AEM-ADV08
Introduction to Computational Linear Algebra
Coursework 1

Student Name: Wenze Liu Chen CID: 01526073

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 handwritten 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 cornensponding 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

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

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 x for the following matrix-vector system $Ax = b$:

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

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 behaviour 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) $||L||_\infty ||U||_\infty$ and (ii) $\alpha_{ALG} = \frac{||L||_\infty ||U||_\infty}{||A||_\infty}$. What do you infer from these variations and how do you relate these variations with partial pivoting?

Question 1:

1.1 The largest absolute value that the coefficients of L can take is 1. When performing partial pivoting, it is always desirable to select the largest possible entry (in absolute value) as the pivot element in order to make sure the matrix calculations is stable and accurate (although not guaranteed). Thus, there only exist 2 possible cases:

- There exists a larger entry in other rows, then the coefficients would be less than 1
- No larger entries are presented in other rows, the pivot element remains unchanged, and dividing itself gives 1

1.2 Consider the first stage of Gaussian Eliminination with partial pivoting to matrix A:

$$\left|A_{i,j}^1\right| = \left|A_{i,j}^0 - \frac{A_{i,1}^0}{A_{1,1}^0} A_{1,j}^0\right|$$

Using the triangle inequality:

$$\begin{aligned}\left|A_{i,j}^1\right| = \left|A_{i,j}^0 - \frac{A_{i,1}^0}{A_{1,1}^0} A_{1,j}^0\right| &\le \left|A_{i,j}^0\right| + \left|-\frac{A_{i,1}^0}{A_{1,1}^0} A_{1,j}^0\right| \\ &\le \left|A_{i,j}^0\right| + \left|\frac{A_{i,1}^0}{A_{1,1}^0} A_{1,j}^0\right| \\ &\le \left|A_{i,j}^0\right| + \left|A_{1,j}^0\right| \\ &\le 2\max_{i,j}\left|A_{i,j}^0\right| = 2\max_{i,j}\left|A_{i,j}\right|\end{aligned}$$

From here the relation at arbitrary stage k can be deduced:

$$\left|A_{i,j}^k\right| \le 2\max_{i,j}\left|A_{i,j}^{k-1}\right|$$

The matrix U is the resultant matrix of Gaussian Elimination from $n \times n$ matrix A, therefore $n-1$ stages are needed, and using the relation above:

$$\begin{aligned}\left|U_{i,j}\right| &= \left|A_{i,j}^{n-1}\right| \\ &\le \left|A_{i,j}^{n-2}\right| + \left|A_{n-1,j}^{n-2}\right| \\ &\le 2\max_{i,j}\left|A_{i,j}^{n-2}\right| \\ &\le 2^2\max_{i,j}\left|A_{i,j}^{n-3}\right| \\ &\le 2^{n-1}\max_{i,j}\left|A_{i,j}^{0}\right| = 2^{n-1}\max_{i,j}\left|A_{i,j}\right|\end{aligned}$$

Hence,

$$\rho = \frac{max_{i,j=1,2,\ldots,n}\left|U_{ij}\right|}{max_{i,j=1,2,\ldots,n}\left|A_{ij}\right|} = \frac{2^{n-1}\max_{i,j}\left|A_{i,j}\right|}{\max_{i,j}\left|A_{i,j}\right|} \le 2^{n-1}$$

1.3 The computed L, U, P and x using the scripts are:

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

$$x = [1 \quad 1 \quad 1 \quad 1 \quad 1]^T$$

Script of L, U, P factorisation:

```
function [L,U,P] = LPU_factorisation(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

PJ_inv = I;% initialisation of gauss tranfomration matrix with pivoting = P*J
P_k_old = I;% initialisation of permutation matrix = P at 0th stage
for k=1:n-1 % the main loop of the GEM starts here

P_k = permutation_matrix(U,k,n);%permutation matrix at kth stage
U = matrix_matrix_mult(P_k,U,n);%permute the U matrix
P = matrix_matrix_mult(P_k,P_k_old,n);%compute the general permutation matrix
P_k_old = P;%update the permutation matrix at last stage


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.

J_inv = matrix_subtraction(I,-LE,n); %compute the inverse of gauss transformation
matrix, J^-1 = I + LE = I - (-LE)
PJ_inv = matrix_matrix_mult(PJ_inv,P_k,n); %compute gauss tranfomration matrix with
pivoting = P*J at each kth stage
PJ_inv = matrix_matrix_mult(PJ_inv,J_inv,n); %compute gauss tranfomration matrix with
pivoting = P*J at each kth stage

end % the main loop of the GEM ends here

L = P*PJ_inv;%compute the lower triangular matrix
```

Script of permutation matrix:

```
function P = permutation_matrix(A,k,n)
% the fucntion outputs the permutation matrix.
% A - Input matrix to do the partial pivoting
% n - the size of the matrix A
% k - the kth stage of gaussian elimination

%partial pivoting
I = eye(n);%identity matrix
i = k;%define a counter to compute the index of largest element
while (abs(A(i,k)) < max(abs(A(:,k)))) %compare each element in kth column with the
largest element
    if i == n
       break; %break the loop when reach the last element
    end
    i = i+1; %obtain the matrix row index of the largest element
end
P = I - (e_vector(i,n)-e_vector(k,n))'*(e_vector(i,n)-e_vector(k,n));% compute the
permutation matrix

end
```

Script of forward and backward substitution:

```
function b = forward_substitution(L,b,n)
% The function performe the forward substitution for a given lower
% triangular matrix L and a given vector b
% L - the lower triangular matrix L
% b - the vector b
% n - the size of the matrix L

%forward substitution
b(1) = b(1)/L(1,1);%compute the first entry of the solution of lower triangular matrix
for i = 2:n %perform the forward substitution from second entry
   b(i) = (b(i)-L(i,1:i-1)*b(1:i-1))/L(i,i);
end

end


function b = backward_substitution(U,b,n)
% The function performe the forward substitution for a given upper
% triangular matrix L and a given vector b
% U - the lower triangular matrix U
% b - the vector b
% n - the size of the matrix U

%backward substitution
b(n) = b(n)/U(n,n); %compute the last entry of the solution of upper triangular matrix
for i = n-1:-1:1 %perform the backward substitution from n-1 entry
   b(i) = (b(i)-U(i,i+1:n)*b(i+1:n))/U(i,i);
end

end
```

Script of matrix A:

```
function A = matrix_A(n)
%this function creates the gvien matrix A
%   n - the size of matrix A

A = zeros(n,n);
for i = 1:n
   for j = 1:n
       if (i==j)
          A(i,j) = 1;
       elseif (j==n)
          A(i,j) = 1;
       elseif (j<i)
          A(i,j) = -1;
       end
   end
end
```

```
end
```

Script of main function:

```
clear all;
n = 5;
A = matrix_A(n);
b = [2;1;0;-1;-3];
[L,U,P] = LPU_factorisation(A,n);
y = forward_substitution(L,b,n);
x = backward_substitution(U,y,n);
```

1.4 Using the formulation of growth factor in 1.2:

$$\rho = \frac{max_{i,j=1,2,\dots,n}|U_{ij}|}{max_{i,j=1,2,\dots,n}|A_{ij}|} = \frac{16}{1} = 16$$

Table 1: Growth factor of $n \times n$ matrix

$\boldsymbol{n}$	8	16	32
$\boldsymbol{\rho}$	128	32768	2147483648

From *Table 1*, it can be observed that the growth factor ρ increases exponentially with the size of matrix. The relation between ρ and n follows the equation $\rho = 2^{n-1}$, which is the worst case of the inequality equation $\rho \leq 2^{n-1}$ deduced in 1.2. This implies that the LU factorisation becomes very unstable when the size of matrix is very big.

1.5

Table 2: Variation of growth parameters with respect to n

$\boldsymbol{n}$	8	16	32
$\||\boldsymbol{L}\||\boldsymbol{U}\||\|_\infty$	262	65550	$4.295 \cdot 10^9$
$\boldsymbol{\alpha_{ALG}}$	32.75	4096.875	$1.342 \cdot 10^8$

The parameters shown in *Table 2* are related with the growth of resultant L and U matrix, these parameters increase uncontrollably with the increment of the matrix size. The variation of these parameters with respect to n implies that the Gaussian Elimination with partial pivoting is potentially unstable, the solution may be negatively affected by the extremely large elements encountered during the elimination process. Although partial pivoting is used to improve the stability, but it does not guarantee the process to be stable, the partial pivoting may still fail.

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 $A^k x = b$ 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:

2.1 (i) For $A^k x = b$, we know that the solution of x vector is given by $x = A^{-k}b$. Now, consider the matrix equation $Ax = b$, for which the solution is $x = A^{-1}b$, if we replace the vector b by the calculated matrix x, then:

$$Ax = b$$

$$x = A^{-1}b \to b = x$$

$$Ax = A^{-1}b$$

$$x = A^{-2}b \to b = x$$

$$\dots$$

$$x = A^{-k}b$$

repeating the process by k times, we will get $x = A^{-k}b$, which corresponds to the solution of $A^k x = b$. The unknown vector x can be solved by performing a Gaussian Elimination with complete pivoting. In complete pivoting, we swap the row and column at each k^{th} stage of Gaussian Elimination to ensure the pivot element is the largest element:

$$\left|c_{k,k}^{k}\right| \geq \left|c_{i,j}^{k}\right| \quad \text{for } i, j > k$$

Remember that row swap is achieved by left multiplication of permutation matrix and column swap is achieved by right multiplication. Thus, after performing Gaussian Elimination with complete pivoting, we will get:

$$P_{row}A^k P_{col} = P_{row}CP_{col} = LU$$

Substitute back this to the system:

$$A^k x = b \to (P_{row}^{-1}LUP_{col}^{-1})x = b$$

$$Ly = P_{row}b$$

$$y = Uz$$

$$z = P_{col}^{-1}x \to x = P_{col}z$$

Using forward and backward substitution, y and x can be easily solved. Do the above process by k times gives the solution of matrix equation $A^k x = b$.

(ii) Pseudo-code

```
%initialisation of some intermediate variables
PJ_inv = I;
P_r_old = I;
P_c_old = I;
%Gussian elimination with complete pivoting
for k=1:n-1
%find the row and column index of the max. value
[m,max_row]=max(abs(A(k:n,k:n)));
[m,c_ind]=max(m);
r_ind = max_row(c_ind);
%transform this coordinates to the coordinates of A
r_ind=r_ind+k-1;
c_ind=c_ind+k-1;
%row and column permutation matrix at kth stage
P_r = I - (e_vector(r_ind,n)-e_vector(k,n))'*(e_vector(r_ind,n)-e_vector(k,n));
P_c = I - (e_vector(c_ind,n)-e_vector(k,n))'*(e_vector(c_ind,n)-e_vector(k,n));
%impose the permutation matrix to A matrix
A = P_r*A*P_c
%compute the permutation matrix
Pr = P_r*P_r_old
Pc = P_c_old*P_c,n
%update the permutation matrix at k-1 stage
P_r_old = Pr;
P_c_old = Pc;
%compute upper triangular matrix
gv = gauss_vector(A,k,n);
evec = e_vector(k,n);
LE = outer_product(gv,evec,n);
J = I-LE
A = J*A
%compute the inverse of J matrix
J_inv = I+LE
%compute the inverse of J matrix with permutation matrix
PJ_inv = PJ_inv*P_r*J_inv
end
%compute the lower triangular matrix
L = Pr*PJ_inv;
U = A;

%solve the (A^k)*x = b
x = [...];
for i = 1:k
%Use forward substitution to compute y
b = x;
b = Pr*b
b(1) = b(1)/L(1,1)
for i = 2:n;
```

```
b(i) = (b(i)-L(i,1:i-1)*b(1:i-1))/L(i,i);
end
y = b;
%Use backward substitution to compute z
y(n) = y(n)/U(n,n)
for i = n-1:-1:1
y(i) = (y(i)-U(i,i+1:n)*y(i+1:n))/U(i,i);
end
z = y → x = Pc*z %Use column permutation to re-order z and get the solution x
end
```

(iii) The problem is solved by performing a Gaussian Elimination with complete pivoting, the process involves the LU factorisation and forward/backward substitution. Remember that complete pivoting on a non-symmetric and dense matrix is a $O(n^3)$ process. Now consider the forward and backward substitution process, since we know that dot product is a $O(n)$ process and both forward/backward substitution is a dot product repeated by n times, so they are $O(n^2)$ process. However, for this problem, the substitution process need to be iterated by k times, so the substitution becomes $O(kn^2)$ process.

2.2 (i) For the given matrix equation $AX = B$, the algorithm to solve is generally the same as to solve the $Ax = b$. The only difference is that X and B are no longer one-dimensional vectors, they have the size $n \times m$, so when doing forward and backward substitution a for-loop is needed to iterate the substitution process for m times. Each iteration will solve one column of X:

$$AX = B$$

$$X = A^{-1}B$$

$$X(:,m) = A^{-1}B(:,m), m = 1 \dots k$$

The algorithm to perform the Gaussian Elimination with complete pivoting is the same as the one explained in 2.1. So, at each stage of GE we aim to find:

$$\left|a_{k,k}^{k}\right| \geq \left|a_{i,j}^{k}\right| \quad \text{for } i,j > k$$

After factorised the A matrix to LU matrices, we solve the equation by using forward and backward substitution:

$$P_{row}AP_{col} = LU$$

$$AX = B \rightarrow (P_{row}^{-1}LUP_{col}^{-1})X = B$$

$$LY = P_{row}B$$

$$Y = UZ$$

$$Z = P_{col}^{-1}X \rightarrow X = P_{col}Z$$

Perform this process my m times will solve the whole unknown Matrix X.

(ii) Pseudo Code:

```
A = […]
%initialisation of some intermediate variables
PJ_inv = I;
P_r_old = I;
P_c_old = I;
%Gussian elimination with complete pivoting
for k=1:n-1
%find the row and column index of the max. value
[m,max_row]=max(abs(A(k:n,k:n)));
[m,c_ind]=max(m);
r_ind = max_row(c_ind);
%transform this coordinates to the coordinates of A
r_ind=r_ind+k-1;
c_ind=c_ind+k-1;
%row and column permutation matrix at kth stage
P_r = I - (e_vector(r_ind,n)-e_vector(k,n))'*(e_vector(r_ind,n)-e_vector(k,n));
P_c = I - (e_vector(c_ind,n)-e_vector(k,n))'*(e_vector(c_ind,n)-e_vector(k,n));
%impose the permutation matrix to A matrix
A = P_r*A*P_c
%compute the permutation matrix
Pr = P_r*P_r_old
Pc = P_c_old*P_c,n
%update the permutation matrix at k-1 stage
P_r_old = Pr;
P_c_old = Pc;
%compute upper triangular matrix
gv = gauss_vector(A,k,n);
evec = e_vector(k,n);
LE = outer_product(gv,evec,n);
J = I-LE
A = J*A
%compute the inverse of J matrix
J_inv = I+LE
%compute the inverse of J matrix with permutation matrix
PJ_inv = PJ_inv*P_r*J_inv
end
%compute the lower triangular matrix
L = Pr*PJ_inv;
U = A;

%Perform forward and backward substitution to solve the matrix X
%Use forward substitution to compute Y of n*m size
B = Pr*B
for mm = 1:m
B(1,mm) = B(1,mm)/L(1,1)
for i = 2:n;
```

```
B(i,mm) = (B(i,mm)-L(i,1:i-1)*B(1:i-1,mm))/L(i,i);
end
end
Y(:,mm) = B(:,mm);

%Use backward substitution to compute Z of n*m size
for mm = 1:m
Y(n,mm) = Y(n,mm)/U(n,n)
for i = n-1:-1:1
Y(i,mm) = (Y(i,mm)-U(i,i+1:n)*Y(i+1:n,mm))/U(i,i);
end
end
Z(:,mm) = Y(:,mm);

%Use column permutation to re-order Z and get the solution X
X = Pc*Z
```

(iii) Very similar to the matrix problem in **2.1**, Gaussian Elimination with complete pivoting has been used to factorize A matrix in this problem, the computational complexity in terms of complete pivoting is already discussed in **2.1(iii)**, which is a $O(n^3)$ process. Remember that for this problem, the unknown X is no longer a vector, it becomes a $n\times m$ matrix. Thus, the computational complexity of forward and backward substitution become a little bit different. As the algorithm explained in **2.2(i)**, an additional for-loop of m times iteration is needed to solve the all the columns of X, so the substitution becomes a $O(mn^3)$ process.

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:

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

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 $\frac{\partial C(x,t)}{\partial t}$ and a central Crank-Nicholson time discretisation scheme for the $D\,\frac{\partial^2 C(x,t)}{\partial x^2}$ term. Use $i = 0,1,\ldots,N$, 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:

$$\boldsymbol{C}^n = (C_1^n C_2^n \ldots\ldots {C_{N-1}}^n)$$

Now explain why the equations can be written in the form

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

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

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

For $N = 10, D = 1, L = 1$ and $C_0 = 1$ write a MATLAB code that implements this discretisation until $t = 0.1s$. 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 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, 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:

(i) Forward in time discretisation is given by:

$$\frac{\partial C(x,t)}{\partial t} = \frac{C_i^{n+1} - C_i^n}{\Delta t}$$

Central Crank-Nicholson time discretisation is the average of the explicit scheme at (i,n) and implicit at $(i,n+1)$:

$$D\frac{\partial^2 C(x,t)}{\partial x^2} = \frac{D}{2}\frac{\left((C_{i+1}^{n+1} - 2C_i^{n+1} + C_{i-1}^{n+1}) + (C_{i+1}^n - 2C_i^n + C_{i-1}^n)\right)}{(\Delta x)^2}$$

Substitute the forward time discretisation and the central Crank-Nicholson time discretisation into the differential equation gives:

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

(ii) Rearrange the above discretised differential equation:

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

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

Substitute the σ into the equation:

$$\sigma = \frac{D \cdot \Delta t}{(\Delta x)^2}$$

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

If we write the equation in matrix form:

$$\begin{pmatrix} 1+2\cdot{}^{\sigma}/_{2} & -{}^{\sigma}/_{2} & 0 & 0 & 0 \\ -{}^{\sigma}/_{2} & 1+2\cdot{}^{\sigma}/_{2} & -{}^{\sigma}/_{2} & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & -{}^{\sigma}/_{2} & 1+2\cdot{}^{\sigma}/_{2} & -{}^{\sigma}/_{2} \\ 0 & 0 & 0 & -{}^{\sigma}/_{2} & 1+2\cdot{}^{\sigma}/_{2} \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ \dots \\ C_{N-2} \\ C_{N-1} \end{pmatrix}^{n+1}$$

$$= \begin{pmatrix} 1-2\cdot{}^{\sigma}/_{2} & {}^{\sigma}/_{2} & 0 & 0 & 0 \\ \frac{\sigma}{2} & 1-2\cdot{}^{\sigma}/_{2} & {}^{\sigma}/_{2} & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & {}^{\sigma}/_{2} & 1-2\cdot{}^{\sigma}/_{2} & {}^{\sigma}/_{2} \\ 0 & 0 & 0 & {}^{\sigma}/_{2} & 1-2\cdot{}^{\sigma}/_{2} \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ \dots \\ C_{N-2} \\ C_{N-1} \end{pmatrix}^{n}$$

Note that the big matrices can be written as:

$$\begin{pmatrix} 1+2\cdot{}^{\sigma}/_{2} & -{}^{\sigma}/_{2} & 0 & 0 & 0 \\ -{}^{\sigma}/_{2} & 1+2\cdot{}^{\sigma}/_{2} & -{}^{\sigma}/_{2} & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & -{}^{\sigma}/_{2} & 1+2\cdot{}^{\sigma}/_{2} & -{}^{\sigma}/_{2} \\ 0 & 0 & 0 & -{}^{\sigma}/_{2} & 1+2\cdot{}^{\sigma}/_{2} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} + \frac{\sigma}{2} \cdot \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 1-2\cdot{}^{\sigma}/_{2} & {}^{\sigma}/_{2} & 0 & 0 & 0 \\ \frac{\sigma}{2} & 1-2\cdot{}^{\sigma}/_{2} & {}^{\sigma}/_{2} & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & {}^{\sigma}/_{2} & 1-2\cdot{}^{\sigma}/_{2} & {}^{\sigma}/_{2} \\ 0 & 0 & 0 & {}^{\sigma}/_{2} & 1-2\cdot{}^{\sigma}/_{2} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} - \frac{\sigma}{2} \cdot \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

Thus, the differential equation can be written in the form:

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

Where:

$$I=\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&\dots&\dots&\dots&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix},B=\begin{pmatrix}2&-1&0&0&0\\-1&2&-1&0&0\\0&\dots&\dots&\dots&0\\0&0&-1&2&-1\\0&0&0&-1&2\end{pmatrix},\boldsymbol{C}=\begin{pmatrix}C_1\\C_2\\\dots\\C_{N-2}\\C_{N-1}\end{pmatrix}$$

(iii) The following function creates the B matrix by specifying its size and outputs the B matrix in sparse matrix format:

```
function B = Matrix1D(n)
% This function creates the B matrix for the finite difference scheme
%   n = size of B matrix

for (i = 1:n)
    B(i,i) = 2; %assign value to the main diagonal
end
for (i = 1:n-1)
    B(i,i+1) = -1; %assign value to the upper band of diagonal
end
for (i = 2:n)
    B(i,i-1) = -1; %assign value to the lower band of diagonal
end

%convert the B matrix into sparse matrix format
B = sparse(B);

end
```

If the input to the function is $n = 4$, it will produce the matrix on left-hand side and will be stored in the format shown in the right-hand side:

$$\begin{pmatrix}2&-1&0&0\\-1&2&-1&0\\0&-1&2&-1\\0&0&-1&2\end{pmatrix}\rightarrow\begin{pmatrix}2&-1&-1&2&-1&-1&2&-1&-1&2\\1&1&2&2&2&3&3&3&4&4\\1&2&1&2&3&2&3&4&3&4\end{pmatrix}$$

(IV) The B matric obtained deduced from the differential equation is a tridiagonal matrix which has the non-zero values on three middle diagonals, all other entries are null. Due to the nature of this tridiagonal matrix, it is unnecessary to include 0 entries when performing Cholesky factorisation. The given Cholesky factorisation for general symmetric positive definite can be simplified to improve the efficiency. Remember the Cholesky factorisation has the form of $B = U^TU$, so:

$$\begin{pmatrix}b_{11}&b_{12}&0&0\\b_{21}&b_{22}&b_{23}&0\\0&b_{32}&b_{33}&b_{34}\\0&0&b_{43}&b_{44}\end{pmatrix}=\begin{pmatrix}u_{11}&0&0&0\\u_{12}&u_{22}&0&0\\0&u_{23}&u_{33}&0\\0&0&u_{34}&u_{44}\end{pmatrix}\begin{pmatrix}u_{11}&u_{12}&0&0\\0&u_{22}&u_{23}&0\\0&0&u_{33}&u_{34}\\0&0&0&u_{44}\end{pmatrix}$$

Matrix multiplication of right hand side gives:

$$B = U^T U = \begin{pmatrix} u_{11}^2 & u_{11}u_{12} & 0 & 0 \\ u_{11}u_{12} & u_{22}^2 + u_{12}^2 & u_{22}u_{23} & 0 \\ 0 & u_{22}u_{23} & u_{33}^2 + u_{23}^2 & u_{33}u_{34} \\ 0 & 0 & u_{33}u_{34} & u_{44}^2 + u_{34}^2 \end{pmatrix}$$

The following expressions can be deduced from the matrix above:

$$b_{11} = u_{11}^2 \rightarrow u_{11} = \sqrt{b_{11}}$$

$$b_{22} = u_{22}^2 + u_{12}^2 \rightarrow u_{22} = \sqrt{b_{22} - u_{12}^2}$$

$$b_{33} = u_{33}^2 + u_{23}^2 \rightarrow u_{33} = \sqrt{b_{33} - u_{23}^2}$$

$$\dots$$

$$b_{12} = u_{11}u_{12} \rightarrow u_{12} = b_{12}/u_{11}$$

$$b_{23} = u_{22}u_{23} \rightarrow u_{23} = b_{23}/u_{22}$$

$$b_{34} = u_{33}u_{34} \rightarrow u_{34} = b_{34}/u_{33}$$

$$\dots$$

So, the elements of U matrix are given by:

$$u_{i+,j+1} = \sqrt{b_{i,j} - u_{i,j+1}^2}\,, i = j = 1 \dots n-1$$

$$u_{i,j+1} = b_{i,j+1}/u_{i,i}\,, i \neq j$$

Since the U matrix has non-zero values on 2 diagonals, the forward and back substitution can be simplified, consider the forward substitution:

$$Bx = b \rightarrow U^T U x = b$$

$$U^T y = b$$

$$\begin{pmatrix} u_{11} & 0 & 0 & 0 \\ u_{12} & u_{22} & 0 & 0 \\ 0 & u_{23} & u_{33} & 0 \\ 0 & 0 & u_{34} & u_{44} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

$$y_1 = \frac{b_1}{u_{11}}$$

$$y_2 = \frac{b_2 - u_{21}y_1}{u_{22}}$$

$$y_3 = \frac{b_3 - u_{32}y_2}{u_{33}}$$

$$y_i = \frac{b_i - u_{i,i-1}y_{i-1}}{u_{ii}}$$

Similarly, for backward substitution:

$$Ux = y$$

$$\begin{pmatrix} u_{11} & u_{12} & 0 & 0 \\ 0 & u_{22} & u_{23} & 0 \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

$$x_4 = \frac{y_4}{u_{44}}$$

$$x_3 = \frac{y_3 - u_{34}y_4}{u_{33}}$$

$$x_2 = \frac{y_2 - u_{23}y_3}{u_{22}}$$

$$x_i = \frac{y_i - u_{i-1,i}y_i}{u_{ii}}$$

Script to perform Cholesky factorisation for a tridiagonal matrix:

```
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);%get the size of input matrix
if (m~=n || max(max(abs((A+A')/2-A)))~=0 )%check if it is symmetric or not
        error('error: A is not symmetric');%if symmetric then print warning message
end
%perform the Cholesky factorisation
for j = 1:n %starting from the 1st column
    s = 0.0;%initialise the intermediate term for calculating the diagonal entries
    for k = 1:j-1
        A(k,j) = A(k,j)/A(k,k);%calculate entries with index i<j
        s = s+A(k,j)*A(k,j);%compute the intermediate term for calculating the
diagonal entries
    end
    s = A(j,j)-s;%compute the intermediate term for calculating the diagonal entries
    if s<0 %check if it is definite positive or not
            error('error: A is not definite positive');%if not definite positive
send message
    end
    A(j,j) = sqrt(s);%calculate the diagonal entries
end
```

Script to perform forward/backward substitution for a tridiagonal matrix:

```
function b = spsl_tridiag(U,b)
%Solve Ax=b for symmetric definite positive matrices after
%the the upper triangular matrix U was computed by spfa.m
%based on Dr Kevin Gouder script for AEM-ADV08 2016-2017

[~,n]=size(U);
%Solve U'y=b the values of y are written directly in b to save memory;
b(1) = b(1)/U(1,1); %calculate the first entry;
for k = 2:n%for loop for the forward substitution starts from the second entry
    t = dot(U(k-1,k),b(k-1)); %compute the term u(i,i-1)*y(i-1)=(u(i-1,i)')*y(i-1)
    b(k) = (b(k)-dot(U(k-1,k),b(k-1)))/U(k,k);%update the kth entry
end
%Solve Ux=y,
%to save memory the values of x are written inside the array b.
for k = n:-1:2%for loops for the backward substitution start from the last entry
    b(k) = b(k)/U(k,k);%update the kth entry
    b(k-1) = b(k-1)-b(k)*U(k-1,k);%compute the (k-1)th term
end
b(1) = b(1)/U(1,1);%compute the first entry
```

A code has been developed to solve the problem analytically and numerically:

```
clear all;
close all;

%given parameters
N = 10;
L = 1;
D = 1;
C0 = 1; %initial condition
t = 0.1;
%dx and x position
dx = L/(N);
x = [0:dx:L];
%initilisation of C matrix
C_a(1:N+1) = 0;
i = 1:N+1;
%compute the sum term in the analytical sol.
for m = 1:100
    C_a(i) = C_a(i)+(1/((2*m-1)*pi)).*exp(-(D*((2*m-1)^2)*(pi^2)*t/(L^2)))*sin((2*m-
1)*pi*x(i)/(L));
end
C_a = 4*C0*C_a;%compute the analytical sol. of C
%plot the results;
figure;
hold on;
plot(x,C_a,'-x');
grid on;
xlabel('X');
ylabel('C');

%compute the dx, dt and sigma
dx = L/(N);
dt = 0.001;
sig = D*dt/(dx^2);
C(1:N-1,1) = C0;%initial condition
b = (eye(N-1)-sig/2*Matrix1D(N-1))*C;%initialisation of RHS of matrix euqation
iter = t/dt;%compute the number of iterations
U = spfa_tridiag(eye(N-1)+sig/2*Matrix1D(N-1));%perform cholesky factorisation
%compute the C numerically
for i = 1:iter
    C = spsl_tridiag(U,b);
    b = (eye(N-1)-sig/2*Matrix1D(N-1))*C;
end
%include the i = 0 and i = N point to the C
C(2:N) = C(1:N-1);
C(1) = 0;
C(N+1) = 0;
x = [0:dx:L];
%plot the results
% figure;
plot(x,C,'r-o');
grid on;
xlabel('X');
ylabel('Concentration');
legend('Analytical Solution','Numerical Solution');
title(strcat('t =',{' '}, num2str(t)))
```

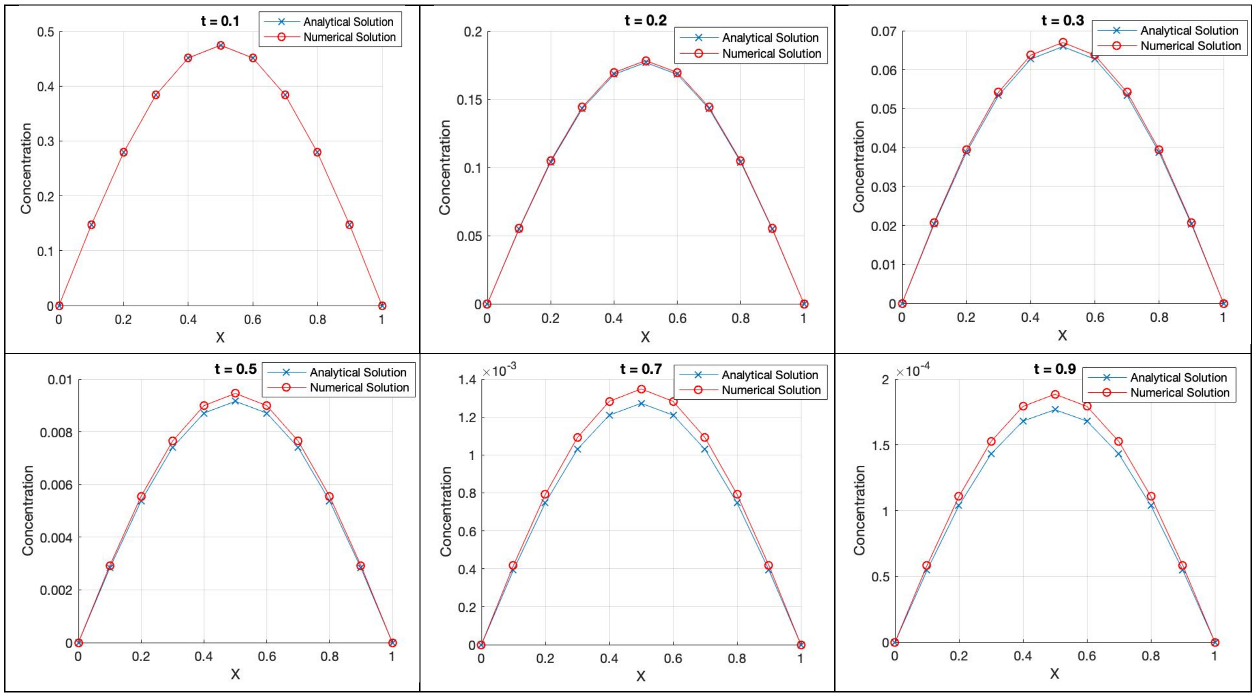

Figure 1:Numerical and analytical solution at different time, N = 10

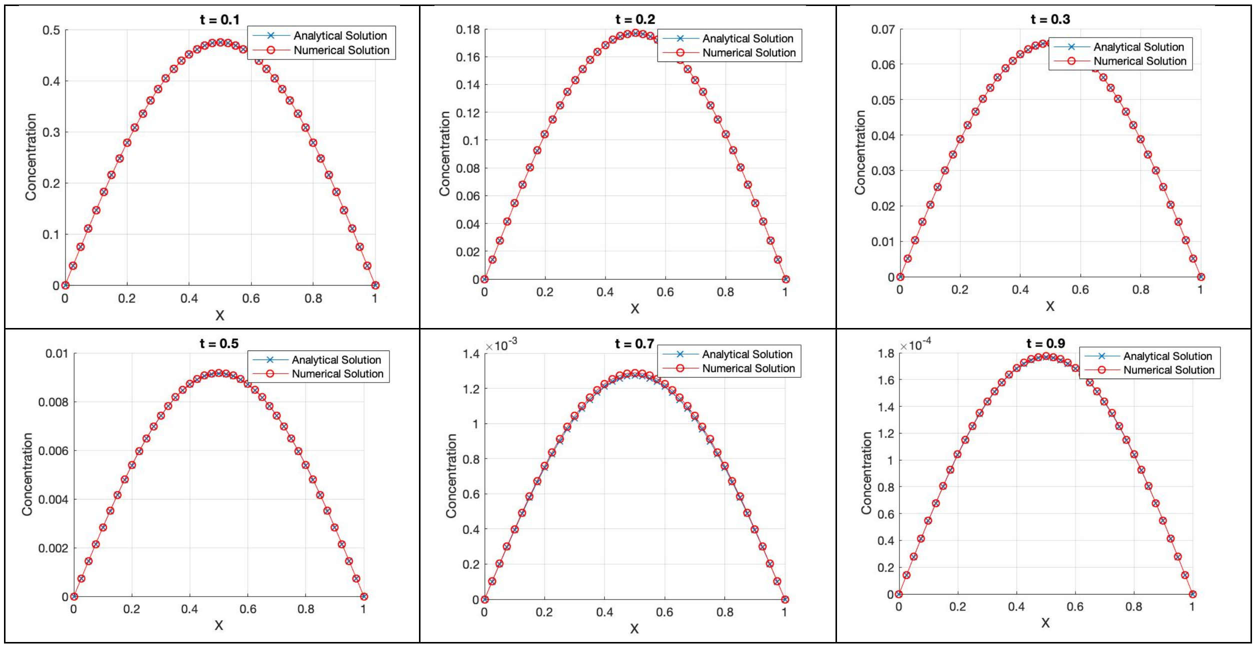

Figure 2: Numerical and analytical solution at different time, N = 40

Figure 1 and *Figure 2* above show the contour plots at different time using different number of points. Observing the *Figure 1*, the numerical solution tends to deviate from analytical solution as the time goes and it seems the analytical solution diffuse faster than the numerical solution. Now consider the *Figure 2*, it can be clearly observed that the contour plots in this figure is much smoother than those in *Figure 1*, as 40 points were used to compute the results. The numerical and analytical results match very well in the case of using 40 points; even at later time, the deviation of numerical from analytical solution is negligible, it is expected to have more accurate solution with larger number of points used.

Question 4: 5% of total

Consider the matrix

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

For what values of α is:

i) the matrix diagonally dominant?

ii) does the Jacobi method converge?

Question 4:

(i) A matrix is said to be diagonally domain if:

$$|a_{ii}| \geq \sum_{i \neq j} |a_{ij}|, for\ all\ i$$

For the given matrix:

$$|a_{ii}| = 1$$

$$\sum_{i \neq j} |a_{ij}| = |\alpha + 0|$$

$$|a_{ii}| \geq \sum_{i \neq j} |a_{ij}| \rightarrow 1 \geq |\alpha|$$

Thus, the matrix is diagonally dominant if the of α is between $-1 < \alpha < 1$.

(ii) The Jacobi method will converge to the unique solution of matrix equation $Ax = b$ if the given matrix A is a strictly diagonally dominant matrix.

Question 5: 25% of total

Consider the two-dimensional boundary value problem of steady state heat conduction in a L-shaped plate region (Fig. 1) governed by the Poisson equation:

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

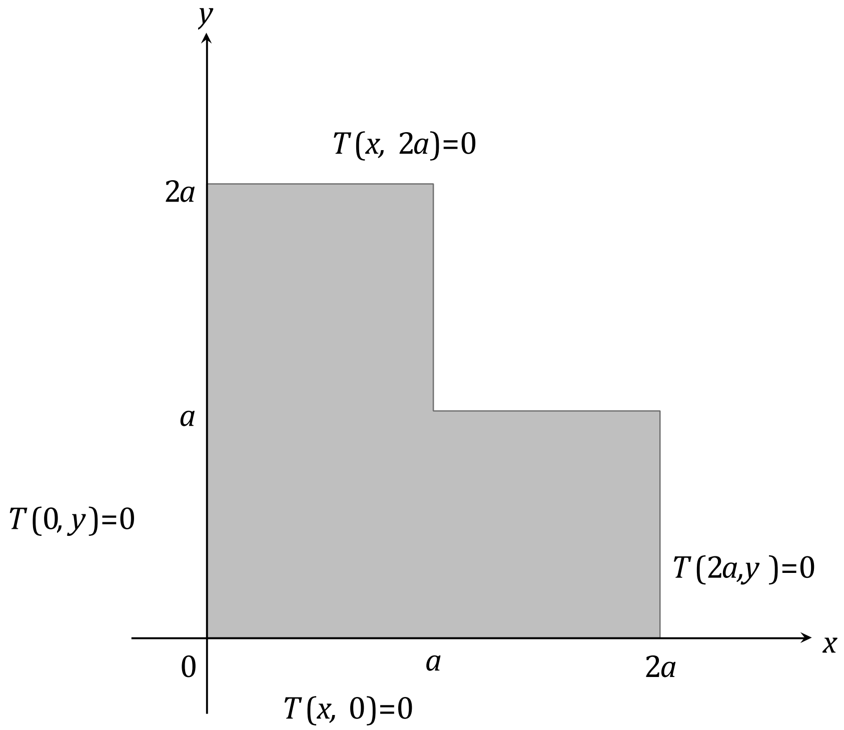

Figure 1: L-shape region subjected to boundary temperature.

and subjected to the boundary conditions:

$$T(0,y) = 0 \quad ; \quad T(2a,y) = 0$$
$$T(x,0) = 0 \quad ; \quad T(x,2a) = 0$$

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

as shown in Fig. 2 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.

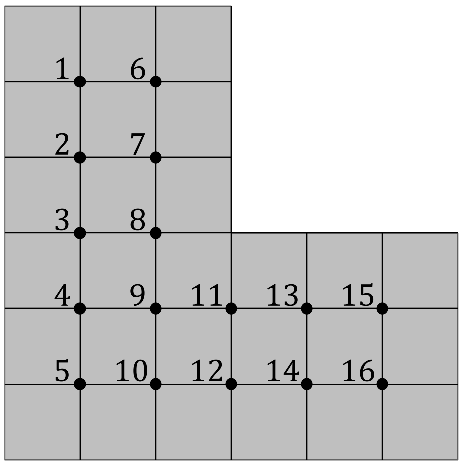

Figure 2: Coarse discretisation for the L-shaped region $N_x = N_y = 3$.

1. Explain why the finite difference matrix A to solve eq. 5 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 $N_x = 5$ and $N_y = 5$, $a = 1\text{m}$, 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. 5 (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, commenzt on the number of iterations needed to reach convergence changing the number of elements inside the domain.

Question 5:

5.1 The partial differential equation to be solved on the domain is a Poisson equation:

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

Using a second order central finite difference scheme give the following discretized form:

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

Assume $\Delta x = \Delta y$:

$$T_{i-1,j} - 2T_{i,j} + T_{i+1,j} + T_{i,j-1} - 2T_{i,j} + T_{i,j+1} = (\Delta x)^2 q_{i,j}$$

$$T_{i-1,j} - 4T_{i,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1} = (\Delta x)^2 q_{i,j}$$

$$-T_{i-1,j} + 4T_{i,j} - T_{i+1,j} - T_{i,j-1} - T_{i,j+1} = -(\Delta x)^2 q_{i,j}$$

Consider a square domain with $N_x = N_y = 3$, which mean we need to solve $2N_x - 1 = 5$ points for the first domain and $N_x - 1 = 2$ points for second domain. The finite difference matrix has the following form:

$$\begin{pmatrix}
4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 4 & -1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 4 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 4 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 4 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 4
\end{pmatrix}$$

Looking at the example finite difference matrix, it is clearly symmetric, as its transpose matrix is equal to itself. It is trivial to show that this matrix is diagonally dominant, remember that a diagonally dominant matrix with all diagonal elements positive is also positive definite, therefore this finite difference matrix is positive definite.

The matrix can be considered as a banded matrix although it has some zero diagonals within the band. The matrix can be think of composed by three types of matrices, a tridiagonal matrix, a negative identity matrix and a null matrix. Let us define the following matrix:

$$T_2 = \begin{pmatrix} T_{1,1} & T_{1,2} \\ T_{2,1} & T_{2,2} \end{pmatrix} \quad -I_2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \quad Z = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

The sub index of the matrix denotes its size. The finite difference matrix can be then represented as the following block matrix:

$$A_{N_x=3} = \begin{pmatrix} T_5 & -I_5 & Z & Z & Z \\ -I_5 & T_5 & -I_2 & Z & Z \\ Z & -I_2 & T_2 & -I_2 & Z \\ Z & Z & -I_2 & T_2 & -I_2 \\ Z & Z & Z & -I_2 & T_2 \end{pmatrix}$$

The band width of such matrix is equal to the size of T_{2N_x-1} plus 1, in other words the bandwidth is given by $(2N_x - 1) + 1 = 2N_x$. Therefore, if the domain is of size $N_x = 10$, the bandwidth of the associated A matrix is $2N_x = 2 \cdot 10 = 20$. The size of the A matrix would be:

$$\left((2N_x - 1)(N_x - 1) + N_x(N_x - 1)\right)\times\left((2N_y - 1)(N_y - 1) + N_y(N_y - 1)\right)$$

$$= (19 \cdot 9 + 10 \cdot 9)(19 \cdot 9 + 10 \cdot 9) = 261\times261$$

$$= 68121\, points$$

5.2 The following code has been developed to create the finite difference matrix for L shape domain:

```
function A = MatrixA2D(n)
% This function creates the finite difference matrix for 2D L-shape domain
%   n - size of the square domain

%calculate the number of points inside the rectangular domain
n1 = 2*n-1;
%calculate the number of points inside the square domain
n2 = n-1;

%create the tridiagonal matrix for the rectangular domain
for i = 1:n1
    T1(i,i) = 4; %assign value to the main diagonal
end
for i = 1:n1-1
    T1(i,i+1) = -1; %assign value to the upper band of diagonal
end
for i = 2:n1
    T1(i,i-1) = -1; %assign value to the lower band of diagonal
end
%create the tridiagonal matrix for the square domain
for i = 1:n2
    T2(i,i) = 4; %assign value to the main diagonal
end
for i = 1:n2-1
    T2(i,i+1) = -1; %assign value to the upper band of diagonal
end
for i = 2:n2
    T2(i,i-1) = -1; %assign value to the lower band of diagonal
end

%create the negative identity matrices for both rectangular and square domain
I1 = -eye(n1);
I2 = -eye(n2);

%create the zero matrices for rectangular and square domain
Z1 = zeros(n1);
Z2 = zeros(n2);

%create the finite difference matrix for rectangular domain
```

```
 for i = 1:n-1
     for j = 1:n-1
        AA{i,j} = Z1; %assign value to the main diagonal
     end
 end
 for i = 1:n-1
    AA{i,i} = T1; %assign value to the main diagonal
 end
for i = 1:(n-1)-1
    AA{i,i+1} = I1; %assign value to the upper band of diagonal
end
for i = 2:n-1
    AA{i,i-1} = I1; %assign value to the lower band of diagonal
end
AA = cell2mat(AA);

%create the finite difference matrix for square domain
 for i = 1:n
     for j = 1:n
        CC{i,j} = Z2; %assign value to the main diagonal
     end
 end
 for i = 1:n
    CC{i,i} = T2; %assign value to the main diagonal
 end
for i = 1:(n-1)
    CC{i,i+1} = I2; %assign value to the upper band of diagonal
end
for i = 2:n
    CC{i,i-1} = I2; %assign value to the lower band of diagonal
end
CC = cell2mat(CC);

%creation the matrix for the rest entries of the big finite difference matrix
BB{1,1} = zeros(length(I2),length(AA)-length(I2));
BB{1,2} = I2;
BB{2,1} = zeros(length(CC)-length(I2),length(AA)-length(I2));
BB{2,2} = zeros(length(CC)-length(I2),length(I2));
BB = cell2mat(BB);

%create the combined rectangular and square finite difference matrix
A{1,1} = AA;
A{1,2} = BB';
A{2,1} = BB;
A{2,2} = CC;
A = cell2mat(A);

%store the A matrix in sparse matrix format
A = sparse(A);
end
```

The above function outputs the finite difference matrix with the pattern explained in the previous section **5.1**. For $N_x = N_y = 5$, the shape of the matrix is shown in the *Figure 3* below. As can be seen, the non-zeros are concentred around the main diagonal and elsewhere are entries of zero value. Since a great portion of the matrix is 0, the matrix has been saved in the sparse matrix format in order to avoid wasting unnecessary storage, only the non-zero values are stored. The *'sparse'* function in MATLAB converts the sparse matrix into the sparse matrix format and allows the user to use the matrix as usual.

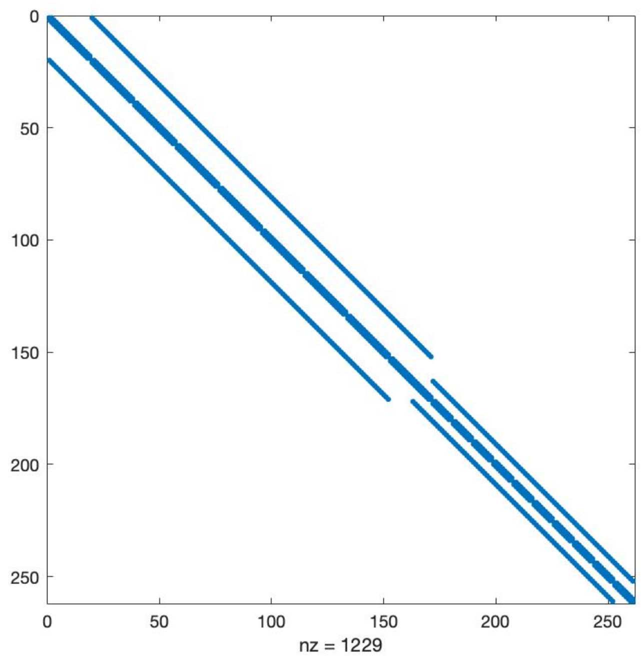

Figure 3: The shape of A matrix

5.3 The following code has been developed to solve the given Poisson equation using Gauss-Seidel iterative method:

```
clear all; close all

% number of elements
n = 5;
%domain length
a = 1;
dx = a/n;

%create the finite difference matrix
A = MatrixA2D(n);

%shape of the matrix
figure;
spy(A);

%initialisation of the unknown matrix
T = zeros(length(A),1);

%define the source term q
q = -ones(length(A),1);
b = (dx^2)*q;

%split the A into L+D+U
L = zeros(length(A));
for i = 2:length(A)
   L(i:length(A),i-1) = A(i:length(A),i-1);
end
for i = 1:length(A)
   D(i,i) = A(i,i);
end
U = zeros(length(A));
for i = 1:length(A)-1
   U(i,i+1:length(A)) = A(i,i+1:length(A));
end

%set the left hand side of the gauss-seidel method
lhs = b-U*T;
```

```
%set convergence criteria
conv_crit = 1e-6;
%initialise teh iteration number
iterNum = 0;
%initialise the err
err = 1;
%Gauss-seidel iterative method
while (max(max(abs(err)))>conv_crit)
    iterNum = iterNum + 1;
    %update the T matrix at each iteration
    T_new = forward_substitution(L+D,lhs,length(L+D));
    %update the lhs of GS method at each iteration
    lhs = b-U*T_new;
    err = T - T_new;
    T = T_new;
end

%generate the meshgrid
x=0:dx:2;
y=2:-dx:0;
[X,Y]=meshgrid(x,y);
%reshape the 1D T matrix into 2D matrix
for j = 1:n-1
sol(:,j+1) = T(1+(2*n-1)*(j-1):1+(2*n-1)*(j-1)+2*n-2);
end
for j = 1:n
sol(n+1:2*n-1,n+j) = T(1+(2*n-1)*(n-2)+2*n-1+(n-1)*(j-1):1+(2*n-1)*(n-2)+2*n-1+(n-
1)*(j-1)+(n-2));
end

%add the values at boundary
sol(2:2*n,:) = sol(1:2*n-1,:);
sol(1,:) = 0;
sol(2*n+1,:) = 0;
sol(:,2*n+1) = 0;
%plot the results
figure;
contourf(X,Y,sol);
figure;
surf(X,Y,sol);
xlabel('X');
ylabel('Y');
zlabel('Temperature');
```

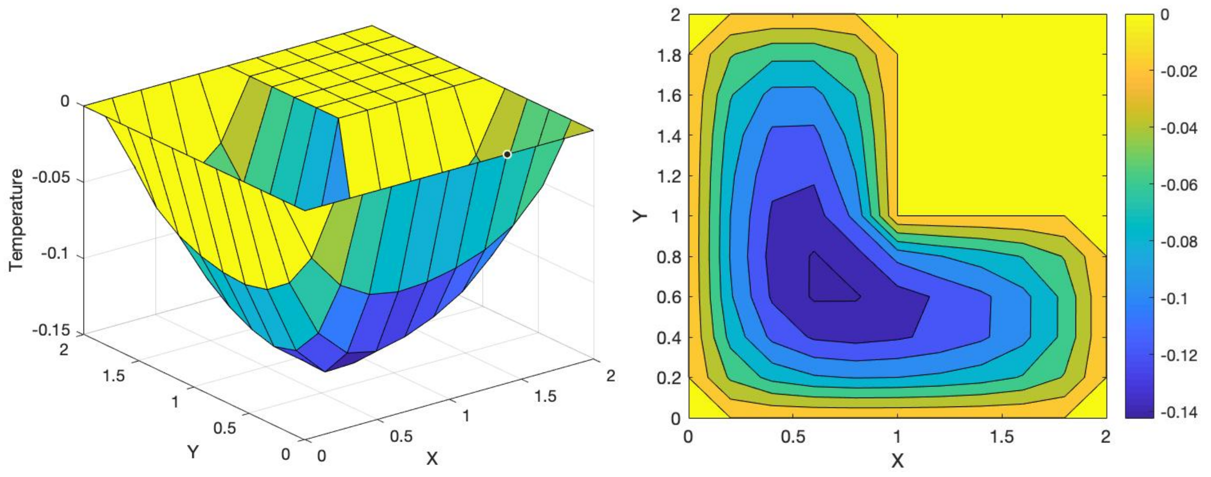

Figure 4: 2D Solution for $N_x = N_y = 5$

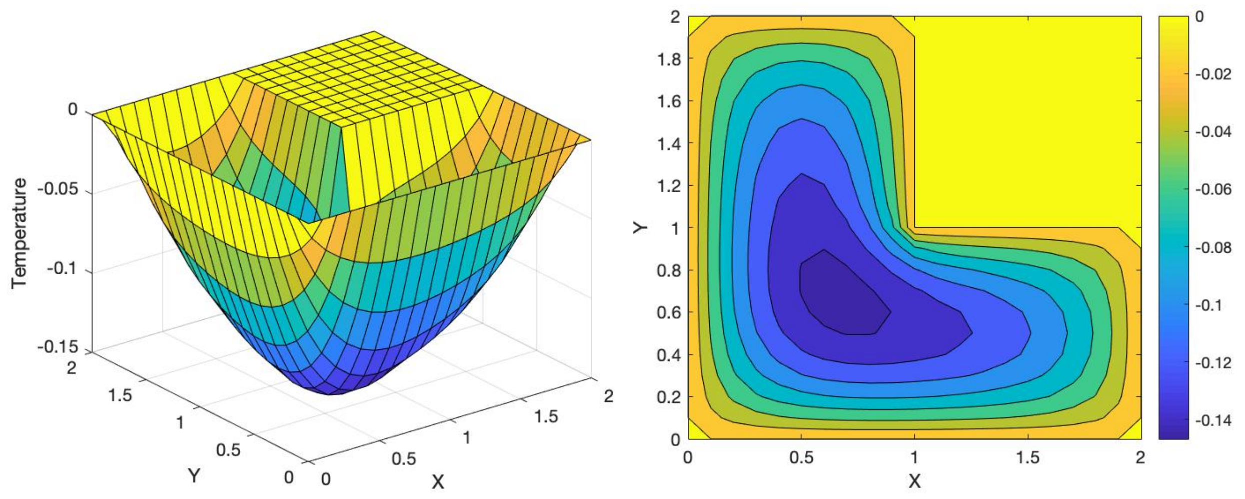

Figure 5: 2D solution for $N_x = N_y = 10$

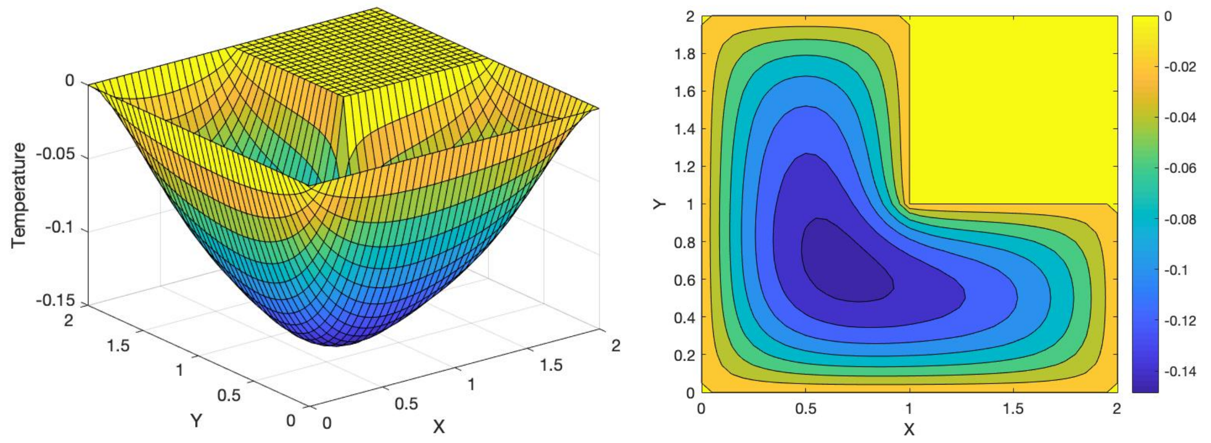

Figure 6: 2D solution for $N_x = N_y = 20$

5.4

Table 3: Number of iteration for different domain size

Number of elements	5	10	20
Number of iteration	53	187	635

Table 3 shows the number of iteration needed to reach the convergence of solution, the convergence criteria is set to be 10^{-6}. This means that once the difference of solution between 2 iteration steps is less than 10^{-6}, then the solution will be considered as converged and the iteration will stop. As can be seen from the table, increasing the number of elements lead to the increment in number of iterations. This is expected since more elements means that the values from boundary need more steps to propagate to the domain so more iteration numbers are needed to reach the same convergence level.