Homework 3

AM213A

Kevin Silberberg

2025-02-06

Part 1: Numerical Coding problems

Structure of Part 1

```
code/
├── include/
│   ├── gauss.h
│   ├── lu.h
│   ├── mat.h
│   └── utils.h
├── Makefile
└── src/
    ├── gauss.c
    ├── lu.c
    ├── main.c
    ├── mat.c
    └── utils.c
```

The code is structured such that inside the `code` directory, there are two subdirectories (`include` and `src` for source files), along with a `Makefile`. The `src` directory contains all `.c` source files, while the `include` directory holds `.h` header files. Running `make` builds the executable `main` inside the `code` directory, which also creates a `build` subdirectory to store compiled object files. Running `make clean` removes the `build` directory and its contents.

Usage

On the command line:

1. Run make to compile the code.
2. Execute one of the following commands:
    - ./main gauss Amat.dat Bmat.dat
    - ./main lu Amat.dat Bmat.dat

Main code driver

The main.c file contains the entry point for the code, it contains the logic for commandline inputs for

Matrix Utility Functions

The `matrix_t` data structure, defined in `mat.c` and `mat.h`, provides core matrix utilities such as:

- Creation and manipulation functions: printMatrix, copyMatrix, createMatrix, etc.

- Matrix operations: multiplication, subtraction, transposition, row swapping, and more.

Miscellaneous Utilities

The files `utils.c` and `utils.h` include functions for:

1. Calculating the error matrix:
$$\mathbf{E} = \mathbf{A}_s\mathbf{X} - \mathbf{B}_s \tag{1}$$
2. Printing the error matrix in scientific notation.
3. Computing the L2 norm of each column of the error matrix and outputting it to stdout.

Gaussian Elimination with Partial Pivoting

`gaussElim`

This function, defined in `gauss.c` and `gauss.h`, accepts two arguments of type `matrix_t` (`A` and `B`) and solves the linear system:
$$\mathbf{AX} = \mathbf{B}, \tag{2}$$
where $\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{B}, \mathbf{X} \in \mathbb{R}^{m\times n}$.

The function returns an upper triangular matrix $\mathbf{U}$, which is stored in the memory allocated for the matrix $\mathbf{A}$. Additionally, the matrix $\mathbf{B}$ undergoes the same row transformations as $\mathbf{A}$.

`backSubGauss`

This function takes an upper triangular matrix $\mathbf{U} \in \mathbb{R}^{m\times m}$ and a transformed matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$ as inputs. It returns a solution matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, which is stored in the memory allocated for the $\mathbf{B}$ matrix.

It performs back-substitution on the system:
$$\mathbf{UX} = \mathbf{B} \tag{3}$$

`mainGauss`

This is the driver code for the Gaussian elimination algorithm. It takes two string arguments, `filename1` and `filename2`, corresponding to the files containing matrices $\mathbf{A}$ and $\mathbf{B}$, respectively. The function performs the following steps:

1. Reads matrices $\mathbf{A}$ and $\mathbf{B}$ from the input files.
2. Stores a copy of the matrices for later error calculation.
3. Prints the matrices before applying Gaussian elimination.
4. Performs Gaussian elimination on $\mathbf{A}$ and $\mathbf{B}$.
5. Prints the matrices after Gaussian elimination.
6. Performs back-substitution on the system $\mathbf{UX} = \mathbf{B}$.
7. Prints the solution matrix $\mathbf{X}$.
8. Calculates the error using (1).
9. Prints the L2 norm of each column of the error matrix $\mathbf{E}$.
10. Deallocates memory for all allocated matrices.

LU Decomposition with Partial Pivoting

`lu`

This function performs LU decomposition with partial pivoting on a given matrix $\mathbf{A}$ and saves the row swaps in a permutation vector $\mathbf{s}$. It takes two arguments of type `matrix_t`:

1. The first argument, $\mathbf{A} \in \mathbb{R}^{m \times m}$, represents the matrix in the system $\mathbf{AX} = \mathbf{B}$, which will later be solved using forward and backward substitution.
2. The second argument, $\mathbf{s}$, is a column vector of size $m \times 1$ stored under the matrix_t struct type. It acts as a permutation vector that records the row swaps applied to $\mathbf{A}$ during LU decomposition.

The function stores the results as follows:

- The upper triangular matrix $\mathbf{U}$ is saved in the diagonal ($i = j$) and upper diagonal ($i < j$) entries of $\mathbf{A}$.
- The lower triangular matrix $\mathbf{L}$ is saved in the lower diagonal ($i > j$) entries of $\mathbf{A}$.

`backSubLU`

The `backSubLU` function performs LU back-substitution on a given matrix $\mathbf{A}$ and matrix $\mathbf{B}$. Here, $\mathbf{A}$ contains:

- The upper triangular matrix $\mathbf{L}$ in its lower diagonal entries ($i > j$).
- The lower triangular matrix $\mathbf{U}$ in its diagonal and upper diagonal entries ($i \leq j$).

The function also takes the permutation vector $\mathbf{s}$, which records the row swaps performed on $\mathbf{A}$ during LU decomposition.

Steps:

1. Permute $\mathbf{B}$
   - Initialize a new matrix $\mathbf{Y}$.
   - Copy the entries of $\mathbf{B}$ into $\mathbf{Y}$ according to the permutation vector $\mathbf{s}$ (using the unsigned integer values stored in $\mathbf{s}$ as indices).
   - Overwrite $\mathbf{B}$ with $\mathbf{Y}$ and deallocate $\mathbf{Y}$.
2. Solve the system
   - Perform forward substitution to solve $\mathbf{LY} = \mathbf{B}$.
   - Perform backward substitution to solve $\mathbf{UX} = \mathbf{Y}$.
   - Store the solution matrix $\mathbf{X}$ in $\mathbf{B}$.

`mainLU`

This is the driver code for the LU factorization algorithm. It takes two string arguments, `filename1` and `filename2`, corresponding to the files containing matrices $\mathbf{A}$ and $\mathbf{B}$, respectively. The function performs the following steps:

1. Reads matrices $\mathbf{A}$ and $\mathbf{B}$ from the input files.
2. Stores a copy of the matrices for later error calculation.
3. Prints the matrices before LU decomposition.
4. Performs LU decomposition on $\mathbf{A}$ with permutation matrix $\mathbf{s}$.
5. Prints the matrix $\mathbf{A}$ after LU.
6. Prints the matrix $\mathbf{L}$.
7. Prints the matrix $\mathbf{U}$.
8. Performs back-substitution on $\mathbf{B}$ with $\mathbf{LU}$ and permutation matrix $\mathbf{s}$.
9. Prints the solution matrix $\mathbf{X}$.
10. Calculates the error using (1).
11. Prints the L2 norm of each column of the error matrix $\mathbf{E}$.
12. Deallocates memory for all allocated matrices.

A very basic application

Consider the following points in $\mathbb{R}^3$,

$$A(1,2,3) \tag{4}$$
$$B(-3,2,5) \tag{5}$$
$$C(\pi, e, -\sqrt{2}) \tag{6}$$

Let

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \tag{7}$$
$$\mathbf{B} = \begin{bmatrix} -3 & 2 & 5 \end{bmatrix} \tag{8}$$
$$\mathbf{C} = \begin{bmatrix} \pi & e & -\sqrt{2} \end{bmatrix} \tag{9}$$

We want to find the equation for a plane that spans the space represented by the vectors $\vec{AB} = \mathbf{A} - \mathbf{B}$ and $\vec{AC} = \mathbf{A} - \mathbf{C}$. To do this numerically we will construct a system of equations in the form of $\mathbf{Ax} = \mathbf{b}$.

Recall that the scalar equation for a plane is as follows:

$$ax + by + cz = d \tag{10}$$

if we divide both sides by d we have,

$$\frac{a}{d}x + \frac{b}{d}y + \frac{c}{d}z = 1 \tag{11}$$

Let

$$\frac{a}{d} = \tilde{a} \quad \frac{b}{d} = \tilde{b} \quad \frac{c}{d} = \tilde{c} \tag{12}$$

such that,

$$\tilde{a}x + \tilde{b}y + \tilde{c}z = 1 \tag{13}$$

this holds as long as $d \neq 0$.

Let

$$\mathbf{x} = \begin{bmatrix} 1 \\ -3 \\ \pi \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 2 \\ 2 \\ e \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} 3 \\ 5 \\ -\sqrt{2} \end{bmatrix} \tag{14}$$

by the equation (13) we can construct a system of equations in the following way,

$$\tilde{a}\mathbf{x} + \tilde{b}\mathbf{y} + \tilde{c}\mathbf{z} = \mathbf{1} \tag{15}$$

$$\tilde{a}\begin{bmatrix}1\\-3\\\pi\end{bmatrix} + \tilde{b}\begin{bmatrix}2\\2\\e\end{bmatrix} + \tilde{c}\begin{bmatrix}3\\5\\-\sqrt{2}\end{bmatrix} = \begin{bmatrix}1\\1\\1\end{bmatrix} \tag{16}$$

$$\begin{bmatrix}1 & 2 & 3\\-3 & 2 & 5\\\pi & e & -\sqrt{2}\end{bmatrix}\begin{bmatrix}\tilde{a}\\\tilde{b}\\\tilde{c}\end{bmatrix} = \begin{bmatrix}1\\1\\1\end{bmatrix} \tag{17}$$

Once the equation is solved we have a solution vector $\tilde{\mathbf{x}} = \begin{bmatrix}\tilde{a}\\\tilde{b}\\\tilde{c}\end{bmatrix}$. We can plot the plane by solving for the z component of the equation (13).

$$\tilde{a}x + \tilde{b}y + \tilde{c}z = 1 \tag{18}$$

$$z = \frac{1 - \tilde{a}x - \tilde{b}y}{\tilde{c}} \tag{19}$$

The following julia code defines a matrix $\mathbf{A}$ containing the columns specified in (14) and a vector $\mathbf{b}$ of size 3 with every entry set to one. It then performs lu depcomposition using the LinearAlgebra library and generates the solution vector $\mathbf{x}$. It then defines a mesh grid and computes the height in the z-axis by (18).

```julia
using GLMakie
using LinearAlgebra

function main()
    # define A matrix
    A = [
        1 2 3;
        -3 2 5;
        π exp(1) -√2
    ]

    # define b vector with every entry set to one
    b = ones(3)

    # perform lu decomposition on A
    F = lu(A)

    # perform back substitution to obtain solution vector x
    x = F \ b

    # decompose solution vector into a, b, c
    a, b, c = x

    # define a mesh grid
    xs = LinRange(-6.0, 6.0, 1000)
    ys = LinRange(-2.0, 6.0, 1000)

    # function to solve for the z component of every mesh point
    zf(x, y) = (1.0 - a*x - b*y)/c
```

```
    zs = [zf(k, l) for k in xs, l in ys]

    fig = Figure()
    ax = Axis3(fig[1, 1],
        title = "Points A, B, C in R3",
        xlabel = "x",
        ylabel = "y",
        zlabel = "z"
    )

    xa = A[:, 1]
    ya = A[:, 2]
    za = A[:, 3]
    colors = [:red, :blue, :green]
    labels = ["A", "B", "C"]
    for i in 1:3
    scatter!(ax, xa[i], ya[i], za[i], color = colors[i], label = labels[i])
    end

    surface!(ax, xs, ys, zs, color = :purple, alpha = 0.3)

    axislegend()

    fig
end
main()
```

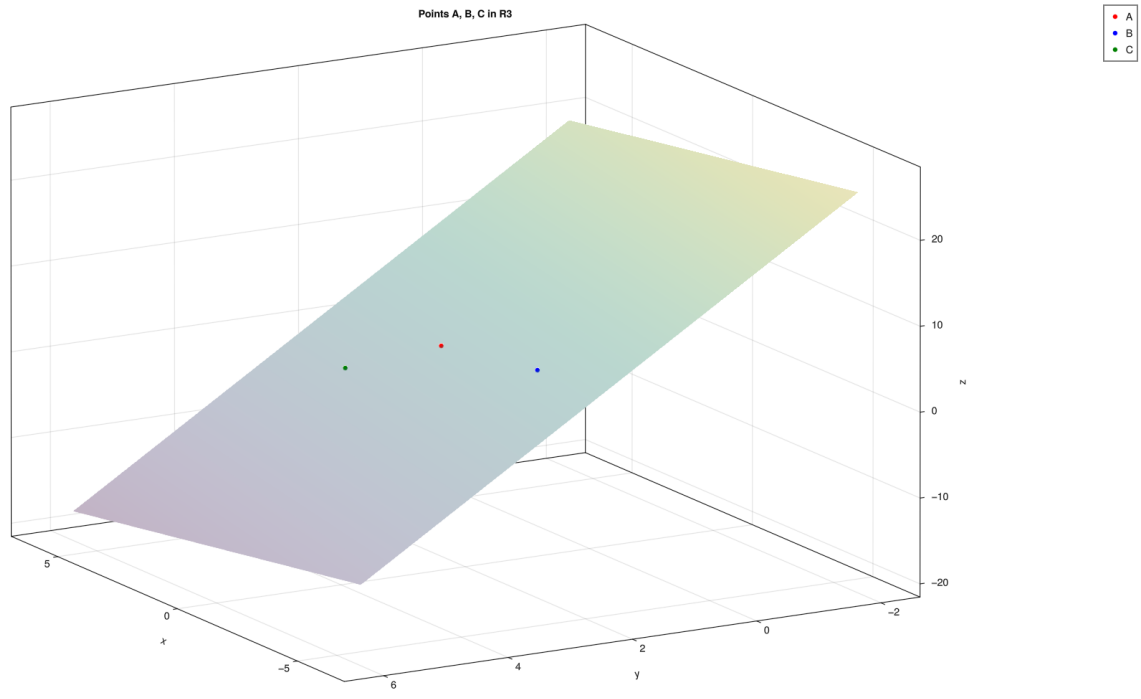

Figure 1: Plot of the plane that is spanned by the points A, B, and C

Part 2: Theory Problems

Question 1

Consider the following system,

$$\begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}. \tag{20}$$

Multiply the last row of the matrix and the right-hand side vector by a large constant c such that $c\epsilon \gg 1$. Perform Gaussian elimination with partial pivoting to the modified row-scaled system and discuss what happens. If solving the resulting system has numerical issues, identify the issues and discus how to improve the method.

Solution

Consider the true solution to the $\mathbf{Ax} = \mathbf{b}$ system in (20).

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{21}$$

$$= \frac{1}{1-\epsilon} \begin{bmatrix} 1 & -1 \\ -\epsilon & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \tag{22}$$

$$= \frac{1}{1-\epsilon} \begin{bmatrix} 1 \\ 1-2\epsilon \end{bmatrix} \tag{23}$$

$$= \begin{bmatrix} \frac{1}{1-\epsilon} \\ \frac{1-2\epsilon}{1-\epsilon} \end{bmatrix} \approx \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{24}$$

Now let us multiply the bottom row by some large number c such that $c\epsilon \gg 1$. Let $c = 10^{20}$.

Let us consider the numerical case when $\epsilon = \epsilon_{\text{mach}} = 10^{-16}$. (20) becomes,

$$\begin{bmatrix} 1 & 1 \\ 10^4 & 10^{20} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 10^{20} \end{bmatrix} \tag{25}$$

Let us proceed now with Gaussian Elimination with partial pivoting. We first swap the two rows such that $R_1 \leftrightarrow R_2$.

$$\begin{bmatrix} 10^4 & 10^{20} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10^{20} \\ 2 \end{bmatrix} \rightarrow R_2 - \frac{R_1}{10^4} \tag{26}$$

$$\rightarrow \begin{bmatrix} 10^4 & 10^{20} \\ 0 & 1-10^{16} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10^{20} \\ 2-10^{16} \end{bmatrix} \tag{27}$$

Numerically, the value $1 - 10^{16} \approx -10^{16}$ and the value $2 - 10^{16} \approx -10^{16}$ such that the system becomes,

$$\begin{bmatrix} 10^4 & 10^{20} \\ 0 & -10^{16} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10^{20} \\ -10^{16} \end{bmatrix} \tag{28}$$

By performing back substitution we have,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{29}$$

To fix this we propose using the implicit pivoting algorithm, where each row of the augmented matrix is first scaled by its largest entry in absolute value.

Question 2

Suppose $\mathbf{A} \in \mathbb{C}^{m \times m}$ is written in the block form

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \tag{30}$$

where $\mathbf{A}_{11} \in \mathbb{C}^{n \times n}$ and $\mathbf{A}_{22} \in \mathbb{C}^{(m-n)\times(m-n)}$. Assume that $\mathbf{A}$ satisfies the condition: $\mathbf{A}$ has an **LU** decomposition if and only if the upper-left $k \times k$ block matrix $\mathbf{A}_{1:k,1:k}$ is non-singular for each k with $1 \leq k \leq m$.

Part a.

Verify the formula

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{0} & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \end{bmatrix}, \tag{31}$$

which eliminate the block $\mathbf{A}_{21}$ from $\mathbf{A}$. The matrix $\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$ is known as the Schur complement of $\mathbf{A}_{11}$ in $\mathbf{A}$, denoted as $\frac{\mathbf{A}}{\mathbf{A}_{11}}$.

Solution

Let us matrix multiply the following matrices to verify (31).

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ -\mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{11} + \mathbf{A}_{21} & -\mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} + \mathbf{A}_{22} \end{bmatrix} \tag{32}$$

Thus in the bottom left block matrix of $\mathbf{A}$ we have that,

$$\mathbf{A}_{21} - \mathbf{A}_{21}\mathbf{I} = \mathbf{0} \tag{33}$$

Thus (31) is identical to (32).

Part b

Suppose that after applying n steps of Gaussian elimination on the matrix $\mathbf{A}$ in (30), $\mathbf{A}_{21}$ is eliminated row by row, resulting in a matrix

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{C} \\ \mathbf{0} & \mathbf{D} \end{bmatrix}. \tag{34}$$

Show that the bottom-right $(m-n) \times (m-n)$ block matrix $\mathbf{D}$ is again $\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$.

Solution

By following the Basic Gaussian elimination algorithm on page 32 of the Numerical Linear Algebra text book, for the matrix (30), we compute the first step in the algorithm, $r_2 = r_2 - \frac{a_{21}}{a_{11}} r_1$. Which translates to the row operation, subtract each element in row two with their corresponding element in row one times $\frac{\mathbf{A}_{21}}{\mathbf{A}_{11}} = \mathbf{A}_{21}\mathbf{A}_{11}^{-1}$.

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{11} & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \end{bmatrix} \tag{35}$$

Which simplifies to

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{0} & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \end{bmatrix} \tag{36}$$

Thus by performing Gaussian Elimination on the block matrices of $\mathbf{A}$, we have shown that indeed the matrix $\mathbf{D}$ is again $\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$.

Question 3

Consider solving $\mathbf{Ax} = \mathbf{b}$, with $\mathbf{A}$ and $\mathbf{b}$ are complex-valued of order m, i.e., $\mathbf{A} \in \mathbb{C}^{m \times m}$ and $\mathbf{b} \in \mathbb{C}^m$.

Part a

Modify this problem to a problem where you only solve a real square system of order $2m$.

Solution

Let the

$$\mathbf{Ax} = \mathbf{b} \tag{37}$$

system be decomposed as,

$$\mathbf{A} = \mathbf{A}_1 + i\mathbf{A}_2 \tag{38}$$
$$\mathbf{x} = \mathbf{x}_1 + i\mathbf{x}_2 \tag{39}$$
$$\mathbf{b} = \mathbf{b}_1 + i\mathbf{b}_2 \tag{40}$$

such that,

$$\mathbf{A}_1 = \mathrm{Re}(\mathbf{A}) \tag{41}$$
$$\mathbf{A}_2 = \mathrm{Im}(\mathbf{A}) \tag{42}$$
$$\mathbf{x}_1 = \mathrm{Re}(\mathbf{x}) \tag{43}$$
$$\mathbf{x}_2 = \mathrm{Im}(\mathbf{x}) \tag{44}$$
$$\mathbf{b}_1 = \mathrm{Re}(\mathbf{b}) \tag{45}$$
$$\mathbf{b}_2 = \mathrm{Im}(\mathbf{b}) \tag{46}$$

Let us substitute the equations into (37)

$$(\mathbf{A}_1 + i\mathbf{A}_2)(\mathbf{x}_1 + i\mathbf{x}_2) = (\mathbf{b}_1 + i\mathbf{b}_2) \tag{47}$$
$$\mathbf{A}_1\mathbf{x}_1 + i\mathbf{A}_1\mathbf{x}_2 + i\mathbf{A}_2\mathbf{x}_1 + i^2\mathbf{A}_2\mathbf{x}_2 = \mathbf{b}_1 + i\mathbf{b}_2 \tag{48}$$
$$\mathbf{A}_1\mathbf{x}_1 + i\mathbf{A}_1\mathbf{x}_2 + i\mathbf{A}_2\mathbf{x}_1 - \mathbf{A}_2\mathbf{x}_2 = \mathbf{b}_1 + i\mathbf{b}_2 \tag{49}$$
$$\tag{50}$$

Let us collect terms for i and write the inequality as a system of equations

$$\mathbf{A}_1\mathbf{x}_1 - \mathbf{A}_2\mathbf{x}_2 = \mathbf{b}_1 \tag{51}$$
$$\mathbf{A}_2\mathbf{x}_1 + \mathbf{A}_1\mathbf{x}_2 = \mathbf{b}_2 \tag{52}$$

Let us write this in block matrix form

$$\begin{bmatrix} \mathbf{A}_1 & -\mathbf{A}_2 \\ \mathbf{A}_2 & \mathbf{A}_1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \iff \tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \tag{53}$$

the above expression (53) is a block matrix where each block matrix in $\tilde{\mathbf{A}}$ is of size $m \times m$, each block vector in $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{b}}$ are of size m.

This is a representation of the original complex system (37) of order m as a real system of order $2m$.

Part b

Determine the storage requirement and the number of floating point operations for the real-valued method in Part a of solving the original complex-valued system $\mathbf{Ax} = \mathbf{b}$. Compare these results with those based on directly solving the complex-valued system using Gaussian Elimination (GE) (without pivoting) and complex arithmetic. Use the fact that the operation count of GE is $\mathcal{O}(\frac{m^3}{3})$ for an $m \times m$ real valued system with one right hand side vector. Pay close attention to the greater expense of complex arithmetic operations. Make your conclusion by quantifying the storage requirement and the operating expense of each method. Draw your conlcusion on which method is computationally advantageous.

Solution

Given the fact that the operation count of Gaussian Elimination for a system of order m is $\mathcal{O}(\frac{m^3}{3})$, for a system of order $2m$ such as the one in part a, we have:

$$\frac{(2m)^3}{3} = \frac{2^3 m^3}{3} = \frac{8}{3}m^3 \tag{54}$$

In a square real-valued system of order m, the space required to store each real-valued floating point number for double precision occupies exactly 8 bytes or 64 bits. This means if we have a matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$ and a corresponding vector $\mathbf{b} \in \mathbb{R}^m$ then the number of bytes required to store every floating point number is $8(m^2 + m)$ bytes.

For a complex-valued matrix of the same order m, each floating point number requires twice the amount of space because both the real part and the imaginary parts of a complex number are themselves double floating point numbers, so the storage requirement is 16 bytes per number. Thus the storage requirement for a complex valued system of order m is $16(m^2 + m)$ bytes.

For system of order $2m$ of real-valued entries discussed in part a the storage requirement is thus

$$8\left((2m)^2 + 2m\right) = 8\left(4m^2 + 2m\right) \tag{55}$$
$$= 32\left(m^2 + \frac{m}{2}\right) \tag{56}$$

Next, let us consider the number of operations in the case for a matrix $\mathbf{A} \in \mathbb{C}^{m\times m}$. The number of computations for GE with real-valued entries for a matrix of size m is given as $\mathcal{O}(\frac{m^3}{3})$ and each multiplication

for complex-values requires foiling, which is four times as many multiplications. Thus the computational cost of perform GE on a matrix with complex values instead of real values is:

$$\mathcal{O}(\frac{4m^3}{3}) \tag{57}$$

This leaves us with the following comparison, organized in the table below:

Order	m	$2m$
Storage	$16(m^2+m)$	$32(m^2+\frac{m}{2})$
Operations	$\mathcal{O}(\frac{4m^3}{3})$	$\mathcal{O}(\frac{8m^3}{3})$

As you can see the method for decomposing the system $\mathbf{Ax} = \mathbf{b}$ into the system $\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ as shown in part a, requires roughly twice the storage and is twice the computational cost of just performing gaussian elimination on a complex valued matrix of size m.