Intro to Numerical Linear Algebra for AMSC 460
Lecture 1

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.

You all should have taken linear algebra before, but let's just remind ourselves of the basics. A matrix $A \in \mathbb{R}^{n\times n}$ applied to a vector $x \in \mathbb{R}^n$ is realized by the formula:

$$Ax = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, \tag{1}$$

$$b_i = \sum_{j=1}^{n} a_{ij} x_j. \tag{2}$$

This formula just says that the entry b_i is the i-th row of the matrix A dotted with x – recall, if $x, y \in \mathbb{R}^n$ are two vectors, the dot product is

$$x \cdot y = \sum_{i=1}^{n} x_i y_i. \tag{3}$$

One can (and should) view matrix multiplication (by a square matrix) as a geometric operation which maps vectors in $\mathbb{R}^n$ to other vectors in $\mathbb{R}^n$.

Matrix-matrix multiplication is given by:

$$AB = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{pmatrix} = \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{pmatrix}, \tag{4}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}. \tag{5}$$

Contemplation will reveal that the formula for AB is the same as saying that each column of AB is just A applied to the corresponding column of B. Recall the very important fact that in general: $AB \neq BA$, that is, matrices do not commute. This is not weird: most things in life do not commute (I bet you have never tried eating an egg sandwich before frying the egg) – the fact that numbers do commute is much more unusual.

1 $O(n)$ and counting floating point operators

When assessing the work required to execute an algorithm, a very common measure is counting floating point operations (abbreviated as 'flop'). These are the addition, subtraction, multiplications, and divides. For example, the assignment:

$$x = a * b + 12$$

is 2 flops: one to multiply and one to add.

We normally are not interested in counting the *exact* number of flops. Any individual flop takes an irrelevantly small amount of time on any modern computer. For linear algebra, this means we generally only care about roughly how the number of flops depends on the dimension n of the matrices and vectors we're working with. For those who have not seen it before, let us recall the following notion of $O(n^p)$ notation.

Definition 1. For a function $f(n)$ and a non-negative function $g(n)$, we say $f(n) = O(g(n))$ as $n \to \infty$ (pronounced "f is Big-Oh of g") if there exists a constant $C > 0$ such that $|f(n)| \leq Cg(n)$ for all n sufficiently large.

Example 1. If $f(n) = 2n^2 - 3n$, we would have $f(n) = O(n^2)$. We would also have $f(n) = n^2 + O(n)$, which more formally means $f(n) - n^2 = O(n)$. The latter is a more specific characterization of how $f(n)$ grows in n.

Example 2. Let us estimate the number of flops required to do a matrix-vector multiplication. Consider this formula: $b_i = \sum_{j=1}^{n} a_{ij}x_j = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n$. We have n multiplications and $n-1$ additions, hence, the cost of computing one entry of b is $2n-1$ flops. Since there are n rows, that brings the total cost to

$$Cost(n) = (2n-1)n = 2n^2 - n. \tag{6}$$

Hence, $Cost(n) = 2n^2 + O(n)$ and $Cost(n) = O(n^2)$.

Example 3. Since there are n columns in the matrix $B \in \mathbb{R}^{n\times n}$, we have that the cost of doing the matrix-matrix multiply AB is $Cost(n) = 2n^3 - n^2 = 2n^3 + O(n^2)$.

2 Linear systems

Let's just remind ourselves of a little theory.

Definition 2. A matrix $A \in \mathbb{R}^{n\times n}$ is called *invertible* if for every $b \in \mathbb{R}^n$ there is a unique $x \in \mathbb{R}^n$ such that $Ax = b$. If A is invertible then there is a matrix $A^{-1} \in \mathbb{R}^{n\times n}$ such that we can always solve for x via $x = A^{-1}b$.

Recall the theorem from linear algebra:

Theorem 1. *Let $A \in \mathbb{R}^{n\times n}$. The following are equivalent:*

1. *The matrix A is invertible.*
2. *The range of A is the whole space: $Range(A) = \{y \in \mathbb{R}^n : \exists\, x \text{ such that } Ax = y\} = \mathbb{R}^n$;*
3. *The nullspace of A is trivial: $Null(A) = \{x \in \mathbb{R}^n : Ax = 0\} = \{0\}$;*
4. *The determinant of A is zero: $det(A) = 0$.*

Do not worry if you don't remember how to compute a determinant, I think there is only one case we need to remember (see below).

Definition 3. A matrix A is called *lower triangular* if all entries of the matrix above the diagonal vanish:

$$A = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ \vdots & & & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \tag{7}$$

A matrix A is called *upper triangular* if all entries of the matrix below the diagonal vanish:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & & & & \vdots \\ 0 & \cdots & 0 & \cdots & a_{nn} \end{pmatrix} \tag{8}$$

It is easy for us to see when triangular matrices are invertible:

Theorem 2. *Let A be triangular (either kind). Then $det(A) = a_{11}a_{22}\cdots a_{nn}$. Hence, a triangular matrix is invertible if and only if $a_{ii} \neq 0$ for all $1 \leq i \leq n$.*

Assuming that A is indeed invertible, it is easy for us to solve the problem $Ax = b$ for x (given b and A) if A is triangular. Indeed, if A is lower triangular, then the linear system $Ax = b$ is just the set of equations

$$a_{11}x_1 = b_1 \tag{9}$$
$$a_{21}x_1 + a_{22}x_2 = b_2 \tag{10}$$
$$\vdots\vdots \tag{11}$$
$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \tag{12}$$

Hence, we first solve for x_1:

$$x_1 = \frac{1}{a_{11}} b_1, \tag{13}$$

which then allows us to solve for x_2:

$$x_2 = \frac{1}{a_{22}} \left(b_2 - a_{21}x_1 \right), \tag{14}$$

and so forth. The algorithm is then: for all j, $1 \leq j \leq n$:

$$x_j = \frac{1}{a_{jj}} \left(b_j - \sum_{i=1}^{j-1} a_{ji}x_i \right). \tag{15}$$

Note that the algorithm never fails as long as the diagonal entries are non-zero, which is the same as A being invertible. Let us now calculate the number of flops. The inner calculation $b_j - \sum_{i=1}^{j-1} a_{ji}x_i$ costs $2(j-1)$ flops: $j-1$ multiplications and $j-1$ subtractions. Hence, to calculate an entry of x_j we need to do $2(j-1)+1$ flops (one more for the divide). This is done for each entry, and hence we have

$$Cost(n) = \sum_{j=1}^{n} 2(j-1) + 1. \tag{16}$$

Since we know the formula $\sum_{j=1}^{n} j = \frac{1}{2}n(n+1)$, then we can compute exactly:

$$Cost(n) = n(n+1) + 2n = n^2 + 3n = n^2 + O(n). \tag{17}$$

More generally, we have the (more useful for our purposes) estimate for $p \geq 1$:

$$\sum_{j=1}^{n} j^p = \frac{n^{p+1}}{p+1} + O(n^p). \tag{18}$$

Hence, the cost of computing x is essentially the same as doing a *single* matrix-vector multiply of the lower triangular matrix (since half the entries are zero, if A is lower triangular, then the cost of computing Ax will be $n^2 + O(n)$ flops, rather than $2n^2 + O(n)$ flops). This is very efficient!