Métodos Numericos

Gauss-Jacobi y Gauss-Seidel

José Sarango

Tabla de Contenidos

Conjunto de ejercicios **2**

1. Encuentre las primeras dos iteraciones del metodo de Jacobi para los siguientes sistemas lineales,por medio de $x^0 = 0$: 2
   a) 2
   b) 3
   c) 3
   d) 3
2. Repita el ejercicio 1 usando el metodo de Gauus-Siedel. 4
3. Utilice el metodo de Jacobi para resolver los sistemas iniciales en el ejercicio 1, con TOL=10^{-3} 6
4. Utilice el método de Gauss-Siedel para resolver los sistemas lineales en el ejercicio 1, con TOL=10^{-3} 7
5. El sistema lineal : 8
   a) Muestre que el metodo de Jacobi con $x_0 = 0$ falla al proporcionar una buena aproximacion despues de 25 iteraciones. 9
   b) Utilice el metodo de Gauus-Siedel con $x_0 = 0$, para aproximar la solucion para el sistema lineal dentro de 10^{-5}. 10
6. El sistema lineal : 11
   a) ¿La matriz de coeficientes 11
   b) Utilice el metodo iterativo de Gauss-Siedel para aproximar la solucion para el sistema lineal con una tolerancia de $10_2 2$ y un maximo de 300 iteraciones. 12
   c) ¿Que pasa en la parte b) cuando el sistema cambia por el siguiente? 13
7. Repita el ejercicio 6 usando el metodo de Jacobi. 13
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. . . 15
   a) ¿La matriz es estrictamente diagonalmente dominante? 15
   b) Resuelva el sistema lineal usando el metodo de Jacobi con $x_0 = 0 y TOL = 10^{-2}$ 17
   c) Repita la parte b) mediante el metodo de Gauss-Siedel. 18

Conjunto de ejercicios

1. Encuentre las primeras dos iteraciones del metodo de Jacobi para los siguientes sistemas lineales,por medio de $x^0 = 0$:

```
import numpy as np
def jacobi_method(A, b, x0, iterations):
    n = len(b)
    x = x0.copy()
    x_new = np.zeros_like(x)
    iter_results = []
    for k in range(iterations):
        for i in range(n):
            sum_ = sum(A[i][j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - sum_) / A[i][i]
        iter_results.append(x_new.copy())
        x = x_new.copy()

    return iter_results
```

a)

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

```
A = np.array([[3, -1, 1],[3, 6, 2],[3, 3, 7]])
b = np.array([1, 0, 4])
x0 = np.zeros(len(b))
iterations = 2
results = jacobi_method(A, b, x0, iterations)
for i, res in enumerate(results):
    print(f"Iteración {i+1}: {res}")
```

```
Iteración 1: [0.33333333 0.         0.57142857]
Iteración 2: [ 0.14285714 -0.35714286  0.42857143]
```

b)

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

```
A = np.array([[10, -1, 0],[-1, 10, -2],[0, -2, 10]])
b = np.array([9, 7, 6])
x0 = np.zeros(len(b))
iterations = 2
results = jacobi_method(A, b, x0, iterations)
for i, res in enumerate(results):
    print(f"Iteración {i+1}: {res}")
```

```
Iteración 1: [0.9 0.7 0.6]
Iteración 2: [0.97 0.91 0.74]
```

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 = 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(len(b))
iterations = 2
results = jacobi_method(A, b, x0, iterations)
for i, res in enumerate(results):
    print(f"Iteración {i+1}: {res}")
```

```
Iteración 1: [ 0.6    2.5   -1.375 -2.2  ]
Iteración 2: [-0.65   1.65  -0.4   -2.475]
```

d)

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

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

```
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]])
b = np.array([6,6, 6,6,6])
x0 = np.zeros(len(b))
iterations = 2
results = jacobi_method(A, b, x0, iterations)
for i, res in enumerate(results):
    print(f"Iteración {i+1}: {res}")
```

```
Iteración 1: [ 1.5 -2.   1.2  1.5  1.5]
Iteración 2: [ 1.325 -1.6    1.6    1.675  2.425]
```

2. Repita el ejercicio 1 usando el metodo de Gauus-Siedel.

```
import numpy as np
def gauss_seidel_method(A, b, x0, iterations):
    n = len(b)
    x = x0.copy()
    iter_results = []
    for k in range(iterations):
        for i in range(n):
            sum_ = sum(A[i][j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - sum_) / A[i][i]
        iter_results.append(x.copy())
    return iter_results
```

```
# Sistema de ecuaciones "a"
A_a = np.array([[3, -1, 1],[3, 6, 2],[3, 3, 7]])
b_a = np.array([1, 0, 4])
# Sistema de ecuaciones "b"
A_b = np.array([[10, -1, 0],[-1, 10, -2],[0, -2, 10]])
b_b = np.array([9, 7, 6])
# Sistema de ecuaciones "c"
A_c = np.array([[10, 5, 0,0],[5, 10, -4,0],[0, -4, 8,-1],[0,0,-1,5]])
b_c = np.array([6, 25, -11,-11])
# Sistema de ecuaciones "d"
A_d = 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]])
b_d = np.array([6,6, 6,6,6])
```

```python
sistemas = {
    'a': (A_a, b_a),
    'b': (A_b, b_b),
    'c': (A_c, b_c),
    'd': (A_d, b_d),
}
iterations = 2
for literal, (A, b) in sistemas.items():
    x0 = np.zeros(len(b))
    print(f"Sistema {literal}:")

    results = gauss_seidel_method(A, b, x0, iterations)

    for i, res in enumerate(results):
        print(f"  Iteración {i+1}: {res}")

    print("\n")
```

```
Sistema a:
  Iteración 1: [ 0.33333333 -0.16666667  0.5       ]
  Iteración 2: [ 0.11111111 -0.22222222  0.61904762]


Sistema b:
  Iteración 1: [0.9   0.79  0.758]
  Iteración 2: [0.979  0.9495 0.7899]


Sistema c:
  Iteración 1: [ 0.6    2.2   -0.275 -2.255]
  Iteración 2: [-0.5       2.64     -0.336875 -2.267375]


Sistema d:
  Iteración 1: [ 1.5     -2.5      1.1      1.525    2.64375]
  Iteración 2: [ 1.1890625  -1.52135417  1.86239583  1.88252604  2.25564453]
```

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

```
import numpy as np

def jacobi_method(A, b, tol, max_iterations):
    n = len(b)
    x = np.zeros(n)
    x_new = np.zeros(n)
    final_x = None
    final_error = None

    for k in range(max_iterations):
        for i in range(n):
            sum_ = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - sum_) / A[i, i]
        error = np.linalg.norm(x_new - x, ord=np.inf)
        final_x = np.copy(x_new)
        final_error = error

        if error < tol:
            break

        x = np.copy(x_new)

    return final_x, k + 1, final_error

tol = 1e-3
max_iterations = 100
sistemas = {
    'a': (np.array([[3, -1, 1], [3, 6, 2], [3, 3, 7]]), np.array([1, 0, 4])),
    'b': (np.array([[10, -1, 0], [-1, 10, -2], [0, -2, 10]]), np.array([9, 7, 6])),
    'c': (np.array([[10, 5, 0, 0], [5, 10, -4, 0], [0, -4, 8, -1], [0, 0, -1, 5]]), np.array
    'd': (np.array([[4, 1, 1, 0, 1], [-1, -3, 1, 1, 0], [2, 1, 5, -1, -1], [0, -1, 0, 4, 0],
}
for literal, (A, b) in sistemas.items():
    print(f"\nSistema {literal.upper()}:")
    solution, iterations, final_error = jacobi_method(A, b, tol, max_iterations)
    print(f"Solución aproximada después de {iterations} iteraciones: {solution}")
    print(f"Error final: {final_error}\n")
```

```
Sistema A:
Solución aproximada después de 9 iteraciones: [ 0.03510079 -0.23663751  0.65812732]
Error final: 0.0007507619179839553


Sistema B:
Solución aproximada después de 6 iteraciones: [0.995725 0.957775 0.79145 ]
Error final: 0.0005250000000001087


Sistema C:
Solución aproximada después de 21 iteraciones: [-0.79710581  2.79517067 -0.25939578 -2.25179
Error final: 0.0009590483248249626


Sistema D:
Solución aproximada después de 12 iteraciones: [ 0.78351258 -1.25474462  1.8323198   1.18647
Error final: 0.000823054560908254
```

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

```python
import numpy as np

def gauss_seidel_method(A, b, tol, max_iterations):
    n = len(b)
    x = np.zeros(n)
    final_x = None
    final_error = None

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

        for i in range(n):
            sum_ = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - sum_) / A[i, i]
        error = np.linalg.norm(x - x_old, ord=np.inf)

        final_x = np.copy(x)
        final_error = error
```

```
        if error < tol:
            return final_x, k + 1

    return final_x, max_iterations
```

```
tol = 1e-3
max_iterations = 100
sistemas = {
    'a': (np.array([[3, -1, 1], [3, 6, 2], [3, 3, 7]]), np.array([1, 0, 4])),
    'b': (np.array([[10, -1, 0], [-1, 10, -2], [0, -2, 10]]), np.array([9, 7, 6])),
    'c': (np.array([[10, 5, 0, 0], [5, 10, -4, 0], [0, -4, 8, -1], [0, 0, -1, 5]]), np.array
    'd': (np.array([[4, 1, 1, 0, 1], [-1, -3, 1, 1, 0], [2, 1, 5, -1, -1], [0, -1, 0, 4, 0],
}
for literal, (A, b) in sistemas.items():
    print(f"\nSistema {literal.upper()}:")
    solution, iterations = gauss_seidel_method(A, b,tol, max_iterations)
    print(f"Solución aproximada después de {iterations} iteraciones: {solution}\n")
```

```
Sistema A:
Solución aproximada después de 6 iteraciones: [ 0.03535107 -0.23678863  0.65775895]


Sistema B:
Solución aproximada después de 4 iteraciones: [0.9957475  0.95787375 0.79157475]


Sistema C:
Solución aproximada después de 9 iteraciones: [-0.79691476  2.79461827 -0.25918081 -2.251836


Sistema D:
Solución aproximada después de 7 iteraciones: [ 0.78324565 -1.2547316   1.8326783   1.186317
```

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 solucion (1,2,-1)

a) Muestre que el metodo de Jacobi con $x_0 = 0$ falla al proporcionar una buena aproximacion despues de 25 iteraciones.

```python
import numpy as np

def jacobi_method(A, b, x0, max_iterations, tol):
    n = len(b)
    x = np.copy(x0)
    final_x = None
    final_error = None

    for k in range(max_iterations):
        x_new = np.copy(x)

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

        error = np.linalg.norm(x_new - x, ord=np.inf)
        x = x_new
        final_x = np.copy(x)
        final_error = error

        if error < tol:
            break

    return final_x, k + 1
```

```python
A = np.array([[2, -1, 1],[2, 2, 2],[-1, -1, 2]])
b = np.array([-1, 4, -5])
x0 = np.zeros(3)
max_iterations = 25
tol = 1e-5

print("Método de Jacobi:")
sol_jacobi, iterations = jacobi_method(A, b, x0, max_iterations, tol)
print(f"Solución aproximada después de {iterations} iteraciones: {sol_jacobi}\n")
```

```
Método de Jacobi:
Solución aproximada después de 25 iteraciones: [-20.82787284   2.         -22.82787284]
```

b) Utilice el metodo de Gauus-Siedel con $x_0 = 0$, para aproximar la solucion para el sistema lineal dentro de 10^{-5}.

```python
import numpy as np

def gauss_seidel_method(A, b, tol, max_iterations):
    n = len(b)
    x = np.zeros(n)
    final_x = None
    final_error = None

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

        for i in range(n):
            sum_ = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - sum_) / A[i, i]
        error = np.linalg.norm(x - x_old, ord=np.inf)

        final_x = np.copy(x)
        final_error = error

        if error < tol:
            return final_x, k + 1

    return final_x, max_iterations

tol = 1e-5
max_iterations = 100
sistemas = {
    'a': (np.array([[2, -1, 1], [2, 2, 2], [-1, -1, 2]]), np.array([-1, 4, -5])),
}

for literal, (A, b) in sistemas.items():
    print(f"\nSistema {literal.upper()}:")
    solution, iterations = gauss_seidel_method(A, b, tol, max_iterations)
    print(f"Solución aproximada después de {iterations} iteraciones: {solution}\n")
```

```
Sistema A:
Solución aproximada después de 23 iteraciones: [ 1.00000226  1.9999975  -1.00000012]
```

6. El sistema lineal :

$x_1 - x_3 = 0.2,$
$-1/2x_1 + x_2 - 1/4x_3 = -1.425,$
$x_1 - 1/2x_2 + x_3 = 2$

Tiene solucion (0.9,-0.8,0.7)

a) ¿La matriz de coeficientes

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

tiene diagonal estrictamente dominante?

```
import numpy as np
def es_estrictamente_diagonal_dominante(A):
    n = A.shape[0]
    es_dominante = True
    for i in range(n):
        diagonal = abs(A[i, i])
        suma_fuera_diagonal = sum(abs(A[i, j]) for j in range(n) if j != i)
        print(f"Fila {i+1}: Elemento diagonal = {diagonal}, Suma fuera diagonal = {suma_fuera
        if diagonal <= suma_fuera_diagonal:
            print(f"-> La fila {i+1} no cumple con la dominancia diagonal estricta.")
            es_dominante = False
        else:
            print(f"-> La fila {i+1} cumple con la dominancia diagonal estricta.")
    return es_dominante

A = np.array([[4,0,-1],[-1/2,1,-1/4],[1,-1/2,1],
])
if es_estrictamente_diagonal_dominante(A):
    print("\nLa matriz es estrictamente diagonalmente dominante.")
else:
    print("\nLa matriz no es estrictamente diagonalmente dominante.")
```

```
Fila 1: Elemento diagonal = 4.0, Suma fuera diagonal = 1.0
-> La fila 1 cumple con la dominancia diagonal estricta.
Fila 2: Elemento diagonal = 1.0, Suma fuera diagonal = 0.75
-> La fila 2 cumple con la dominancia diagonal estricta.
Fila 3: Elemento diagonal = 1.0, Suma fuera diagonal = 1.5
```

-> La fila 3 no cumple con la dominancia diagonal estricta.

La matriz no es estrictamente diagonalmente dominante.

b) Utilice el metodo iterativo de Gauss-Siedel para aproximar la solucion para el sistema lineal con una tolerancia de $10_2 2$ y un maximo de 300 iteraciones.

```
import numpy as np
def gauss_seidel_method(A, b, tol, max_iterations):
    n = len(b)
    x = np.zeros(n)

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

        for i in range(n):
            sum_ = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - sum_) / A[i, i]

        error = np.linalg.norm(x - x_old, ord=np.inf)
        if error < tol:
            print(f"Convergencia alcanzada después de {k+1} iteraciones.")
            print(f"Última Iteración {k+1}: x = {x}, Error = {error:.2e}")
            return x, k + 1
    print(f"No se alcanzó la convergencia después de {max_iterations} iteraciones.")
    print(f"Última Iteración {max_iterations}: x = {x}, Error = {error:.2e}")
    return x, max_iterations
```

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

b = np.array([0.2, -1.425, 2])


x0 = np.zeros(3)

tol = 1e-22
max_iterations = 300
print("Sistema inicial (b):")
solution_b = gauss_seidel_method(A, b, tol, max_iterations)
print(f"Solución aproximada: {solution_b}\n")
```

```
Sistema inicial (b):
No se alcanzó la convergencia después de 300 iteraciones.
Última Iteración 300: x = [ 0.9 -0.8  0.7], Error = 2.22e-16
Solución aproximada: (array([ 0.9, -0.8,  0.7]), 300)
```

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

$x_1 - 2x_3 = 0.2,$
$-1/2x_1 + x_2 - 1/4x_3 = -1.425,$
$x_1 - 1/2x_2 + x_3 = 2$

```python
A = np.array([[1, 0, -2],[-1/2, 1, -1/4],[1, -1/2, 1]])
b = np.array([0.2, -1.425, 2])
x0 = np.zeros(3)
tol = 1e-22
max_iterations = 300
print("Sistema modificado (c) con el método de Gauss-Seidel:")
solution_c= gauss_seidel_method(A, b, tol, max_iterations)
print(f"Solución aproximada: {solution_c}\n")
```

```
Sistema modificado (c) con el método de Gauss-Seidel:
No se alcanzó la convergencia después de 300 iteraciones.
Última Iteración 300: x = [ 2.15687283e+41  1.34804552e+41 -1.48285007e+41], Error = 3.73e+41
Solución aproximada: (array([ 2.15687283e+41,  1.34804552e+41, -1.48285007e+41]), 300)
```

7. Repita el ejercicio 6 usando el metodo de Jacobi.

```python
import numpy as np

def jacobi(A, b, x0, tol, max_iterations):
    n = len(b)
    x = np.copy(x0)
    x_new = np.copy(x0)
    last_error = None
    last_iteration = 0

    for k in range(max_iterations):
        for i in range(n):
            sum_ = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - sum_) / A[i, i]
```

```
        error = np.linalg.norm(x_new - x, ord=np.inf)
        last_error = error
        last_iteration = k+1
        if error < tol:
            break

        x = np.copy(x_new)
    print(f"Última Iteración {last_iteration}: x = {x_new}, Error = {last_error:.2e}")
    if last_error < tol:
        print(f"Convergencia alcanzada después de {last_iteration} iteraciones.")
    else:
        print(f"No se alcanzó la convergencia después de {last_iteration} iteraciones.")

    return x_new

# Sistema inicial (b)
A_b = np.array([[1, 0, -1],[-1/2, 1, -1/4],[1, -1/2, 1]])
b_b = np.array([0.2, -1.425, 2])

# Sistema modificado (c)
A_c = np.array([
    [1, 0, -2],
    [-1/2, 1, -1/4],
    [1, -1/2, 1]
])

b_c = np.array([0.2, -1.425, 2])
x0 = np.zeros(3)
tol = 1e-22
max_iterations = 300
print("Sistema inicial (b) con el método de Jacobi:")
solution_b = jacobi(A_b, b_b, x0, tol, max_iterations)
print(f"Solución aproximada: {solution_b}\n")

print("Sistema modificado (c) con el método de Jacobi:")
solution_c = jacobi(A_c, b_c, x0, tol, max_iterations)
print(f"Solución aproximada: {solution_c}\n")
```

```
Sistema inicial (b) con el método de Jacobi:
Última Iteración 300: x = [ 0.90025541 -0.80004033  0.70012933], Error = 4.64e-04
No se alcanzó la convergencia después de 300 iteraciones.
Solución aproximada: [ 0.90025541 -0.80004033  0.70012933]
```

```
Sistema modificado (c) con el método de Jacobi:
Última Iteración 300: x = [ 1.34478913e+43 -7.28028158e+42  1.56650340e+43], Error = 3.14e+43
No se alcanzó la convergencia después de 300 iteraciones.
Solución aproximada: [ 1.34478913e+43 -7.28028158e+42  1.56650340e+43]
```

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.

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

a) ¿La matriz es estrictamente diagonalmente dominante?

```python
import numpy as np
def es_estrictamente_diagonal_dominante(A):
    n = A.shape[0]
    es_dominante = True
    for i in range(n):
        diagonal = abs(A[i, i])
        suma_fuera_diagonal = sum(abs(A[i, j]) for j in range(n) if j != i)
        print(f"Fila {i+1}: Elemento diagonal = {diagonal}, Suma fuera diagonal = {suma_fuera
```

```
        if diagonal <= suma_fuera_diagonal:
            print(f"-> La fila {i+1} no cumple con la dominancia diagonal estricta.")
            es_dominante = False
        else:
            print(f"-> La fila {i+1} cumple con la dominancia diagonal estricta.")

    return es_dominante

A = np.array([
    [4, -1, 0, 0, 0, -1, 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, 0, 0, 0, 0, 0, 0, -1],
    [-1, 0, 0, 0, 4, 0, 0, 0, 0, -1, 0, 0],
    [0, 0, 0, 0, 0, 4, -1, 0, 0, 0, -1, 0],
    [0, 0, 0, 0, 0, -1, 4, -1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, -1, 4, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 4, -1, 0, -1],
    [0, 0, 0, 0, -1, 0, 0, 0, -1, 4, 0, 0],
    [0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 4, -1],
    [0, 0, 0, -1, 0, 0, 0, 0, -1, 0, -1, 4]])
if es_estrictamente_diagonal_dominante(A):
    print("\nLa matriz es estrictamente diagonalmente dominante.")
else:
    print("\nLa matriz no es estrictamente diagonalmente dominante.")
```

```
Fila 1: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 1 cumple con la dominancia diagonal estricta.
Fila 2: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 2 cumple con la dominancia diagonal estricta.
Fila 3: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 3 cumple con la dominancia diagonal estricta.
Fila 4: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 4 cumple con la dominancia diagonal estricta.
Fila 5: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 5 cumple con la dominancia diagonal estricta.
Fila 6: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 6 cumple con la dominancia diagonal estricta.
Fila 7: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 7 cumple con la dominancia diagonal estricta.
Fila 8: Elemento diagonal = 4, Suma fuera diagonal = 1
-> La fila 8 cumple con la dominancia diagonal estricta.
```

```
Fila 9: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 9 cumple con la dominancia diagonal estricta.
Fila 10: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 10 cumple con la dominancia diagonal estricta.
Fila 11: Elemento diagonal = 4, Suma fuera diagonal = 2
-> La fila 11 cumple con la dominancia diagonal estricta.
Fila 12: Elemento diagonal = 4, Suma fuera diagonal = 3
-> La fila 12 cumple con la dominancia diagonal estricta.

La matriz es estrictamente diagonalmente dominante.
```

b) Resuelva el sistema lineal usando el metodo de Jacobi con $x_0 = 0 y TOL = 10^{-2}$

```python
import numpy as np

def jacobi_method(A, b, x0, tol, max_iterations):
    x = x0.copy()
    n = len(b)
    x_prev = np.zeros(n)
    final_x = None
    final_error = None

    for k in range(max_iterations):
        x_prev = x.copy()

        for i in range(n):
            sum_ = sum(A[i][j] * x_prev[j] for j in range(n) if j != i)
            x[i] = (b[i] - sum_) / A[i][i]
        error = np.linalg.norm(x - x_prev, ord=np.inf)
        final_x = x.copy()
        final_error = error

        if error < tol:
            break

    return final_x, k + 1, final_error
```

```python
b = np.array([220, 110, 110, 220, 110, 110, 110, 110, 220, 110, 110, 220])
x0 = np.zeros(len(b))
tol = 1e-2
max_iterations = 300
```

```
solution, iterations, final_error = jacobi_method(A, b, x0, tol, max_iterations)
print(f"Solución aproximada después de {iterations} iteraciones: {solution}")
print(f"Error final: {final_error:.2e}")
```

```
Solución aproximada después de 15 iteraciones: [ 86.40227661  66.62768292  70.1176931  103.85
  58.98986709  52.39574392  40.59828363 103.85466119  70.1176931
  73.572989   125.31533062]
Error final: 9.20e-03
```

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

```
import numpy as np

def gauss_seidel_method(A, b, x0, tol, max_iterations):
    x = x0.copy()
    n = len(b)
    final_x = None
    final_error = None
    iterations = 0

    for k in range(max_iterations):
        x_prev = x.copy()

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

        error = np.linalg.norm(x - x_prev, ord=np.inf)
        final_x = x.copy()
        final_error = error
        iterations = k + 1

        if error < tol:
            break

    return final_x, iterations, final_error
```

```
solution_gauss_seidel, iterations, final_error = gauss_seidel_method(A, b, x0, tol, max_itera
print(f"\nSolución final después de {iterations} iteraciones: {solution_gauss_seidel}")
print(f"Error final: {final_error:.2e}")
```

```
Solución final después de 10 iteraciones: [ 86.40549601  66.63171896  70.12309751 103.861992
  58.99409476  52.39839298  40.59959824 103.86190161  70.12348967
  73.57974223 125.32590919]
Error final: 3.32e-03
```

Link de repositorio: https://github.com/armando-2002/Metodos_Numericos.git