**AM 213A, Winter 2022**
**Homework 2 (100 points)**

**Posted on Tue, Feb 1st, 2022**
**Due 11:59 pm, Thu, Feb 10th, 2022**

**Submit your homework to your Git repository by 11:59 pm**

---

## 1. General Guidelines

### 1.1. Two parts

You have two parts in this homework set:

- **Part 1 (50 pts)**: Numerical coding problems (Make sure to use double precision!!!)
- **Part 2 (50 pts)**: Theory problems

Your final set of answers should consist of

- Your report needs to have relevant discussions on each problem to describe what you demonstrate. For all coding problems, showing the screen output from your code execution is considered to be not sufficient.
- For Part 1: a pdf file with your written answers. This pdf *must* be created using a word processor.
- For Part 2: solutions to the theory problems in a pdf file (a cleanly handwritten scanned pdf is also allowed).
- For both Part 1 and Part 2: Please submit all your answers to your git. Copy the first 10 digits of your git hash that corresponds to your final Git HW2 submission and paste them into a text file (.txt file). Submit the git hash file to Canvas. Use subdirectories named as **Part1** and **Part2** under your top directory called **HW2**.

Please organize and present the material in the best possible way, in particular, for your Part 1 solutions:

- For any written material, be informative but concise. Mathematical derivations, if appropriate, should contain enough pertinent steps to show the reader how you proceeded. All included figures should be clearly annotated and have an informative caption. Each problem should be addressed in its own Section (e.g., "Section 1: Problem 1", "Section 2: Problem 2", etc.). References should be included if you are using any material as part of your discussions and/or calculations.

- For the codes: All codes must be written in Fortran 90 (or more recent Fortran, but not Fortran 77) or C, but *not* in any other language. MATLAB or Python is to be used to (i) generate figures, and/or (ii) perform simple analysis (e.g., calculating errors) the data produced by your Fortran/C codes. Put all the codes relevant for each problem in separate directories called Prob1, Prob2, etc, as needed. Annotate your codes carefully so the reader knows what each part of the code does. For each routine and function in your Fortran/C, include a header (i.e., comment sections at the beginning explanations) explaining the inputs and outputs of the code, and what the routine does. In the main program in each directory, include a header that contains (i) the command required to compile the code, and (ii) the structure of the inputs and outputs of the main program.

- Template codes are provided in Fortran 90 only, which means you need to translate the template Fortran codes to C if your language choice is C. Be careful about the column-wise convention of arrays in Fortran vs. the row-wise convention in C.

### 1.2. Template Implementations (provided) for Part 1

Download the tarball `hw2_template_code.tar` which includes `LinAl.f90` module. The module contains the routine `readMat`. Carefully read the module and the others included in the tar file. Also check out the data file `Amat.dat` which contains the matrix $\mathbf{A}$ from equation (2.18) in the lecture note. In `Amat.dat` the two numbers in the first line represent the dimension of the matrix $\mathbf{A}$.

## 2. Warming-up: Basic Fortran routines and functions for Part 1 (10 pts)

In scientific computing, it is always good to practice writing your codes so that they are well equipped with some debugging capabilities. To meet this, add a few more useful routines to the template module `LinAl.f90` and test them. They will be used later throughout the quarter.

For all function (and/or subroutine) implementations within the module:

- Write a function (and/or subroutine) that takes three arguments including two inputs and one output. The first is an $m \times m$ square matrix $\mathbf{A}$, the second is the first dimension of $\mathbf{A}$ (i.e., $m$), and finally the last is the trace of $\mathbf{A}$ as a return value.

- Write a function (and/or subroutine) that takes three arguments. Two input arguments are a vector and its dimension, and one output argument is its Euclidean norm (i.e., 2-norm).

- Write a function (and/or subroutine) that takes two input arguments of (i) an $m \times n$ matrix $\mathbf{A}$, and (ii) both of its dimensions. This routine then prints the matrix and its dimensions (i.e., $m$ and $n$) to the screen in a human-readable form (i.e., screen output).

Now write a driver (or calling) program that takes a filename as an argument, reads the matrix $\mathbf{A}$ from the `Amat.dat` file, and performs the following tests:

- Prints the matrix to the screen in a human-readable form (i.e., screen output).
- Calculates its trace by calling your function (or subroutine) implementation(s), and prints it to the screen.
- Calculates the norm of each column vector of $\mathbf{A}$, and prints them to the screen.

You need to make sure your implementations are correct by evaluating analytically the requested trace and norms and comparing the analytical and the numerical results from your test program. Debug if necessary until they match. Briefly discuss your answer (mathematical derivation of the analytical results, and comparison with the numerical answer) in the pdf for Part 2.

### 3. Gaussian Elimination with partial pivoting (15 pts)

In your `LinAl.f90` module, create a routine to perform a Gaussian elimination with partial pivoting on $\mathbf{A}$ with corresponding operations on a matrix $\mathbf{B}$ containing $n$ rhs vectors. The arguments of the routine are

- a square matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$ (e.g., use `Amat.dat`)
- a non-square matrix $\mathbf{B}$ which contains $n$ rhs vectors, i.e., $\mathbf{B} = [\mathbf{b}_1 \sqcup \mathbf{b}_2 \sqcup \cdots \sqcup \mathbf{b}_n]$, $\mathbf{b}_i \in \mathbb{R}^m$. (e.g., use `Bmat.dat`)
- their respective dimensions (the first line in `Amat.dat` and `Bmat.dat`)
- a logical flag (i.e. `.TRUE.` or `.FALSE.`) that indicates whether the problem is singular or not

On entry, $\mathbf{A}$ and $\mathbf{B}$ are the input matrices. On exit, $\mathbf{A}$ should be upper triangular, and $\mathbf{B}$ contains the modified set of rhs vectors. The logical is set to be `.FALSE.` if the problem is not singular, and `.TRUE.` if it is.

Then create a second routine that takes as input an upper-diagonal matrix $\mathbf{U}$ (i.e., for instance from the output of the previous routine), and a rhs matrix $\mathbf{B}$ (i.e., again, for instance, from the output of the previous routine), and performs a backsubstitution to return the solution to the problem $\mathbf{AX} = \mathbf{B}$, where $\mathbf{X}$ is a matrix containing the $n$ solution vectors. You may chose to over-write the vector $\mathbf{B}$ on exit with $\mathbf{X}$, or return the solution as a separate vector. Note that the argument list of this back-substitution routine must reflect your choice. Test your routines on the equation $\mathbf{AX} = \mathbf{B}$. To do so create a program that

- reads and allocates the matrices in `Amat.dat` and `Bmat.dat`
- copies $\mathbf{A}$ and $\mathbf{B}$ into saved arrays $\mathbf{A}_s$ and $\mathbf{B}_s$
- prints the matrices $\mathbf{A}$ and $\mathbf{B}$ *before* application of the Gaussian elimination

- performs Gaussian Elimination
- prints the matrices $\mathbf{A}$ and $\mathbf{B}$ *after* application of the Gaussian elimination
- performs a backsubstitution
- prints the matrix $\mathbf{X}$ (the solution vector) after backsubstitution, as well as the error matrix $\mathbf{E} = \mathbf{A}_s\mathbf{X} - \mathbf{B}_s$
- calculates the norm of each of the column vectors of the error matrix $\mathbf{E}$ and prints each of them to the screen

If the maximum value of these norms is not within machine accuracy (or close to) for your selected precision, debug your code until it is.

## 4. LU decomposition with partial pivoting (15 pts)

In your `LinAl.f90` module, create an LU decomposition routine with partial pivoting that takes as argument

- a square matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$ (e.g., either use `Amat.dat` or your own this time)
- its first dimension (i.e, $m$)
- a logical flag (i.e. `.TRUE.` or `.FALSE.`) that indicates whether the problem is singular or not
- a permutation vector $\mathbf{s}$

On exit, $\mathbf{A}$ is replaced by its LU decomposition (with $\mathbf{U}$ and $\mathbf{L}$ stored as in the lecture note – see Eq. (2.55)). The logical is set to be `.FALSE.` if the problem is not singular, and `.TRUE.` if it is, and $\mathbf{s}$ contains the permutation vector.

Then create a LU backsubstitution routine that solves $\mathbf{Ax} = \mathbf{b}$, and takes the followings as argument:

- the matrix $\mathbf{A}$ already decomposed into LU
- its first dimension
- a vector $\mathbf{b}$ (Note: you can generalize this and input a matrix $\mathbf{B}$ containing $n$ rhs vectors if you prefer, i.e., you can either use the provided `Bmat.dat` or create a new one)
- the permutation vector $\mathbf{s}$

You may chose to over-write the vector $\mathbf{b}$ on exit with the solution $\mathbf{x}$, or return the solution as a separate vector (in which case it has to be added to the argument list).

Test your routines as in the previous section. To do so create a program that

- reads and allocates the matrices in `Amat.dat` and `Bmat.dat` (either by using the provided ones or creating your own data files)
- copies $\mathbf{A}$ and $\mathbf{B}$ into saved arrays $\mathbf{A}_s$ and $\mathbf{B}_s$
- prints the matrix $\mathbf{A}$ before application of the LU decomposition
- performs LU decomposition
- prints the matrix $\mathbf{A}$, $\mathbf{L}$, and $\mathbf{U}$, after LU decomposition
- for each of the column vectors of $\mathbf{b}_i$ of $\mathbf{B}$ (either one by one, or all at the same time):
  - performs a backsubstitution
  - prints the solution $\mathbf{x}_i$ after backsubstitution, as well as the error vector $\mathbf{A}_s\mathbf{x}_i - \mathbf{b}_{i,s}$.
  - calculates the norm of $\mathbf{A}_s\mathbf{x}_i - \mathbf{b}_{i,s}$ and prints it to the screen

If the maximum value of these norms is not within machine accuracy (or close to) for your selected precision, debug your code until it is.

## 5. A very basic application (10 pts)

Consider the following three points in 3D space, with corresponding $(x, y, z)$ coordinates: $A(1, 2, 3)$, $B(-3, 2, 5)$ and $C(\pi, e, -\sqrt{2})$. Explain what equations you need to solve to find the equation of the plane passing through all three points, and do so numerically, using either the Gaussian elimination routine, or the LU decomposition routine (as you prefer). Use Python or Matlab to produce a 3D figure with a graph of this plane, as well as the three points clearly marked.

**Part 2: Theory Problems (10 pts each)**

**1.** The Schur decomposition theorem states that if $A \in \mathbb{C}^{m\times m}$, then there exist a unitary matrix $Q$ and an upper triangular matrix $U$ such that $A = QUQ^{-1}$. Use the Schur decomposition theorem to show that a real symmetric matrix $A$ is diagonalizable by an orthogonal matrix, i.e., $\exists$ an orthogonal matrix $Q$ such that $Q^TAQ = D$, where $D$ is a diagonal matrix with its eigenvalues in the diagonal.

**2.** Consider the following system

$$\begin{pmatrix} 1 & 1 \\ \varepsilon & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}. \tag{1}$$

Multiply the last row of the matrix and the right-hand side vector by a large constant $c$ such that $c\varepsilon \gg 1$. 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 discuss how to improve the method.

**3.** What can you say about the diagonal entries of a symmetric positive definite matrix? Justify your assertion.

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

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \tag{2}$$

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

(a) Verify the formula

$$\begin{pmatrix} I & \mathbf{0} \\ -A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ \mathbf{0} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}, \tag{3}$$

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

(b) Suppose that after applying $n$ steps of Gaussian elimination on the matrix $A$ in (2), $A_{21}$ is eliminated row by row, resulting in a matrix

$$\begin{pmatrix} A_{11} & C \\ \mathbf{0} & D \end{pmatrix}. \tag{4}$$

Show that the bottom-right $(m-n)\times(m-n)$ block matrix $D$ is again $A_{22} - A_{21}A_{11}^{-1}A_{12}$. (Note: Part (b) is separate from Part (a)).

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

(a) Modify this problem to a problem where you only solve a *real* square system of order $2n$. (Hint: Decompose $A = A_1 + iA_2$, where $A_1 = \mathrm{Re}(A)$ and $A_2 = \mathrm{Im}(A)$, and similarly for $b$ and $x$. Determine equations to be satisfied by $x_1 = \mathrm{Re}(x)$ and $x_2 = \mathrm{Im}(x)$.

(b) Determine the storage requirement and the number of floating-point operations for the real-valued method in (a) of solving the original complex-valued system $Ax = b$. Compare these results with those based on directly solving the original complex-valued system using Gaussian elimination (without pivoting) and complex arithmetic. Use the fact that the operation count of Gaussian elimination is $\mathcal{O}(m^3/3)$ 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 conclusion on which method is computationally advantageous.