Abstract

The project integrated our use of fortran, python, latex and webpage design. I developed a numerical linear algebra solver that applied Gaussian elimination with pivoting for solving linear systems. The algorithm includes LU factorization and partial pivoting for Gaussian elimination. The solver is developed with fortran implementation, while python helps to initialize the data inputs, compile fortran code, check correctness of solution and visualize data. With three linear examples, my solver successfully passed the correctness check and output the corresponding solutions for each example.

1 Introduction

The project aims to develop a numerical linear algebra solver that applies Gaussian elimination with pivoting for solving linear systems. It consists of two parts. Firstly, a linear algebra solver is developed with Fortran implementation. Secondly, we use python implementation to run a setup, scheduler, and a data visualizer for the solver. The solver is then tested with three examples. The results are briefly presented and discussed in this report.

2 Methods

I implemented Gaussian elimination with pivoting to develop the solver. To solve the linear system $Ax = b$, with $A$ being a $n \times n$ square matrix and $x$ and $b$ being $n$-vectors, I decompose the problem with LU factorization and pivoting.

2.1 LU factorization

LU factorization is to devise a nonsingular linear transformation that transforms a given general linear system into a triangular linear system. Specifically, we transform $A$ matrix with matrix $M$,

$$M = M_{n-1} \cdots M_1.$$  

Then the $U$ and $L$ matrices are defined as,

$$U = MA$$

$$L = M_1^{-1} \cdots M_{n-1}^{-1}.$$  

Therefore, we transform matrix $A$ with upper triangular matrix $U$ and lower triangular matrix $L$, $A = LU$. The system is easily solved to derive vector $x$,

$$Ly = b$$

$$Ux = y.$$
2.2 Partial Pivoting

In the cases that the choice of a divisor, i.e., a pivot, is zero or close enough to zero, the Gaussian elimination algorithm breaks down. Therefore, if the pivot entry is zero at stake $k$, we interchange row $k$ of both the matrix and the right-hand side vector with a subsequent row whose entry in column $k$ is in the largest magnitude within the column. The transformation is achieved by a permutation matrix $P$,

$$P = P_{n-1} \cdots P_1$$

$$M_{n-1}P_{n-1} \cdots M_1P_1A = U.$$ 

3 Code implementation

3.1 Fortran implementation

- Linear_solve.f90: this is the main driver routine, within which I call the following subroutines:
  - read_data.f90: this reads in $A$ and $b$ from data file $A_i.dat$ and $b_i.dat$, respectively.
  - write_to_screen.f90: this writes both $A$ and $b$ to screen for sanity check
  - LU_decomp.f90: Gaussian elimination with partial pivoting
  - forward_solve.f90: this solves $Ly = b$
  - backward_solve.f90: this solves $Ux = y$
  - write_data.f90: this outputs the result onto screen as well as to a file, $x_i.dat$ for each $i$

3.2 Python implementation

- PyRun_linAlg.py: this is the python routine, within which I implement the following tasks:
  - Run setup: It initializes three input examples of a pair $A$ and $b$ and writes them to files, $A_i.dat$ and $b_i.dat$ for each $i$. It compiles the Fortran code.
  - Run scheduler: Run three different cases, by executing the Python routine from the Python directory to run the Fortran algorithm.
  - Solution check: Implement a numpy linear algebra algorithm to check the correctness of the Fortran solutions. This outputs Pass/Fail by comparing the Fortran and Python solutions. If Fail, output the error together with the Fortran and Python solutions to screen.
  - Data visualizer: Produce three plots of $A$ for three examples, and another three plots of $x$ and $b$ using subploting methods.

4 Examples

4.1 Inputs

I input the three linear system examples into python and produce data files with matrix $A$ and vector $b$ to be read into fortran. The first example has a dimension of 3 and the latter two examples have dimension of 4.

4.2 Outputs

The Figure 2 presents output that is printed to the screen. The solutions are the fortran solution and they are compared with python numpy solution for correctness check. It turned out that our examples successfully pass all three checks.
4.3 Results and Discussions

The matrices and vectors could be visualized with python visualizer. Figure 3, 5 and 7 present the visualization for matrices A, while figure 4, 6 and 8 present the visualization for vectors $x$ and $b$.

Although our examples have no need for pivoting, my code works perfectly with cases in which pivoting is needed for Gaussian elimination.

5 Conclusions

This project integrated our use of fortran, python, latex and webpage design. I developed a numerical linear algebra solver that applied Gaussian elimination with pivoting for solving linear systems. The algorithm includes LU factorization and partial pivoting for Gaussian elimination. The solver is developed with fortran implementation with modular programming. I have multiple subroutines that read data, LU decompose, backward solve, forward solve, output data, etc. A main driver routine then calls all the subroutines and a makefile helps compile and execute the code. I use python to initialize the data inputs, compile fortran code, check correctness of solution and visualize data. I run the routines with three linear examples. The solver successfully passed the correctness check and output the corresponding solutions for each example.

\[
A = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & -2 \\ -2 & 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}
\]

\[
A = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}
\]

\[
A = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 3 & -3 & 3 \\ 2 & -4 & 7 & -7 \\ -3 & 7 & -10 & 14 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 2 \\ -2 \\ -8 \end{bmatrix}
\]

Figure 1: Inputs for three examples
Figure 2: Fortran and Python output for three examples

**Problem 1**

**A Matrix**

\[
\begin{bmatrix}
1 & 1 & -1 \\
1 & 2 & -2 \\
-2 & 1 & 1 \\
\end{bmatrix}
\]

**b vector**

\[
\begin{bmatrix}
1 \\
0 \\
1 \\
\end{bmatrix}
\]

**Solution**

\[
\begin{bmatrix}
2.000 \\
2.000 \\
3.000 \\
\end{bmatrix}
\]

**Problem 2**

**A Matrix**

\[
\begin{bmatrix}
4 & 3 & 2 & 1 \\
3 & 4 & 3 & 2 \\
2 & 3 & 4 & 3 \\
1 & 2 & 3 & 4 \\
\end{bmatrix}
\]

**b vector**

\[
\begin{bmatrix}
1 \\
1 \\
-1 \\
-1 \\
\end{bmatrix}
\]

**Solution**

\[
\begin{bmatrix}
0.000 \\
1.000 \\
-1.000 \\
0.000 \\
\end{bmatrix}
\]

**Problem 3**

**A Matrix**

\[
\begin{bmatrix}
1 & -1 & 1 & -1 \\
-1 & 3 & -3 & 3 \\
2 & -4 & 7 & -7 \\
-3 & 7 & -10 & 14 \\
\end{bmatrix}
\]

**b vector**

\[
\begin{bmatrix}
0 \\
2 \\
-2 \\
-8 \\
\end{bmatrix}
\]

**Solution**

\[
\begin{bmatrix}
1.000 \\
1.000 \\
-3.000 \\
-3.000 \\
\end{bmatrix}
\]

PASS
PASS
PASS
Figure 3: Data visualization for matrix $A$ in case 1

Figure 4: Data visualization for matrix $x$ and $b$ in case 1

Figure 5: Data visualization for matrix $A$ in case 2
Figure 6: Data visualization for matrix $x$ and $b$ in case 2

Figure 7: Data visualization for matrix $A$ in case 3

Figure 8: Data visualization for matrix $x$ and $b$ in case 3