Tarea Metodos iterativos

Angel Caceres Licona

July 11, 2020

1 Aproxime la solución de los siguientes sistemas usando el método de Jacobi y el método de Gauss-Seidel

1.1 Para el primer sistema tenemos

Para Jacobi lo hace en 499 iteraciones y obtengo este resultado: [7.07951054e+171 6.04439046e+171 -2.95318927e+172 7.68350811e+172 -3.93286362e+172]
Gauss Seidel lo hace en 10 iteraciones y obtenemos el siguiente resultado

1.2 Para el segundo sistema tenemos

para Jacobi lo calcula en 13 iteraciones y da el siguiente resultado: [2.71355235 2.37696729 0.59407288 2.47846564 1.20059366 1.94860919]
Para Gauss Seidel lo obtiene en 3 iteraciones y obtengo: [2 1 0 2 0 1]

2 Problema 2

2.1 Investigue qué es una matriz diagonal dominante...

Tenemos la siguiente matriz:

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

Si, la matriz es diagonal dominante.

2.2 Ahora calcule las soluciones usando el método de Jacobi...

Para el método de Jacobi obtuve: [0.88790326 -0.76901297 0.63891982] en 108 iteraciones
Para Gauus Seidel obtuve [0.90397904 -0.79701572 0.6975131] en 13 iteraciones

2.3 Ahora cambiando la matriz...

Para Jacobi hizo 300 iteraciones y calculó: [1.34478913e+43 -7.28028158e+42 1.56650340e+43]
Para Gauss-Seidel hizo [-1.56863478e+41 -9.80396739e+40 1.07843641e+41] en 300 iteraciones

Podemos concluir que la convergencia del método depende de si la matriz es diagonal dominante

```
import numpy as np

def jacobi(A, b, x_init, epsilon=10e-2, max_iterations=300):
    D = np.diag(np.diag(A))
    LU = A - D
    x = x_init
    for i in range(max_iterations):
        D_inv = np.diag(1 / np.diag(D))
        x_new = np.dot(D_inv, b - np.dot(LU, x))
        if np.linalg.norm(x_new - x) < epsilon:
            return x_new
            break
        x = x_new
        print("i=", i)
    return x

A = np.array([
    [1, 0, -2],
    [-1/2, 1, -1/4],
    [1, -1/2, 1]
])
b = np.array([0.2, -1.425, 2])

x_init = np.zeros(len(b))
x = jacobi(A, b, x_init)

print('x:', x)
print('computed b:', np.dot(A, x))
print('real b:', b)


import numpy as np

ITERATION_LIMIT = 300


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

```
    [-1/2, 1, -1/4],
    [1, -1/2, 1]])

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

print("System of equations:")
for i in range(A.shape[0]):
    row = ["{0:3g}*x{1}".format(A[i, j], j + 1) for j in range(A.shape[1])]
    print("[{0}] = [{1:3g}]".format(" + ".join(row), b[i]))

x = np.zeros_like(b)
for it_count in range(1, ITERATION_LIMIT):
    x_new = np.zeros_like(x)
    print("Iteration {0}: {1}".format(it_count, x))
    for i in range(A.shape[0]):
        s1 = np.dot(A[i, :i], x_new[:i])
        s2 = np.dot(A[i, i + 1:], x[i + 1:])
        x_new[i] = (b[i] - s1 - s2) / A[i, i]
    if np.allclose(x, x_new, rtol=10e-2):
        break
    x = x_new

print("Solucion: {0}".format(x))
error = np.dot(A, x) - b
print("Error: {0}".format(error))
```