Lecture 1

Introduction

1.1 Thumbnail

- The Course
  - Purpose
  - Pre-requisite knowledge
  - Course Structure
  - Materials
  - Manner of working
- This week
  - Motivation and application examples
  - Problems to be addressed in this course
  - Learning outcomes
  - Gaussian elimination
  - LU factrisation

1.2 The Course

1.2.1 Purpose

Matrix analysis and numerical linear algebra play a key role in mathematics and engineering problems with rigours theory and many real world applications. Matrix analysis and algorithms have different objectives, depending on the field of application. These include,

- Efficiently solving linear systems.
- Solving polynomial matrix equations.
- Optimisation problems: ex: systems' control, systems' modelling, machine learning.
- Analysing large scale problems.
- Analysing the computational complexity and efficiency of various algorithms.

The aim in this course is to understand the mathematical principles underlying the design and the analysis of effective methods and algorithms for solving these problems.

1.2.2 Pre-requisite Knowledge

- MA106 Linear Algebra - providing methodological foundations
- MA124 Mathematics by Computer - being a useful introduction to some of the scientific computing aspects and programming elements of the module
- MA251 Algebra 1: Advanced Linear Algebra - developing more complex techniques in matrix classification and their properties in particular mathematical settings of interest
- MA259 Multivariable Calculus - providing the language and concepts needed for some of the typical proofs we will encounter.

1.2.3 Course Structure

This is a tentative plan for the course.

- Week 1: Course Introduction and Overview, Gaussian Elimination, and LU factorisation.
- Week 2: Gaussian Elimination with Pivoting, Matrix Norms, and Eigenvalues and Eigenvectors.
- Week 3: Matrix Norms, Floating point representation, and Error Analysis.
- Week 4: Conditioning, Backward Error Analysis, and Error Analysis of the Gaussian Elimination.
- Week 5: Computational Cost, Divide & Conquer Algorithms, and Least Squares Problems.
- Week 6: Singular Value Decomposition, Conditioning of LSQ, and QR factorisation.
- Week 7: QR factorisation with Householder reflections, Linear Iterative Methods, and the Jacobi method.
- Week 8: Computational Complexity of Linear Iterative Methods, Steepest Descent, and Conjugate Gradient method.
- Week 9: CGM Comparison of SD and CG, and Computational Complexity and Preconditioning.
- Week 10: Eigenvalue Problems, Power Iteration, and Simultaneous Iteration and the QR Method.

1.2.4 Materials and Manner of Working

- Required Reading:
  - Extensive Lecture notes (available on Moodle)
  - AM Stuart and J Voss, Matrix Analysis and Algorithms, script. I refer to relevant chapters/sections/pages in my lecture notes.
- I will break the concepts down using relevant examples and tutorial materials to facilitate learning

- I will discuss with you the solution methodology and how to recall, reconstruct and apply ideas rather than just memorizing some steps for solving a particular problem which guarantees achievement of high marks.
- I will work on strengthening your understanding of the concept by carefully preparing my exams and evaluating your acquisition of material that concentrate on developing methodological solution and critical thinking to the raised problem.
- I am also happy to deviate in a lecture in response to constructive feedback given by you.

1.2.5 Reading list

- AM Stuart and J Voss, Matrix Analysis and Algorithms, script.
- G Golub and C van Loan, Matrix Computations, 3. ed., Johns Hopkins Univ. Press, London 1996.
- NJ Higham, Accuracy and Stability of Numerical Algorithms, SIAM 1996.
- RA Horn and CR Johnson, Matrix Analysis, Cambridge University Press 1985.
- D Kincaid and W Cheney, Numerical Analysis, 3. ed., AMS 2002.
- LN Trefethen and D Bau, Numerical Linear Algebra, SIAM 1997.

Lecture notes week 1

The purpose of this lecture

- Introduction to Matrix Applications: Provide an overview of the diverse applications of matrices, showcasing examples in fields like biomedical, stock market, and digital image processing to illustrate the relevance and versatility of matrices in real-world scenarios.
- Outline of Course Problems: Briefly discuss the specific problems and topics that will be addressed throughout the course, giving you a roadmap of the learning journey ahead.
- Highlight of Learning Outcomes: Present the expected learning outcomes, emphasizing the acquisition of skills in modern problem-solving techniques, matrix manipulations, and application of these methods to real-world problems.

1.3 Motivation and application examples

The use of matrices is extremely important in analysing and understanding many systems such as: engineering systems, image processing, quantum mechanics, stock prices prediction, graph theory,biomedical applications, and many others. Systems could be static/memoryless (systems whose response is due to present input alone) or dynamic (systems which are evolving in time). Although the theory in this course will focus on static systems, its generalisation to dynamical systems should be straight forward.
Application examples where matrices and numerical analysis are used,

- Biomedical and Bioinformatics (EEG,MEG, patient records).
  https://link.springer.com/chapter/10.1007/978-3-030-26036-1_25
- Stock market.
- meteorological (sunspot activity, weather patterns).
- consumer databases (airline passenger demand, electricity load demand).
- Engineering (crack propagation, fault diagnosis).

1.3.1 Application Examples

- **Solving Systems of Equations**

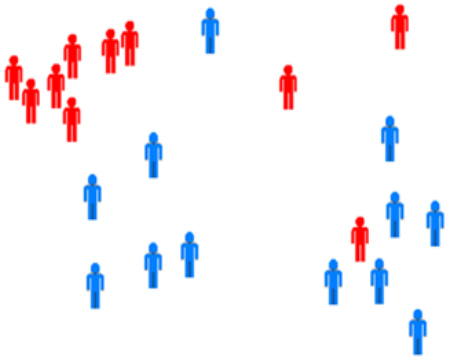
$$\begin{aligned} 5x - 2y &= 6 \\ -3x + 4y &= 2 \end{aligned} \tag{1.1}$$

This is equivalent to,

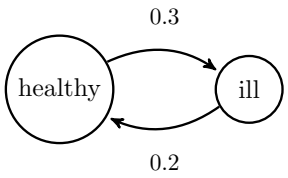
$$\begin{pmatrix} 5 & -2 \\ -3 & 4 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 6 \\ 2 \end{pmatrix} \tag{1.2}$$

Question: which input vector $(x\ y)^T$ does this matrix map to the vector $(6\ 2)^T$?

- **Epidemic breakout**



Question: what is going to happen in the long run.



In the first day.
Healthy people: $0.7 \times (200)$ will stay healthy, $0.2 \times (200)$ of the ill become healthy, thus,

$$0.7 \times (200) + 0.2 \times (200) = 180$$

Ill people: $0.8 \times (200)$ will stay ill, $0.3 \times (200)$ of the healthy become ill, thus,

$$0.8 \times (200) + 0.3 \times (200) = 220$$

In the second day.
Healthy people: $0.7 \times (180)$ will stay healthy, $0.2 \times (220)$ of the ill become healthy. Thus,

$$0.7 \times (180) + 0.2 \times (220) = 170$$

Ill people: $0.8 \times (220)$ will stay ill, $0.3 \times (180)$ of the healthy become ill, thus,

$$0.8 \times (220) + 0.3 \times (180) = 230$$

We need to continue in order to know what will happen in the long run.

Note that the above can be written in matrix-vector form as follows,

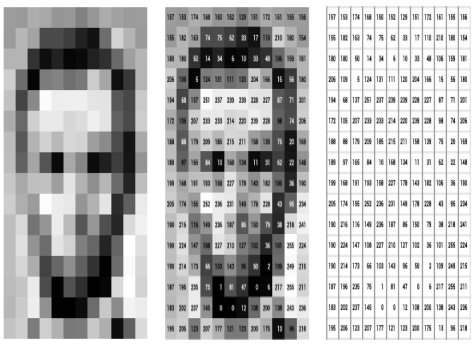
$$\underbrace{\begin{pmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{pmatrix}}_{\text{Markov Matrix}} \times \begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} \tag{1.3}$$

where x_k is the population of healthy people and y_k is the population of ill people.

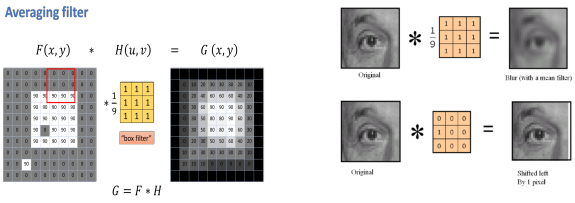
- **Digital image processing** A digital image is made of a number of pixels each of single color. Those colors can be represented by some numerical values.
  Why image processing?
  - Numbers identification

  - Suspects in a crime identification

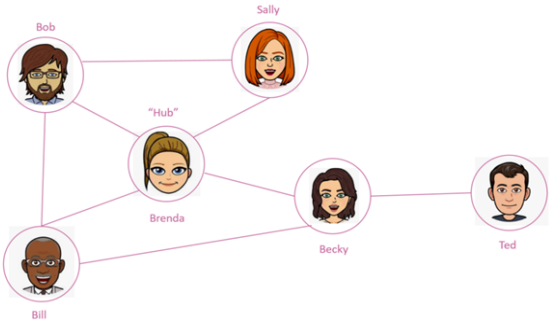
  - Face recognition



  Examples of image processing: blurring an image, detecting edges, sharpening an image, etc, all is about manipulating the pixels in a very specific way



- **Networks or graph theory**
  Example: a group of co-workers and their mutual friendships.



The table representing the above graph is,

	Bob	Sally	Brenda	Bill	Becky	Ted
Bob	0	1	1	1	0	0
Sally	1	0	1	0	0	0
Brenda	1	1	0	1	1	0
Bill	1	0	1	0	1	0
Becky	0	0	1	1	0	1
Ted	0	0	0	0	1	0

Or equivalently, its matrix representation is as follows,

$$A = \begin{array}{c} \\ Bob \\ Sally \\ Brenda \\ Bill \\ Becky \\ Ted \end{array} \begin{array}{c} \begin{array}{cccccc} Bob & Sally & Brenda & Bill & Becky & Ted \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \tag{1.4}$$

In graph theory, this is known as adjacency matrix. It tells us how many paths of length 1 exists between any two nodes. Note: path means any sequence of edges that joins a sequence of vertices.

But now what if I want to see how many mutual matches two people have? This can be obtained by just multiplying the adjacency matrix by itself or squaring it.

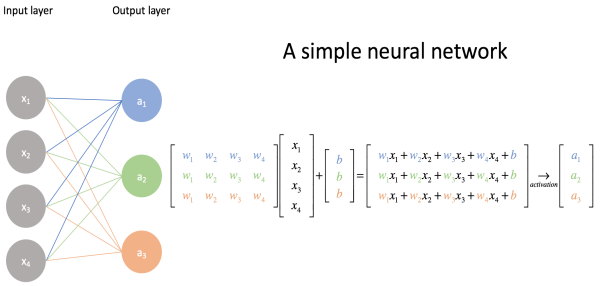
$$A^2 = \begin{array}{c} \\ Bob \\ Sally \\ Brenda \\ Bill \\ Becky \\ Ted \end{array} \begin{array}{c} \begin{array}{cccccc} Bob & Sally & Brenda & Bill & Becky & Ted \end{array} \\ \begin{pmatrix} 3 & 1 & 2 & 1 & 2 & 0 \\ 1 & 2 & 1 & 2 & 1 & 0 \\ 2 & 1 & 4 & 2 & 1 & 1 \\ 1 & 2 & 2 & 3 & 1 & 1 \\ 2 & 1 & 1 & 1 & 3 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \end{array} \tag{1.5}$$

Then if we cube the matrix,

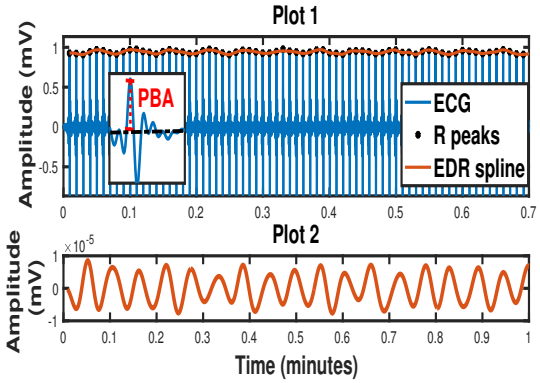
$$A^3 = \begin{array}{c} \\ Bob \\ Sally \\ Brenda \\ Bill \\ Becky \\ Ted \end{array} \begin{array}{c} \begin{array}{cccccc} Bob & Sally & Brenda & Bill & Becky & Ted \end{array} \\ \begin{pmatrix} 4 & 5 & 7 & 7 & 3 & 2 \\ 5 & 2 & 6 & 3 & 3 & 1 \\ 7 & 6 & 6 & 7 & 7 & 1 \\ 7 & 3 & 7 & 4 & 6 & 1 \\ 3 & 3 & 7 & 6 & 2 & 3 \\ 2 & 1 & 1 & 1 & 3 & 0 \end{pmatrix} \end{array} \tag{1.6}$$

we get all the paths of length 3 from one person to another. So, the original matrix to some power, tells us how many paths of that length exist between any two nodes.

- **Machine learning and neural networks** These are coded with and manipulated by matrices.



- **ECG signal** https://link.springer.com/chapter/10.1007/978-3-030-26036-1_25



1.4 Problems to be addressed in this course

Systems of Linear Equations (SLE)

Given $A \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$

$$\text{find } x \in \mathbb{C}^n \text{ such that } Ax = b.$$

Lots of theory has already been provided in courses on linear algebra such as solvability (non-singular matrices), characterisation of linear maps (Jordan canonical form).
Example of methods you have already studied (Gaussian elimination).
Questions:

- When do we expect these direct methods not work?
- How do they scale for large systems?

An understanding to how to differentiate between direct and iterative methods and when it is best to use each will also be developed.

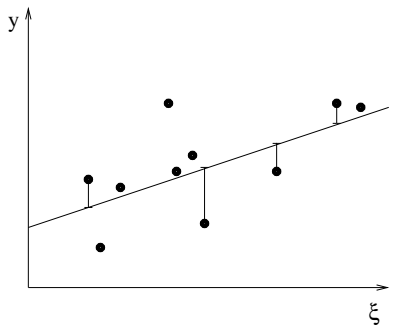
Least SQuares problems (LSQ)

Given $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ (typically $m \geq n$)

$$\text{find } x \in \mathbb{R}^n \text{ such that } \|Ax - b\|_2 \text{ is minimal.}$$

Example: Regression (linear). Given points $(\xi_i, y_i)_{i=1}^m$ find a linear function $\xi \mapsto x_1 + x_2\xi$ such that

$$g(x) := \Big(\sum_{i=1}^{m} \big(x_1 + x_2\xi_i - y_i\big)^2\Big)^{1/2} \quad \text{is minimal.}$$



In this case,

$$A = \begin{pmatrix} 1 & \xi_1 \\ \vdots & \vdots \\ 1 & \xi_m \end{pmatrix}, \quad b = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}.$$

EigenValue Problems (EVP)

Given $A \in \mathbb{C}^{n\times n}$

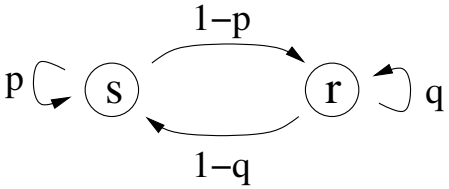
$$\text{find } (x, \lambda) \in (\mathbb{C}^n \backslash \{0\}) \times \mathbb{C} \text{ such that } Ax = \lambda x.$$

Sometimes, an eigenvector or an eigenvalue may be known, and the goal then is to compute the corresponding eigenvalue or, respectively, a corresponding eigenvector.

Example: We consider a highly simplified weather model. We distinguish only two states: $\underline{s}$un and $\underline{r}$ain. If one day's weather is s then the next day's weather is

- s with probability $p \in (0, 1)$,
- r with probability $1 - p$.

Similarly, there is a probability $q \in (0, 1)$ for no change of state r and $1 - q$ for a change from r to s.



Let us denote the actual state by a vector $\mu \in \mathbb{R}^2$ such that $(1, 0)$ corresponds to s (i.e., we associate state s with index 1) and $(0, 1)$ corresponds to r (associate r with index 2). Let us furthermore introduce the probability matrix $P = (p_{i,j})_{i,j=1}^2$ where $p_{i,j}$ denotes the probability that the system being in state i changes to state j, i.e.,

$$P = \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix}.$$

Multiplying the actual state with P from the right then yields a vector of probabilities for the next day's state, for instance if we start with a sunny day, $\mu^{(0)} := (1, 0)$, then

$$\mu^{(0)} P = (1, 0) \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix} = (p, 1-p) =: \mu^{(1)}$$

indeed contains the correct probabilities for s and r. The nice thing about this notation is that we simply may go on multiplying with P from the right to obtain the probabilities for the subsequent days, for instance

$$\mu^{(2)} := \mu^{(1)}P = (p^2 + (1-p)(1-q), p(1-q) + (1-p)q)$$

contains the probabilities for s and r on the second day, and

$$\mu^{(k)} := \mu^{(k-1)}P\Big(= \mu^{(0)}P^k\Big) \tag{1.7}$$

the probabilities for the k^{th} day.
Experts will recognise this iteration as a discrete Markov chain. In applications, one often is interested in stationary distributions π satisfying

$$\pi = \pi P, \quad \pi_i \geq 0 \, \forall i, \quad \sum_i \pi_i = 1$$

which means that π is a left eigenvector of P for the eigenvalue 1. In fact, 1 always is an eigenvalue of such probability matrices so the goal is just to compute a corresponding eigenvector. And of further interest is when Markov chains often converge to such stationary states.
We will see such type of iterations as the Markov chain (1.7) again in a slightly more general context of the so-called power iteration. Apart from the convergence, another questions immediately arising is how long one has to iterate in order to obtain an acceptable approximation to a stationary state (as this is an iterative method one will not expect to get an exact solution).

1.5 Learning Outcomes

- Gain expertise in modern-problem solving: learn various techniques for performing algebraic and analytic manipulations on matrices, understand their computational cost, memory usage, and use the most appropriate method for a particular application.
- Apply these methods to real world problems.
- Develop an understanding to linear system solving strategies including matrix factorisations, error analyses, computational costs, and differences in the analysis of direct versus iterative methods.

Lecture 2

Gaussian Elimination

Learning Outcomes

- Understand and apply the method of Gaussian Elimination to find the solution vector x for a given square matrix A and vector b in the equation $Ax = b$.
- Acquire skills in matrix transformation techniques to convert any given matrix into an upper triangular form, enabling simplified solutions to systems of equations.
- Learn and implement the Backward Substitution algorithm to solve for x once the matrix is in upper triangular form and gain knowledge of the analogous Forward Substitution method for solving systems with lower triangular matrices.

Methodology of Gaussian elimination

Problem: Given $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$ find $x \in \mathbb{C}^n$ such that $Ax = b$.
Observation: If $A = (a_{i,j})_{i,j=1}^n$ is (upper) triangular, i.e.,

$$A = \begin{pmatrix} a_{1,1} & \cdots & \cdots & a_{1,n} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n,n} \end{pmatrix}$$

then the x_i can be recursively computed as follows:

$$\begin{aligned}
&\text{last row :} && a_{n,n}x_n = b_n \Rightarrow x_n = b_n/a_{nn}, \\
&\text{row } n-1 : && a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1} \Rightarrow x_{n-1} = (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1}, \\
&\vdots && \vdots \\
&\text{first row:} && \ldots \Rightarrow x_1 = (b_1 - \sum_{j=2}^{n} a_{1,j}x_j)/a_{1,1}, \\
&\text{for any i:} && \ldots \Rightarrow x_i = b_i - \sum_{j=i+1}^{n} a_{ij}x_j/a_{ii}.
\end{aligned}$$

Algorithm 1 BS (backward substitution)

input: $U = (u_{i,j})_{i,j=1}^{n} \in \mathbb{C}^{n\times n}$ upper triangular, $u_{i,i} \neq 0$ for all $i = 1, \ldots, n$, $b = (b_i)_{i=1}^{n} \in \mathbb{C}^n$.
output: $x \in \mathbb{C}^n$ solution to $Ux = b$.

```
x_n := b_n / u_{n,n}
for i = n − 1 to 1 do
    h := 0
    for j = i + 1 to n do
        h := h + u_{i,j} x_j
    end for
    x_i := (b_i − h) / u_{i,i}
end for
```

Assume now that A is an arbitrary matrix.
The following operations do not change the solution space:

- multiplying an equation with a number $r \in \mathbb{C}\backslash\{0\}$,
- adding an equation to another equation.

Idea: Use such operations to transform A into a triangular matrix.

Example: Consider the system of equations written as a matrix equation,

$$\begin{array}{rcrcrcl} 2x_1 & + & x_2 & + & x_3 & = & 4 \\ 4x_1 & + & 3x_2 & + & 3x_3 & = & 10 \\ 8x_1 & + & 7x_2 & + & 9x_3 & = & 24 \end{array} \implies \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 4 \\ 10 \\ 24 \end{pmatrix}}_{b}.$$

$$[A|b] = \left(\begin{array}{ccc|c} 2 & 1 & 1 & 4 \\ 4 & 3 & 3 & 10 \\ 8 & 7 & 9 & 24 \end{array}\right) \begin{array}{l} \\ -2 * R_1 + R_2 \\ -4 * R_1 + R_3 \end{array} \implies \left(\begin{array}{ccc|c} 2 & 1 & 1 & 4 \\ 0 & 1 & 1 & 2 \\ 0 & 3 & 5 & 8 \end{array}\right) \tag{2.1}$$

Then,

$$[A|b] = \left(\begin{array}{ccc|c} 2 & 1 & 1 & 4 \\ 0 & 1 & 1 & 2 \\ 0 & 3 & 5 & 8 \end{array}\right) \begin{array}{l} \\ \\ -3 * R_2 + R_3 \end{array} \implies \left(\begin{array}{ccc|c} 2 & 1 & 1 & 4 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 2 & 2 \end{array}\right) \tag{2.2}$$

Now, use **BS** to compute the solution $x = (1, 1, 1)^T$, which is obtained as follows,

$$\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix} \rightsquigarrow$$

$$2x_3 = 2 \rightsquigarrow x_3 = 1$$

$$x_3 + x_2 = 2 \rightsquigarrow x_2 = 1$$

$$x_3 + x_2 + 2x_1 = 4 \rightsquigarrow x_1 = 1$$

This system can also be solved by factorising the matrix A as the product of a unit lower triangular matrix, L and an upper triangular regular (non-singular) matrix, U. This factorisation is called Lower upper (LU) factorisation, or LU decomposition. Here, $A = LU$.

Definition: A matrix $A = (a_{i,j}) \in \mathbb{C}^{n\times n}$ is unit lower triangular if

1. $a_{i,j} = 0$ if $1 \leq i < j \leq n$,
2. $a_{i,i} = 1$ for all $i = 1, \ldots, n$.

Substitute $A = LU$ into $Ax = b$ and then solve $LUx = b$ for x to solve the system. To achieve this, Let $Ux = y$, and $Ly = b$, and then perform the following two steps,

1. Solve $Ly = b$ for y
2. Solve $Ux = y$ for x

The above two steps need to be performed after finding the LU decomposition of the matrix A. The LU decomposition of the A matrix can be obtained by performing row operations on the A matrix to form an upper triangular matrix. The opposite sign of the multipliers can then be used to build the lower triangular matrix as will be seen shortly.
Considering the A matrix in the previous example,

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix} \begin{matrix} \\ -2 * R_1 + R_2 \\ -4 * R_1 + R_3 \end{matrix} \equiv \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -4 & 0 & 1 \end{pmatrix}}_{\tilde{L}_1} \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix}}_{A} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 5 \end{pmatrix} \begin{matrix} \\ \\ -3 * R_2 + R_3 \end{matrix} \equiv \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix}}_{\tilde{L}_2} \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -4 & 0 & 1 \end{pmatrix}}_{\tilde{L}_1} \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}}_{U}$$

It can be seen from the above equation that,

$$\tilde{L}_2\tilde{L}_1 A = U \implies (\tilde{L}_2\tilde{L}_1)^{-1}U = A \implies \tilde{L}_1^{-1}\tilde{L}_2^{-1}U = A \implies LU = A$$

Lemma 2.1. [5.1] *Let $L \in \mathbb{C}^{n\times n}$ unit lower triangular with non-zero entries only in column k. Then L^{-1} is unit lower triangular with non-zero entries only below the diagonal in column k with entries*

$$(L^{-1})_{i,k} = -L_{i,k}, \quad i = k+1, \ldots, n.$$

which is equivalent to taking the opposite sign of the multipliers to build the lower triangular matrix

Thus, the LU decomposition of A is,

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

To solve the SLE,

- By forward substitution (FS), solve $Ly = b$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 10 \\ 24 \end{pmatrix}$$

  This yields,

$$y_1 = 4$$
$$2y_1 + y_2 = 10 \implies y_2 = 2$$
$$4y_1 + 3y_2 + y_3 = 24 \implies y_3 = 2$$

- By backward substitution (BS), solve $Ux = y$

$$U = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix}$$

  This yields,

$$2x_3 = 2 \implies x_3 = 1$$
$$x_2 + x_3 = 2 \implies x_2 = 1$$
$$2x_1 + x_2 + x_3 = 4 \implies x_3 = 1$$

Exercise: formulate the (**FS**) in analogy to the (**BS**).