not isinstance(b, np.ndarray):
        b = np.array(b, dtype=float)
    assert b.shape[0] == A.shape[0], "El vector b debe ser de tamaño n."

    if not isinstance(x0, np.ndarray):
        x0 = np.array(x0, dtype=float)
    assert x0.shape[0] == A.shape[0], "El vector x0 debe ser de tamaño n."

    n = A.shape[0]
    x = x0.copy()
    print(f"i= {0} x: {x.T}")
    for k in range(1, max_iter + 1):
        for i in range(n):
            suma = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - suma) / A[i, i]
        print(f"i= {k} x: {x.T}")
        if np.linalg.norm(x - x0) < tol:
            print(f"Convergencia alcanzada en {k} iteraciones.")
            break
        x0 = x.copy()
    else:
        print(f"No se alcanzó la convergencia en {max_iter} iteraciones.")
    return x

tol = 1e-2
max_iter = 2

A_mod = np.array([[1, 0, -2], [-0.5, 1, -0.25], [1, -0.5, 1]], dtype=float)
b_mod = np.array([0.2, -1.425, 2], dtype=float)
x0_mod = np.zeros((3,), dtype=float)

solution_mod = gauss_seidel(A=A_mod, b=b_mod, x0=x0_mod, tol=tol, max_iter=m
print("Solución del sistema lineal  método de Gauss-Seidel:\n", solution_moc
```

```
i= 0 x: [0. 0. 0.]
i= 1 x: [ 0.2    -1.325   1.1375]
i= 2 x: [ 2.475      0.096875  -0.4265625]
No se alcanzó la convergencia en 2 iteraciones.
Solución del sistema lineal  método de Gauss-Seidel:
 [ 2.475      0.096875  -0.4265625]
```

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

In [45]:
```python
def gauss_jacobi(*, A: np.array, b: np.array, x0: np.array, tol: float, max_

    # --- Validación de los argumentos de la función ---
    if not isinstance(A, np.ndarray):
        A = np.array(A, dtype=float)
    assert A.shape[0] == A.shape[1], "La matriz A debe ser de tamaño n-by-(r
```

```
    if not isinstance(b, np.ndarray):
        b = np.array(b, dtype=float)
    assert b.shape[0] == A.shape[0], "El vector b debe ser de tamaño n."

    if not isinstance(x0, np.ndarray):
        x0 = np.array(x0, dtype=float)
    assert x0.shape[0] == A.shape[0], "El vector x0 debe ser de tamaño n."

    # --- Algoritmo ---
    n = A.shape[0]
    x = x0.copy()
    print(f"i= {0} x: {x.T}")
    for k in range(1, max_iter + 1):
        x_new = np.zeros_like(x0)  # prealloc
        for i in range(n):
            suma = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - suma) / A[i, i]
        print(f"i= {k} x: {x_new.T}")
        if np.linalg.norm(x_new - x) < tol:
            print(f"Convergencia alcanzada en {k} iteraciones.")
            return x_new
        x = x_new.copy()
    else:
        print(f"No se alcanzó la convergencia en {max_iter} iteraciones.")
    return x

# Definimos la tolerancia y el número máximo de iteraciones
tol = 1e-2
max_iter = 300

# Sistema lineal del ejercicio 6 literal b
A = np.array([[1, 0, -1], [-0.5, 1, -0.25], [1, -0.5, 1]], dtype=float)
b = np.array([0.2, -1.425, 2], dtype=float)
x0 = np.zeros((3,), dtype=float)

# Resultado
solution = gauss_jacobi(A=A, b=b, x0=x0, tol=tol, max_iter=max_iter)
print("Solución del sistema lineal método de Jacobi:\n", solution)
```

```
i= 0 x: [0. 0. 0.]
i= 1 x: [ 0.2   -1.425  2.   ]
i= 2 x: [ 2.2    -0.825   1.0875]
i= 3 x: [ 1.2875   -0.053125 -0.6125  ]
i= 4 x: [-0.4125    -0.934375   0.6859375]
i= 5 x: [ 0.8859375  -1.45976563  1.9453125 ]
i= 6 x: [ 2.1453125  -0.49570312  0.38417969]
i= 7 x: [ 0.58417969 -0.25629883 -0.39316406]
i= 8 x: [-0.19316406 -1.23120117  1.2876709 ]
i= 9 x: [ 1.4876709  -1.19966431  1.57756348]
i= 10 x: [ 1.77756348 -0.28677368 -0.08750305]
i= 11 x: [ 0.11249695 -0.55809402  0.07904968]
i= 12 x: [ 0.27904968 -1.34898911  1.60845604]
i= 13 x: [ 1.80845604 -0.88336115  1.04645576]
i= 14 x: [ 1.24645576 -0.25915804 -0.25013661]
i= 15 x: [-0.05013661 -0.86430627  0.62396522]
i= 16 x: [ 0.82396522 -1.294077    1.61798348]
i= 17 x: [ 1.81798348 -0.60852152  0.52899628]
i= 18 x: [ 0.72899628 -0.38375919 -0.12224424]
i= 19 x: [ 0.07775576 -1.09106292  1.07912412]
i= 20 x: [ 1.27912412 -1.11634109  1.37671278]
i= 21 x: [ 1.57671278 -0.44125974  0.16270533]
i= 22 x: [ 0.36270533 -0.59596728  0.20265735]
i= 23 x: [ 0.40265735 -1.192983    1.33931103]
i= 24 x: [ 1.53931103 -0.88884357  1.00085115]
i= 25 x: [ 1.20085115 -0.4051317   0.01626719]
i= 26 x: [ 0.21626719 -0.82050763  0.596583  ]
i= 27 x: [ 0.796583   -1.16772066  1.373479  ]
i= 28 x: [ 1.573479   -0.68333875  0.61955667]
i= 29 x: [ 0.81955667 -0.48337133  0.08485162]
i= 30 x: [ 0.28485162 -0.99400876  0.93875766]
i= 31 x: [ 1.13875766 -1.04788477  1.218144  ]
i= 32 x: [ 1.418144   -0.55108517  0.33729995]
i= 33 x: [ 0.53729995 -0.63160301  0.30631342]
i= 34 x: [ 0.50631342 -1.07977167  1.14689854]
i= 35 x: [ 1.34689854 -0.88511866  0.95380075]
i= 36 x: [ 1.15380075 -0.51310054  0.21054213]
i= 37 x: [ 0.41054213 -0.79546409  0.58964898]
i= 38 x: [ 0.78964898 -1.07231669  1.19172582]
i= 39 x: [ 1.39172582 -0.73224405  0.67419268]
i= 40 x: [ 0.87419268 -0.56058892  0.24215215]
i= 41 x: [ 0.44215215 -0.92736562  0.84551286]
i= 42 x: [ 1.04551286 -0.99254571  1.09416504]
i= 43 x: [ 1.29416504 -0.62870231  0.45821428]
i= 44 x: [ 0.65821428 -0.66336391  0.39148381]
i= 45 x: [ 0.59148381 -0.99802191  1.01010376]
i= 46 x: [ 1.21010376 -0.87673215  0.90950524]
i= 47 x: [ 1.10950524 -0.59257181  0.35153016]
i= 48 x: [ 0.55153016 -0.78236484  0.59420886]
i= 49 x: [ 0.79420886 -1.00068271  1.05728742]
i= 50 x: [ 1.25728742 -0.76357372  0.70544979]
i= 51 x: [ 0.90544979 -0.61999384  0.36092572]
i= 52 x: [ 0.56092572 -0.88204367  0.78455329]
i= 53 x: [ 0.98455329 -0.94839882  0.99805244]
i= 54 x: [ 1.19805244 -0.68321025  0.5412473 ]
i= 55 x: [ 0.7412473  -0.69066195  0.46034244]
```

```
i= 56 x: [ 0.66034244 -0.93929074  0.91342172]
i= 57 x: [ 1.11342172 -0.86647335  0.87001219]
i= 58 x: [ 1.07001219 -0.65078609  0.4533416 ]
i= 59 x: [ 0.6533416  -0.7766585   0.60459476]
i= 60 x: [ 0.80459476 -0.94718051  0.95832914]
i= 61 x: [ 1.15832914 -0.78312033  0.72181498]
i= 62 x: [ 0.92181498 -0.66538168  0.45011069]
i= 63 x: [ 0.65011069 -0.85156484  0.74549417]
i= 64 x: [ 0.94549417 -0.91357111  0.92410689]
i= 65 x: [ 1.12410689 -0.72122619  0.59772027]
i= 66 x: [ 0.79772027 -0.71351649  0.51528001]
i= 67 x: [ 0.71528001 -0.89731986  0.84552149]
i= 68 x: [ 1.04552149 -0.85597962  0.83606006]
i= 69 x: [ 1.03606006 -0.69322424  0.5264887 ]
i= 70 x: [ 0.7264887  -0.7753478   0.61732782]
i= 71 x: [ 0.81732782 -0.90742369  0.8858374 ]
i= 72 x: [ 1.0858374  -0.79487674  0.72896033]
i= 73 x: [ 0.92896033 -0.69984122  0.51672423]
i= 74 x: [ 0.71672423 -0.83133878  0.72111906]
i= 75 x: [ 0.92111906 -0.88635812  0.86760638]
i= 76 x: [ 1.06760638 -0.74753887  0.63570188]
i= 77 x: [ 0.83570188 -0.73227134  0.55862418]
i= 78 x: [ 0.75862418 -0.86749301  0.79816245]
i= 79 x: [ 0.99816245 -0.8461473   0.80762931]
i= 80 x: [ 1.00762931 -0.72401145  0.5787639 ]
i= 81 x: [ 0.7787639  -0.77649437  0.63036497]
i= 82 x: [ 0.83036497 -0.87802681  0.83298891]
i= 83 x: [ 1.03298891 -0.80157029  0.73062163]
i= 84 x: [ 0.93062163 -0.72585014  0.56622594]
i= 85 x: [ 0.76622594 -0.8181327   0.7064533 ]
i= 86 x: [ 0.9064533  -0.8652737   0.82470771]
i= 87 x: [ 1.02470771 -0.76559642  0.66090985]
i= 88 x: [ 0.86090985 -0.74741868  0.59249408]
i= 89 x: [ 0.79249408 -0.84642156  0.76538081]
i= 90 x: [ 0.96538081 -0.83740776  0.78429514]
i= 91 x: [ 0.98429514 -0.74623581  0.61591531]
i= 92 x: [ 0.81591531 -0.7788736   0.64258695]
i= 93 x: [ 0.84258695 -0.85639561  0.79464789]
i= 94 x: [ 0.99464789 -0.80504455  0.72921524]
i= 95 x: [ 0.92921524 -0.74537224  0.60282984]
i= 96 x: [ 0.80282984 -0.80968492  0.69809864]
i= 97 x: [ 0.89809864 -0.84906042  0.7923277 ]
i= 98 x: [ 0.9923277  -0.77786876  0.67737115]
i= 99 x: [ 0.87737115 -0.75949336  0.61873792]
i= 100 x: [ 0.81873792 -0.83162994  0.74288217]
i= 101 x: [ 0.94288217 -0.8299105   0.76544711]
i= 102 x: [ 0.96544711 -0.76219714  0.64216258]
i= 103 x: [ 0.84216258 -0.7817358   0.65345432]
i= 104 x: [ 0.85345432 -0.84055513  0.76696952]
i= 105 x: [ 0.96696952 -0.80653046  0.72626812]
i= 106 x: [ 0.92626812 -0.75994821  0.62976525]
i= 107 x: [ 0.82976525 -0.80442463  0.69375778]
i= 108 x: [ 0.89375778 -0.83667793  0.76802243]
i= 109 x: [ 0.96802243 -0.7861155   0.68790326]
i= 110 x: [ 0.88790326 -0.76901297  0.63891982]
i= 111 x: [ 0.83891982 -0.82131842  0.72759026]
```

```
i= 112 x: [ 0.92759026 -0.82364253  0.75042098]
i= 113 x: [ 0.95042098 -0.77359963  0.66058848]
i= 114 x: [ 0.86058848 -0.78464239  0.66277921]
i= 115 x: [ 0.86277921 -0.82901096  0.74709033]
i= 116 x: [ 0.94709033 -0.80683781  0.72271531]
i= 117 x: [ 0.92271531 -0.77077601  0.64949077]
i= 118 x: [ 0.84949077 -0.80126965  0.69189669]
i= 119 x: [ 0.89189669 -0.82728045  0.74987441]
i= 120 x: [ 0.94987441 -0.79158306  0.69446309]
i= 121 x: [ 0.89446309 -0.77644702  0.65433407]
i= 122 x: [ 0.85433407 -0.81418494  0.7173134 ]
i= 123 x: [ 0.9173134  -0.81850462  0.73857347]
i= 124 x: [ 0.93857347 -0.78169994  0.6734343 ]
i= 125 x: [ 0.8734343  -0.78735469  0.67057657]
i= 126 x: [ 0.87057657 -0.82063871  0.73288836]
i= 127 x: [ 0.93288836 -0.80648963  0.71910408]
i= 128 x: [ 0.91910408 -0.7787798   0.66386683]
i= 129 x: [ 0.86386683 -0.79948125  0.69150602]
i= 130 x: [ 0.89150602 -0.82019008  0.73639254]
i= 131 x: [ 0.93639254 -0.79514885  0.69839894]
i= 132 x: [ 0.89839894 -0.78220399  0.66603303]
i= 133 x: [ 0.86603303 -0.80929227  0.71049906]
i= 134 x: [ 0.91049906 -0.81435872  0.72932083]
i= 135 x: [ 0.92932083 -0.78742026  0.68232158]
i= 136 x: [ 0.88232158 -0.78975919  0.67696904]
i= 137 x: [ 0.87696904 -0.81459695  0.72279883]
i= 138 x: [ 0.92279883 -0.80581577  0.71573249]
i= 139 x: [ 0.91573249 -0.78466746  0.67429328]
i= 140 x: [ 0.87429328 -0.79856044  0.69193378]
i= 141 x: [ 0.89193378 -0.81486991  0.7264265 ]
i= 142 x: [ 0.9264265  -0.79742648  0.70063126]
i= 143 x: [ 0.90063126 -0.78662893  0.67486026]
i= 144 x: [ 0.87486026 -0.8059693   0.70605427]
i= 145 x: [ 0.90605427 -0.8110563   0.72215509]
i= 146 x: [ 0.92215509 -0.79143409  0.68841758]
i= 147 x: [ 0.88841758 -0.79181806  0.68212786]
i= 148 x: [ 0.88212786 -0.81025925  0.71567339]
i= 149 x: [ 0.91567339 -0.80501772  0.71274251]
i= 150 x: [ 0.91274251 -0.78897768  0.68181775]
i= 151 x: [ 0.88181775 -0.79817431  0.69276865]
i= 152 x: [ 0.89276865 -0.81089896  0.7190951 ]
i= 153 x: [ 0.9190951  -0.7988419   0.70178187]
i= 154 x: [ 0.90178187 -0.79000698  0.68148395]
i= 155 x: [ 0.88148395 -0.80373808  0.70321464]
i= 156 x: [ 0.90321464 -0.80845436  0.71664701]
i= 157 x: [ 0.91664701 -0.79423093  0.69255818]
i= 158 x: [ 0.89255818 -0.79353695  0.68623753]
i= 159 x: [ 0.88623753 -0.80716153  0.71067335]
i= 160 x: [ 0.91067335 -0.8042129   0.71018171]
i= 161 x: [ 0.91018171 -0.7921179   0.6872202 ]
i= 162 x: [ 0.8872202  -0.79810409  0.69375934]
i= 163 x: [ 0.89375934 -0.80795006  0.71372775]
i= 164 x: [ 0.91372775 -0.79968839  0.70226563]
i= 165 x: [ 0.90226563 -0.79256972  0.68642805]
i= 166 x: [ 0.88642805 -0.80226017  0.70144951]
i= 167 x: [ 0.90144951 -0.80642359  0.71244186]
```

```
i= 168 x: [ 0.91244186 -0.79616478  0.69533869]
i= 169 x: [ 0.89533869 -0.7949444   0.68947575]
i= 170 x: [ 0.88947575 -0.80496172  0.70718911]
i= 171 x: [ 0.90718911 -0.80346485  0.70804339]
i= 172 x: [ 0.90804339 -0.7943946   0.69107847]
i= 173 x: [ 0.89107847 -0.79820869  0.69475931]
i= 174 x: [ 0.89475931 -0.80577094  0.70981719]
i= 175 x: [ 0.90981719 -0.80016605  0.70235522]
i= 176 x: [ 0.90235522 -0.7945026   0.69009979]
i= 177 x: [ 0.89009979 -0.80129744  0.70039348]
i= 178 x: [ 0.90039348 -0.80485174  0.70925149]
i= 179 x: [ 0.90925149 -0.79749039  0.69718065]
i= 180 x: [ 0.89718065 -0.79607909  0.69200332]
i= 181 x: [ 0.89200332 -0.80340885  0.7047798 ]
i= 182 x: [ 0.9047798  -0.80280339  0.70629226]
i= 183 x: [ 0.90629226 -0.79603703  0.6938185 ]
i= 184 x: [ 0.8938185  -0.79839924  0.69568922]
i= 185 x: [ 0.89568922 -0.80416844  0.70698188]
i= 186 x: [ 0.90698188 -0.80040992  0.70222656]
i= 187 x: [ 0.90222656 -0.79595242  0.69281316]
i= 188 x: [ 0.89281316 -0.80068343  0.69979723]
i= 189 x: [ 0.89979723 -0.80364411  0.70684512]
i= 190 x: [ 0.90684512 -0.7983901   0.69838071]
i= 191 x: [ 0.89838071 -0.79698226  0.69395983]
Convergencia alcanzada en 191 iteraciones.
Solución del sistema lineal método de Jacobi:
 [ 0.89838071 -0.79698226  0.69395983]
```

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.

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

In [46]:
```python
#### a. ¿La matriz es estrictamente diagonalmente dominante?
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, -1, 0, 0, 0, 0, -1, 4]
], dtype=float)

# Verificamos si la matriz es diagonalmente dominante
def es_diagonalmente_dominante(A):
    n = A.shape[0]
    for i in range(n):
        suma = sum(abs(A[i, j]) for j in range(n) if j != i)
        if abs(A[i, i]) <= suma:
            return False
    return True

print("La matriz es diagonalmente dominante:", es_diagonalmente_dominante(A))
```

```
La matriz es diagonalmente dominante: True
```

In [47]:

```python
#### b. Resuelva el sistema lineal usando el método de Jacobi con $x_{(0)} =

def gauss_jacobi(*, A: np.array, b: np.array, x0: np.array, tol: float, max_
    if not isinstance(A, np.ndarray):
        A = np.array(A, dtype=float)
    assert A.shape[0] == A.shape[1], "La matriz A debe ser de tamaño n-by-(r

    if not isinstance(b, np.ndarray):
        b = np.array(b, dtype=float)
    assert b.shape[0] == A.shape[0], "El vector b debe ser de tamaño n."

    if not isinstance(x0, np.ndarray):
        x0 = np.array(x0, dtype=float)
    assert x0.shape[0] == A.shape[0], "El vector x0 debe ser de tamaño n."

    n = A.shape[0]
    x = x0.copy()
    print(f"i= {0} x: {x.T}")
    for k in range(1, max_iter + 1):
        x_new = np.zeros_like(x0)
        for i in range(n):
            suma = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - suma) / A[i, i]
        print(f"i= {k} x: {x_new.T}")
        if np.linalg.norm(x_new - x) < tol:
            print(f"Convergencia alcanzada en {k} iteraciones.")
            return x_new
        x = x_new.copy()
    else:
        print(f"No se alcanzó la convergencia en {max_iter} iteraciones.")
    return x

# Definimos la tolerancia y el número máximo de iteraciones
tol = 1e-2
max_iter = 8

# Matriz
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, -1, 0, 0, 0, 0, -1, 4]
], dtype=float)

# Sistema lineal literal b
b = np.array([220, 110, 110, 220, 110, 110, 110, 110, 220, 110, 110, 220], c
```

```
x0 = np.zeros(12, dtype=float)

# Resultado
solution = gauss_jacobi(A=A, b=b, x0=x0, tol=tol, max_iter=max_iter)
print("Solución del sistema lineal método de Jacobi:\n", solution)
```

```
i= 0 x: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
i= 1 x: [55.  27.5 27.5 55.  27.5 27.5 27.5 27.5 55.  27.5 27.5 55. ]
i= 2 x: [68.75  48.125 48.125 68.75  48.125 48.125 48.125 48.125 68.75  48.1
25
 48.125 68.75 ]
i= 3 x: [79.0625  56.71875 56.71875 79.0625  56.71875 56.71875 56.71875 56.7
1875
 79.0625  56.71875 56.71875 79.0625 ]
i= 4 x: [83.359375  61.4453125 61.4453125 83.359375  61.4453125 61.4453125
 61.4453125 61.4453125 83.359375  61.4453125 61.4453125 83.359375 ]
i= 5 x: [85.72265625 63.70117188 63.70117188 85.72265625 63.70117188 63.7011
7188
 63.70117188 63.70117188 85.72265625 63.70117188 63.70117188 85.72265625]
i= 6 x: [86.85058594 64.85595703 64.85595703 86.85058594 64.85595703 64.8559
5703
 64.85595703 64.85595703 86.85058594 64.85595703 64.85595703 86.85058594]
i= 7 x: [87.42797852 65.42663574 65.42663574 87.42797852 65.42663574 65.4266
3574
 65.42663574 65.42663574 87.42797852 65.42663574 65.42663574 87.42797852]
i= 8 x: [87.71331787 65.71365356 65.71365356 87.71331787 65.71365356 65.7136
5356
 65.71365356 65.71365356 87.71331787 65.71365356 65.71365356 87.71331787]
No se alcanzó la convergencia en 8 iteraciones.
Solución del sistema lineal método de Jacobi:
 [87.71331787 65.71365356 65.71365356 87.71331787 65.71365356 65.71365356
 65.71365356 65.71365356 87.71331787 65.71365356 65.71365356 87.71331787]
```

In [48]:

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

def gauss_seidel(*, A: np.array, b: np.array, x0: np.array, tol: float, max_
    if not isinstance(A, np.ndarray):
        A = np.array(A, dtype=float)
    assert A.shape[0] == A.shape[1], "La matriz A debe ser de tamaño n-by-(r

    if not isinstance(b, np.ndarray):
        b = np.array(b, dtype=float)
    assert b.shape[0] == A.shape[0], "El vector b debe ser de tamaño n."

    if not isinstance(x0, np.ndarray):
        x0 = np.array(x0, dtype=float)
    assert x0.shape[0] == A.shape[0], "El vector x0 debe ser de tamaño n."

    n = A.shape[0]
    x = x0.copy()
    print(f"i= {0} x: {x.T}")
    for k in range(1, max_iter + 1):
        for i in range(n):
            suma = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - suma) / A[i, i]
        print(f"i= {k} x: {x.T}")
```

```
        if np.linalg.norm(x - x0) < tol:
            print(f"Convergencia alcanzada en {k} iteraciones.")
            break
        x0 = x.copy()
    else:
        print(f"No se alcanzó la convergencia en {max_iter} iteraciones.")
    return x

# Definimos la tolerancia y el número máximo de iteraciones
tol = 1e-2
max_iter = 15

# Matriz
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, -1, 0, 0, 0, 0, -1, 4]
], dtype=float)

# Sistema lineal literal c
b = np.array([220, 110, 110, 220, 110, 110, 110, 110, 220, 110, 110, 220], d
x0 = np.zeros(12, dtype=float)

# Resultado
solution = gauss_seidel(A=A, b=b, x0=x0, tol=tol, max_iter=max_iter)
print("Solución del sistema lineal método de Gauss-Seidel:\n", solution)
```

```
i= 0 x: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
i= 1 x: [55.         41.25       37.8125     64.453125   41.25       43.6132
8125
 37.8125     38.40332031 64.453125   43.61328125 38.40332031 75.50415039]
i= 2 x: [75.625      55.859375   57.578125   80.29785156 55.859375   57.1752
9297
 57.578125   60.66986084 80.29785156 57.17529297 60.66986084 84.46128845]
i= 3 x: [82.9296875  62.62695312 63.23120117 85.10162354 62.62695312 63.9428
7109
 63.23120117 64.60103989 85.10162354 63.94287109 64.60103989 87.13597775]
i= 4 x: [86.31347656 64.88616943 64.99694824 87.23495483 64.88616943 65.4589
9868
 64.99694824 65.64874411 87.23495483 65.45899868 65.64874411 87.7769357 ]
i= 5 x: [87.44308472 65.61000824 65.71124077 87.79255986 65.61000824 65.8603
2599
 65.71124077 65.90931542 87.79255986 65.86032599 65.90931542 87.94241035]
i= 6 x: [87.80500412 65.87906122 65.91790527 87.94455782 65.87906122 65.9634
6831
 65.91790527 65.97646967 87.94455782 65.96346831 65.97646967 87.98498449]
i= 7 x: [87.93953061 65.96435897 65.9772292  87.98517438 65.96435897 65.9904
1101
 65.9772292  65.99384888 87.98517438 65.99041101 65.99384888 87.99606497]
i= 8 x: [87.98217949 65.98985217 65.99375664 87.99604191 65.98985217 65.9974
727
 65.99375664 65.99838442 87.99604191 65.9974727  65.99838442 87.99896428]
i= 9 x: [87.99492609 65.99717068 65.99830315 87.99894396 65.99717068 65.9993
3209
 65.99830315 65.99957409 87.99894396 65.99933209 65.99957409 87.99972655]
i= 10 x: [87.99858534 65.99922212 65.99954152 87.9997184  65.99922212 65.999
82312
 65.99954152 65.99988742 87.9997184  65.99982312 65.99988742 87.99992764]
Convergencia alcanzada en 10 iteraciones.
Solución del sistema lineal método de Gauss-Seidel:
 [87.99858534 65.99922212 65.99954152 87.9997184  65.99922212 65.99982312
 65.99954152 65.99988742 87.9997184  65.99982312 65.99988742 87.99992764]
```

link del repositorio de Git-Hub

https://github.com/JuanfranPinto/Metodos-Numericos-/blob/main/Tarea-11-Ejercicios-Unidad-04-D-Gauss-Jacobi-y-Gauss-Seidel-PJF.ipynb