Informe Tarea 11

Anthony Goyes

2024-07-28

Tabla de Contenidos

Conjunto de ejercicios tarea 11

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

Literal a

$$3x_1 - x_2 + x_3 = -1$$

$$3x_1 + 6x_2 + 2x_3 = 0$$

$$3x_1 + 3x_2 + 7x_3 = 4$$

```
import numpy as np
def gauss_jacobi(
    A: np.array, b: np.array, x0: np.array, tol: float, max_iter: int
) -> np.array:
    # --- 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-(n)."

    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()
    for k in range(1, max_iter):
        x_new = np.zeros((n, 1))  # 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]

        if np.linalg.norm(x_new - x) < tol:
            return x_new

        x = x_new.copy()

    return x
```

```
A = [
    [3, -1, 1],
    [3, 6, 2],
    [3, 3, 7]
]

b = [1, 0, 4]

x = gauss_jacobi(A, b, [0, 0, 0], 10e-5, 100)
print(x)
```

```
[[ 0.0350863 ]
 [-0.23685698]
 [ 0.65787809]]
```

Literal b

$$10x_1 - x_2 = 9$$

$$-x_1 + 10x_2 - 2x_3 = 7$$

$$2x_2 + 10x_3 = 6$$

```
A = [
    [10, -1, 0],
    [-1, 10, -2],
    [0, -2, 10]
]

b = [9, 7, 6]

x = gauss_jacobi(A, b, [0, 0, 0], 10e-5, 100)
print(x)
```

```
[[0.99578625]
 [0.95788875]
 [0.7915725 ]]
```

Literal c

$$10x_1 + 5x_2 = 6$$

$$5x_1 + 10x_2 - 4x_3 = 25$$

$$-4x_2 + 8x_3 - x_4 = -11$$

$$-x_3 + 5x_4 = -11$$

```
A = [
    [10, 5, 0, 0],
    [5, 10, -4, 0],
    [0, -4, -8, 1],
    [0, 0, -1, 5]
]
```

```
b = [6, 25, -11, -11]

x = gauss_jacobi(A, b, [0, 0, 0, 0], 10e-5, 100)
print(x)
```

```
[[-0.78792172]
 [ 2.77583088]
 [-0.29530611]
 [-2.25906474]]
```

Literal d)

$$4x_1 + x_2 + x_3 + x_5 = 6$$

$$-x_1 - 3x_2 + x_3 + x_4 = 25$$

$$2x_1 + x_2 + 5x_3 - x_4 - x_5 = 6$$

$$-x_1 - x_2 - x_3 + 4x_4 = 6$$

$$2x_2 - x_3 + x_4 + 4x_5 = 6$$

```
A = [
    [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]
]

b = [6, 6, 6, 6, 6]

x = gauss_jacobi(A, b, [0, 0, 0, 0, 0], 10e-5, 100)
print(x)
```

```
[[ 0.78661584]
 [-1.00257369]
 [ 1.86634212]
 [ 1.91259293]
 [ 1.98974776]]
```

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

Literal a

$$3x_1 - x_2 + x_3 = -1$$

$$3x_1 + 6x_2 + 2x_3 = 0$$

$$3x_1 + 3x_2 + 7x_3 = 4$$

```python
def gauss_seidel(
    A: np.array, b: np.array, x0: np.array, tol: float, max_iter: int
) -> np.array:
    # --- 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-(n)."

    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()

    for k in range(1, max_iter):
        x_new = np.zeros((n, 1))  # prealloc
        for i in range(n):
            suma = sum([A[i, j] * x_new[j] for j in range(i) if j != i]) + sum(
```

```
                [A[i, j] * x[j] for j in range(i, n) if j != i]
            )
            x_new[i] = (b[i] - suma) / A[i, i]

        if np.linalg.norm(x_new - x) < tol:
            return x_new

        x = x_new.copy()

    return x

A = [
    [3, -1, 1],
    [3, 6, 2],
    [3, 3, 7]
]

b = [1, 0, 4]

x = gauss_seidel(A, b, [0, 0, 0], 10e-5, 100)
print(x)
```

```
[[ 0.03510326]
 [-0.23683891]
 [ 0.6578867 ]]
```

Literal b

$$10x_1 - x_2 = 9$$

$$-x_1 + 10x_2 - 2x_3 = 7$$

$$2x_2 + 10x_3 = 6$$

```
A = [
    [10, -1, 0],
    [-1, 10, -2],
    [0, -2, 10]
]
```

```
b = [9, 7, 6]

x = gauss_seidel(A, b, [0, 0, 0], 10e-5, 100)
print(x)
```

```
[[0.99578737]
 [0.95789369]
 [0.79157874]]
```

Literal c

$$10x_1 + 5x_2 = 6$$

$$5x_1 + 10x_2 - 4x_3 = 25$$

$$-4x_2 + 8x_3 - x_4 = -11$$

$$-x_3 + 5x_4 = -11$$

```
A = [
    [10, 5, 0, 0],
    [5, 10, -4, 0],
    [0, -4, -8, 1],
    [0, 0, -1, 5]
]

b = [6, 25, -11, -11]

x = gauss_seidel(A, b, [0, 0, 0, 0], 10e-5, 100)
print(x)
```

```
[[-0.78791707]
 [ 2.77583885]
 [-0.29530191]
 [-2.25906038]]
```

Literal d

$$4x_1 + x_2 + x_3 + x_5 = 6$$

$$-x_1 - 3x_2 + x_3 + x_4 = 25$$

$$2x_1 + x_2 + 5x_3 - x_4 - x_5 = 6$$

$$-x_1 - x_2 - x_3 + 4x_4 = 6$$

$$2x_2 - x_3 + x_4 + 4x_5 = 6$$

```
A = [
    [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]
]

b = [6, 6, 6, 6, 6]

x = gauss_seidel(A, b, [0, 0, 0, 0, 0], 10e-5, 100)
print(x)
```

```
[[ 0.78663577]
 [-1.00257108]
 [ 1.86632614]
 [ 1.91259771]
 [ 1.98971765]]
```

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

Literal a

```
A = [
    [3, -1, 1],
    [3, 6, 2],
    [3, 3, 7]
]

b = [1, 0, 4]

x = gauss_jacobi(A, b, [0, 0, 0], 10e-3, 100)
print(x)
```

```
[[ 0.03516089]
 [-0.23570619]
 [ 0.65922185]]
```

Literal b

```
A = [
    [10, -1, 1],
    [-1, 10, -2],
    [0, -2, 10]
]

b = [9, 7, 6]

x = gauss_jacobi(A, b, [0, 0, 0], 10e-3, 100)
print(x)
```

```
[[0.91603]
 [0.94913]
 [0.78962]]
```

Literal c

```
A = [
    [10, 5, 0, 0],
    [5, 10, -4, 0],
    [0, -4, -8, 1],
    [0, 0, -1, 5]
]

b = [6, 25, -11, -11]

x = gauss_jacobi(A, b, [0, 0, 0, 0], 10e-3, 100)
print(x)
```

```
[[-0.788375  ]
 [ 2.77715625]
 [-0.29553125]
 [-2.26032813]]
```

Literal d

```
A = [
    [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]
]

b = [6, 6, 6, 6, 6]

x = gauss_jacobi(A, b, [0, 0, 0, 0, 0], 10e-3, 100)
print(x)
```

```
[[ 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.

Literal a

```
A = [
    [3, -1, 1],
    [3, 6, 2],
    [3, 3, 7]
]

b = [1, 0, 4]

x = gauss_seidel(A, b, [0, 0, 0], 10e-3, 100)
print(x)
```

```
[[ 0.0361492 ]
 [-0.23660752]
 [ 0.65733928]]
```

Literal b

```
A = [
    [10, -1, 1],
    [-1, 10, -2],
    [0, -2, 10]
]

b = [9, 7, 6]

x = gauss_seidel(A, b, [0, 0, 0], 10e-3, 100)
print(x)
```

```
[[0.91593697]
 [0.94956218]
 [0.78991244]]
```

Literal c

```
A = [
    [10, 5, 0, 0],
    [5, 10, -4, 0],
    [0, -4, -8, 1],
    [0, 0, -1, 5]
]

b = [6, 25, -11, -11]

x = gauss_seidel(A, b, [0, 0, 0, 0], 10e-3, 100)
print(x)
```

```
[[-0.78802812]
 [ 2.77579328]
 [-0.29528544]
 [-2.25905709]]
```

Literal d

```
A = [
    [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]
]

b = [6, 6, 6, 6, 6]

x = gauss_seidel(A, b, [0, 0, 0, 0, 0], 10e-3, 100)
print(x)
```

```
[[ 0.78616258]
 [-1.00240703]
 [ 1.86606999]
 [ 1.91245638]
 [ 1.98960692]]
```

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 la 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.

```
A = [
    [2, -1, 1],
    [2, 2, 2],
    [-1, -1, 2]
]

b = [-1, 4, -5]

x = gauss_jacobi(A, b, [0, 0, 0], 10e-5, 25)
print(x)
```

```
[[ -7.73114914]
 [-32.92459655]
 [  7.73114914]]
```

b) Utilice el método de Gauss-Siedel con x(0) = 0:para aproximar la solución para el sistema lineal dentro de 10−5.

```
A = [
    [2, -1, 1],
    [2, 2, 2],
    [-1, -1, 2]
]

b = [-1, 4, -5]
```

```
x = gauss_seidel(A, b, [0, 0, 0], 10e-5, 25)
print(x)
```

```
[[ 0.99998474]
 [ 2.00001717]
 [-0.99999905]]
```

6. El sistema lineal

$$x_1 - x_3 = 0.2$$

$$-\frac{1}{2}x_1 + x_2 - \frac{1}{4}x_3 = -1.425$$

$$x_1 - \frac{1}{2}x_2 + 5x_3 = 2$$

tiene la solución (0.9, -0.8, 0.7).

a) ¿La matriz de coeficientes

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

tiene diagonal estrictamente dominante?

```
import numpy as np

def is_strictly_diagonally_dominant(matrix):
    """
    Verifica si una matriz es estrictamente diagonalmente dominante.
    :param matrix: Matriz cuadrada
    :return: True si la matriz es estrictamente diagonalmente dominante, False en caso contra
    """
    n = matrix.shape[0]
    for i in range(n):
        # Valor absoluto del elemento diagonal
        diag_element = abs(matrix[i, i])
```

```
        # Suma de los valores absolutos de los elementos no diagonales en la fila i
        off_diag_sum = np.sum(np.abs(matrix[i, :])) - diag_element

        # Verificación de la condición de diagonal estrictamente dominante
        if diag_element <= off_diag_sum:
            return False
    return True

# Matriz de coeficientes
A = np.array([
    [1, 0, -1],
    [-0.5, 1, -0.25],
    [1, -0.5, 1]
], dtype=np.float32)

# Verificar si la matriz A es estrictamente diagonalmente dominante
resultado = is_strictly_diagonally_dominant(A)
print("La matriz A es estrictamente diagonalmente dominante:", resultado)
```

```
La matriz A es estrictamente diagonalmente dominante: False
```

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

```
import numpy as np

def gauss_seidel(A: np.array, b: np.array, x0: np.array, tol: float,
                 max_iter: int) -> np.array:
    # --- 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-(n)."

    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()

    for k in range(1, max_iter):
        x_new = np.zeros((n,), dtype=np.float32)  # prealloc
        for i in range(n):
            suma = sum([A[i, j] * x_new[j] for j in range(i) if j != i]) + sum(
                [A[i, j] * x[j] for j in range(i, n) if j != i]
            )
            x_new[i] = (b[i] - suma) / A[i, i]

        if np.linalg.norm(x_new - x) < tol:
            return x_new

        x = x_new.copy()

    return x

# Datos del sistema
A = np.array([
    [1, 0, -1],
    [-0.5, 1, -0.25],
    [1, -0.5, 1]
], dtype=np.float32)

b = np.array([0.2, -1.425, 2], dtype=np.float32)
x0 = np.zeros_like(b, dtype=np.float32)
tol = 1e22
max_iter = 300

# Resolver el sistema usando el método de Gauss-Seidel
solucion = gauss_seidel(A, b, x0, tol, max_iter)

print("La solución aproximada es:", solucion)
```

```
La solución aproximada es: [ 0.2       -1.3249999  1.1375   ]
```

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

```
import numpy as np
def gauss_seidel(A: np.array, b: np.array, x0: np.array, tol: float,
                 max_iter: int) -> np.array:
    # --- 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-(n)."

    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()

    for k in range(1, max_iter):
        x_new = np.zeros((n,), dtype=np.float32)  # prealloc
        for i in range(n):
            suma = sum([A[i, j] * x_new[j] for j in range(i) if j != i]) + sum(
                [A[i, j] * x[j] for j in range(i, n) if j != i]
            )
            x_new[i] = (b[i] - suma) / A[i, i]

        if np.linalg.norm(x_new - x) < tol:
            return x_new

        x = x_new.copy()

    return x

# Datos del sistema
A = np.array([
    [1, 0, -2],
    [-0.5, 1, -0.25],
    [1, -0.5, 1]
```

```
], dtype=np.float32)

b = np.array([0.2, -1.425, 2], dtype=np.float32)
x0 = np.zeros_like(b, dtype=np.float32)
tol = 1e22
max_iter = 300

# Resolver el sistema usando el método de Gauss-Seidel
solucion = gauss_seidel(A, b, x0, tol, max_iter)

print("La solución aproximada es:", solucion)
```

```
La solución aproximada es: [ 0.2       -1.3249999  1.1375   ]
```

Se puede observar que el resultado no cambió en comparación al literal b.

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

No encontré ningún ejercicio 11.

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.

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

Figura 1: Matriz

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.

a. ¿La matriz es estrictamente diagonalmente dominante?

Cada elemento en la diagonal es 4, y la suma de los valores absolutos de los elementos no diagonales en cualquier fila es siempre 2. Esto implica que la matriz es estrictamente diagonalmente dominante.

b. Resuelva el sistema lineal usando el método de Jacobi con x(0) = 0 y TOL = 10-2.

```
A = [
    [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]
]

b = [220, 110, 110, 220, 110, 110, 110, 110, 220, 110, 110, 220]

x = gauss_jacobi(A, b, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 10e-2, 100)
print(x)
```

```
[[87.98209548]
 [65.98209679]
 [65.98209679]
 [87.98209548]
 [65.98209679]
 [65.98209679]
 [65.98209679]
 [65.98209679]
 [87.98209548]
 [65.98209679]
 [65.98209679]
 [87.98209548]]
```

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

```
A = [
    [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]
]

b = [220, 110, 110, 220, 110, 110, 110, 110, 220, 110, 110, 220]

x = gauss_seidel(A, b, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 10e-2, 100)
print(x)
```

```
[87.98218  65.98985  65.99376  87.99604  65.98985  65.997475 65.99376
 65.99838  87.99604  65.997475 65.99838  87.99896 ]
```

Enlace al repositorio de github

https://github.com/anthonypgq/Tareas9-10-11_MetodosNumericos