Lec 11

Pivoting Strategies.

Recall that last time we considered a system in which a pivot was not zero, but was small.

$$\varepsilon x_1 + x_2 = 1$$
$$x_1 + x_2 = 2 \qquad \varepsilon \ll 1$$

We showed that

$$x_{1,exact} = \frac{1}{1-\varepsilon} \simeq 1$$

but that, on the machine,

$$x_{1,mach} \cong 0$$

Thus relative error is

$$\left| \frac{x_{1,mach} - x_{1,exact}}{x_{1,exact}} \right| = 100\%$$

This is terrible!

Can we remedy this by finding "first" non-zero element below pivot? Perhaps, but what if that element is also small? Better would be to choose "largest" element below pivot, because this maximizes chance that new pivot is not small :

PARTIAL PIVOTING

ALGORITHM : Choose row p that corresponds to the largest element from the set $\{|a_{i,i}|, |a_{i+1,i}| \dots |a_{m,i}|\}$. Then perform the row exchange $R_i \leftrightarrow R_p$ [DO THIS EVEN IF PIVOT IS NON-ZERO]

Let's apply to our example:

Recall : $\varepsilon x_1 + x_2 = 1$
$x_1 + x_2 = 2$

partial pivoting $\Rightarrow$ $R_1 \leftrightarrow R_2$

$$x_1 + x_2 = 2$$
$$\varepsilon x_1 + x_2 = 1$$

$$\left(\begin{array}{cc|c} 1 & 1 & 2 \\ \varepsilon & 1 & 1 \end{array}\right)$$

$$\rightarrow \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1-\varepsilon & 1-2\varepsilon \end{array}\right)$$

$$x_2 = \frac{1-2\varepsilon}{1-\varepsilon}$$

$$x_1 + x_2 = 2$$
$$x_1 = 2 - x_2$$

$$
\begin{aligned}
1-2\varepsilon &= (1000.0 - 0.2)\times 10^{-3} \\
&= 999.8 \times 10^{-3} \\
&= 0.9998 \times 10^{0} \\
&\overset{\text{mach}}{=} 1.000 \times 10^{0} \\
&\equiv 0.100 \times 10^{1}
\end{aligned}
$$

Similarly

$$1-\varepsilon \overset{\text{mach}}{=} 1$$

Thus $x_2 \simeq 1$

Now, however,

$$
\begin{aligned}
x_1 &= 2 - x_2 \\
&\overset{\text{mach}}{=} 2-1 \\
&= 1
\end{aligned}
$$

which is much closer to the exact sol^{n}. than 0 (the answer that standard GE w/ BS gives, cf Lec 10).

In partial pivoting we must find the location of the element in the set w/ the largest magnitude. This procedure occurs often in computer science (eg machine learning) and is known as "argmax"

$$p = \underset{q}{\operatorname{argmax}} \{ |a_{qi}| : q = i, \dots, n \}$$

How might you implement this in Python? (Go to ipynb file).

Ex Perform G.E. w/ partial pivoting to solve $Ax=b$.

Sol^{n}

$$A \left\{ \begin{array}{cccc|c} 1 & -1 & 1 & 1 & 1 \\ 2 & -2 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{array} \right\} b$$

$$\begin{array}{cccc|c} 1 & -1 & 1 & 1 & 1 \\ 0 & 0 & -1 & -1 & -1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 2 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{cccc|c} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & -1 & -1 & -1 \end{array}$$

$$\begin{array}{cccc|c} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & -1 & -1 \end{array}$$

$$\begin{array}{cccc|c} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 1 \end{array}$$

- $x_4 = 1$
- $-x_3 - x_4 = -1 \Rightarrow -x_3 = 0 \Rightarrow x_3 = 0$
- $2x_2 = 0 \Rightarrow x_2 = 0$
- $x_1 - x_2 + x_3 + x_4 = 1 \Rightarrow x_1 + 1 = 1 \Rightarrow x_1 = 0$

$$Ax = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 2 & -2 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = b$$

... as required.

BUT: <u>Even partial pivoting can fail!</u>

Here's how ... Recall:

$$\varepsilon x_1 + x_2 = 1$$
$$x_1 + x_2 = 2 \qquad \varepsilon \ll 1.$$

Multiply 1st eq$^{\underline{n}}$ by $10/\varepsilon$

$$10 x_1 + \frac{10}{\varepsilon} x_2 = \frac{10}{\varepsilon}$$
$$x_1 + x_2 = 2$$

(of course, sol$^{\underline{n}}$ is unchanged).

Now partial pivoting has no effect since "10" is already the largest element in the 1st column. This leads to the same round-off problem as before (See Lec 10)

$$\begin{array}{r|cc|c} & 10 & \frac{10}{\varepsilon} & \frac{10}{\varepsilon} \\ R_2 - \frac{1}{10}R_1 & 0 & 1-\frac{1}{\varepsilon} & 2-\frac{1}{\varepsilon} \end{array}$$

$$x_2 = \frac{2-1/\varepsilon}{1-1/\varepsilon} \qquad (*)$$

$$x_1 = \frac{\frac{10}{\varepsilon} - \frac{10}{\varepsilon}x_2}{10} \qquad (**)$$

$(*) \Rightarrow x_2 \simeq 1$

$(**) \Rightarrow x_1 \simeq 0.$ (instead of $x_1 \simeq 1$)

What this example teaches us is that not just $|a_{11}|$ vs $|a_{21}|$ matters, but also $|a_{11}|$ vs $|a_{12}|$ and $|b_1|$.

Specifically $|a_{11}| \ll |a_{12}|, |b_1|$ regardless of whether the 1^{st} $eq^{\underline{n}}$ is multiplied by $10/\varepsilon$ or not.

Take this into account by choosing a pivot that is largest relative to the entries in its row

SCALED PARTIAL PIVOTING

ALGORITHM: Choose

$$s_i = \max\{|a_{ij}| : j = 1, \dots, n\}.$$

If $s_i = 0 \Rightarrow$ no unique $sol^{\underline{n}}$, since matrix is rank deficient. Otherwise, choose row p that corresponds to the largest element from the set $\left\{ \frac{|a_{ii}|}{s_i}, \frac{|a_{i+1,i}|}{s_{i+1}}, \dots, \frac{|a_{n,i}|}{s_n} \right\}$. Then perform the row exchange $R_i \Leftrightarrow R_p$.

As before

$$p = \underset{q}{\operatorname{argmax}} \left\{ \frac{|a_{qi}|}{s_q} : q = i \dots n \right\}$$

Revisit the problematic example:

$$\left.\begin{array}{cc|c} 10 & \frac{10}{\varepsilon} & \frac{10}{\varepsilon} \\ 1 & 1 & 2 \end{array}\right. \qquad \begin{array}{l} s_1 = \frac{10}{\varepsilon} \\ s_2 = 1 \end{array} \qquad \left.\begin{array}{l} \frac{|a_{11}|}{s_1} = \varepsilon \\ \frac{|a_{21}|}{s_2} = 1 \end{array}\right\} R_1 \leftrightarrow R_2$$

$$\left.\begin{array}{cc|c} 1 & 1 & 2 \\ 10 & 10/\varepsilon & 10/\varepsilon \end{array}\right.$$

$$\left.\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & \frac{10}{\varepsilon}-10 & \frac{10}{\varepsilon}-10(2) \end{array}\right.$$

$$x_2 = \frac{\frac{10}{\varepsilon}-10(2)}{\frac{10}{\varepsilon}-10} \overset{\text{mach.}}{\simeq} 1$$

$x_1 = 2 - x_2 \overset{\text{mach}}{\simeq} 1.$ .... which is a much better approx. of true answer.

Scaled partial pivoting:
Additional computational cost

- Let's count only comparisons (not divisions).
- $n-1$ comparisons for each of n rows to determine $s_1, s_2, \dots, s_n$

$\Rightarrow n(n-1)$ comparisons

- Having figured out the $\{s_i\}$, we then use them to determine the pivot for each of n columns. For any given column i, we must perform comparisons among $\left\{ \frac{|a_{qi}|}{s_q} : q = i \dots n \right\}$:

column	# comparisons
1	$n-1$
2	$n-2$
⋮	
$n-1$	1

- total # comparisons

$$= n(n-1) + \sum_{k=1}^{n-1} k$$

$$= n(n-1) + \frac{(n-1)(n)}{2}$$

$$= \frac{3}{2} n(n-1) = O(n^2)$$

Similar computation for # divisions also yields $O(n^2)$ [Text p379].

Thus additional computations are insignificant [compared to $O(n^3)$].

If, on the other hand, the scale factors were recomputed after each column had been zero'd, then the additional cost would be $O(n^3)$, see Text p379, and therefore comparable to standard Gaussian elimination.