Chapter 1.8: When $A(i, i)$ is 0 (or small). So far we have no at all addressed the obvious issue of what to do when one of the diagonal entries is 0 (or, given that we will work in floating point arithmetic, close to 0). This may and will happen.

The answer will be to employ one of the other elementary operations to help us out: swapping rows.

First a bit of notation. Following MATLAB, we will denote the following submatrices as follows

- $A(i : j, k : l)$
- $A(i : j, :) = A(i : j, 1 : n)$
- $A(i, j : k) = A(i : i, j : k)$

To quickly outline the method, each time we iterate through the first for loop, we will search for the largest entry in $A(i : n, i)$, swap the $i$th row with the row in which that largest entry is, and continue with the rest of the code, recording all the swaps. The result will be a three-pronged output, $L, U, P$ such that $LU = PA$, where $P$ will be a permutation matrix. More on this next time.

Motivation. Why do we swap rows to make sure the largest entry is the one that becomes $A(i, i)$? The answer is that when we divide by small quantities (not necessarily non-zero), we incur significant errors, and these errors accumulate. As such, we will want to divide by the largest quantities possible, in order to minimize errors; and since we can keep track of row swaps and do almost no extra work, there is no downside to swapping rows to keep things as accurate as possible in a floating point computing environment.

Permutations. To record whatever swaps of rows we will effect, we need to use a permutation matrix $P$.

Remark 1. We define a permutation matrix $P$ of size $n \times n$ as a $0 - 1$ matrix which has precisely one 1 in each row and column, and this means that such a matrix has an associated function $i \rightarrow p(i)$ with $(i, p(i))$ being the entry which is 1 in row $i$. Such a function is a bijection from $\{1, 2, \ldots, n\}$ to itself, aka a permutation, and this is what gives the matrix its name. Also, for an $n \times n$ matrix $A$, $PA$ is the matrix obtained from $A$ by rearranging the rows of $A$ in the order given by the permutation. See example below.
Example.

\[
P = \begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}, \quad p(1) = 3, \; p(2) = 1, \; p(3) = 2;\]

and if

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}, \quad \text{then } PA = \begin{bmatrix}
7 & 8 & 9 \\
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}.
\]

To come back to the discussion of a stable Gaussian elimination or variant of the LU factorization, the output of this code would then be three-fold: \([L, U, P]\), where \(P\) would be the permutation matrix encoding the row swaps that have been done, and so \(PA = LU\). (Alternately, this can be achieved by returning a vector that represents the permutation itself.)

To write this code, you will have to take the following ingredients. For each outer iteration \(i\),

- look down the column and find the maximum entry (in absolute value) and its index. This can be achieved by doing, at step \(i\),

\[
[m, \text{index}] = \max(\text{abs}(A(:, i))) \quad \text{index} = \text{index} + i - 1;
\]

as \(\text{index} = 2\), for example, means the largest entry is in row \(i + 1\).

- if \(i \neq \text{index}\), swap rows \(A(i, :)\) and \(A(\text{index}, :)\), but also \(L(i, 1:(i-1))\) and \(L(\text{index}, 1:(i-1))\) (provided \(i \geq 2\))

- record the swapping either in \(P\) or in the vector \(p\)

- continue with the rest of the code.

The amount of work thus added is \(O(n^3)\) assignments, so that does not change the \(2/3n^3 + O(n^2)\) flop count.

The above is called partial pivoting; even more accuracy can be obtained by doing total pivoting, where one searches for the pivot not only among rows but also among columns.

**Remark 2.** It’s notable that the LU factorization does not always exist, and even when it does, it is unstable. By contrast, the PLU factorization always exists and it is stable.