Homework #2
CS 575: Numerical Linear Algebra
Spring 2022

John Tran

February 7, 2023

Important Notes

- `Python` 3.11 was used to run the notebook (i.e., to use the `match` statement)
- the partially completed notebook was used to complete the coding assignment, so many of the things asked (e.g., verification and testing) was done for us and they were modified for the mat-mat portion
- the `README` was done in `Markdown` but the raw text can still be viewed

Problem 1

We are given vectors $x, y \in \mathbb{R}^n$ with the assumption $n\epsilon_m \ll 1$ and we want to show that $\text{fl}\Big(x^T y\Big) \approx \sum_{i=1}^{n} x_i y_i (1 + \delta_i)$ where $\delta_i \leq n\epsilon_m$.

We can first expand $\text{fl}\Big(x^T y\Big)$:

$$
\begin{aligned}
\text{fl}\Big(x^T y\Big) &= \text{fl}\left(\sum_{i=1}^{n} x_i y_i\right) \quad \text{expand inner product} \\
&= \sum_{i=1}^{n} \text{fl}(x_i y_i) \quad \text{linearity of fl()} \\
&= \sum_{i=1}^{n} \text{fl}(x_i)\text{fl}(y_i) \quad \text{separable cases of fl()} \\
&\quad \text{(intermediate representation)} \\
&= \sum_{i=1}^{n} [x_i(1+\epsilon_m)][y_i(1+\epsilon_m)] \quad \text{def. of fl()} \\
&= \sum_{i=1}^{n} x_i y_i (1+\epsilon_m)^2 \\
&= \sum_{i=1}^{n} x_i y_i \Big(1 + 2\epsilon_m + \epsilon_m^2\Big) \\
&\approx \sum_{i=1}^{n} x_i y_i (1 + 2\epsilon_m) \quad \text{first-order approximation}
\end{aligned}
$$

where the approximation is from the fact that $\epsilon_m \ll 1$

If we match our solution with what was given, we can see that as long as $n \geq 2$, then $\text{fl}\left(x^T y\right) \approx \sum_{i=1}^{n} x_i y_i (1 + \delta_i)$ holds where $\delta_i = 2\epsilon_m$ and the equality of $\delta_i \leq n\epsilon_m$ is from $n = 2$.

Problem 2

We are asked to solve $Ax = b$ using Gaussian Elimination without Row Swapping where

$$A = \begin{bmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} -1 \\ -7 \\ -6 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

If we use the augmented form, we have

$$\left[\begin{array}{ccc|c} -3 & 2 & -1 & -1 \\ 6 & -6 & 7 & -7 \\ 3 & -4 & 4 & -6 \end{array}\right]$$

First we introduce 0s in the first column (after the first row) by adding 2 times the first row to the second row

$$\left[\begin{array}{ccc|c} -3 & 2 & -1 & -1 \\ 0 & -2 & 5 & -9 \\ 3 & -4 & 4 & -6 \end{array}\right]$$

and add the first row to the third row

$$\left[\begin{array}{ccc|c} -3 & 2 & -1 & -1 \\ 0 & -2 & 5 & -9 \\ 0 & -2 & 3 & -7 \end{array}\right]$$

Now we can introduce 0s in the second column (after the second row) by subtracting the second row from the third row

$$\left[\begin{array}{ccc|c} -3 & 2 & -1 & -1 \\ 0 & -2 & 5 & -9 \\ 0 & 0 & -2 & 2 \end{array}\right]$$

From here, we can start using back-substitution to solve our system

$$\begin{bmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -9 \\ 2 \end{bmatrix}$$

So, we have

$$\begin{aligned} x_3 &= \frac{2}{-2} \\ &= -1 \\ x_2 &= \frac{-9 - 5(-1)}{-2} \\ &= \frac{-4}{-2} \\ &= 2 \\ x_1 &= \frac{-1 - (-1)(-1) - 2(2)}{-3} \\ &= \frac{-1 - 1 - 4}{-3} \\ &= \frac{-6}{-3} \\ &= 2 \end{aligned}$$

After back-substituting, we have our solution

$$x = \begin{bmatrix} 2 \\ 2 \\ -1 \end{bmatrix}$$

Problem 3

We are asked to solve $Ax = b$ using Gaussian Elimination with Partial Pivoting where

$$A = \begin{bmatrix} 2 & 4 & -2 & -2 \\ 1 & 2 & 4 & -3 \\ -3 & -3 & 8 & -2 \\ -1 & 1 & 6 & -3 \end{bmatrix} \qquad b = \begin{bmatrix} -4 \\ 5 \\ 7 \\ 7 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

If we use the augmented form, we have

$$\left[\begin{array}{rrrr|r} 2 & 4 & -2 & -2 & -4 \\ 1 & 2 & 4 & -3 & 5 \\ -3 & -3 & 8 & -2 & 7 \\ -1 & 1 & 6 & -3 & 7 \end{array}\right]$$

For Partial Pivoting, we have to look at the corresponding element in a given column (initially in the first column) with the highest magnitude (with all rows ***below*** the current one). So for the first row swap, we have the third row with the highest magnitude of 3, swapping the first and third rows:

If we use the augmented form, we have

$$\left[\begin{array}{rrrr|r} -3 & -3 & 8 & -2 & 7 \\ 1 & 2 & 4 & -3 & 5 \\ 2 & 4 & -2 & -2 & -4 \\ -1 & 1 & 6 & -3 & 7 \end{array}\right]$$

Then we continue with eliminating zeroes in the first column like before, starting with adding 1/3 times the (new) first row to the second row:

$$\left[\begin{array}{rrrr|r} -3 & -3 & 8 & -2 & 7 \\ 0 & 1 & \frac{20}{3} & -\frac{11}{3} & \frac{22}{3} \\ 2 & 4 & -2 & -2 & -4 \\ -1 & 1 & 6 & -3 & 7 \end{array}\right]$$

and adding 2/3 times the first row to the third row:

$$\left[\begin{array}{rrrr|r} -3 & -3 & 8 & -2 & 7 \\ 0 & 1 & \frac{20}{3} & -\frac{11}{3} & \frac{22}{3} \\ 0 & 2 & \frac{10}{3} & -\frac{10}{3} & \frac{2}{3} \\ -1 & 1 & 6 & -3 & 7 \end{array}\right]$$

and finally subtracting 1/3 times the first row from the fourth row:

$$\left[\begin{array}{rrrr|r} -3 & -3 & 8 & -2 & 7 \\ 0 & 1 & \frac{20}{3} & -\frac{11}{3} & \frac{22}{3} \\ 0 & 2 & \frac{10}{3} & -\frac{10}{3} & \frac{2}{3} \\ 0 & 2 & \frac{10}{3} & -\frac{7}{3} & \frac{14}{3} \end{array}\right]$$

Now we check the row swapping again for the second column. Since both the third and fourth rows have the same magnitude, either can be chosen to be swapped with the second row. So, the third row was chosen for this example:

$$\left[\begin{array}{cccc|c} -3 & -3 & 8 & -2 & 7 \\ 0 & 2 & \frac{10}{3} & -\frac{10}{3} & \frac{2}{3} \\ 0 & 1 & \frac{20}{3} & -\frac{11}{3} & \frac{22}{3} \\ 0 & 2 & \frac{10}{3} & -\frac{7}{3} & \frac{14}{3} \end{array}\right]$$

Again, we can introduce 0s in the second column by first subtracting 1/2 times the second row from the third row:

$$\left[\begin{array}{cccc|c} -3 & -3 & 8 & -2 & 7 \\ 0 & 2 & \frac{10}{3} & -\frac{10}{3} & \frac{2}{3} \\ 0 & 0 & \frac{15}{3} & -2 & 7 \\ 0 & 2 & \frac{10}{3} & -\frac{7}{3} & \frac{14}{3} \end{array}\right]$$

and subtract the second row from the fourth row:

$$\left[\begin{array}{cccc|c} -3 & -3 & 8 & -2 & 7 \\ 0 & 2 & -\frac{10}{3} & -\frac{10}{3} & \frac{2}{3} \\ 0 & 0 & \frac{15}{3} & -2 & 7 \\ 0 & 0 & 0 & 1 & 4 \end{array}\right]$$

After this step, we are done since the last row operation not only created a 0 in the second column, but also in the third column and allowed us to save a row swap on the third column and any subsequent row operations to get a 0 in rows below the third row.

From here, we can start using back-substitution to solve our system

$$\begin{bmatrix} -3 & -3 & 8 & -2 \\ 0 & 2 & \frac{10}{3} & -\frac{10}{3} \\ 0 & 0 & \frac{15}{3} & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 7 \\ \frac{2}{3} \\ 7 \\ 4 \end{bmatrix}$$

So, we have

$$
\begin{aligned}
x_4 &= 4 \\
x_3 &= \frac{7-(-2)(4)}{\frac{15}{3}} \\
&= \frac{7+8}{\frac{15}{3}} \\
&= \frac{15}{\frac{15}{3}} \\
&= 3 \\
x_2 &= \frac{\frac{2}{3}-(-\frac{10}{3})(4)-(10/3)(3)}{2} \\
&= \frac{\frac{2}{3}+\frac{40}{3}-10}{2} \\
&= \frac{14-10}{2} \\
&= \frac{4}{2} \\
&= 2 \\
x_1 &= \frac{7-(-2)(4)-8(3)-(-3)(2)}{-3} \\
&= \frac{7+8-24+6}{-3} \\
&= \frac{21-24}{-3} \\
&= \frac{-3}{-3} \\
&= 1
\end{aligned}
$$

After back-substituting, we have our solution

$$
x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}
$$

Problem 4

The pseudo code provided to us by the Gaussian Elimination slides (both without Row Swapping and with Partial Pivoting) was adapted for this homework. The code can be found in the `Jupyter` notebook `cs575_hw2_notebook_JTran.ipynb`.

Gaussian Elimination Without Row Swapping

If we look at the code, we can see that we need to be careful about the internal representation of the matrix (i.e., set the `numpy` array to be a `float` array as intermediate steps may result in non-integer values which may be rounded if left to the default `int` type – crucial for the next part). Then looking at the intermediate values for A and b, they match with what we got when doing it by hand above (after the forward elimination). Then the final solution x does indeed match with what we got above (after the backward substitution).

The variable in the code `x` represents the solution formed by the implementation and `actual x` represents the hard-coded solution we got by hand above. We can see the difference by the `.` at the end of each of the values that shows a floating-point number and the lack thereof with just plain integers for `x` and `actual x`, respectively.

```
(3, 3)
(3,)
A: [[-3.  2. -1.]
 [ 6. -6.  7.]
 [ 3. -4.  4.]]
b: [-1. -7. -6.]
intermediate before
A: [[-3.  2. -1.]
 [ 0. -2.  5.]
 [ 0.  0. -2.]]
b: [-1. -9.  2.]
intermediate after
x: [ 2.  2. -1.]
actual x: [ 2  2 -1]
```

Figure 1: Output of Gaussian Elimination without Row Swapping

Gaussian Elimination With Partial Pivoting

Again, we need to be careful about the internal representation of the values in A and b. However, swapping of the rows in A can be tricky because they contain the ***pointer*** to the row. So, to make the coding simpler, the entire row was copied instead of keeping track of pointers to switch values between two rows. Besides the extra implementation to deal with the row swapping (i.e., find the max magnitude and swap the rows, if necessary) for the forward elimination, everything else was pretty much the same as before – the backward substitution was abstracted and reused for this Gaussian elimination with row swapping with minor modifications to the forward elimination portion. Then looking at the intermediate values for A and b, they match with what we got when doing it by hand above (after the forward elimination). Then the final solution x does indeed match with what we got above (after the backward substitution).

```
(4, 4)
(4,)
A: [[ 2.  4. -2. -2.]
 [ 1.  2.  4. -3.]
 [-3. -3.  8. -2.]
 [-1.  1.  6. -3.]]
b: [-4.  5.  7.  7.]
intermediate before
A: [[-3.         -3.          8.         -2.        ]
 [ 0.          2.          3.33333333 -3.33333333]
 [ 0.          0.          5.         -2.        ]
 [ 0.          0.          0.          1.        ]]
b: [7.         0.66666667 7.         4.        ]
intermediate after
x: [1. 2. 3. 4.]
actual x: [1 2 3 4]
```

Figure 2: Output of Gaussian Elimination with Partial Pivoting

Problem 5

We are asked to compare our Gaussian Elimination implementation with Partial Pivoting with the built-in solver. To verify that the randomly generated matrix (code used from the previous homework), we used the `numpy.det()` function to make sure the determinant of the matrix was non-zero corresponding to a singular matrix (or above some tolerance due to limited numerical precision). From there, we used the specified sizes $N = 10, 20, 40, 80$ to run our implementation and built-in solver against.

When running our code multiple times, we got varied results – no direct correlation between the matrix/vector size and norm of the error or the timings of our implementation and the built-in solver. Sometimes, the norm would increase then decrease again with size, but it should depend on the

generated matrix A and vector b. For the timings (which was not explicitly stated in the question but completed to compare how efficient our implementation was compared to the built-in solver), the results were arguably even more inconsistent as a lot of runs resulted in `0.0`s time (negligible). Especially for the built-in solver, most of the runs were not able to be captured (`0.0`s time).

Looking at the magnitude of the norm of errors, we can see that they are very small (around `1e-15` to `1e-11` across multiple runs) and small errors are expected for this scenario. In terms of timing, both finish execution relatively fast where the built-in solver has consistently faster timings, as expected with the optimizations made in `Python`.

The results of one of the runs are shown below:

matrix/vector size (N)	norm of error	timing of built-in solver	timing of GE (partial pivoting)
10	2.217302348033905e-15	0.0	0.0
20	5.107044768930868e-15	0.0	0.0
40	5.317810023770905e-15	0.0	0.0
80	4.800009874308177e-14	0.015620708465576172	0.046869754791259766

Figure 3: Table of the norm of the error and the timings of our implementation and the built-in solver for various sizes

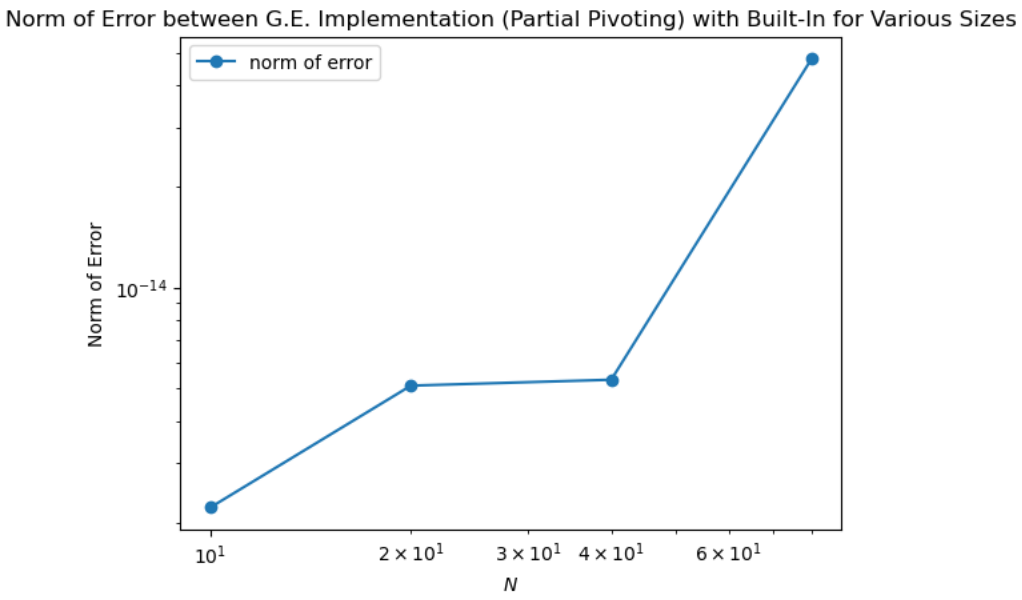

Figure 4: Plot of the norm of the error for various sizes

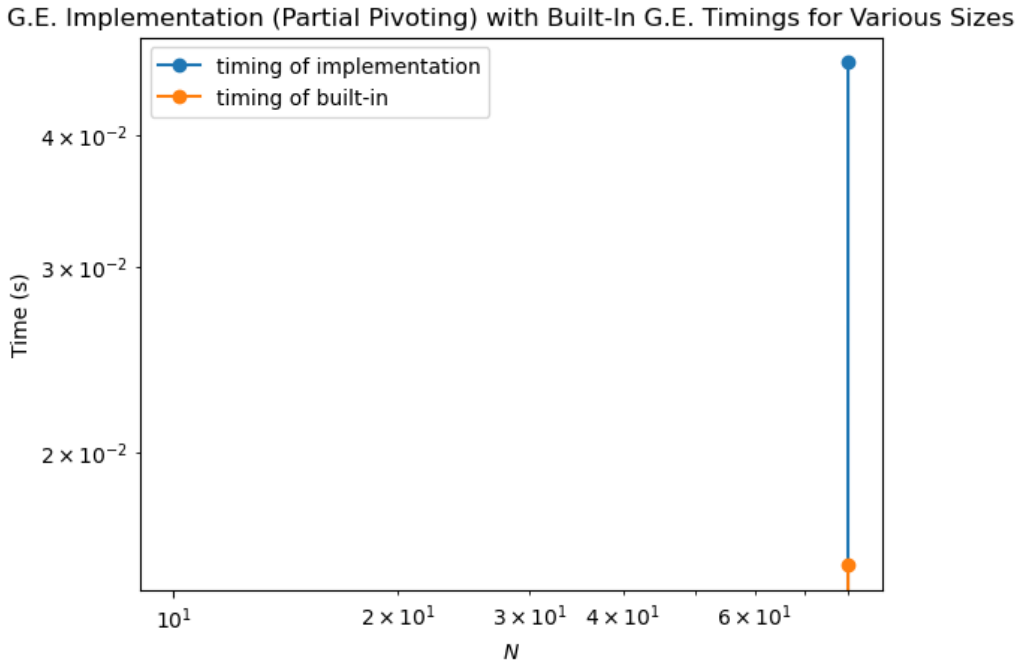

Figure 5: Plot of the timings for various sizes