Intro to Numerical Linear Algebra for AMSC 460
Lecture 2

Prof. Jacob Bedrossian
University of Maryland, College Park

These notes will supplement the lectures on numerical linear algebra. NLA is one of the most important, if not the most important, branches of numerical analysis. For many scientific computing calculations, most of the compute time is spent doing linear algebra, and so doing NLA effectively is crucial to useful code. Probably the best reference I've ever seen to introduce the topic of NLA is the book Trefethen and Bau, but its too advanced to follow directly in AMSC 460. These notes are a distillation of Bindle+Goodman's notes on scientific computing and Trefethen and Bau's book.

1 Gaussian elimination and the LU decomposition

It would be very convenient if we could write every matrix $A \in \mathbb{R}^{n\times n}$ as the product of a lower triangular matrix and an upper triangular matrix. That is, if we could write

$$A = LU, \tag{1}$$

where, for example,

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \ell_{n1} & \ell_{n2} & \cdots & \cdots & 1 \end{pmatrix},$$

$$U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}.$$

This would be great because then we could very easily solve

$$Ax = b \tag{2}$$

for x. That is, we first use forward substitution to solve $Ay = b$. and then use backwards substitution to solve $Ux = y$. The total cost for this solve in flops would be $Cost(n) = 2n^2 + O(n)$, which is extremely fast for a linear system solve, as it is essentially the time it would take to compute Ax given A and given x.

An identity like (1) is called a *matrix factorization* – a factorization of a matrix into two or more other matrices. There is a classical algorithm you have probably already learned in your introduction to linear algebra course precisely for taking a given matrix A and reducing it to an upper triangular matrix – *Gaussian elimination.*

Let us carry this out on an explicit example (this example is from Bindle and Goodman's book) of trying to solve the following linear system:

$$\begin{pmatrix} 4 & 4 & 2 \\ 4 & 5 & 3 \\ 2 & 3 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{3}$$

We will apply row operations to zero out the entries a_{21} and a_{31} of the matrix by subtracting the first row from the second and subtracting half the first row from the third row. That is:

$$\begin{pmatrix} 4 & 4 & 2 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 - b_1 \\ b_3 - \frac{1}{2}b_1 \end{pmatrix} \tag{4}$$

Notice that this row operation can be encoded into a lower triangular matrix:

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}$$

then what we did was apply L_1 to both sides of (3):

$$L_1 Ax = L_1 b. \tag{5}$$

Next, we eliminate the 1 in the third row, second column from the matrix $L_1 A$ by subtracting the second row from the third, which is the matrix on the left-hand side of (4):

$$\begin{pmatrix} 4 & 4 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 - b_1 \\ b_3 + \frac{1}{2}b_1 - b_2 \end{pmatrix} \tag{6}$$

The resulting linear system can be solved now using back substitution for x This last step was again multiplication on both sides by a lower triangular matrix L_2:

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}$$

and so (6) is

$$Ux = L_2 L_1 Ax = L_2 L_1 b, \tag{7}$$

where U is the upper triangular matrix

$$U = \begin{pmatrix} 4 & 4 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Let us now mention a few lemmas.

Lemma 1. *If C and D are both lower triangular, then so is CD.*

Proof. Follows from the formula for matrix-matrix multiplication. □

Lemma 2. *If C is an invertible, lower triangular matrix, then so is C^{-1}.*

Proof. Recall that C^{-1} is the matrix such that $x = C^{-1}b$ is the solution to $Cx = b$. From the forward substitution algorithm we have

$$x_1 = \frac{1}{c_{11}} b_1 \tag{8a}$$

$$x_2 = \frac{1}{c_{22}} \left(b_2 - c_{21} x_1 \right) \tag{8b}$$

$$\vdots\!\vdots \tag{8c}$$

$$x_j = \frac{1}{c_{jj}} \left(b_j - \sum_{i=1}^{j-1} c_{ji} x_i \right), \tag{8d}$$

if we write

$$C = \begin{pmatrix} c_{11} & 0 & 0 & \cdots & 0 \\ c_{21} & c_{22} & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ c_{n1} & c_{n2} & \cdots & \cdots & c_{nn} \end{pmatrix}.$$

From the formula (8), you can see that x_j depends only on b_i for $i \leq j$ (and C of course). Hence, if we can represent $x = C^{-1}b$ for some matrix C^{-1} (which we can), the matrix C^{-1} must be lower triangular. □

Hence, our Gaussian elimination strategy in (7) succeeded, theoretically speaking, in factorizing A as follows for $L = L_1^{-1} L_2^{-1}$:

$$LU = A.$$

Now the above example does not really clarify how to factorize an arbitrary matrix in any reasonable sense. Figuring out ways to implement it is a little different. Let us work backwards. Suppose we have matrices L and U with

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \ell_{n1} & \ell_{n2} & \cdots & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}. \tag{9}$$

By the matrix multiplication formula, we know we need to have

$$u_{1j} = a_{1j} \tag{10}$$

$$\ell_{i1} u_{11} = a_{i1}, \tag{11}$$

for all $1 \leq i, j \leq n$. Hence, we have a simple formula for the first row of U and the first column of L.

The key insight in how to go further is remembering that during Gaussian elimination, once you have eliminated the sub-diagonal elements in the first column, you don't think about that column again, and you simply run the exact same algorithm on the next column. That is, step one of Gaussian elimination step is just to do:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \mapsto \begin{pmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ 0 & * & \cdots & * \\ 0 & * & \cdots & * \\ 0 & * & \cdots & * \end{pmatrix}. \tag{12}$$

The $*$'s denote non-zero elements – in particular, note that *they are no longer the original elements of A*. However, the next thing we do is just to apply the *exact same elimination algorithm* to that matrix of $*$'s. This gives us the following algorithm.

Algorithm 1 (Beginner's *LU* factorization)**.** I will use #'s to denote inline comments. This is pseudo-code, not quite actual computer code.

```
L = I, U = A
for k=1 to n-1:                      #loop over columns (note last column requires no work)
    for j=k+1 to n:                  #loop over sub-diagonal rows
        L[j,k] = U[j,k]/U[k,k]
        for i=k to n:                #loop over columns to the left of k
            U[j,i] = U[j,i] - L[j,k]*U[k,i]
```

Counting the flops (approximately) in Algorithm 1 is good practice. Each for loop becomes a summation. Translating directly from the pseudo-code gives:

$$\begin{aligned} Cost(n) &= \sum_{k=1}^{n-1} \sum_{j=k+1}^{n} \left(1 + \sum_{i=k}^{n} 2\right) \\ &= \sum_{k=1}^{n-1} \sum_{j=k+1}^{n} (1 + 2(n-k)) \\ &= \sum_{k=1}^{n-1} (n-k-1) + 2(n-k)(n-k-1). \quad = \sum_{k=1}^{n-1} (n-k-1) + 2n^2 + 2k(k-1) - 4n(k-1) \end{aligned}$$

Now recall that $\sum_{k=1}^{m} k^p = \frac{1}{p+1} m^{p+1} + O(m^p)$ and hence,

$$\sum_{k=1}^{n-1} (n-k-1) = (n-1)^2 - \frac{1}{2}k^2 + O(k)$$

$$\begin{aligned} \sum_{k=1}^{n-1} 2n^2 + 2k(k-1) - 4n(k-1) &= 2n^2(n-1) + \frac{2}{3}n^3 - 2n(n-1)^2 + O(n^2) \\ &= \frac{2}{3}n^3 + O(n^2). \end{aligned}$$

Hence, the cost of doing the *LU* factorization is about $\approx \frac{2}{3}n^3$. This is much more costly than a single matrix-vector multiply, its about on par with multiplying two arbitrary $n \times n$ matrices. This speed of $O(n^3)$ cannot really be beaten when it comes to solving the linear system $Ax = b$ unless A has special properties (which thankfully, in real life, it often does!).

Unfortunately, Algorithm 1 is *terrible* and *useless*, as we will see. Next time, we will discuss why Algorithm 1 is useless, and discuss how we can fix it to make it useful...