AMS 209, Fall 2016
Final Project Type A – Numerical Linear Algebra:
Gaussian Elimination with Pivoting for Solving Linear Systems

1. Project Description

There are two parts in the final term project:

- Fortran implementation of a linear algebra solver in a Fortran directory, `"project/LinAlg"`,
- Python implementation of a run setup, a run scheduler, and a data visualizer, in a Python directory, `"project/PyRun"`.

1.1. Fortran Implementation

In the Fortran part of the project, you are going to implement:

- Gaussian elimination *with and without* partial pivoting (two options!),

in order to solve $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is an $n \times n$ square matrix and $\mathbf{b}$ is an n-vector. Your code is to take $\mathbf{A}$ and $\mathbf{b}$ as inputs by reading in them from ascii input files, for instance:

- `A_i.dat` for matrix inputs, where for each `i = 1,2, ..., n` is associated with each different matrix, and
- `b_i.dat` for right hand side vector inputs, similarly for each `i`.

Right after reading those from files, the code writes $\mathbf{A}$ and $\mathbf{b}$ onto screen so that users can check if the inputs are correct. When outputting to screen, the code needs to write $\mathbf{A}$ and $\mathbf{b}$ in a human readable format so that users can easily read the entries of $\mathbf{A}$ and $\mathbf{b}$ in a square matrix form and a vector form, respectively.

Also when reading in $\mathbf{A}$ and $\mathbf{b}$ from files, the code should not assume anything on the sizes of the matrix $\mathbf{A}$ and $\mathbf{b}$. In other words, this means that you need to implement your code in such a way that the code should be able to determine the size n as an input and use it to initialize each pair of matrix and vector from files.

For each algorithm, you are going to write three major code components (e.g., Step 1, Step 2a and Step 2b) to do the followings in step:

- Step 1: Decompose $\mathbf{A}$ into a lower $\mathbf{L}$ and an upper $\mathbf{U}$ triangular matrices using Gaussian elimination with partial pivoting ($\mathbf{A} = \mathbf{LU}$),
- Step 2: Solve two successive triangular systems:
  - Step 2a: $\mathbf{Ly} = \mathbf{b}$ (forward substitution),
  - Step 2b: $\mathbf{Ux} = \mathbf{y}$ (backward substitution),

to finally solve the linear system $\mathbf{Ax} = \mathbf{b}$.

Once the code finishes solving for $\mathbf{x}$, the solution $\mathbf{x}$ is going to be written to screen in a human readable format, keeping its entries in a vector form. Please also implement to save your solution to a file, `x_i.dat` for each `i`.

Use your code to solve the following linear systems $\mathbf{Ax} = \mathbf{b}$:

- $$\mathbf{A} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & -2 \\ -2 & 1 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$
- $$\mathbf{A} = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \tag{2}$$
- $$\mathbf{A} = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 3 & -3 & 3 \\ 2 & -4 & 7 & -7 \\ -3 & 7 & -10 & 14 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 2 \\ -2 \\ -8 \end{bmatrix} \tag{3}$$

Modular Programming: You're going to implement your code in modular form for which you need to design and organize your code components in several subroutines and functions appropriately. Your code should have one main driver routine which calls necessary subroutines and functions. For example, you would want to have the following code structures:

- `linear_solve.f90` – this is going to be your main driver routine, within which you call the followings subroutines:
  - `read_data.f90` – this reads in $\mathbf{A}$ and $\mathbf{b}$ (you will need to prepare them using Python implementations)
  - `write_to_screen.f90` – this writes both $\mathbf{A}$ and $\mathbf{b}$ to screen for sanity check
  - `LU_decomp.f90` – Gaussian elimination with or without partial pivoting
  - `forward_solve.f90` – this solves $\mathbf{Ly} = \mathbf{b}$
  - `backward_solve.f90` – this solves $\mathbf{Ux} = \mathbf{y}$
  - `write_data.f90` – this outputs your result onto screen as well as to a file, `x_i.dat` for each `i`

Makefile: Please compile your code using a makefile, which is going to be executed by Python. When coding, please make sure you use useful *debugging Fortran flags* for easy debugging processes, for instance, with gdb. Later, you run your code with *optimization flags* only after you are convinced with the

code. See sections on **Fortran Flags** and **Makefiles** in the lecture note.

Cross Comparison: Please make sure you get a correct solution using Python's linear algebra routines in numpy.

No External Linear Algebra Libraries: You can absolutely get some coding ideas from other resources, but you are *not allowed* to copy and paste other's codes in any circumstances, nor use Fortran's linear algebra libraries (e.g., LAPACK or BLAS).

1.2. Python Implementations

Your Python implementation needs to conduct the following roles:

- Run setup:
  - Initialize three different inputs of a pair **A** and **b** and write them to files, `A_i.dat` and `b_i.dat` for each `i`.
  - Compile (e.g., execute `make`) your Fortran code in `"project/LinAlg"`, from within the Python directory `"project/PyRun"`. Similar to homework 6, you first do `make clean` if the code has been already compiled, and then `make`. You do this compilation step only once before running three different cases.
- Run scheduler:
  - Run three different cases in Eqs. (1) – (3), by executing your Python routine from the Python directory to run the Fortran algorithm.
- Solution check and visualizer:
  - Implement a short numpy linear algebra algorithm (direct inversion is fine) to check the correctness of your Fortran solutions. This outputs `Pass/Fail` by comparing the Fortran and Python solutions. You can use a small threshold value to quantify an error in absolute value between the two solutions as previously done in the class homework sets. If `Fail`, output the error together with the Fortran and Python solutions to screen.
  - Produce three plots of **A** in Eqs. (1) – (3), and another three plots of **x** and **b** using `plt.subplots(1,2,1)` and `plt.subplots(1,2,2)`, respectively.
    See how to plot a $n \times n$ matrix in:
    http://stackoverflow.com/questions/21071128/matplotlib-plot-numpy-matrix-as-0-index.

1.3. LaTeX Report

Write your final report using LaTeX (7-page limit including figures). You have to write three parts in your report:

- Abstract

- Body: methods, results, findings, comments, etc.
- Conclusion

1.4. Website Update

Upload your LaTeX report and your source codes to your website, under a new tab, "`Project`".