MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF MECHANICAL ENGINEERING

ME 310 NUMERICAL METHODS

FALL 2022

PROGRAMMING PROJECT 2

Murat Berk Buzluk 2377653

Gökberk Çiçek 2377810

Contents

ME 310
Project 2 07.12.2022

Abstract

ME310 lecture students create this report to develop proper programs to solve the linear system of equations by using two different methods. Python is used in programming because the broader application area of Python makes it preferable for students to learn. We have learned how to code and deal with different problems caused by other functions. All the mistakes that we have done while writing the code lead us to a better understanding of the course objectives.

1. INTRODUCTION

There are N particles positioned at equally spaced points, where $x_i, x_{i+1}..,x_{N-1}$, $(\, i \geq 0)\ in\ the\ domain\ \ of\ [0,1]$. The particles are denoted as $p_{i,}$ and their current positions are denoted as x_i with index i. The most left-positioned particle is changing its location with the function of $f(t) = 0.25\ x \sin(\pi t)$, where t is time, and the most right-positioned particle is not moving. The relation of the location of moving particles is obtained as follows:

$$x'_{i+1} - 2x'_i + x'_{i-1} = 0, \qquad x_i'{:}\ the\ new\ location\ of\ the\ particle\ p_i \qquad \textbf{(1)}$$

In this programming project, after obtaining an NxN matrix, rows represent the equation (1), and the algorithm of the program solves all the new positions of the N particles in the domain of $[0,1]$, using Gauss elimination and Gauss-Seidel method, respectively.

2. HAND CALCULATIONS

Since p_0 is the most-left positioned particle,

$$x'_0 = f(t) = 0.25\,x\sin(\pi t) \tag{2}$$

Furthermore, because the most-right positioned particle is not moving,

$$x'_{N-1} = x_{N-1} \tag{3}$$

Now assume $i = 1$. Then according to equation (1),

$$x'_2 - 2x'_1 + x'_0 = 0 \tag{4}$$

Similarly, $i = 2$

$$x'_3 - 2x'_2 + x'_1 = 0 \tag{5}$$

So, combining the linear equations derived above, one may obtain $Ax = b$

- A is an NxN matrix system consisting of the coefficients of x',
- x is a vector, $x^T = [x'_0, x'_1 \dots \dots x'_{N-1}]$,
- b is the right-hand side of linear equations.

As a result, $Ax = b$ is equal to:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & 0 & 1 & -2 & 1 \\ 0 & \ddots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_0 \\ x'_1 \\ x'_2 \\ x'_3 \\ \vdots \\ \vdots \\ x'_{N-1} \end{bmatrix} = \begin{bmatrix} f(t) \\ 0 \\ 0 \\ 0 \\ \vdots \\ \vdots \\ x'_{N-1} \end{bmatrix}$$

3. NUMERICAL RESULTS

N=10 (constant) with increasing time t(s):

$Particle\ p_i$	$x'\ at\ t = 0s$	$x'\ at\ t = 0.25s$	$x'\ at\ t = 0.5s$	$x'\ at\ t = 0.75s$	$x'\ at\ t = 1.0s$
0	0.000	0.177	0.250	0.177	0.000
1	0.111	0.268	0.333	0.268	0.111
2	0.222	0.360	0.417	0.360	0.222
3	0.333	0.451	0.500	0.451	0.333
4	0.444	0.543	0.583	0.543	0.444
5	0.556	0.634	0.667	0.634	0.556
6	0.667	0.726	0.750	0.726	0.667
7	0.778	0.817	0.833	0.817	0.778
8	0.889	0.909	0.917	0.909	0.889
9	1.000	1.000	1.000	1.000	1.000

$t = 0.1667s$ (constant) with increasing N:

N = 5	0.13	0.34	0.56	0.78	1.00										
N = 10	0.13	0.22	0.32	0.42	0.51	0.61	0.71	0.81	0.90	1.00					
N = 15	0.13	0.19	0.25	0.31	0.38	0.44	0.50	0.56	0.63	0.69	0.75	0.81	0.88	0.94	1.00

4. GRAPHICS OF NUMERICAL RESULTS

Location of the ten particles vs. time:

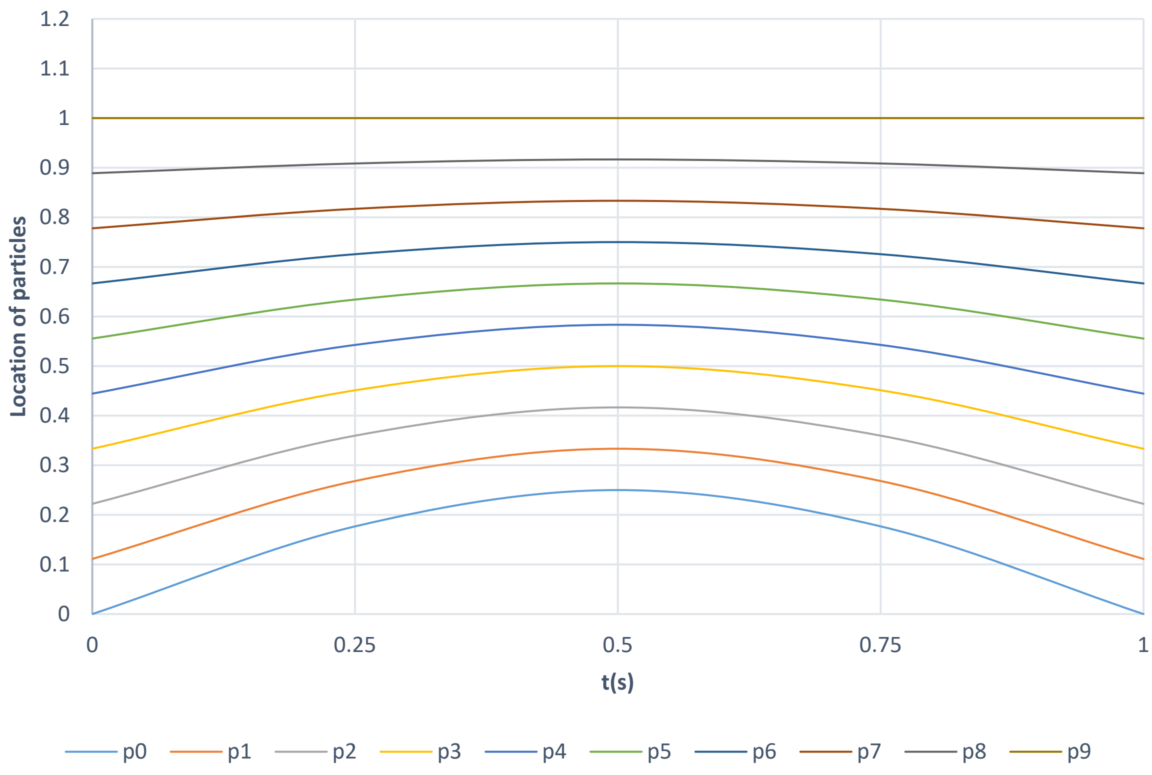

Figure 1 Location of the particles x' vs. time t(s)

Location of particles vs. increasing number of particles:

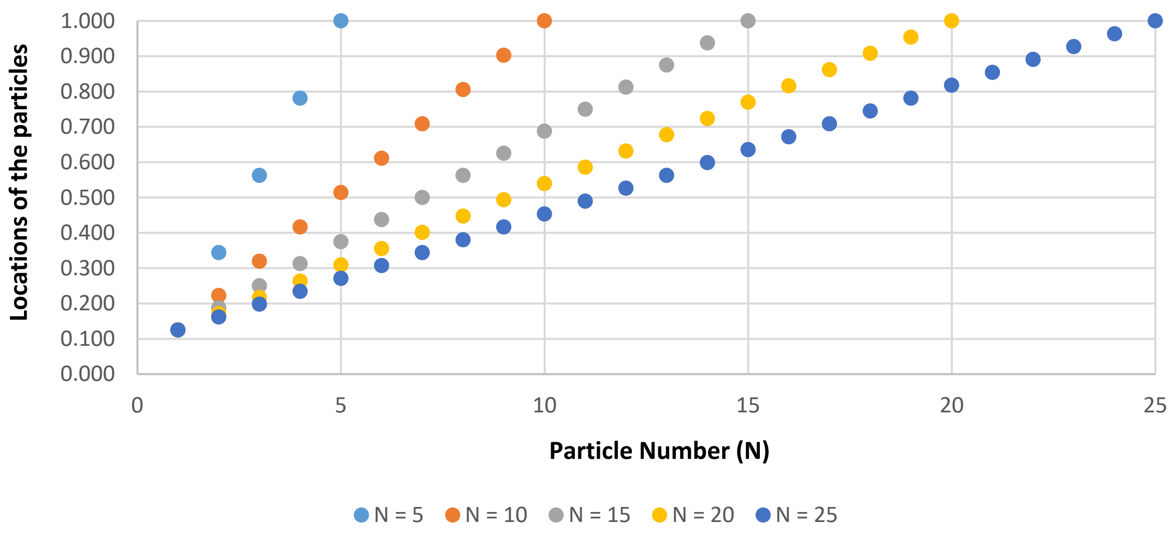

Figure 2 Location of the particles x’ vs. particle number N

5. DISCUSSION AND CONCLUSION

The problem is to find new positions of each particle in the domain that moves according to the function $f(t) = 0.25\,x\sin(\pi t)$, and it is found that each new position of the particles with neighbor particles creates a linear system. Therefore, solving the matrix that includes these linear systems resolves the particles' positions. The purpose of a sample code might be using the naive Gauss elimination method and Gause-Seidel method, solving $NxN\ \ matrix,$ which is found in the introduction part. However, this way needs a lot of storage, so a better way than solving with matrix might be solving with just simple equations such as for Seidel method instead of doing:

```
for i in range(N):
     SUM = 0
     for k in range(N):
         if i != k:
             SUM += x[k]*A[i][k]
     x[i] = (b[i]-SUM)/A[i][i]
```

This calculation we can do this simpler:

```
for i in range(1,N-1):
     x[i] = (x[i-1]+x[i+1])/2
```

The code is executed in two situations: The first one is, increasing the time that

$t(s) \in [0,1]$, and the second one is while t is constant, $t = 0.1667\ s$, N is increasing. For the first case, both methods found the same results. From figure 1, distances between particles do not change because they follow a sinusoidal trajectory except for the last particle. Therefore, the previous distance and the following distance between particles are equal to each other except between the particles $p_{N-2}\ and\ p_{N-1}$. Furthermore, due to sinusoidal movement, the beginning and the final position $(at\ t = 0s\ and\ t = 1s)$ of the particles and also, at $t = 0.25s, t = 0.75s$ are the same. For the second case, again, both methods found the same result. While using the Gauss-Seidel method, with an increasing number of particles, the number of iterations also should be increased. Otherwise, $\epsilon_a\%$ increases and the results get slightly different from naive Gauss elimination results. Entering a small tolerance value causes fewer iterations, leading to different results compared to the true solution.

To compare these two methods, even though both methods provide the same results, the naive Gauss elimination method causes a longer runtime than the Gauss-Seidel method. In this project, the matrix system is not large; in other words, the number of linear equations is not significant to consider runtime; however, for large systems, using the Gauss-Seidel method is more reasonable and provides less runtime in the programming operations.

Finally, in addition to the matrix solution, solving the linear equations by hand, it is found that there is a relation between the position of particles such as:

$$x_i = \frac{\frac{f(t)}{i-1} + x_{i+1}}{\frac{i}{i-1}}$$

When this relation is executed by coding, it prevents storing sparces in the matrix and provides less time consumption and more efficiency. Therefore, it derives a better solution compared to the Gauss elimination method.

7. Appendices

Code of the Program

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 25 21:44:41 2022

@author: Murat Berk      2377653
         Gökberk Çiçek   2377810
"""
import numpy as np
import itertools
import math

def f(x):                        ####♥ Definition of the first particle
function
    return 0.25*math.sin(np.pi*x)

N = input('Enter the number of particles: ')          ### Input taking
N = int(N)
t = input('Enter the time instant: ')                 ### Input taking
t = float(t)
Tol = input('Enter the tolerance to terminate: ')     ### Input taking
Tol = float(Tol)
Niter = input('Enter the maximum number of iterations to terminate: ')
Niter = int(Niter)



temp = itertools.count(1)
M = [[next(temp) for i in range(N)] for i in range(N)]   ##Creating
coefficient matrix
r = [[next(temp) for i in range(1)] for i in range(N)]   ##Creating result
matrix

"""
Coefficient Matrix is created below
"""
for i in range(N):
    for k in range(N):
        M[i][k] = 0

for i in range(N):
    for k in range(N):
        if i == 0:
            M[i][i] = 1
            if k != 0:
                M[i][k] = 0
        elif i == N-1:
            M[i][i] = 1
            if k != N-1:
                M[i][k] = 0
        else:
            M[i][i-1] = 1
            M[i][i] = -2
            M[i][i+1] = 1

"""
Result Matrix is created below
"""
for i in range(N):
    if i == 0:
        r[i] = f(t)
    elif i == N-1:
        r[i] = 1
    else:
        r[i]= 0

"""
Gauss Elimination operations (using a lot of storage)
                              (Less storage using code provided as
                               comment at the bottom of the code)
"""

# forward elimination
def forward(A, b):

  # go ever each row except the last one (pivots)
    for k in range(0,N-1,1):  # total N-1
      # go over from row k to n (row number of elimination)
      for i in range(k+1,N,1):   #  total n-k

        factor = A[i][k]/A[k][k]  # 1 divison
        # go over each column (column number for elimination)
        for j in range(k,N,1):
          A[i][j] = A[i][j] - factor*A[k][j]  # n-k x and -

        b[i] = b[i] - factor*b[k] # 1 -> x and -
    return (A, b)

def back(A, b):

    x = np.zeros_like(b)
    x[-1] = b[-1]/A[-1][-1]

    # go over each row to n
    for i in range(N-2,-1,-1):
      sum = 0.0
      #go over each column
      for j in range(i+1,N,1):
        sum = sum + A[i][j]*x[j]

      x[i] = (b[i] - sum)/A[i][i]

    return x
```

```
"""
Gauss Seidel operations       (using a lot of storage)
                              (Less storage using code provided as
                               comment at the bottom of the code)
"""
def seidel(A, b):
    n = 0
    x = np.zeros_like(b)
    error_max = 100000
    error_list = [ 0 for i in range(N)]
    x_old = [ 0 for i in range(N)]
    while n < (Niter+1) and error_max > Tol:
        n += 1
        for i in range(N):
            x_old[i] = x[i]

        for i in range(N):
            SUM = 0
            for k in range(N):
                if i != k:
                    SUM += x[k]*A[i][k]
            x[i] = (b[i]-SUM)/A[i][i]
        for i in range(N):
            if x[i] != 0:  ### not to calculate error with initial
condition t = 0
                error_list[i] = abs(x[i]-x_old[i]) / x[i]
                error_max = max(error_list)

    return  x, n

print('for gaus elimination answers are: ')
(A,b) = forward(M,r)
x = back(M, r)
for i in range(N):
    print ('x',i,' :%.10g' %x[i])


x, n = seidel(M, r)
print('for gaus-seidel with',n,'iteration answers are: ')
for i in range(N):
    print ('x',i,' :%.10g' %x[i])



"""
###### We can REDUCE the STORAGE USEAGE by implamenting simply this code:
def GaussElim():
    x = [ 0 for i in range(N)]
    x[0] = f(t)    ### First particle position
    x[N-1] = 1     ### Last particle position
    print(x)
    for i in range(1, N-1):
        x[N-(i+1)] = (f(t)/(N-(i+1)) + x[N-(i)]) / ((N - i) / (N - (i+1)))
    return x
"""
```

```
"""
###### We can REDUCE the STORAGE USEAGE by implamenting simply this code:
def seidel():
    n = 0
    x = [ 0 for i in range(N)]
    error_max = 100000
    error_list = [ 0 for i in range(N)]
    x_old = [ 0 for i in range(N)]
    x[0] = f(t)    ### First particle position
    x[N-1] = 1 ### Last particle position
    while n < (Niter+1) and error_max > Tol:
        n += 1
        for i in range(N):
            x_old[i] = x[i]

        for i in range(1,N-1):
            x[i] = (x[i-1]+x[i+1])/2

        for i in range(N):
            if x[i] != 0: ### not to calculate error with initial condition
t = 0
                error_list[i] = abs(x[i]-x_old[i]) / x[i]
                error_max = max(error_list)
    return  x, n
"""
```