MA5470 - Assignment 1

MM16B023[1]

[a]*Indian Institute of Technology Madras*

Keyword: Jacobi, Gauss-Siedel, error, convergence

Abstract: This paper presents the solutions to fist assignment of the Numerical Analysis course (MA5470) at *IIT Madras*. All the notations used are as according with the textbook Numerical Analysis: Mathematics of Scientific Computing

Iterative schemes

To solve Ax = b,

$$A_{ij} = \begin{cases} N*(i+j+2) & i = j \\ \dfrac{i+j+2}{N} & i \neq j \end{cases}$$

Clearly, this matrix is diagonally dominant and therefore, any inital guess would lead to convergence for the following two schemes.

a) Jacobi

$$A = D - L - U$$
$$Dx = (L+U)x + b$$
$$x = D^{-1}(L+U)x + D^{-1}b$$

Therefore, the Jacobi-iteration is $x^{(n+1)} = D^{-1}(L+U)x^{(n)} + D^{-1}b$

b) Gauss-Siedel

$$Dx^{(n+1)} = Lx^{(n+1)} + Ux^{(n)} + D^{-1}b$$

Therefore, the Gauss-Siedel iterative scheme is $x^{(n+1)} = (D-L)^{-1}Ux^{(n)} + (D-L)^{-1}b$

The above 2 schemes are implemented in Python as follows:

```
import numpy as np

''' Jacobi and Gauss-Siedel iterative schemes to solve Ax = b'''

### JACOBI method ###

def jacobi_solver(A,b,x0,num_itr):

    D = np.diag(A)
    R = A - np.diagflat(D)
    x = x0

    for i in range(num_itr):

        temp = (b - np.dot(R,x))/ D
```

```python
        x = temp

    return x

### GAUSS-SIEDEL method ###

def seidel_solver(A,b,x0,num_itr):

    x = x0
    N = len(A)
    for j in range(0, N):
        temp = b[j]
        for i in range(0, N):
            if(j != i):
                temp -= A[j][i] * x[i]

        x[j] = temp / A[j][j]
    return x

### MAIN CODE ###

# Enter the dimension of the matrix A
N = 5

# Assign values for the matrix A
A = np.zeros((N,N))
for i in range(N):
    for j in range(N):
        if i == j:
            A[i,j] = N*(i+j+2)
        else:
            A[i,j] = (i+j+2)/N

# Entries of the b vector
b = np.random.random(N)

# Random guess for the solution
x0 = np.zeros(N)

x = jacobi_solver(A,b,x0,num_itr = 200)
y = seidel_solver(A,b,x0,num_itr = 200)

# Print statements
print("A =",np.matrix(A),"\n")
print("b =",np.matrix(b),"\n")

print("Initial Guess =", np.zeros(N),"\n")

print("Solution from Jacobi method =", x)
print("Solution from Gauss-Siedel method =",y)
print('Solution from built-in functions = %s' % np.linalg.solve(A, b),"\n")

print("Error in Jacobi method =",np.linalg.norm(x-np.linalg.solve(A,b)))
print("Error in Gauss-Siedel method =",np.linalg.norm(y-np.linalg.solve(A,b)))
```

Output of the above code is shown overleaf

Output

```
A = [[10.   0.6  0.8  1.   1.2]
 [ 0.6 20.   1.   1.2  1.4]
 [ 0.8  1.  30.   1.4  1.6]
 [ 1.   1.2  1.4 40.   1.8]
 [ 1.2  1.4  1.6  1.8 50. ]]

b = [[0.48817126 0.66637875 0.74816905 0.65817666 0.06086219]]

Initial Guess = [0. 0. 0. 0. 0.]

Solution from Jacobi method = [ 0.0440799   0.03019293  0.02221612  0.01375421 -0.00189214]
Solution from Gauss-Siedel method = [ 0.04881713  0.03185442  0.02257536  0.01348822 -0.00205428]
Solution from built-in functions = [ 0.0440799   0.03019293  0.02221612  0.01375421 -0.00189214]

Error in Jacobi method = 8.50114066984862e-18
Error in Gauss-Siedel method = 0.005042618184882898
```

For this particular case, Jacobi method converges to the solution with lower error value. In general, the rate of convergence and error depends on the condition number of the matrix $\kappa(A)$ and the initial guess x_0
