Week 4: Wednesday, Sep 16

Administrative notes

1. I will be traveling from the afternoon of 9/18 through the evening of 9/20, and will have (at best) intermittent email contact. If you have questions to ask me about the homework that is due on Monday, ask before then!

2. The MathWorks is hosting free technical seminars on MATLAB on 9/23 and 9/24. If you are interested, information is available here:

   `http://www.mathworks.com/seminars/cu0909`

A teaser revisited

In a "teaser" problem I gave before class last week (related to the earlier railroad problem teaser), I asked for a routine to accurately determine θ from δ, $0 < \delta \ll 1$, in

$$f(\theta) = \frac{\sin\theta}{\theta} - (1-\delta) = 0.$$

The most obvious approach is Newton's iteration; the only problem is that roundoff in the evaluation of $f(\theta)$ kills the accuracy when δ (and hence θ) is small. A simple way around this is to Taylor expand the sine to write

$$f(\theta) = \delta - \frac{\theta^2}{3!} + \frac{\theta^4}{5!} - \frac{\theta^6}{7!} + \ldots$$

Now write $g(\psi) = f(\theta)$ where $\psi = \theta^2$. Then $g'(0) = 1/6$, and an inexact Newton iteration on g gives

$$\begin{aligned}\psi_{k+1} &= \psi_k + \frac{g(\psi_k)}{g'(0)} = 6\left(\delta + \frac{\psi_k^2}{5!} - \frac{\psi_k^3}{7!} + \ldots\right) \\ &= 6\delta + \frac{\psi_k^2}{4\cdot 5}\left(1 - \frac{\psi_k}{6\cdot 7}\left(1 - \frac{\psi_k}{8\cdot 9}\left(1 - \frac{\psi_k}{10\cdot 11}\left(\ldots\right)\right)\right)\right).\end{aligned}$$

From a starting guess of $\psi_0 = 6\delta$, this iteration converges ferociously quickly, with good relative accuracy at each step. We can solve the equation accurately because we evaluate the *residual* $f(\theta)$ accurately. This same idea will be key to our discussion of iterative refinement (next week).

LU factorization by example

Last time, we developed the idea of LU factorization through the following steps:

1. We walked through the elimination procedure with an augmented matrix that one would normally learn in an introductory class, and wrote MATLAB code for that procedure.

2. We rewrote the main loop body of our MATLAB code in terms of a single line corresponding to a rank-1 update of a trailing submatrix.

3. We described these rank-1 update steps in terms of a linear transformation (a Gauss transformation) described by a unit lower triangular matrix.

4. We claimed that the product of these transformations was a unit lower triangular matrix L^{-1}, so that $L^{-1}A = U$. We also claimed that the inverse of a unit lower triangular matrix is unit lower triangular, so that $A = LU$ is a factorization of A into a unit lower triangular matrix L and an upper triangular matrix U.

Let's start this lecture by working through these ideas with a concrete example:

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix}.$$

To eliminate the subdiagonal entries a_{21} and a_{31}, we subtract twice the first row from the second row, and thrice the second row from the third row:

$$A^{(1)} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix} - \begin{bmatrix} 0\cdot 1 & 0\cdot 4 & 0\cdot 7 \\ 2\cdot 1 & 2\cdot 4 & 2\cdot 7 \\ 3\cdot 1 & 3\cdot 4 & 3\cdot 7 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix}.$$

That is, the step comes from a rank-1 update to the matrix:

$$A^{(1)} = A - \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \end{bmatrix}.$$

Another way to think of this step is as a linear transformation $A^{(1)} = M_1 A$, where the rows of M_1 describe the multiples of rows of the original matrix that go into rows of the updated matrix:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} = I - \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} = I - \tau_1 e_1^T.$$

Similarly, in the second step of the algorithm, we subtract twice the second row from the third row:

$$\begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix} = \left(I - \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \right) A^{(1)}.$$

More compactly: $U = (I - \tau_2 e_2^T) A^{(1)}$.

Putting everything together, we have computed

$$U = (I - \tau_2 e_2^T)(I - \tau_1 e_1^T) A.$$

Therefore,

$$A = (I - \tau_1 e_1^T)^{-1} (I - \tau_2 e_2^T)^{-1} U = LU.$$

Now, note that

$$(I - \tau_1 e_1^T)(I + \tau_1 e_1^T) = I - \tau_1 e_1^T + \tau_1 e_1^T - \tau_1 e_1^T \tau_1 e_1^T = I,$$

since $e_1^T \tau_1$ (the first entry of τ_1) is zero. Therefore,

$$(I - \tau_1 e_1^T)^{-1} = (I + \tau_1 e_1^T)$$

Similarly,

$$(I - \tau_2 e_2^T)^{-1} = (I + \tau_2 e_2^T)$$

Thus,

$$L = (I + \tau_1 e_1^T)(I + \tau_2 e_2^T).$$

Now, note that because τ_2 is only nonzero in the third element, $e_1^T \tau_2 = 0$; thus,

$$\begin{aligned} L &= (I + \tau_1 e_1^T)(I + \tau_2 e_2^T) \\ &= (I + \tau_1 e_1^T + \tau_2 e_2^T + \tau_1 (e_1^T \tau_2) e_2^T \\ &= I + \tau_1 e_1^T + \tau_2 e_2^T \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix}. \end{aligned}$$

The final factorization is

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix} = LU.$$

Note that the subdiagonal elements of L are easy to read off: for $j > i$, l_{ij} is the multiple of row j that we subtract from row i during elimination. This means that it is easy to read off the subdiagonal entries of L during the elimination process. On one more (related) implementation note, observe that we only really need to *explicitly* represent l_{ij} for $j > i$ — and we only need to explicitly represent u_{ij} for $j \leq i$. Thus, we can re-use the storage space for A to store L and U.

Schur complements

The idea of expressing a step of Gaussian elimination as a low-rank submatrix update turns out to be sufficiently useful that we give it a name. At any given step of Gaussian elimination, the trailing submatrix is called a *Schur complement*. We investigate the structure of the Schur complements by looking at an LU factorization in block 2-by-2 form:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{22}U_{22} + L_{21}U_{12} \end{bmatrix}.$$

We can compute L_{11} and U_{11} as LU factors of the leading sub-block A_{11}, and

$$\begin{aligned} U_{12} &= L_{11}^{-1} A_{12} \\ L_{21} &= A_{21} U_{11}^{-1}. \end{aligned}$$

What about L_{22} and U_{22}? We have

$$\begin{aligned} L_{22}U_{22} &= A_{22} - L_{21}U_{12} \\ &= A_{22} - A_{21}U_{11}^{-1}L_{11}^{-1}A_{12} \\ &= A_{22} - A_{21}A_{22}^{-1}A_{12}. \end{aligned}$$

This matrix $S = A_{22} - A_{21}A_{22}^{-1}A_{12}$ is the block analogue of the rank-1 update computed in the first step of the standard Gaussian elimination algorithm.

For our purposes, the idea of a Schur complement is important because it will allow us to write blocked variants of Gaussian elimination — an idea we will take up in more detail next time.