NA

Assignment 1

Comparision between Jacobi Iterative method and Gauss-Seidel Iterative method

By:

Abdul Rehman Ansari

24487

Function Descriptions

generateMatrix(size : int):

* Takes an integer as an input and returns a Diagonally Dominant matrix of size NxN
* A value of Diagonal\_value (Stored above as 1000 as a constant) is placed on the diagonal values
* The other elements are generated using a function that generates random numbers from x to y [random.randint(x,y)]. In my implementation:
  + X is kept 0
  + Y is Diagonal\_value/N for a matrix of size NxN
* Matrix is generated row-by-row.

generateResult(size : int):

* Takes an integer as an input and returns a matrix of size Nx1
* The elements are generated using a function that generates random numbers from x to y [random.randint(x,y)]. In my implementation:
  + X is kept 0
  + Y is Diagonal\_value (Stored above as 1000 as a constant)

Residual(matrix, coeff, result):

* Takes 3 matrices as input namely:
  + Matrix = matrix of order NxN
  + Coeff = matrix containing calculated values of x
  + Result = matrix of order Nx1
* Calculates the residual matrix using formula (Matrix) x (coeff) – Result [Ax – B]
* Returns the magnitude of the residual matrix ( sqrt( R1^2 + R2^2 … + Rn^2)
  + Where R1,R2…Rn are the elements of the matrix

Gauss\_Seidel(matrix, result):

* Takes 2 matrices as input namely:
  + Matrix = matrix of order NxN
  + Result = matrix of order Nx1
* It maintains a matrix of order Nx1 where it stores calculated values of x (initialized to a null matrix)
* The residual is calculated using the function described previously and stored as residual
* Then a while loop is started where values of x are calculated using the formula by:
  + Xi = [Bi + (**Σ ( j = 0 to N)** AijXj )] / Aii
* For each iteration, the above formula is done N times ( once for each value of x) These values of x are stored in the matrix we previously initialized
* After every 5 iterations, the residual is calculated and stored in the residual variable
* Once the residual value is lower than the tolerance value ( 10 ^ -20), the while loop is terminated and the matrix with calculated values of x are returned

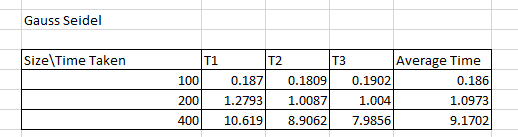
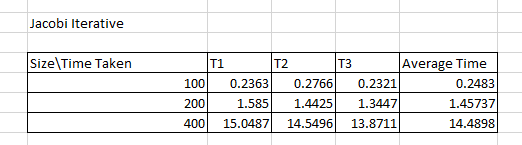
Jacobi\_iterative(matrix, result):

* Works the same as Gauss\_seidel function, the only difference is that in every while loop, a clone of the x matrix is generated to calculate x values for the next iteration while using all the x values acquired in the previous iteration.

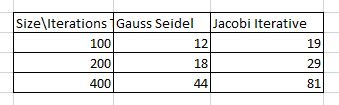
Comparison

Computation Time Comparison:

(In seconds)

Iterations Taken comparision:



Conclusion:

Gauss Seidel is better in both the Iterations taken to solve a particular system of equations and also the real-time taken to compute the answers. The difference in time taken and number of iterations needed increase as the size of the system of equations increases.

Code

Language used: Python

import numpy as np  
import math  
import random  
from decimal import Decimal  
import time  
  
# Constants  
  
DIAGONAL\_VALUE: int = 1000  
TOLERANCE\_VALUE: int = 10 \*\* -20  
TOLERANCE\_CHECK: int = 5  
  
  
# Return matrix of size n  
  
  
  
def generateMatrix(size: int):  
 matrix = []  
  
 max\_num = int(DIAGONAL\_VALUE / (size - 1))  
 for i in range(0, size):  
 row = []  
  
 for j in range(0, size):  
 if i == j:  
 row.insert(j, DIAGONAL\_VALUE)  
 else:   
 row.insert(j, random.randint(0, max\_num))   
  
 matrix.insert(i, row)  
 return matrix  
  
# Returns the Result matrix of size n  
  
def generateResult(size: int):  
 result = []  
  
 for i in range(0, size):  
 result.insert(i, random.randint(1, 10))  
  
 return result

# Returns Residual of Ax - B = 0  
  
def Residual(matrix, coeff, result):  
 residualMatrix = np.subtract(np.dot(matrix, coeff), result)  
 sum = 0  
  
 for i in range(0, len(coeff)):  
 sum += Decimal(residualMatrix[i] \*\* 2)  
  
 residual = math.sqrt(sum)  
  
 return residual  
  
  
# Performs Gauss Seidel method on A and B matrices and returns values of coefficients in an array  
  
def Gauss\_Seidel(matrix, result):  
 coeff = []  
  
 error = []  
  
 for i in range(0, len(result)):  
 coeff.insert(i, Decimal(0))  
  
 counter = 0  
 residual = Residual(matrix, coeff, result)  
 error.append(residual)  
  
 while residual > TOLERANCE\_VALUE:  
  
 for i in range(0, len(coeff)):  
  
 sum: float = 0  
  
 for j in range(0, len(coeff)):  
 if (i == j):  
 continue  
 sum += matrix[i][j] \* coeff[j]  
  
 coeff[i] = Decimal((result[i] - sum) / matrix[i][i])  
  
 counter += 1  
  
 if counter % TOLERANCE\_CHECK == 0:  
 residual = Residual(matrix, coeff, result)  
  
 print("Gauss-Seidel needed iterations = ", counter)  
 return coeff

# Performs Jacobi iterative method on A and B matrices and returns values of coefficients in an array  
  
def Jacobi\_iterative(matrix, result):  
 coeff = []  
 error = []  
  
 for i in range(0, len(result)):  
 value = Decimal(0)  
 coeff.insert(i, value)  
  
 counter = 0  
 residual = Residual(matrix, coeff, result)  
 error.insert(0, residual)  
  
 while (residual > TOLERANCE\_VALUE):  
 currCoeff = coeff.copy()  
 for i in range(0, len(currCoeff)):  
  
 sum: float = 0  
  
 for j in range(0, len(currCoeff)):  
 if (i == j):  
 continue  
 sum += matrix[i][j] \* currCoeff[j]  
  
 coeff[i] = Decimal((result[i] - sum) / matrix[i][i])  
  
 counter += 1  
  
 if counter % TOLERANCE\_CHECK == 0:  
 residual = Residual(matrix, coeff, result)  
  
  
 print("Jacobi Iterative needed iterations = ", counter)  
 return coeff  
  
  
  
def main():  
 size = 400  
 matrix = generateMatrix(size)  
 #print(matrix)  
  
 result = generateResult(size)  
  
  
 start\_time = time.time()  
 coeff = Gauss\_Seidel(matrix, result)  
 #print("Gauss Seidel co-efficients = ", coeff)  
  
 print("time taken for Gauss Seidel Method = ","{:0.4f}".format(time.time() - start\_time)," seconds")  
  
 print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n")  
  
 start\_time = time.time()  
 coeff = Jacobi\_iterative(matrix, result)  
 #print("Jacobi iterative Co-efficients = ", coeff)  
 print("time taken for Jacobi Iterative Method = ","{:0.4f}".format(time.time() - start\_time)," seconds")  
 print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n")  
  
  
main()