AEM-ADV08
Introduction to Computational Linear Algebra
Coursework 1

Student Name: Ismael El Houas Ghouddana CID: 01539303

Due date: Friday, 8 February 2019 (Week 19).

You should submit the following documents for this coursework:

1. A professionally-typed pdf document (no hand-writing, or smartphone photos of hand-written solutions), formatted in Latex or MS Word using the provided templates, containing answers to all the questions. Do not modify the margins, use a font size of 12 and use a line spacing set at 1.15. You must answer the questions in order as they are in this document. Include any Matlab functions and scripts (appropriately commented to facilitate reading) used to attempt the coursework **within** your answers. If you are using the Latex template, the matlab code can be included with the command **lstinputlisting**{yourfile.m} Each answer should be written below the corresponding question. Upload the final documents with all the Matlab scripts in a compressed folder names **your_username.zip**, having one subfolder per question call Q1scripts Q3scripts...

You should write your own individual code and answers to questions. The programs and question answers will be checked for similarities.

Questions have different weight as indicated at the start of each question.

1 Question 1: 25% of total

Consider Gaussian elimination with partial pivoting $PA = LU$ for a generic square matrix A of size $n \times n$ in the real number space.

- What is the largest absolute value that the coefficients of L can take? Reason and comment your answer.
- We define the growth factor as

$$\rho = \frac{max_{i,j=1,2,.....,n}|U_{ij}|}{max_{i,j=1,2,.....,n}|A_{ij}|}$$

  Show that $\rho \leq 2^{n-1}$.

- The MATLAB code for the upper-triangulisation of a generic square matrix A of size $n \times n$ is given in the blackboard. Building on these functions, write your own scripts for: (i) lower triangulisation, (ii) forward and backward substitutions, (iii) partial pivoting process (i.e. swapping of rows through matrix multiplication). The scripts must be clear with proper comments and they must be presented in this report. Using these scripts, compute P, L and U and the unknown vector $\boldsymbol{x}$ for the following matrix-vector system $A\boldsymbol{x} = \boldsymbol{b}$:

$$\begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ -1 \\ -3 \end{bmatrix}$$

- Find ρ for the above square matrix A using the formula given in point 2. Apply your MATLAB scripts (with partial pivoting) to find the LU factorisations for the matrix of the form given above but for n=8,16 and 32. For all n values systematically record the corresponding matrices L, U and P. Then, find ρ and comment your observation on the behavior of ρ with respect to n.
- For the above problem (i.e. the square matrix A of the form of point 3 but with n=8,16 and 32), evaluate the variation of the following parameters with respect to n: (i) $|||L||U|||_\infty$ and (ii) $\alpha_{ALG} = \frac{|||L||U|||_\infty}{\|A\|_\infty}$. What do you infer from this variations and how do you relate these variations with partial pivoting?

Question 1: Answer

1. **Largest values of L**

   In this question, a Gaussian elimination with partial pivoting is being considered $PA = LU$ for a square matrix A of size nxn.

   Then, the method uses **LU factorisation**, where A is factored into a unit lower triangular L and an upper triangular matrix U.

   The lower triangular matrix L of size n is made with n-1 column vectors with the whole main diagonals with 1:

$$L_k = J_1^{-1} J_2^{-1} \dots J_{k-1}^{-1} \tag{1}$$

   where,

$$J_k = I_n - I_k e_k^T and \begin{bmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1} \\ \vdots \\ l_n \end{bmatrix} \tag{2}$$

   Therefore, L_{k+1} is given by the following matrix (the whole process to obtain could be seen in Lecture 4):

$$L_{k+1} = \begin{pmatrix} 1 & & & & & & & & & \\ l_{2,1} & 1 & & & & & & & & \\ l_{3,1} & l_{3,2} & 1 & & & & & & & \\ \vdots & \vdots & \ddots & \ddots & & & & & & \\ \vdots & \vdots & & \ddots & 1 & & & & & \\ \vdots & \vdots & & & l_{k+1,k} & 1 & & & & \\ \vdots & \vdots & & & \vdots & 0 & 1 & & & \\ \vdots & \vdots & & & \vdots & \vdots & 0 & \ddots & & \\ \vdots & \vdots & & & \vdots & \vdots & & \ddots & \ddots & \\ l_{n,1} & l_{n,1} & \dots & \dots & l_{n,k} & 0 & \dots & \dots & 0 & 1 \end{pmatrix}$$

For partial pivoting in the Gaussian elimination method, the highest absolute value of each column has to be found because whch will become the pivot.

As $l_{ik} = \frac{a_{ik}}{a_{kk}}$ with a_{kk} being the pivot which is the greatest absolute value of the column. As the main diagonal is formed by 1 and the condition for Gaussian elimination is $|l_{ik}| \leq 1$, it can be assumed that **the largest absolute value that the coefficients of L can take is 1**.

2. **Show that for the growth factor $\rho \leq 2^{n-1}$.**

The growth factor, also called Wilkinson growth factor, is a key quantity in the roundoff error analysis which is defined as:

$$\rho = \frac{max_{i,j=1,2,.....,n}|U_{ij}|}{max_{i,j=1,2,.....,n}|A_{ij}|}$$

If ρ the algorithm for the elimination method becomes unstable. As Gaussian elimination with partial pivoting is unstable due to the accumulation of errors associated with large elements found on the elimination process.

Considering Neville elimination which is an alternative procedure to Gaussian elimination where zeros in a column of a matrix are obtained by adding to each row a multiple of the previous one. If A is an nxn matrix, it consists of at most n - 1 successive major steps.

$$A = A^{(1)} \to \tilde{A}^{(1)} \to A^{(2)} \to \tilde{A}^{(2)} \to \cdots \to A^{(n)} \to \tilde{A}^{(n)} = U \tag{3}$$

A is a square matrix, then $A^{(t)}$has only zeros bellows the main diagonal.

We introduce now the elements: $p_{ij} = \tilde{a}_{ij}$ with $1 \leq \text{i} \leq \text{j} \leq \text{n}$

We now define:

$$m_{ij} = \begin{cases} \frac{\tilde{a}_{ij}}{\tilde{a}_{i-1j}} = \frac{\tilde{p}_{ij}}{\tilde{p}_{i-1j}}, if \tilde{a}^{(j)}_{i-1j} \neq 0 \\ 0, if \tilde{a}^{(j)}_{i-1j} = 0 \end{cases} \tag{4}$$

Therefore:

$$a^{(t+1)}_{i-1j} = \tilde{a}^{(t+1)}_{i-1j} - m_{it}\tilde{a}^{(t)}_{i-1j} \tag{5}$$

And:

$$|a_{i-1j}^{(t+1)}| \leq |\tilde{a}_{i-1j}^{(t+1)}| + |m_{it}||\tilde{a}_{i-1j}^{(t)}| \leq |\tilde{a}_{i-1j}^{(t)}| + |\tilde{a}_{i-1j}^{(t)}| \tag{6}$$

We can then affirm:

$$|\tilde{a}_{ij}^{(t+1)}| \leq 2max_{i,j}|A_{ij}^{(t)}| \tag{7}$$

Therefore :

$$\rho = \frac{max_{i,j,k}|a_{ij}^{(k)}|}{max_{i,j}|a_{ij}|} \leq \frac{2^{n-1}max_{i,j}|a_{ij}|}{max_{i,j}|a_{ij}|} = 2^{n-1}$$

For k = n,

$$\rho = \frac{max_{i,j}|U_{ij}|}{max_{i,j}|A_{ij}|}$$

Then,

$$\rho = \frac{max_{i,j}|U_{ij}|}{max_{i,j}|A_{ij}|} \leq \frac{2^{n-1}max_{i,j}|a_{ij}|}{max_{i,j}|a_{ij}|} = 2^{n-1}$$

3. **Solve Linear system**

Then, we need to solve the following linear system:

$$\begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ -1 \\ -3 \end{bmatrix}$$

In order to solve it, we need to use Gaussian-elimination method using matrix operation. So, we need to obtain the matrices P, L and U using the functions **upper**

triangulisation, **lower triangulisation** and **partial pivoting**. The scripts for these functions are presented as follows:

Upper Triangulisation

This function converts a matrix in upper triangular.

```
function U = upper_triangulisation(A,n)
% The function computes the upper triangularisation of a given square
    matrix A using the Gauss elimination method (GEM).
% A - the coefficient matrix
% n - the size of the matrix A
for i=1:n %lines 5-15: initialisation of the identity (I) and upper-
    triangular (U) matrices
     for j=1:n
         if (i==j)
             I(i,j)=1.0;
             U(i,j) = A(i,j);
         else
         I(i,j) = 0.0;
         U(i,j) = A(i,j);
end
end
end

for k=1:n-1 % the main loop of the GEM starts here
gv = gauss_vector(U,k,n); % the first step in GEM is to construct Gauss
    vector
evec = e_vector(k,n); %second, I construct my e vector for the
    corresponding stage of the GEM
LE = outer_product(gv,evec,n); %Here, I take outer product between the
    Gauss and e vectors for the given stage
J = matrix_subtraction(I,LE,n); %Here, I construct my Gauss
    transformation matrix by subtracting the previously computed outer
    product from the identity matrix
U = matrix_matrix_mult(J,U,n); %Here, I compute the updated upper-
    triangular matrix through matrix multiplication with the Gauss
    transformation matrix.
end % the main loop of the GEM ends here
end
```

Lower Triangulisation

This function converts a matrix in lower triangular.

```
function L = Lower_triangulisation(A,n)

% The function computes the lower triangularisation of a given
%square matrix A using the Gauss elimination method (GEM).
% A - the coefficient matrix
% n - the size of the matrix A
%Initialisation of the identity (I) and upper-triangular (U) matrices

for i=1:n
    for j=1:n
        if (i==j)
            I(i,j)=1.0;
            U(i,j) = A(i,j);
        else
            I(i,j) = 0.0;
            U(i,j) = A(i,j);
        end
    end
end

%The Lower triangularisation matrix needs to be computed by adding some
%more steps to the uppertriangularisation code.

L=I;  %Initialize L as the identity matrix

for k=1:n-1 % the main loop of the GEM starts here

% First step in GEM is to construct Gauss vector
gv = gauss_vector(U,k,n);

%Secondly, I construct e vector for the corresponding stage of the GEM
evec = e_vector(k,n);

%Here, I take outer product between the Gauss and e vectors for
%the given stage
LE = outer_product(gv,evec,n);

%Here, I construct the Gauss transformation matrix by subtracting
%the previously computed outer product from the identity matrix
J = matrix_subtraction(I,LE,n);

%Here, I compute the inverse of the matrix J in order to calculate L
J_inv=inv(J);
```

```
44
45 %We compute the updated upper-triangular matrix through matrix
46 %multiplication with the Gauss transformation matrix.
47 %It will be used in the loop to calculate J and therefore: L
48 U = matrix_matrix_mult(J,U,n);
49
50 %According to lecture 4, L=inv(J1)*...*inv(k), with k?n-1
51 L=matrix_matrix_mult(L,J_inv,n);
52
53 end % the main loop of the GEM ends here
54 end
```

Forward Substitution

We use the forward substitution to calculate an unknown column vector y, through the equation: Ly=b, with L as a lower-triangular matrix and b a column vector.

```
1  function b = Forward_substitution(L,b,n)
2
3  %LECTURE 3
4
5  %This program compute solution "y" with the forward substitution
6  % applying Ly=b with L as a lower triangular matrix.
7
8  b(1)=b(1)/L(1,1);
9
10 for i=2:n
11
12 %We start with the first row which contains only one value so there is
13 %only one division.
14
15 %We compute xi as a function of the values of xj, j<i, as they
16 % have been calculated in the previous steps. It is not the real
17 % value of xi yet.
18
19 b(i) = (b(i)-inner_product(L(i,1:i-1),b(1:i-1),i-1))/L(i,i);
20
21
22 %Forward substitution is an O(n^{2}) process
23
24 end
25
26 end
```

Backward Substitution

We use the backward substitution to calculate an unknown column vector solution x, through the equation: Ux=b, with U as an upper-triangular matrix and b a column vector.

```
function b = Backward_substitution(U,b,n)

%LECTURE 3

%This program compute solution "y" with the forward substitution applying
% Ux=b with U as a upper triangular matrix.


%We compute the last element first, which is the n element to avoid
%singularity
b(n) = b(n)/U(n,n);

%Loop backwards to go from the last row
for i=n-1:-1:1

%We compute xi as a function of the values of xj, j>i, as
%they have been calculated in the previous steps. It is not
%the real value of xi yet.

b(i) = (b(i)-inner_product(U(i,i+1:n),b(i+1:n),n-i))/U(i,i);

end

end
```

Partial Pivoting

Pivoting is based in the process of swapping rows by the multiplication of a matrix with a permutation matrix (in our case called P).

```
function [I] = Partial_pivoting(A,k,n)

%This script compute the matrix I (in general P, the permutation matrix)
%which swaps rows.
%'a' and 'b' are the positions of the rows that we want to swap.

%First, we define I as the identity matrix.
I=eye(n);

%We search the pivot for the column j
a=k;
max=0;

for j=k:n
    if abs(A(j,k))>max
        max=abs(A(j,k));
        index=j;
    end
end

b=index;

%The justification of this process is explained with more details
%in the report.

I(a,a)=0;%the coefficient 'Iaa' and 'Ibb' are now 0.
I(a,b)=1;%'Iab' is now 1.
I(b,b)=0;
I(b,a)=1;%And 'Iba'=1.

%We multiply the matrix for which we want to swap rows with I
%and get A with rows 'a' and 'b' switched.
end
```

Main program

```
clear all
%%CLA_Q1

%Declare the matrix of the system Ax=b given in the coursework

A=[1 0 0 0 1; -1 1 0 0 1; -1 -1 1 0 1; -1 -1 -1 1 1; -1 -1 -1 -1 1];
b=[2;1;0;-1;-3];

%Get size of matrix A
[~,n]=size(A);

I=eye(n);          %Initialize the identity matrix
U=A;               %Initialize A matrix as U matrix before modifying it
Po=eye(n);         %Initialize the permutation matrix as identity matrix
L=zeros(n);        %Initialize the lower triangular matrix

for k=1:n

P=Partial_pivoting(U,k,n); %Perform partial pivoting to obtain
                           %the permutation matrix
U=P*U;                     %Compute the U matrix with pivoting matrix
L=P*L;                     %Compute the L matrix with pivoting matrix
Po = P*Po;

gv = gauss_vector(U,k,n); % the first step in GEM is to construct Gauss
                           %vector
evec = e_vector(k,n); %construct e vector for the corresponding stage
                      %of the GEM
LE = outer_product(gv,evec,n); %Here, I take outer product between the
                               %Gauss and e vectors for the given stage
J = matrix_subtraction(I,LE,n); %Here, I construct my Gauss
    transformation
                           %matrix by subtracting the previously
                           %computed outer product from the identity
                              matrix

U = matrix_matrix_mult(J,U,n); %Here, I compute the updated
                     %upper-triangular matrix through matrix
                        multiplication
                     %with the Gauss transformation matrix.
L=L+LE;
```

```
39 end
40
41 %L is the sum of l and e vectors
42 L=L+eye(n);
43
44 %We now calculate LY=Pb
45 y = Forward_substitution(L,P*b,n);
46
47 %Then, Ux=Y
48 x = Backward_substitution(U,y,n);
49
50 %And we finally get the unknown vector X
```

Then, we get the matrices U, L and P and the solution column vector x:

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 0 & 16 \end{bmatrix} L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Growth factor

In this section we need to find the LU factorisation for the matrix of the form given above but for n =8,16 and 32. Then, we are going to compute the L, U and P matrices for the different values of n.

The growth factor indicates the instability of the Gaussian elimination method because it indicates the worst scenario matrix. Also, it is bigger as the values inside the matrix have different order of magnitude.

The U matrix has a shape as:

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 & 2^0 \\ 0 & 1 & 0 & 0 & 2^1 \\ \vdots & 0 & \ddots & 0 & \vdots \\ \vdots & \vdots & \vdots & \ddots & 2^{n-2} \\ 0 & 0 & 0 & 0 & 2^{n-1} \end{bmatrix} \tag{8}$$

Then, we already know what are going to be the values of U and the growth factor as the it is defined as

$$\rho = \frac{max_{i,j}|U_{ij}|}{max_{i,j}|A_{ij}|} \tag{9}$$

In the Matlab script GrowthFactor.m presented below, the code for compute the growth factor is presented (the user could change the n value, in this case is n=8), it uses the code from the previous exercise to obtain the matrices L, U and P and afterwards, compute ρ:

Note that the as the size of matrix L, U and P is too big to show them on the report, they have been recorded and saved in the subfolder Matrices inside the folder for exercise 1, Q1.m

```
clear all;

%Define size of matrix A
n=8;

%We create A as a function of 'n'

A = eye(n);
```

```
A = A+tril(-ones(n,n),-1);

for i=1:n
    A(i,n) = 1;
end

[~,n]=size(A);

Pivf=eye(n);          %Initialize the permutation matrix as identity matrix
L=zeros(n);           %Initialize the lower triangular matrix
U=A;                  %Initialize A matrix as U matrix before modifying it
I=eye(n);             %Initialize the identity matrix

for k=1:n

P=Partial_pivoting(U,k,n); %Perform partial pivoting to obtain
                           %the permutation matrix
U=P*U;                     %Compute the U matrix with pivoting matrix
L=P*L;                     %Compute the L matrix with pivoting matrix
Pivf = P*Pivf;
gv = gauss_vector(U,k,n); % the first step in GEM is to construct Gauss
                          %vector
evec = e_vector(k,n); %construct e vector for the corresponding stage
                      %of the GEM
LE = outer_product(gv,evec,n); %Here, I take outer product between the
                               %Gauss and e vectors for the given stage
J = matrix_subtraction(I,LE,n); %Here, I construct my Gauss
   transformation
                           %matrix by subtracting the previously
                           %computed outer product from the identity
                              matrix

U = matrix_matrix_mult(J,U,n); %Here, I compute the updated
                    %upper-triangular matrix through matrix
                       multiplication
                    %with the Gauss transformation matrix.
L=L+LE;
end

%L is the sum of l and e vectors
L=L+eye(n);

%We calculate the maximum absolute value of U
```

```
U_max = max(max(abs(U)));
%We calculate the maximum absolute value of A
A_max = max(max(abs(A)));
%Thus, we get rho according to the formula
rho = U_max/A_max;
```

We can assure for that the defined matrix A the growth factor follows the mathematical definition:

$$\rho = 2^n - 1 \tag{10}$$

The results are in Table I:

n	ρ **Numerical**	ρ **Theoretical**
8	128	128
16	32,768	32,768
32	2,147,483,648	2,147,483,648

Table I: Growth factor values for different size matrices.

Note that the growth factor grows exponentially with the size of n and we have obtained the greatest value for a matrix of this size.

4. **Norms**

$|||L||U|||_\infty$:

In the Matlab script NormLU.m presented below, uses the code from the previous exercise to obtain the matrices L, U and P and afterwards, sums the coefficients of the product of $|||L||U|||_\infty$ line by line and returns the largest value of the sum.

$$||A||_\infty = max_{1 \leq i \leq n} \sum_{j=1}^{n} |A_{ij}| \tag{11}$$

```
clear all;
%Define size of matrix A
n=16;

%We create A as a function of 'n'

A = eye(n);
```

```
A = A+tril(-ones(n,n),-1);

for i=1:n
    A(i,n) = 1;
end

[~,n]=size(A);

Pivf=eye(n);          %Initialize the permutation matrix as identity matrix
L=zeros(n);           %Initialize the lower triangular matrix
U=A;                  %Initialize A matrix as U matrix before modifying it
I=eye(n);             %Initialize the identity matrix

for k=1:n

P=Partial_pivoting(U,k,n); %Perform partial pivoting to obtain
                           %the permutation matrix
U=P*U;                     %Compute the U matrix with pivoting matrix
L=P*L;                     %Compute the L matrix with pivoting matrix
Pivf = P*Pivf;
gv = gauss_vector(U,k,n); % the first step in GEM is to construct Gauss
                          %vector
evec = e_vector(k,n); %construct e vector for the corresponding stage
                      %of the GEM
LE = outer_product(gv,evec,n); %Here, I take outer product between the
                               %Gauss and e vectors for the given stage
J = matrix_subtraction(I,LE,n); %Here, I construct my Gauss
   transformation
                           %matrix by subtracting the previously
                           %computed outer product from the identity
                              matrix

U = matrix_matrix_mult(J,U,n); %Here, I compute the updated
                     %upper-triangular matrix through matrix
                        multiplication
                     %with the Gauss transformation matrix.
L=L+LE;
end

%L is the sum of l and e vectors
L=L+eye(n);

L_abs=abs(L);%We define |L|
```

```
48
49 U_abs=abs(U);%We define |U|
50
51 R=L_abs*U_abs; %We compute the product |L|.|U|
52
53 %We create the vector which will contain the value of the sum of each row
54 S = zeros(1,n);
55
56 %We sum the coefficents row by row
57
58 for i=1:n
59     v=0;
60         for j=1:n
61             v=v+R(i,j);
62         end
63
64         %The value of the sum of the ith row is saved as the ith value of
               c
65 S(i)=v;
66 end
67 %We get the highest value
68 m=max(S) ;
```

n	$\|\|\|L\|\|U\|\|\|_\infty$
8	262
16	65,550
32	4,294,967,326

Table II: Growth factor values for different size matrices.

$\alpha_{ALG} = \frac{\|\|\|L\|\|U\|\|\|_\infty}{\|A\|_\infty}$:

In the Matlab script AlphaNorm.m presented below, uses the code from the previous exercise and adds the sums the coefficients of $|A|_\infty$ line by line and returns the largest value of the sum. Ten, divides the product of $\|\|\|L\|\|U\|\|\|_\infty$ by $|A|_\infty$ to obtain alpha for the different values of n

```
1 clear all;
2 %Define size of matrix A
3 n=8;
4
5 %We create A as a function of 'n'
6
```

```
A = eye(n);
A = A+tril(-ones(n,n),-1);

for i=1:n
    A(i,n) = 1;
end

[~,n]=size(A);

Pivf=eye(n);          %Initialize the permutation matrix as identity matrix
L=zeros(n);           %Initialize the lower triangular matrix
U=A;                  %Initialize A matrix as U matrix before modifying it
I=eye(n);             %Initialize the identity matrix

for k=1:n

P=Partial_pivoting(U,k,n); %Perform partial pivoting to obtain
                           %the permutation matrix
U=P*U;                     %Compute the U matrix with pivoting matrix
L=P*L;                     %Compute the L matrix with pivoting matrix
Pivf = P*Pivf;
gv = gauss_vector(U,k,n); % the first step in GEM is to construct Gauss
                          %vector
evec = e_vector(k,n); %construct e vector for the corresponding stage
                      %of the GEM
LE = outer_product(gv,evec,n); %Here, I take outer product between the
                               %Gauss and e vectors for the given stage
J = matrix_subtraction(I,LE,n); %Here, I construct my Gauss
    transformation
                           %matrix by subtracting the previously
                           %computed outer product from the identity
                              matrix

U = matrix_matrix_mult(J,U,n); %Here, I compute the updated
                     %upper-triangular matrix through matrix
                        multiplication
                     %with the Gauss transformation matrix.
L=L+LE;
end

%L is the sum of l and e vectors
L=L+eye(n);

```

```
47 L_abs=abs(L);%We define |L|
48
49 U_abs=abs(U);%We define |U|
50
51 R=L_abs*U_abs; %We compute the product |L|.|U|
52
53 %We create the vector which will contain the value of the sum of each row
54 S = zeros(1,n);
55
56 %We sum the coefficents row by row
57
58 for i=1:n
59     v=0;
60         for j=1:n
61             v=v+R(i,j);
62         end
63
64         %The value of the sum of the ith row is saved as the ith value of
               c
65 S(i)=v;
66 end
67
68 %We get the highest value at S_max
69 S_max=max(S);
70
71
72 A_abs=abs(A);%We compute its absolute value
73
74 %We create the Sum vector which will contain the value of the sum of
75 %each row
76
77 T = zeros(1,n);
78 %We sum the coefficents row by row
79
80 for i=1:n
81     c=0;
82         for j=1:n
83             c=c+A_abs(i,j);
84         end
85
86         %The value of the sum of the ith row is saved as the ith value of
               c
87 T(i)=c;
```

```
88 end
89
90 Norm_max = max(T); %n is the result of the matrix norm for A
91 alpha = S_max/Norm_max; %alpha is the formula set by the coursework
       statement
```

n	$\alpha_{ALG} = \frac{\|\|L\|U\|\|_\infty}{\|A\|_\infty}$
8	32.75
16	4,096.875
32	134,217,728.9375

Table III: Growth factor values for different size matrices.

We could see how the two parameters grow exponentially as the size of the matrix increases.

2 Question 2: 25% of total

Let A be a non-singular $n \times n$ matrix. Explain how to efficiently solve the following problems using Gaussian elimination with complete pivoting. In each case, (i) describe the algorithm, (ii) give a pseudo-code, (iii) discuss complexity logically.

- Solve $A^k \boldsymbol{x} = \boldsymbol{b}$ where k is a positive integer.
- Solve the matrix equation $AX = B$, where X and B are matrices of size $n \times m$.

Question 2: Answer

In complete pivoting we have a quite similar algorithm as partial pivoting, but in this case each elimination step is preced with a permutation P_k of the rows applied on the left and also a permutation Q_1 on the right:

$$L_{m-1}P_{m-1}\dots L_1P_1Q_1\dots Q_{m-1} = U \tag{12}$$

As functions created in Q1 (Backward....) are used in the pseudo codes, we have only elaborate the content that is not treated in Q1.

- Solve $A^k \boldsymbol{x} = \boldsymbol{b}$ where k is a positive integer.

  A is a non-singular nxn matrix and the problem we need to solve is $A^k x = b$ with $k \geq 0$

  In order to solve it, we split the matrix A^k in the product of A with another matrix A^{k-1}.

$$A^k x = b \rightarrow AA^{k-1}x = b \tag{13}$$

  Then we set: $A^{k-1}x = x_1$ as we know how to compute x_0 from $Ax_1 = b$ through complete pivoting.

  After using the complete pivoting algorithm, we obtain the value of $x_1 = A^{k-1}x$. For more clarity we set: $x_1 = b_1$ which leads to the second equation:

$$x_1 = b_1 \rightarrow A^{k-1}x = b_1 \rightarrow AA^{k-2}x = b_1 \tag{14}$$

  The following relation is obtained: $A^{k-n}x = b_n$

  The process is repeated until n=k-1,so:

$$A^k - k - 1x = b_{k-1} \rightarrow Ax = b_{k-1} \tag{15}$$

  Through Gaussian elimination with complete pivoting we can compute X as we know A and b_{k-1}

  The pseudo code for complete pivoting is presented below:

Algorithm 1 Complete Pivoting

```
procedure Complete_Pivoting()
    U = A, L = I, P=I
    for k=1 to n-1
    Select i ≥ to maximize |u_ik| and |u_ki|
    u_{k,k:m} ↔ u_{i,k:m} (interchange two rows)
    u_{k:m,k} ↔ u_{k:m,i} (interchange two columns)
    l_{k,1:k-1} ↔ l_{i,1:k-1}
    p_{k,:} ↔ p_{i,:} (define permutation matrix P from swapping rows)
    q_{:,k} ↔ q_{:,k} (define permutation matrix P from swapping columns)
```

Algorithm 2 Compute X

```
1: procedure ComputeX
2:     While k ≥ 1
3:     x_1 = Backward Substitution(U,b,n) (Perform backward substitution function used in exercise 1)
4:
5:     b = x_1
6:
7:     k = k-1
8:     end
```

Complete pivoting has a complexity of $O(m * n^3)$. Then, as it is repeated k times the complexity is:

$$O(k * n^3)$$

If k < n we can affirm that the complexity is:

$$O(n^3)$$

If k ≥ n, then the complexity is:

$$O(n^4)$$

- Solve the matrix equation $Ax = B$, where x and B are matrices of size $n \times m$.

  For this problem we are going to apply the same process as the previous exercise, but we need to solve each column from 1 to m and afterwards perform backwards substitution.

Algorithm 3 Compute X non square

```
1: procedure ComputeXnonsquare
2:     [m,n] get size of A
3:     Complete Pivoting(A)
4:     for j=1 to m
5:     x_1(1:n,j) = Backward Substitution(U,b,n) (Perform backward substitution function used in Q1)
6:     end
```

Complete pivoting has a complexity of $O(m * n^3)$. Then, as it is repeated k times the complexity is:

$$O(k * n^3)$$

If m $\leq$ n we can affirm that the complexity is:

$$O(n^3)$$

If m $\geq$ n, then the complexity is:

$$O(n^4)$$

The conclusions from complete pivoting are:

- In complete pivoting, the selection of pivots adds significant cost to the Gaussian elimination process.
- It further imposes potential difficulties of global communication in an unpredictable pattern across all matrix entries
- In practice this is rarely done, because the improvement instability is marginal.
- Equally good pivots can be obtained from a much smaller set of entries.
- The standard method for doing this is partial pivoting.

Question 3: 20% of total

Consider a problem where a large region of a porous medium having a finite thickness L, and an initial concentration C_0 of a chemical specie is suddenly exposed on an environment having concentration 0 of the same chemical specie. The partial differential equation governing the mass transfer problem is given by the one dimensional form:

$$D\frac{\partial^2 C(x,t)}{\partial x^2} = \frac{\partial C(x,t)}{\partial t} \tag{16}$$

where D is the diffusivity, $C(x,t)$ is the concentration of the chemical specie along the x-direction at time t, having boundary conditions $C(0,t) = 0$, $C(L,t) = 0$, and initial condition $C(x,0) = C_0$.

i) Discretise this equation using a Forward-in-Time discretisation for the term $\frac{\partial C}{\partial t}(x,t)$ and a central Crank-Nicholson time discretisation scheme for the $D\frac{\partial^2 C(x,t)}{\partial x^2}$ term. Use

$i = 0, 1, 2..., N$, where C_i^n is the concentration at time n at points $x = i\Delta x$, $t = n\Delta t$, and $N+1$ is the number of grid points in the x direction. The equation does not need to be solved for $i = 0$ and $i = N$, because the concentration C is known to be fixed at 0. Write down the differential equation in its discretised form.

ii) Write the vector of concentration values as:

$$\boldsymbol{C^n} = (C_1^n, C_2^n, ..., C_{N-1}^n) \tag{17}$$

Now explain why the equations can be written in the form

$$\left(I + \frac{\sigma}{2}B\right)\boldsymbol{C^{n+1}} = \left(I - \frac{\sigma}{2}B\right)\boldsymbol{C^n}, \tag{18}$$

where I is the identity matrix and $\sigma = D\Delta t/(\Delta x)^2$, and B a tridiagonal matrix having all 2 on the main diagonal and -1 on the upper and lower bands.

iii) Write a MATLAB code Matrix1D.m to form the matrix B in *sparse matrix format*. Present your code in the report with sufficient comments to explain each line.

iv) An analytical solution for the problem is

$C(x,t) = 4C_0 \sum_{m=1}^{\infty} \frac{1}{(2m-1)\pi} e^{-\left(D(2m-1)^2\pi^2 t/L^2\right)} \sin(\frac{(2m-1)\pi x}{L})$.

For $N = 10$, $D = 1$, $L = 1$ and $C_0 = 1$ write a MATLAB code that implements this discretisation until $t = 0.1$s. The code should use a modified version, written for tridiagonal matrices, of the scripts spfa.m and spsl.m within the time stepping loop. The scripts spsl.m compute the solution of the generic system $Ax = b$ using the matrix U obtained via the script spfa.m which implements the Cholesky factorisation for symmetric positive definite matrices. The scripts should be saved in spfa_tridiag.m and spsl_tridiag.m respectively, and should be presented in the report with sufficient comments to explain each line.

Compare the converged solution of C with the analytical solution, calculated truncating the sum at $n = 100$, at different time steps producing 6 contour plots at various times.

Repeat the precedent step with an increased spatial discretisation $N = 40$.

Question 3: Answer

i) **Transient heat equation in its discretised form**

The partial differential equation governing the mass transfer problem is a parabolic

equation. Also, the problem is well defined as we have Dirichlet boundary conditions and the initial condition.

Once we are sure that the problem is well defined, we proceed to use the Crank-Nicholson numerical scheme to disretize the equation. This method is forward in time and central difference in space, as you can see in Fig.(1).

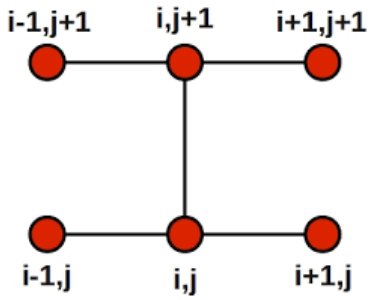

Figure 1: Crank-Nicholson numerical method.

The discretized form of the equation is:

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} = D\frac{(C_{i+1}^{n+1} - 2C_i^{n+1} + C_{i-1}^{n+1}) + (C_{i+1}^n - 2C_i^n + C_{i-1}^n)}{2\Delta x^2} \tag{19}$$

ii) **Discretized heat equation in matrix form**

We want to write our heat equation in a matrix forms as follows:

$$\left(I + \frac{\sigma}{2}B\right)\boldsymbol{C^{n+1}} = \left(I - \frac{\sigma}{2}B\right)\boldsymbol{C^n}, \tag{20}$$

First, we need to write the concentration terms in a column vector form.

Rearranging the discretized form of the equation we obtain:

$$\frac{C_i^{n+1}}{\Delta t} - D\frac{(C_{i+1}^{n+1} - 2C_i^{n+1} + C_{i-1}^{n+1})}{2\Delta x^2} = \frac{C_i^n}{\Delta t} + D\frac{(C_{i+1}^n - 2C_i^n + C_{i-1}^n)}{2\Delta x^2} \tag{21}$$

Then, we move the elements which depend on *n+1* to the left-hand side and the ones depending from n to the right-hand side

$$C_i^{n+1} - \frac{D\Delta t}{2\Delta x^2}(C_{i+1}^{n+1} - 2C_i^{n+1} + C_{i-1}^{n+1}) = C_i^n + \frac{D\Delta t}{2\Delta x^2}(C_{i+1}^n - 2C_i^n + C_{i-1}^n) \tag{22}$$

where $\sigma = \frac{D\Delta t}{\Delta x^2}$

$$C_i^{n+1} + \frac{\sigma}{2}(-C_{i+1}^{n+1} + 2C_i^{n+1} - C_{i-1}^{n+1}) = C_i^n - \frac{\sigma}{2}(-C_{i+1}^n + 2C_i^n - C_{i-1}^n) \qquad (23)$$

Define the equation in matrix form in order to get the concentration vectors.

$$\begin{pmatrix} C_i^{n+1} \\ \vdots \\ C_N^{n+1} \end{pmatrix} + \frac{\sigma}{2} \begin{pmatrix} -1 & 2 & -1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & -1 & 2 & -1 \end{pmatrix} \begin{pmatrix} C_{i-1}^{n+1} \\ C_i^{n+1} \\ C_{i+1}^{n+1} \end{pmatrix} = \begin{pmatrix} C_{i-1}^n \\ C_i^n \\ C_{i+1}^n \end{pmatrix} - \frac{\sigma}{2} \begin{pmatrix} -1 & 2 & -1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & -1 & 2 & -1 \end{pmatrix} \begin{pmatrix} C_i^n \\ \vdots \\ C_N^n \end{pmatrix} \qquad (24)$$

Now, we have a matrix equation quite similar to the Eq.(20).

Secondly, the boundary conditions of the domain are already defined at i=0 and i=N where the concentration value is 0. Then, the equation does not need to be solved for these points. The equation becomes:

$$\begin{pmatrix} C_1^{n+1} \\ \vdots \\ C_{N-1}^{n+1} \end{pmatrix} + \frac{\sigma}{2} \begin{pmatrix} 2 & -1 & \dots & 0 \\ -1 & 2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \dots & -1 & 2 \end{pmatrix} \begin{pmatrix} C_1^{n+1} \\ \vdots \\ C_{N-1}^{n+1} \end{pmatrix} = \begin{pmatrix} C_1^n \\ \vdots \\ C_{N-1}^n \end{pmatrix} - \frac{\sigma}{2} \begin{pmatrix} 2 & -1 & \dots & 0 \\ -1 & 2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \dots & -1 & 2 \end{pmatrix} \begin{pmatrix} C_1^n \\ \vdots \\ C_{N-1}^n \end{pmatrix} \qquad (25)$$

Finally, we take out the matrix common factor $\begin{pmatrix} C_1^{n+1} \\ \vdots \\ C_{N-1}^{n+1} \end{pmatrix}$ and $\begin{pmatrix} C_1^n \\ \vdots \\ C_{N-1}^n \end{pmatrix}$. We obtain the following matrix equation:

$$\left(I_{N-1} + \frac{\sigma}{2} \begin{pmatrix} 2 & -1 & \dots & 0 \\ -1 & 2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \dots & -1 & 2 \end{pmatrix} \right) \begin{pmatrix} C_1^{n+1} \\ \vdots \\ C_{N-1}^{n+1} \end{pmatrix} = \left(I_{N-1} - \frac{\sigma}{2} \begin{pmatrix} 2 & -1 & \dots & 0 \\ -1 & 2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \dots & -1 & 2 \end{pmatrix} \right) \begin{pmatrix} C_1^n \\ \vdots \\ C_{N-1}^n \end{pmatrix} \qquad (26)$$

where I is the identity matrix and $\sigma = D\Delta t/(\Delta x)^2$, and B a tridiagonal matrix having all 2 on the main diagonal and -1 on the upper and lower bands.

Then, we have proved that the equations could be written in the form of Eq.(20), that we were looking for.

iii) **Matrix B in sparse matrix format using function Matrix1D.m**

Matrix1D.m function creates a sparse matrix from B, which is a tridiagonal matrix and we are only interested in the non-zero values. The Matlab code is presented as follows:

```
%%Matrix1D function to compute sparse matrix B  with a defined number of
%%points n
function [B] = Matrix1D(n)
%The function receives the n points of the domain. As it is a square
%matrix, we have the same number of points in x and y diretion

%We create a column vector of ones of length n-1 in order to be able to
%define the diagonals of the matrix
Ones = ones(n-1, 1);

%Define tridiagonal matrix of size (n-1)X(n-1) with the main diagonal of
   2 and the upper and
%lower diagonal of -1

%We use the Matlab function diag, which allows us to create a matrix with
%a diagonal that where we can specify the length and the position.
%Then, combining the three matrices we obtain the tridiagonal matrix.

B = diag(2 * Ones, 0) - diag(Ones(1:n-2), -1) - diag(Ones(1:n-2), 1);

%Using Matlab sparse function to store pnly the non-zero values of the
   tridiagonal
%matrix
B = sparse(B);
end
```

Then, the tridiagonal matrix is transformed in a sparse matrix, which will speed up the computations because only the non-zero terms are used in the computations which are the values which have influence in the problem.

iv) **Analytical and numerical solution comparison**

In this section, the analytical and numerical solutions for the scheme are compared.

In the following Matlab script, the program used for compute the analytical and numerical solution and plot the results is presented.

Note that in order to take advantage of the tridiagonal matrix sorted as a sparse matrix, the functions *spfa-tridiag.m* and *spsl-tridiag.m* have been created. These scripts compute the solution of the generic system $Ax = b$ using the matrix U obtained via the script *spfa-tridiag.m* which implements the Cholesky factorisation for symmetric positive definite matrices.

As these scripts work with the sparse matrix B, they do not loop over the zero values, so the program runs faster than using the given scripts.

The code of these two functions is presented as follows:

spfa tridiag:

```
function A = spfa_tridiag(A)
%Returns the matrix U computed via the Cholesky factorisation
%of a symmetric definite positive matrix A, writing the values of U
%directly in the upper triangular part of A
%based on Dr Kevin Gouder script for AEM-ADV08 2016-2017

[m,n]=size(A);
if (m~=n || max(max(abs((A+A')/2-A)))~=0 )
        error('error: A is not symmetric');
end

%Use  function to compute only the non-zero values
nZ=find(A(1,1:n)~=0);

b=nZ(end)-1;
for j = 1:n
    s = 0.0;
        for k = max(1,j-b):j-1
            t = A(k,j);
            t = t/A(k,k);
            A(k,j) = t;
            s = s+t*t;
        end

        q = min(j+b,n);
        s = A(j,j)-s;
        if s<0
```

```
28                error('error: A is not definite positive');
29            end
30            A(j,j) = sqrt(s);
31    end
32
33    end
```

spsl tridiag:

```
1   function b = spsl_tridiag(U,b)
2
3   %Solve Ax=b for symmetric definite positive matrices after
4   %the the upper triangular matrix U was computed by spfa.m
5   %based on Dr Kevin Gouder script for AEM-ADV08 2016-2017
6   [~,n]=size(U);
7
8   %Solve U'y=b the values of y are written directly in b to save memory;
9
10  b(1)=b(1)/(U(1,1)); %We computhe the first element outside the loop
11                      %to avoid singularity
12
13  for k = 2:n %Starting loop from the second element
14      if(k-1 ~= 0) % The values are computed, but the ones that give zero
15                   %are neglected
16          t = dot(U(k-1,k),b(k-1));
17          b(k) = (b(k)-t)/U(k,k);
18      end
19  end
20
21  %Solve Ux=y,
22  %to save memory the values of x are written inside the array b.
23  for k = n:-1:2
24      if(k-1 ~= 0) % The values are computed, but the ones that give zero
25                   %are neglected
26          b(k) = b(k)/U(k,k);
27          b(k-1) = b(k-1)-b(k)*U(k-1,k);
28      end
29  end
30
31  b(1)=b(1)/(U(1,1));%We computhe the first element outside the loop
32                     %to avoid singularity
33
34  end
```

These two functions are used to compute the numerical solution of the discretized equation.

The main program is presented implemented as follows, where the analytical, calculated truncating the sum at n= 100, and numerical solution are compared. Note that the plots are made in the same main program.

Main program:

```
clear all
%% Question 3
% Heat equation using Crank-Nicholson discretization

%% Compute the Concentration matrix for the analytical solution
%Define the constant values N,D,L,Co,t,n

N = 40;             %Number of points in the domain
D = 1;              %Diffusivity constant
L = 1;              %Length of the domain
Co = 1;             %Initial Concentration
t = 0.1;            %Discretization time
n = 100;            %Number of time steps
DeltaT = t/n;       %Increment of time
DeltaX = L/(N-1);   %Increment of space
S = 0;              %Initialazation of a variable to Summation term

%Loop over the Fourier series
for j = 1:n %We loop over each time step
    T = DeltaT*j;
    for i = 1:N-2
        x = DeltaX*i;
        for m = 1:n  %Compute sumation terms from m=1 truncating at m=100
            %We assign each term of the summation factor to a variable,
               in
            %order to be have a clear code
            Y = (D*((2*m)-1)^2*(pi^2)*T/(L^2));
            P = (((2*m)-1)*pi*x)/(L);
            Q = 1/(((2*m)-1)*pi);
            S = S + (Q*exp(-Y)*sin(P));
            Y = 0;
        end
        C_a(i,j) = 4*Co*S;
        S = 0;
```

```
    end
end

%Add boundary conditions in the first and last point,
%which are specified as zero

Ci=0;
Cf=0;
C_a = [Ci*ones(1,n);C_a;Cf*ones(1,n)];

X = linspace(0,1,N);  %Space vector to plot the specified number of
    points

figure (1)
hold on

for j=10:16:100 %Plot concentration distribution for different time steps
 T = DeltaT*j;
 time = ['t = ', num2str(T)];
 plot(X, C_a(:,j), 'DisplayName',time)
 title('Concentration distribution for analytical solution N=40');
 ylabel('Concentration');
 xlabel('Distance');
end
hold off
legend show

%% Compute the Concentration matrix for the numerical solution

B = Matrix1D(N);                    % Sparse Matrix using  Matrix1D.m

sigma = D * (DeltaT/((DeltaX)^2));  %Sigma value otbtained from DT and DX
I = eye(size(B,1));                 %Identity matrix of the size of B
Z1 = (I +(sigma/2) * B);            %Matrix multipliying C^(n+1)
Z2 = (I -(sigma/2) * B);            %Matrix multipliying C^(n)
C_n=Co*ones(N-2,n);                 %Initialize concentration values at 0

U = spfa_tridiag(Z1);               %Obtain U matrix to solve Ax=b
    problem
                                    %It is computed using spfa_tridiag.m


for j = 1:n-1                       %Loop until number of time steps=100
```

```
75      b = Z2*C_n(:,j);                %Compute column vecctor solution b
76      C_n(:,j+1) = spsl_tridiag(U, b);%Compute the new concentration
            solving
77  end                                 %Ax=b using spsl_triridiag.m
78
79  %Add boundary conditions in the first and last point,
80  %which are specified as zero
81
82  Ci = 0;
83  Cf = 0;
84  C_n = [Ci*ones(1,n);C_n;Cf*ones(1,n)];
85
86  figure (2)
87  hold on
88
89  for j=10:15:100 %Plot concentration distribution for different time steps
90   T = DeltaT*j;
91   time = ['t = ', num2str(T)];
92   plot(X, C_n(:,j), 'DisplayName',time)
93   title('Concentration distribution for numerical solution N=40');
94   ylabel('Concentration');
95   xlabel('Distance');
96  end
97  hold off
98  legend show
99
100 %% Plots
101
102 figure (3)
103
104 subplot(3,2,1); plot(X, C_a(:,10),'b',X,C_n(:,10),'r'), title('t=0.01s'),
        axis([0 1 0 1]),xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]),
        yticks([0 0.2 0.4 0.6 0.8 1])
105 ylabel('Concentration');
106 xlabel('Distance');
107
108 subplot(3,2,2); plot(X, C_a(:,30),'b',X,C_n(:,30),'r'), title('t=0.03s'),
        axis([0 1 0 1]),xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]),
        yticks([0 0.2 0.4 0.6 0.8 1])
109 ylabel('Concentration');
110 xlabel('Distance');
111
112 subplot(3,2,3); plot(X, C_a(:,50),'b',X,C_n(:,50),'r'), title('t=0.05s'),
```

```
    axis([0 1 0 1]),xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]),
    yticks([0 0.2 0.4 0.6 0.8 1])
ylabel('Concentration');
xlabel('Distance');

subplot(3,2,4); plot(X, C_a(:,70),'b',X,C_n(:,70),'r'), title('t=0.07s'),
    axis([0 1 0 1]),xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]),
    yticks([0 0.2 0.4 0.6 0.8 1])
ylabel('Concentration');
xlabel('Distance');

subplot(3,2,5); plot(X, C_a(:,90),'b',X,C_n(:,90),'r'), title('t=0.09s'),
    axis([0 1 0 1]),xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]),
    yticks([0 0.2 0.4 0.6 0.8 1])
ylabel('Concentration');
xlabel('Distance');

subplot(3,2,6); plot(X, C_a(:,100),'b',X,C_n(:,100),'r'), title('t=0.1s')
    ,axis([0 1 0 1]),xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]),
    yticks([0 0.2 0.4 0.6 0.8 1])
ylabel('Concentration');
xlabel('Distance');

sgtitle('Concentration for analytical and numerical solution with N=40')
```

From the obtained graphs for N=10, we can see how the analytical and numerical solutions, Fig(2) (as the units are consistent, adimensional units of distance and concentration have been used), tends to zero as time increases, so at $t \to \infty$, C(x,t)=0. The maximum value is achieved at t=0.1, where C(x,0.1) = 1, from this point the temperature decreases.

When the numerical and analytical solution are computed and plotted together, it could be seen that there is not any difference, the two solutions are almost equal. In the multiple plots, the red graph corresponds to the analytical and the blue to the numerical, but the red is over the blue so it is barely appreciable.

If the number of points is increased to N=40, the same pattern could be observed. Nevertheless, the graph for this points are smoother as expected, because we have a fine mesh which improves the resolution which lead to more accurate results. Although, it need more running time than the coarser mesh.

Then, it could be concluded that the numerical scheme using the Crank-Nickolson scheme is an appropriate scheme to discretize this equation due to it computes a

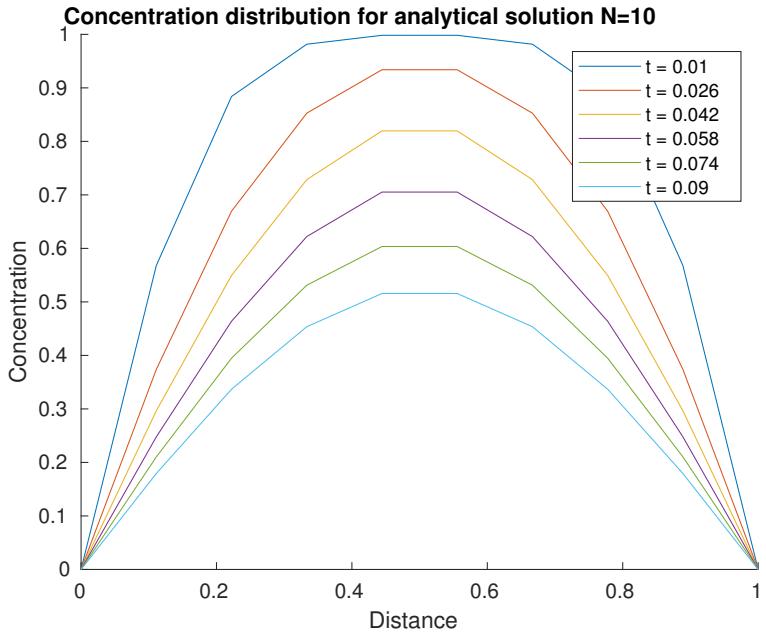

Figure 2: Concentration evolution in time for analytical solution with N=10.

solution close to the analytical one, without considerable error.

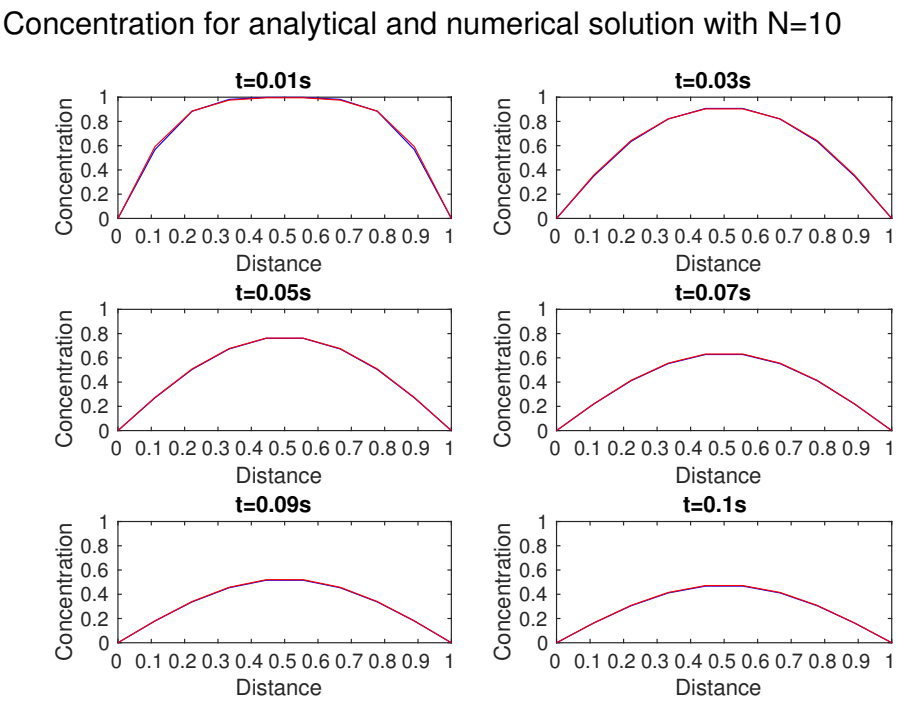

Figure 3: Concentration evolution in time for numerical and analytical solution at different time steps at N=10.

Question 4: 5% of total

Consider the matrix

$$\begin{pmatrix} 1 & \alpha & 0 \\ \alpha & 1 & 0 \\ 0 & \alpha & 1 \end{pmatrix} \tag{27}$$

For what values of α is:

i) the matrix diagonally dominant?

ii) does the Jacobi method converge?

Question 4: Answer

i) α **values for matrix diagonally dominant using Matlab**

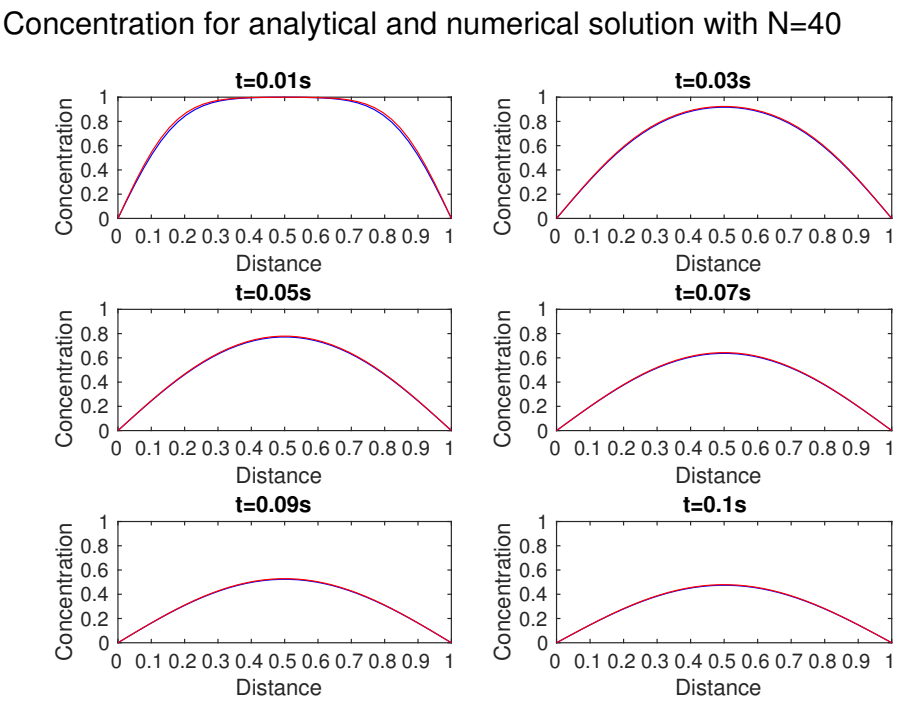

Figure 4: Concentration evolution in time for numerical and analytical solution at different time steps at N=40.

From lecture 11, a matrix is diagonal dominant if, for each row, the the absolute value of the diagonal element is strictly greater than the sum of the absolute values of the rest of the elements of that row. More precisely, an n x n matrix B is diagonally dominant if

$$|B_{ii}| \geq \sum_{j \neq i}^{n} |Bij| \quad \text{for all } i, \tag{28}$$

Therefore, a Matlab script is created in order to determine the values of alpha that makes matrix diagonally dominant

```
clear all
%% Question 4
% Demonstrate that the matrix is diagonal dominant

a=0;    %Initialize the first value of alpha
f = 0; %Initialize condition that states when alpha makes the matrix non
```

```
    diagonal dominant

while (f~=1) %Condition to break the loop. If f=1, alpha will not satisfy
             %the condition for a non diagonal dominant matrix

B=[1 a 0; a 1 0; 0 a 1]; %The matrix defined in the coursework

[m,n]=size(B); %Get the size paramters of B

R=zeros(1,3); %Vector which will store the sum of the absolute values of
              %each row

    for i=1:m     %We loop until the end of the row
        S=0;      %Initialize the sum of the non-diagonal elements
        for j=1:n
            %Sum the absolute values of a row without the values
            %on the main diagonal.
            if i~=j %Check if the value is not placed in the main
                diagonal.
                S=S+abs(B(i,j)); %Sum the absolute value
            end
        end
        %We save each sum in the vector R.
        R(i)=S;
    end

    Delta = 0.0001; %Increment values for alpha

    for i=1:m
        %Compare the sum of the absolute values on the row with the
           absolute
        %values on the main diagonal.
        if R(i)>B(i,i)
            f=1; %An alpha value which makes the matrix non diagonal
                dominant
                 %has been FOUND
        else
            a=a+Delta; %Delta is added to alpha until the condition of no
                diagonal
                      %dominant is not satisfied
        end
    end
end
```

From our small program, we found that the matrix is diagonal dominant until $\alpha = 1$. As, the algorithm checks absolute values, it can be concluded that the matrix is diagonal dominant for $\forall\ \alpha \in [-1, 1]$

ii) **Convergence of Jacobi method**

From lecture 11, a diagonal **dominant matrix** assures that the **Jacobi method converges** and its residual decreases in each iteration. Then, a Jacobi method applied to a problem which involves this matri will converge for $\forall\ \alpha \in [-1, 1]$

Question 5: 25% of total

Consider the two-dimensional boundary value problem of steady state heat conduction in a L-shaped plate region (Fig. 5) governed by the Poisson equation:

$$\boldsymbol{\nabla}^2 T(x, y) = \frac{\partial^2 T(x, y)}{\partial x^2} + \frac{\partial^2 T(x, y)}{\partial y^2} = q(x, y) \tag{29}$$

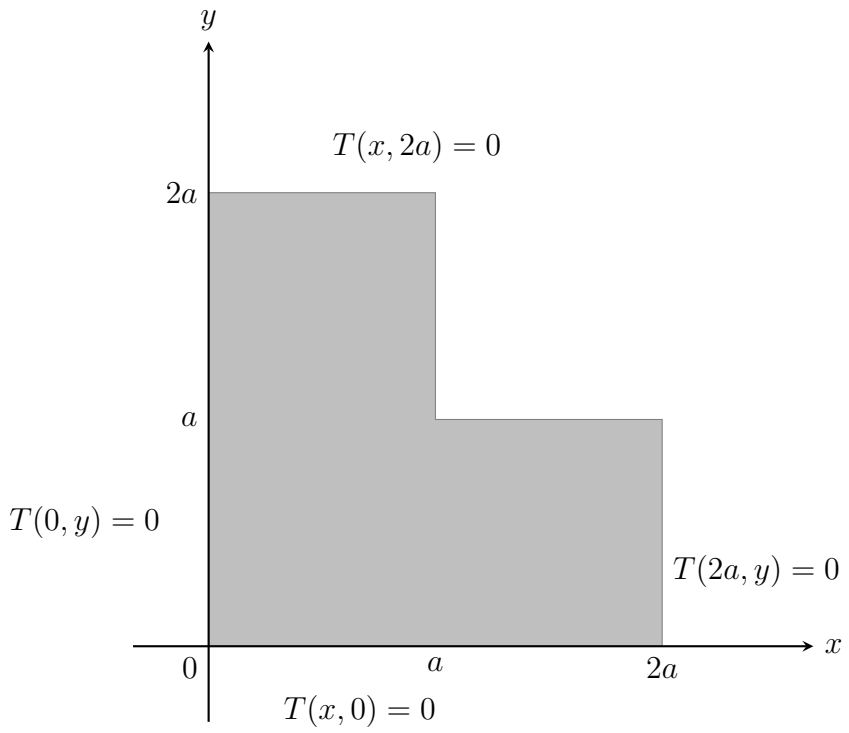

Figure 5: L-shape region subjected to boundary temperature.

and subjected to the boundary conditions:

$$\begin{aligned} T(0,y) = 0 \quad &; \quad T(2a,y) = 0 \\ T(x,0) = 0 \quad &; \quad T(x,2a) = 0 \end{aligned} \tag{30}$$

The region will be decomposed in two domains: a small square having $(N_x + 1) \times (N_y + 1)$ elements (including the boundary elements), and a rectangle with $(N_x + 1) \times (2N_y + 1)$ elements (including the boundary elements). Each grid point inside the domain will be enumerated

- from the the top to the bottom of the domain,
- from left to right,

as shown in Fig. 6 for a coarse discretisation having $N_x = N_y = 3$. The elements on the outer boundaries of the domains won't be enumerated because the temperature T is known to be fixed at 0.

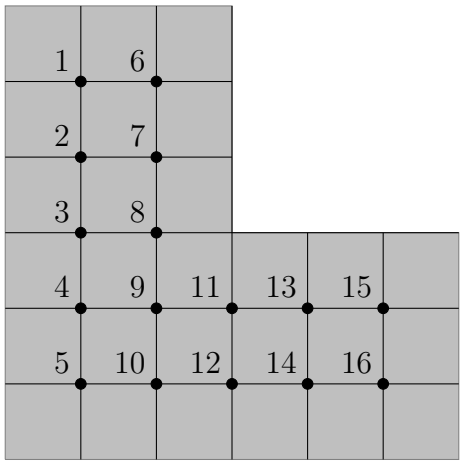

Figure 6: Coarse discretisation for the L-shaped region, $N_x = N_y = 3$.

1. Explain why the finite difference matrix A to solve eq. 29 inside the domain is a symmetric positive definite banded matrix, and calculate the bandwidth of A for $N_x = 10$ and $N_y = 10$. What will the total size of A be?

2. For $N_x = 5$, $N_y = 5$, $a = 1$m, write a MATLAB code MatrixA2D.m to form the finite difference matrix A in a *sparse matrix format* and show the shape of the matrix A with the MATLAB function *spy(A)*. Present your code in the report with sufficient

comments to explain each line. The code should be accompanied with an explanation of the general form of the matrix A and how you plan to store and use it in your code.

3. Plot the 2D solution of eq. 29 (including the boundaries) for the given boundary conditions with $q(x,t) = 1$: for $N_x = N_y = 5$, $N_x = N_y = 10$ and $N_x = N_y = 20$. The code, stored in SolveMatrixA2D.m, should use an *iterative solver*, which checks that the convergence of the solution is under a certain tolerance, defined by the user.

4. Keeping the tolerance constant, comment on the number of iterations needed to reach convergence changing the number of elements inside the domain.

Question 5: Answer

1. **Positive definite matrix**

We need to solve the Laplace equation in 2D (31)

$$\nabla T^2 = \left(\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} \right) = 0 \tag{31}$$

Using the central difference of second order, the discretized form of the Poisson equation the x and y direction is:

$$\begin{aligned} \frac{\delta^2 T}{\delta x^2}|_{i,j} &\approx \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} \\ \frac{\delta^2 T}{\delta y^2}|_{i,j} &\approx \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} \end{aligned} \tag{32}$$

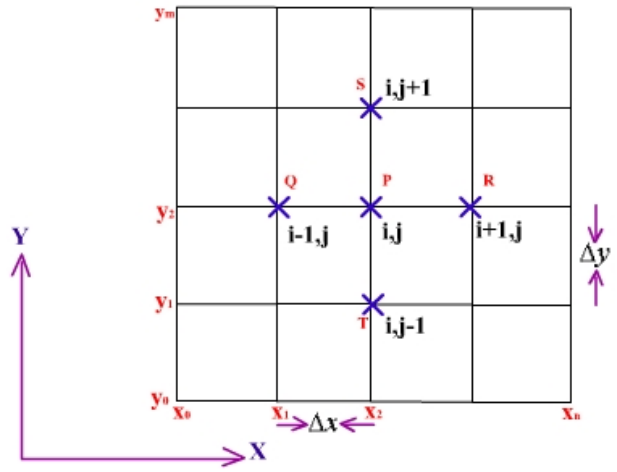

Figure 7: Discretized square domain.

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} = q(x, y) \tag{33}$$

In this problem $\Delta x = \Delta y$, so the discretization equation becomes:

$$4T_{i,j} - (T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1}) = q * \Delta x^2 \tag{34}$$

Then, the system that has to be solves is of the form:

$$Ax = b \tag{35}$$

The number of nodes is specified to be $N = 2 * N_x + 1$, but as the boundary conditions are specified to be a zero at the edges, the final number of nodes is:

$$N = 2 * N_x - 1 \tag{36}$$

And x is of the form:

$$x = \begin{pmatrix} T_{11} \\ T_{21} \\ \vdots \\ T_{NN} \end{pmatrix} \tag{37}$$

Therefore, the matrix A will have the size $N^2 x N^2$ and will be composed of N^2 square matrices of size N x N.

$$A = \begin{pmatrix} D & -I & \dots & 0 \\ -I & D & \ddots & \vdots \\ \vdots & \ddots & \ddots & -I \\ 0 & \dots & -I & D \end{pmatrix} \tag{38}$$

Substituting the values of the discretized Poisson equation, the matrix A becomes:

$$A = \begin{pmatrix} 4 & -1 & \dots & 0 \\ -1 & 4 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \dots & -1 & 4 \end{pmatrix} \tag{39}$$

In the case of $N_x = 10$ and $N_y = 10$, number of nodes will be $N = 2 * N_x - 1 = 19$, because the boundaries are not considered which are zero.

The bandwidth of the matrix is equal to number of nodes N = 19

$$m_u = m_l = 19 \tag{40}$$

The total size of A will be $N^2 x N^2 = 19^2 x 19^2 = 361$*361

Our matrix will have the shape of the matrix in Fig.(8), in this case $N_x = 4$.

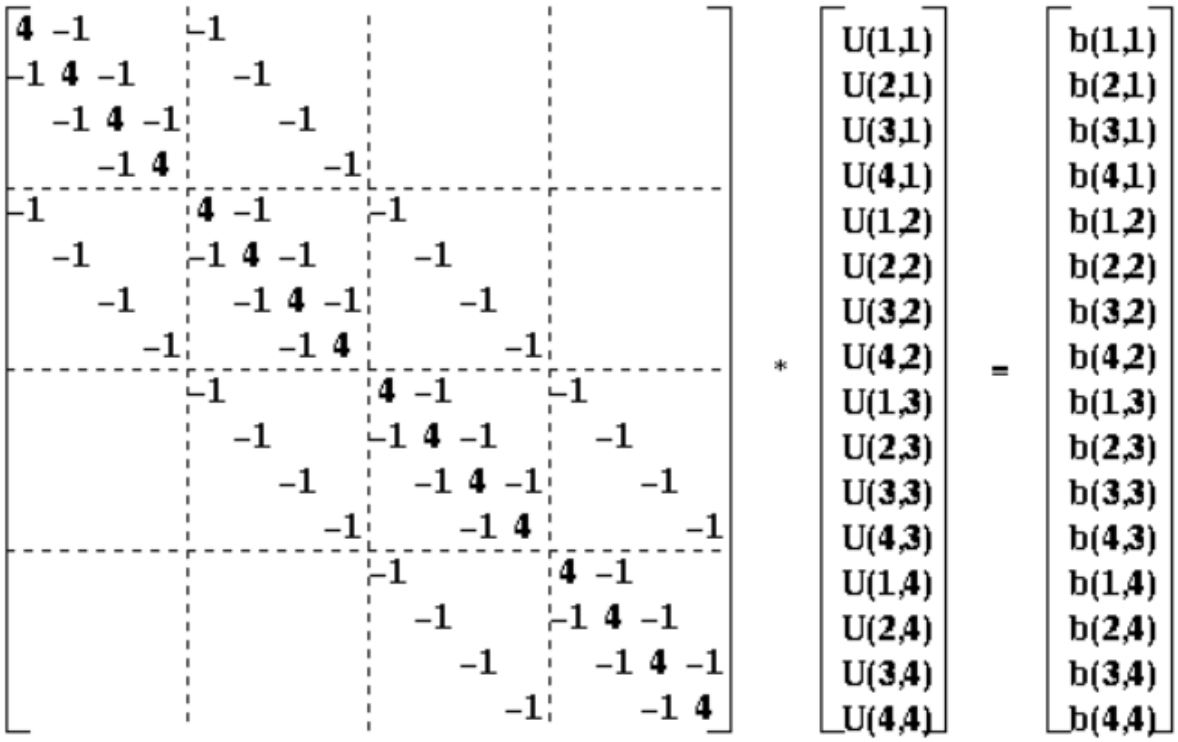

Figure 8: Discrete Poisson equation grid for n=4.

As you can see, we have a clear banded matrix, where the upper and lower band are well defined. Recall that a matrix is symmetric positive definite if and only if A = A^T and $x^t\ Ax > 0$ for all x $\neq$ 0.

Through using the *transpose()* function of Matlab we have proved the theorem and found that the matrix is symmetric positive definite

2. **Sparsity pattern matrix**

From the discretization of our equation, we can obtain the coefficients matrix. An example for a n=4 grid could be seen in Fig.(8). In this matrix, we can see then elements of the main diagonal an the off diagonal elements.

In our case, the domain has a size of $N = 2 * N_x + 1 = 11$, but as the boundaries are not considered the size is $(2 * N_x - 1)^2 X (2 * N_x - 1)^2 \rightarrow 9^2 X 9^2$. Then, we will end with a $81X81$ matrix. In order to obtain this matrix shape in Matlab, the function *kron()* has been used in order to repeat the top left matrix from Eq.39 along the main diagonal. and very important to avoid coupling between elements.

The *kron()* function uses the Kronecker tensor product to expand a small square matrix

to a bigger one. The key point of this function is that we avoid coupling between the last elements from the first small block matrix with the following ones

If A is an m-by-n matrix and B is a p-by-q matrix, then the Kronecker tensor product of A and B is a large matrix formed by multiplying B by each element of A; Ref.[?]

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$$

Theorem

$$A = \begin{bmatrix} 1 & -2 \\ -1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & -3 \\ 2 & 3 \end{bmatrix}$$

$$A \otimes B = \begin{bmatrix} 1 \cdot 4 & 1 \cdot -3 & -2 \cdot 4 & -2 \cdot -3 \\ 1 \cdot 2 & 1 \cdot 3 & -2 \cdot 2 & -2 \cdot 3 \\ -1 \cdot 4 & -1 \cdot -3 & 0 \cdot 4 & 0 \cdot -3 \\ -1 \cdot 2 & -1 \cdot 3 & 0 \cdot 2 & 0 \cdot 3 \end{bmatrix} = \begin{bmatrix} 4 & -3 & -8 & 6 \\ 2 & 3 & -4 & -6 \\ -4 & 3 & 0 & 0 \\ -2 & -3 & 0 & 0 \end{bmatrix}$$

Example

Figure 9: Kronecker tensor product.

Afterwards, the elements of the off diagonal have been added using the build Matlab function *diag()*. Note that to locate this diagonal elements we need to understand carefully what will be the size and the location. Then, in Fig.(8) you can see that the off daigonal situated at the right is shifted $N_x = 4$ points from the first element of the matrix and the length of the diagonal is $(N_x - 1) * (N_x) = 3 * 4 = 12$ elements.

It is important to note that in this function, we eliminate already the points in the top right corner in order to get the desired L-shape. In order to eliminate these points, we selected them through a loop in order to identify the position p that they occupy in our coefficients matrix. We loop over the rows and columns, i j, from the last point of our matrix A which corresponds to the top right corner point of the square domain, so the loop goes from $2 * N_x - 1$ to N_x. Then, we select each point using the following equation which we have determined by hand:

$$p = i + (j - 1) * (2 * N_x - 1) \tag{41}$$

This equation gives us the position of each deleted position and then we store in a matrix to use it later to plot the L-shape temperature surface.

Then, the column an row with the identified index are eliminated, thanks to that we get rid off all the points that have dependence with the ones that are outside from our L-shape domain.

The used Matlab script MatrixA2D is presented as follows:

```
%%MatrixA2D function to compute sparse matrix B  with a defined number of
%%points n
function [A,N,pos] = MatrixA2D(Nx)

%Define number of points and length and we substract two points due to
%the boundaries which are zero
N=(2*(Nx)+1)-2;

%Define the longitude of the domain without the boundaries and define the
%width space in x.

Lx = 2*Nx+1;
DeltaX = 2/Lx;

%Create vector column of ones of length N
Ones = ones(N,1);

%Define tridiagonal matrix of size Nx^2 X Ny^2 with the main diagonal of
%4 and the upper and lower diagonal of -1

%We use the Matlab function diag, which allows us to create a matrix with
%a diagonal that where we can specify the length and the position.
%Then, combining the 5 matrices we obtain the desired matrix.

A= diag((4)/(DeltaX^2) * Ones, 0) + diag((-1)/(DeltaX^2)*Ones(1:N-1), -1)
     + diag((-1)/(DeltaX^2)*Ones(1:N-1), 1);

A = kron(eye(N), A);

%Shift the other non zero diagonal Ny positions and the length of these
%diagonals is Nx-1 X Ny

off_diag = ((-1)/(DeltaX^2))*ones((N-1)*(N),1);

A = A + diag(off_diag, N)+ diag(off_diag, -N) ;

%As the problem is defined over a L shaped domain, the points of the
```

```
%square matrix that are out this domain have to be deleted from the
   matrix
%Delete these points now will make our computations faster

%The zone to elimiate it's defined as a small in the top right corner
%square starting at (n-1) to (2*n-1)

for j = (2*Nx-1):-1:(Nx) %It loops backwards from the top-right point to
                         %the half of the domain in the x-direction

    for i = (2*Nx-1):-1:(Nx)%It loops backwards from the top-right point
                            %to the half of the domain in the x-direction

         p = i+(j-1)*(2*Nx-1);%This equation gives us the position of the
                              %points we want to eliminate from our matrix

            A(p,:) = [];%The rows and columns that have a dependence with
            A(:,p) = [];%the target points are eliminated

            pos(i,j)=p;
    end
end

%Now we are storing a smaller sparse matrix than if we apply the BC later
A = sparse(A);
end
```

Using the Matlab function *spy()*, the sparsity pattern of the matrix A is plotted. This plots shows the locations of the nonzero elements.

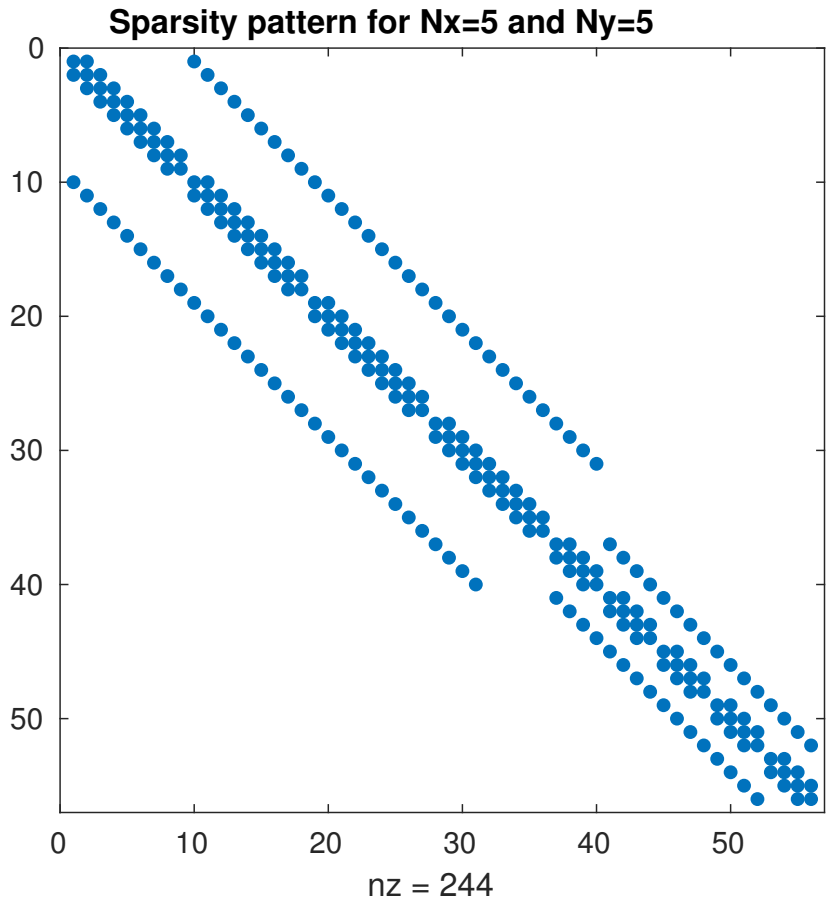

Figure 10: Sparsity pattern of the matrix A.

3. **Iterative solver using Gauss-Seidel method**

   In order to solve the sparse matrix obtained, we are going to use the Gauss-Seidel method.

   This method splits the matrix A = L + D + U, where:

   - **L** is a strictly lower triangular matrix obtained from Matlab function *tril()*.

   - **D** is a diagonal matrix obtained from applying twice the Matlab function *diag()*

   - **U** is a strictly upper triangular matrix obtained from Matlab function *triu()*.

   Precisely, we have used the forward Gauss-Seidel method which is:

$$(L + D)x^(k + 1) = b - Ux^k \tag{42}$$

   This method is faster than the Jacobi method because it uses new values as soon as

possible. In addition, is a method that takes less memory because it only needs to store one array due to its overwritten.

The Matlab script SolveMatrixA2D created to obtain the tcolumn vector solution of temperatures is presented as follows:

```
function[T,timeElapsed,iterations] = SolveMatrixA2D(A,Nx)

%A time counter will be used to determine the needed time for the
    iterative
%solver to converge. It is used the tic-toc Matlab feature.

tic; %The time counter is activitated

%Get matrix size, by obtaining the number of points in x and y
[ny,nx]=size(A);


q = 1;                          %Source term
b = -q*ones(ny,1);              %Value resulted for multiplying q and DeltaX
T = zeros(ny,1);                %Initialize the Temperature columb vector
                                %with zeros

%The problem is going to use an iterative solver which checks that
%the convergence of the solution is under a certain tolerance,
%defined by the user

tol = 10^-6;                     %Error tolerance used for check convergence

%http://employees.oneonta.edu/GoutziCJ/fall_1999/math323/matlab/lesson_05
    .pdf

%The itertive solver will use the LDU decomposition, which allows to
%solve the problem Ax=b, using Gauss-Seidel forward substitution,
%by decomposing A=L+D+U. Then, we have the following eq:
%(L+D)x^(k+1) = b-Ux^k)

%First, the lower and upper triangular matrices and a main diagonal are
%obtained matrix uisng defined Matlab functions

L = tril(A,-1);      %Lower triangular matrix using Matlab function tril()
D = diag(diag(A));   %Main diagonal matrix using Matlab function diag()
U = triu(A,1);       %Upper triangular matrixusing Matlab function triu()
```

```
36
37 Tn = zeros(ny,1);              %Preallocating Tn
38 a=0; %Condition to break loop
39 m=1; %Counter of the matrices computed inside the loop
40
41 while (a~=1)
42     Tn=T;
43     T = (L+D)\(b-U*Tn);
44     %Error betwen p and pn
45         dif = abs(T - Tn);
46         eps= dif;
47
48         for i=2:ny-1
49             eps(i,1) = abs(dif(i,1)/T(i,1));
50         end
51
52         e_max(m) = max(eps(:));
53
54         if(e_max(m) < tol)
55             a=1;
56         end
57         m=m+1;
58
59 end
60
61      timeElapsed = toc;
62      iterations = m;
63
64 end
```

The temperature distribution obtained from the code for diffent values of T is presented in Fig.(13) for N=5, Fig.(16) for N=10 and Fig.(19) for N=20.

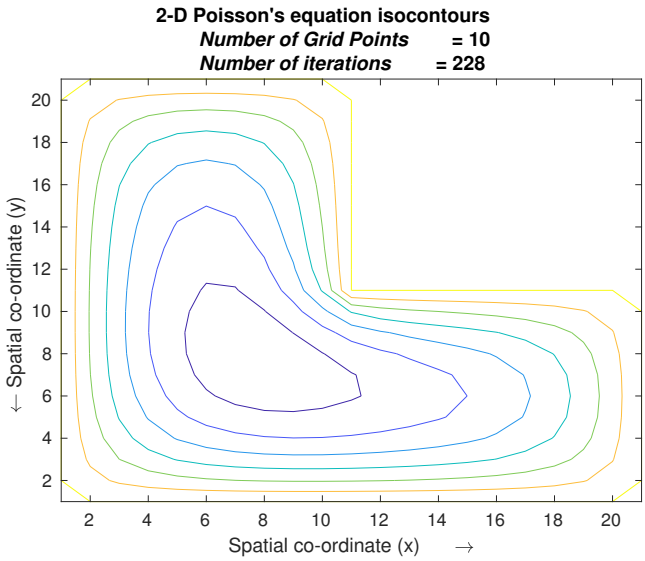

Figure 14: Isocontours.

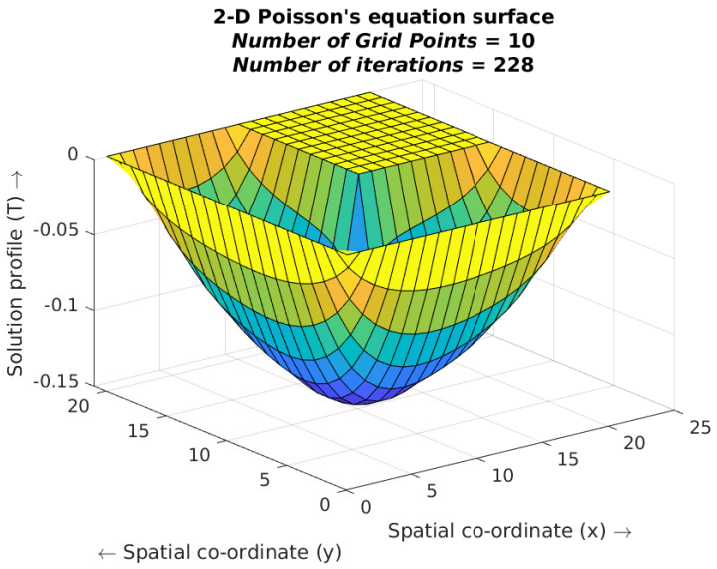

Figure 15: Surface.

Figure 16: Poisson equation For Nx=5 with Gauss-Seidel method.

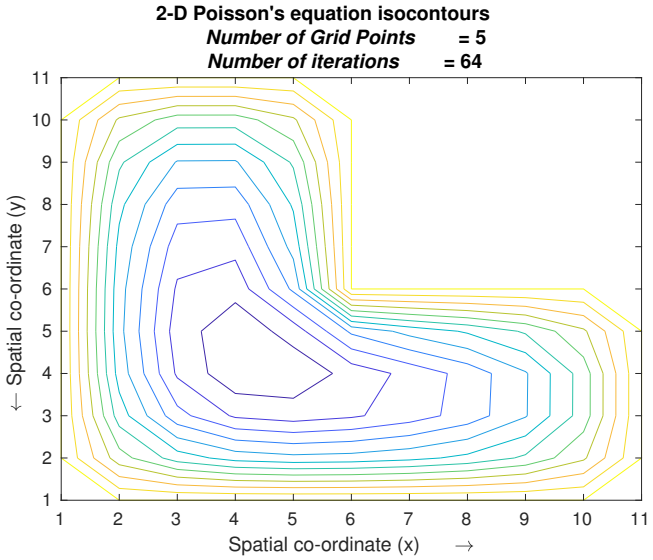

Figure 11: Isocontours.

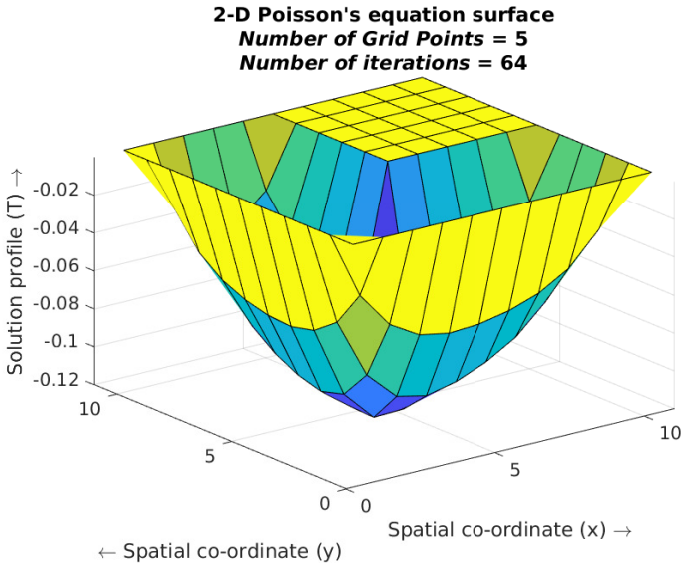

Figure 12: Surface.

Figure 13: Poisson equation For Nx=5 with Gauss-Seidel method.

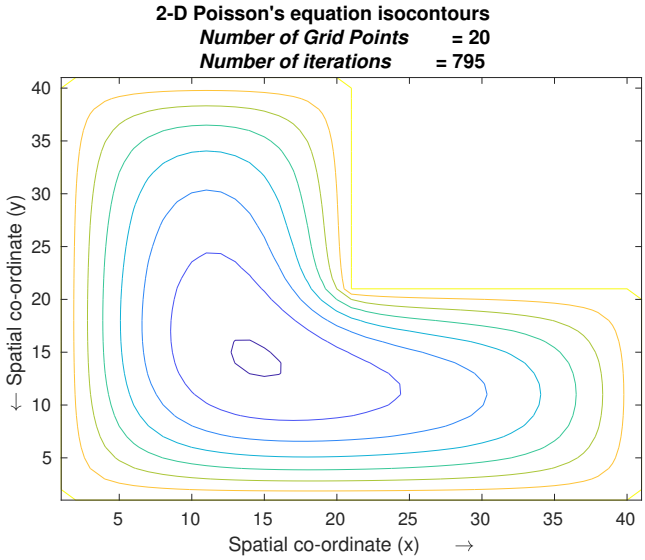

Figure 17: Isocontours.

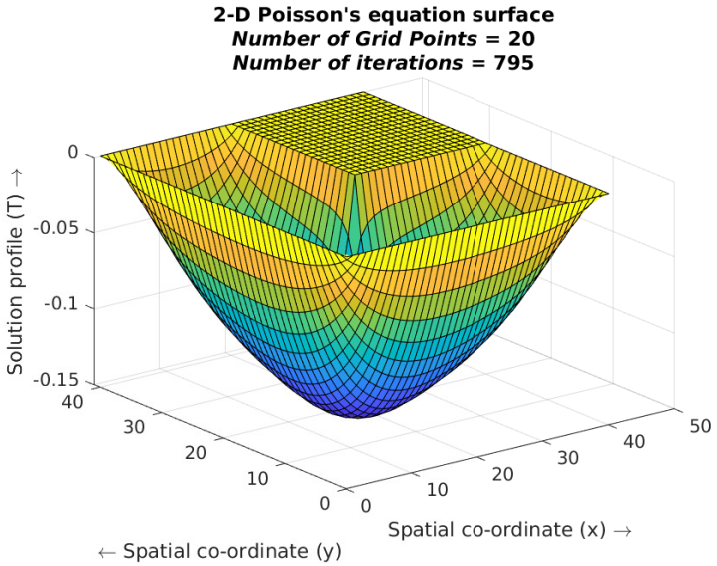

Figure 18: Surface.

Figure 19: Poisson equation For Nx=5 with Gauss-Seidel method.

4. **Results analysis**

   Keeping the tolerance constant for a value of 10^{-6}, we have computed the number of iterations needed to reach convergence in function of the number of elements inside the domain.

   The results for each mesh size are shown in Table (IV)

No. grid points (Nx)	**Mesh size (Δx)**	**No. iterations**	**Run time**
5	0.1818	64	0.004s
10	0.0952	228	0.011s
20	0.0488	795	0.035s

Table IV: Number of iterations and run time for each mesh size with Gauss method.

   We can observe how the number of iterations grows exponentially and then, it will arrive a point that is not worthy to increase the points to obtain more accuracy because the simulation will take a long time to finish.

   So, we have to find a mesh size that gives us the most approximate value withthe less possible run time.

References

[1] Lui, S.H., 2011. Numerical Linear Algebra. In Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Hoboken, NJ, USA: John Wiley Sons, Inc., pp. 169220.

[2] Demmel, J.W. Society for Industrial Applied Mathematics, 1997. Applied numerical linear algebra, Philadelphia: Society for Industrial and Applied Mathematics.+

[3] Alonso et al., 2011. Growth factors of pivoting strategies associated with Neville elimination. Journal of Computational and Applied Mathematics, 235(7), pp.17551762.