ME343 – Numerical Methods for Engineering

Homework 1 – Gauss Elimination and Gauss-Siedel Iteration

Berk Ilgar Özalp – 250203016

Instructor: Ayşe Korucu

IZTECH Department of Mechanical Engineering

December 12th, 2021

Table of Contents

INTRODUCTION 2

SCRIPT EVALUATION 3

isLessThan10x10() Function 3

gaussElimination() Function 3

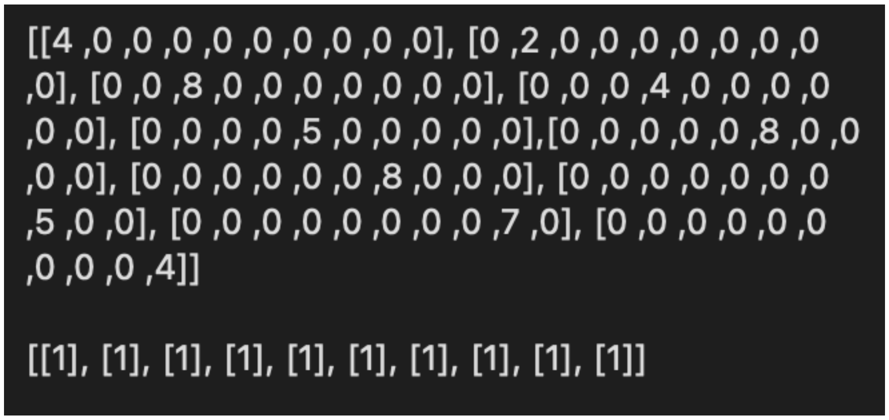
isDominant() Function 4

gaussSiedel() Function 5

Command Part 6

Introduction

The assignment of Gauss Elimination and Gauss-Siedel Iteration method was coded in Python 3. Python 3 is a powerful high-level language in which has a wide variety of libraries to work with. In this case, “Numpy” and “Scipy” libraries were used for enhancing the calculative properties of Python the coding language. Easy installation method of these dependencies were mentioned in readme.txt file. Readme.txt file lists the limitations and some warnings about the code itself, such as 10 by 10 coefficient matrix limitation required by the assignment conditions. Moreover, the value of the matrix is taken from the user in a certain format in order to be initialized. The matrix is needed to be typed line by line, separated by commas, in squared parentheses. An example of proper input can be seen in “Figure 1”.



```
[[4 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0], [0 ,2 ,0 ,0 ,0 ,0 ,0 ,0 ,0
,0], [0 ,0 ,8 ,0 ,0 ,0 ,0 ,0 ,0 ,0], [0 ,0 ,0 ,4 ,0 ,0 ,0 ,0
,0 ,0], [0 ,0 ,0 ,0 ,5 ,0 ,0 ,0 ,0 ,0],[0 ,0 ,0 ,0 ,0 ,8 ,0 ,0
,0 ,0], [0 ,0 ,0 ,0 ,0 ,0 ,8 ,0 ,0 ,0], [0 ,0 ,0 ,0 ,0 ,0 ,0
,5 ,0 ,0], [0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0], [0 ,0 ,0 ,0 ,0 ,0
,0 ,0 ,0 ,4]]

[[1], [1], [1], [1], [1], [1], [1], [1], [1], [1]]
```

Figure 1: Example Input Format of the Matrix

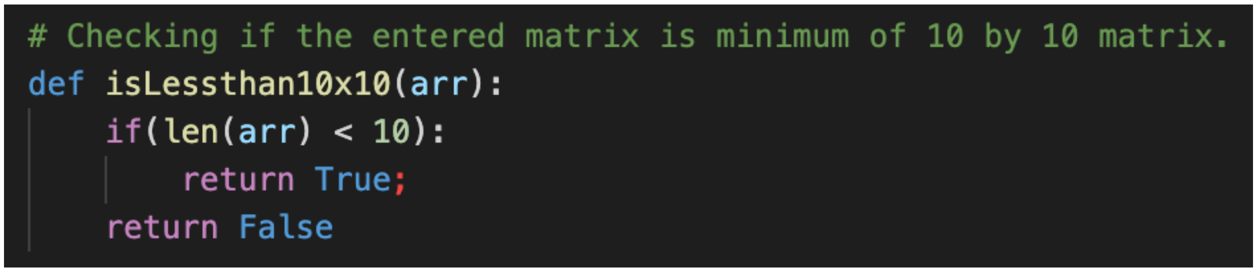
In addition to that, the script checks diagonally dominance which is needed for Gauss-Siedel method (further explained in script section of this paper). Thus, the user must enter a value which is diagonally dominant for the script to run properly, else the code will prompt an error message stating that the value is not diagonally dominant.

Script Evaluation

This section covers the evaluation of the script one by one separated by functions defined during the development process.

isLessThan10x10() Function

This function checks the length of the value typed by the user and fulfills the requirement of 10x10 matrix mentioned by the assignment paper. In order to bypass this function, user can disable the if/else condition in the section titled as "command part".

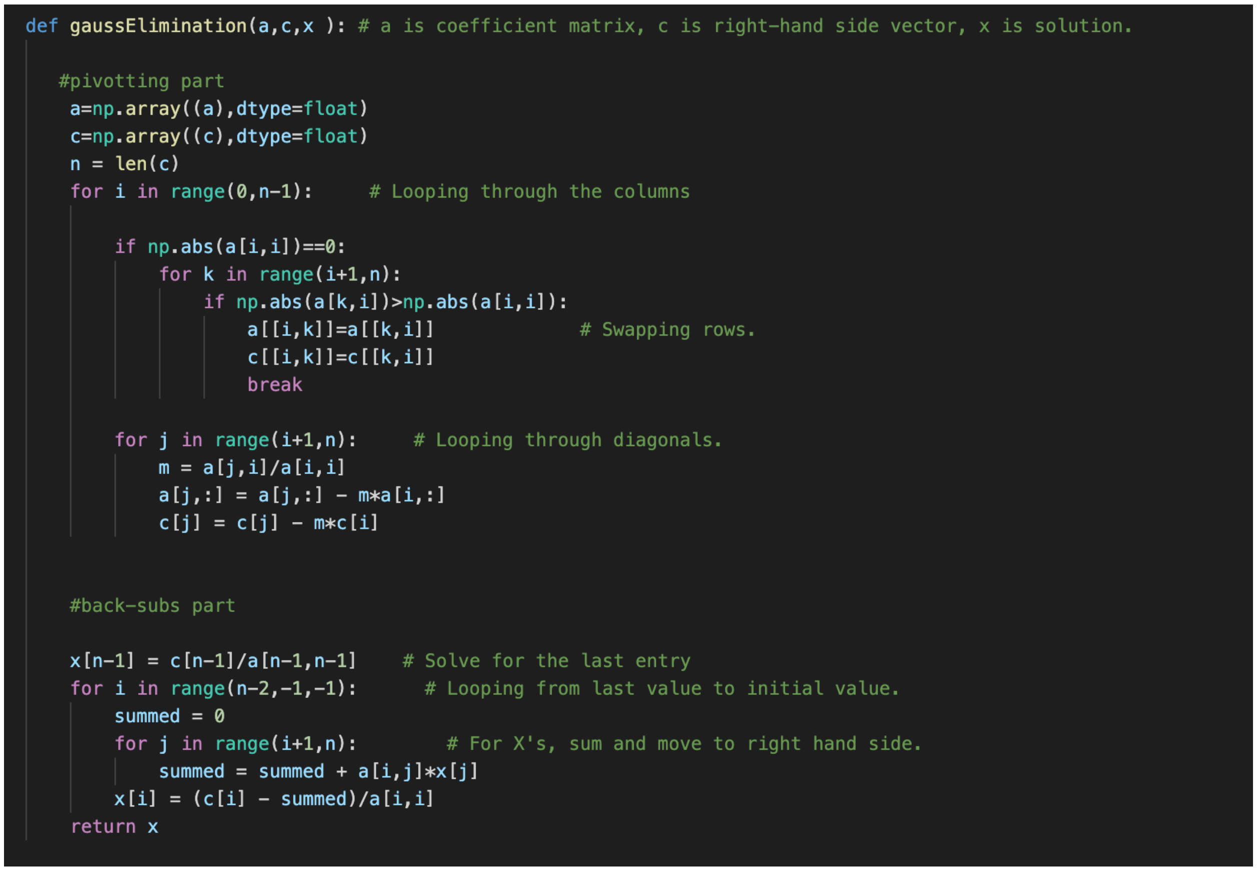


```python
# Checking if the entered matrix is minimum of 10 by 10 matrix.
def isLessthan10x10(arr):
    if(len(arr) < 10):
        return True;
    return False
```

Figure 2: 10 by 10 Matrix Check

gaussElimination() Function

This function is the process of the gauss elimination. When the input is provided from the user as A and C, as stated in assignment value the code creates an array of them using the Numpy library. After taking the length of the input C (which is needed for looping through the rows of the arrays), the code travels across the columns and swaps the rows if absolute of k,ith value is greater than i,i th value of the row. Then the code goes across the diagonal dividing the values by upper rows and subtracting from the initial value. In the second part of the Gauss Elimination function, the code performs the back substruction starting from the last entry. It first solves the x array by values of c and a given by the user input. Than loops from the last value to initial value by multiplying a and x values for each row, and moving it to the right hand side. Lastly the function returns the x array, completing the Gauss Elimination process. Figure 3 shows that portion of the script with proper comments added for crucial turning points of the script.

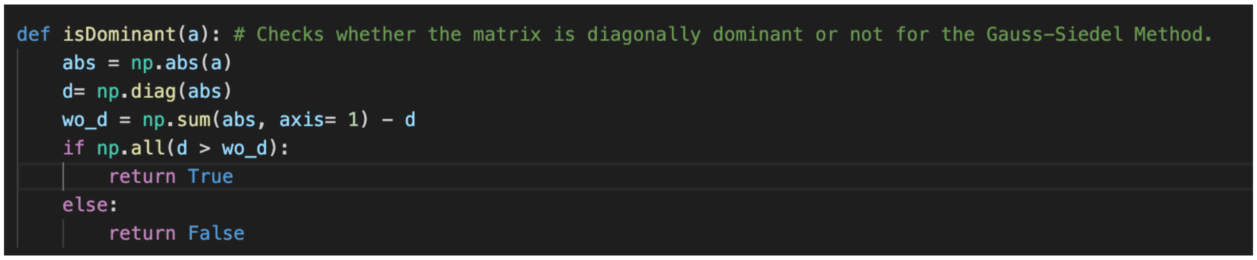


```python
def gaussElimination(a,c,x ): # a is coefficient matrix, c is right-hand side vector, x is solution.

   #pivotting part
    a=np.array((a),dtype=float)
    c=np.array((c),dtype=float)
    n = len(c)
    for i in range(0,n-1):     # Looping through the columns

        if np.abs(a[i,i])==0:
            for k in range(i+1,n):
                if np.abs(a[k,i])>np.abs(a[i,i]):
                    a[[i,k]]=a[[k,i]]             # Swapping rows.
                    c[[i,k]]=c[[k,i]]
                    break

        for j in range(i+1,n):     # Looping through diagonals.
            m = a[j,i]/a[i,i]
            a[j,:] = a[j,:] - m*a[i,:]
            c[j] = c[j] - m*c[i]


    #back-subs part

    x[n-1] = c[n-1]/a[n-1,n-1]    # Solve for the last entry
    for i in range(n-2,-1,-1):      # Looping from last value to initial value.
        summed = 0
        for j in range(i+1,n):        # For X's, sum and move to right hand side.
            summed = summed + a[i,j]*x[j]
        x[i] = (c[i] - summed)/a[i,i]
    return x
```

Figure 3: Gauss Elimination Function within Script

isDominant() Function

One of the conditions for Gauss-Siedel Iteration method to work is making sure that the matrix is diagonally dominant. In the case of that the matrix is not diagonally dominant, the iteration would not result with a proper answer since not converging properly to the value we want. “isDominant” function is there to check this condition. It takes the absolute value of the matrix, defines “d” as the diagonal of this matrix, takes all the values without the diagonal while summing them up as “wo_d” and checks whether the absolute value of diagonal is greater than wo_d or not. The outcome of this function is used in the gaussSiedel() function in the script which is mentioned later in this paper.

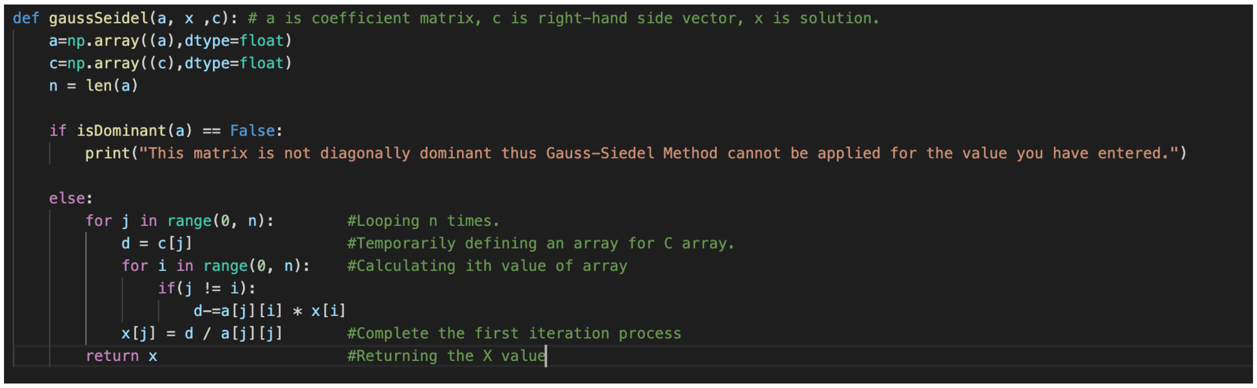


```python
def isDominant(a): # Checks whether the matrix is diagonally dominant or not for the Gauss-Siedel Method.
    abs = np.abs(a)
    d= np.diag(abs)
    wo_d = np.sum(abs, axis= 1) - d
    if np.all(d > wo_d):
        return True
    else:
        return False
```

Figure 4: isDominant() Function Used for Diagonally Dominance Check

gaussSiedel() Function

The script first takes the values entered by the user unaltered and defines arrays using the values. Then, defined a value "n" for the length of the entered values (which is needed since one can enter any matrix bigger than 10). Later, it checks the condition of diagonal dominance as mentioned in section above. If the condition is False, it exits the function, else it continues the iteration of the values from 0 to n number of lines repeatedly, which was determined by the length of C matrix, for I and J values. Then, function defines a temporary value of d for storing the c value, later calculating the ith value of the array. Later it completes the first step of iteration and returns the x value at the end of all iteration, which is defined later in the main part of the code.

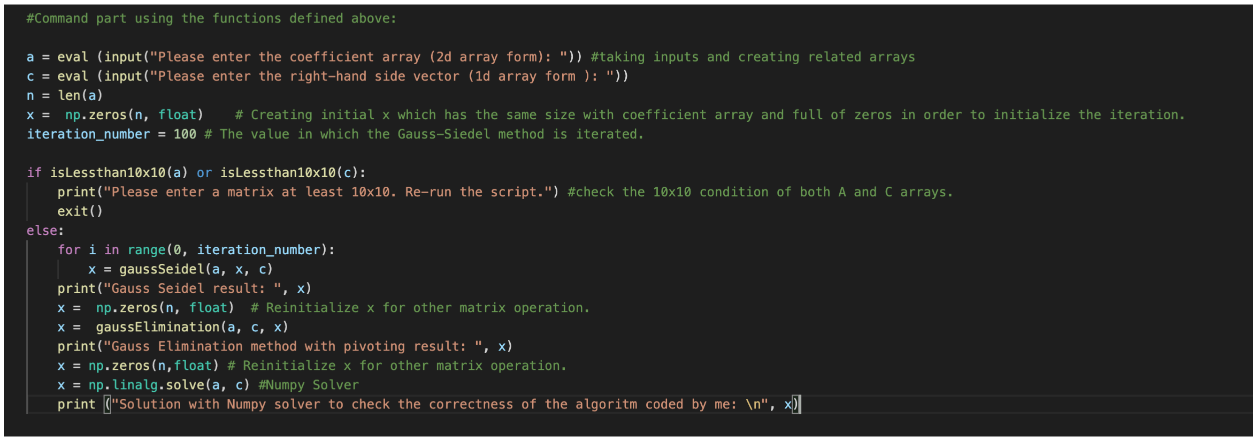


```python
def gaussSeidel(a, x ,c): # a is coefficient matrix, c is right-hand side vector, x is solution.
    a=np.array((a),dtype=float)
    c=np.array((c),dtype=float)
    n = len(a)

    if isDominant(a) == False:
        print("This matrix is not diagonally dominant thus Gauss-Siedel Method cannot be applied for the value you have entered.")

    else:
        for j in range(0, n):        #Looping n times.
            d = c[j]                 #Temporarily defining an array for C array.
            for i in range(0, n):    #Calculating ith value of array
                if(j != i):
                    d-=a[j][i] * x[i]
            x[j] = d / a[j][j]       #Complete the first iteration process
        return x                     #Returning the X value
```

Figure 5: gaussSiedel() Function

Command Part

After the definitions of the functions are completed, the script uses the mentioned functions to correctly evaluate the arrays. Iteration value is pre-determined as 100, which generously given for the correct evaluation of the array. Figure 6 contains the main section of the code with proper comments in the steps taking place in the evaluation. Last value is an extension of “Numpy” which solves the array with ease. It is there to check the consistency of the code written by me.



```python
#Command part using the functions defined above:

a = eval (input("Please enter the coefficient array (2d array form): ")) #taking inputs and creating related arrays
c = eval (input("Please enter the right-hand side vector (1d array form ): "))
n = len(a)
x =  np.zeros(n, float)    # Creating initial x which has the same size with coefficient array and full of zeros in order to initialize the iteration.
iteration_number = 100 # The value in which the Gauss-Siedel method is iterated.

if isLessthan10x10(a) or isLessthan10x10(c):
    print("Please enter a matrix at least 10x10. Re-run the script.") #check the 10x10 condition of both A and C arrays.
    exit()
else:
    for i in range(0, iteration_number):
        x = gaussSeidel(a, x, c)
    print("Gauss Seidel result: ", x)
    x =  np.zeros(n, float)  # Reinitialize x for other matrix operation.
    x =  gaussElimination(a, c, x)
    print("Gauss Elimination method with pivoting result: ", x)
    x = np.zeros(n,float) # Reinitialize x for other matrix operation.
    x = np.linalg.solve(a, c) #Numpy Solver
    print ("Solution with Numpy solver to check the correctness of the algoritm coded by me: \n", x)
```

Figure 6: Main Part of the Code