Unit 9: Numerical linear algebra

October 26, 2018

References:

- Gentle: Numerical Linear Algebra for Applications in Statistics (my notes here are based primarily on this source) [Gentle-NLA]
  - Unfortunately, this is not in the UCB library system - I have a copy that you can take a look at.
- Gentle: Computational Statistics [Gentle-CS]
- Lange: Numerical Analysis for Statisticians
- Monahan: Numerical Methods of Statistics

In working through how to compute something or understanding an algorithm, it can be very helpful to depict the matrices and vectors graphically. We'll see this on the board in class.

1 Preliminaries

1.1 Context

Many statistical and machine learning methods involve linear algebra of some sort - at the very least matrix multiplication and very often some sort of matrix decomposition to fit models and do analysis: linear regression, various more sophisticated forms of regression, deep neural networks, principle components analysis (PCA) and the wide varieties of generalizations and variations on PCA, etc., etc.

1.2 Goals

Here's what I'd like you to get out of this unit:

1. How to think about the computational order (number of computations involved) of a problem
2. How to choose a computational approach to a given linear algebra calculation you need to do.
3. An understanding of how issues with computer numbers (Unit 6) affect linear algebra calculations.

1.3 Key principle

The form of a mathematical expression and how it should be evaluated on a computer may be very different. Better computational approaches can increase speed and improve the numerical properties of the calculation.

Example 1 (already seen in Unit 4): If X and Y are matrices and z is a vector, we should compute $X(Yz)$ rather than $(XY)z$; the former is much more computationally efficient.

Example 2: We do not compute $(X^\top X)^{-1}X^\top Y$ by computing $X^\top X$ and finding its inverse. In fact, perhaps more surprisingly, we may never actually form $X^\top X$ in some implementations.

Example 3: Suppose I have a matrix A, and I want to permute (switch) two rows. I can do this with a permutation matrix, P, which is mostly zeroes. On a computer, in general I wouldn't need to even change the values of A in memory in some cases (e.g., if I were to calculate PAB). Why not?

1.4 Computational complexity

We can assess the computational complexity of a linear algebra calculation by counting the number multiplys/divides and the number of adds/subtracts. Sidenote: addition is a bit faster than multiplication, so some algorithms attempt to trade multiplication for addition.

In general we do not try to count the actual number of calculations, but just their order, though in some cases in this unit we'll actually get a more exact count. In general, we denote this as $O(f(n))$ which means that the number of calculations approaches $cf(n)$ as $n \to \infty$ (i.e., we know the calculation is approximately proportional to $f(n)$). Consider matrix multiplication, AB, with matrices of size $a \times b$ and $b \times c$. Each column of the second matrix is multiplied by all the rows of the first. For any given inner product of a row by a column, we have b multiplies. We repeat these operations for each column and then for each row, so we have abc multiplies so $O(abc)$ operations. We could count the additions as well, but there's usually an addition for each multiply, so we can

usually just count the multiplys and then say there are such and such {multiply and add}s. This is Monahan's approach, but you may see other counting approaches where one counts the multiplys and the adds separately.

For two symmetric, $n \times n$ matrices, this is $O(n^3)$. Similarly, matrix factorization (e.g., the Cholesky decomposition) is $O(n^3)$ unless the matrix has special structure, such as being sparse. As matrices get large, the speed of calculations decreases drastically because of the scaling as n^3 and memory use increases drastically. In terms of memory use, to hold the result of the multiply indicated above, we need to hold $ab + bc + ac$ total elements, which for symmetric matrices sums to $3n^2$. So for a matrix with $n = 10000$, we have $3 \cdot 10000^2 \cdot 8/1e9 = 2.4$Gb.

When we have $O(n^q)$ this is known as polynomial time. Much worse is $O(b^n)$ (exponential time), while much better is $O(\log n)$ (log time). Computer scientists talk about NP-complete problems; these are essentially problems for which there is not a polynomial time algorithm - it turns out all such problems can be rewritten such that they are equivalent to one another.

In real calculations, it's possible to have the actual time ordering of two approaches differ from what the order approximations tell us. For example, something that involves n^2 operations may be faster than one that involves $1000(n \log n + n)$ even though the former is $O(n^2)$ and the latter $O(n \log n)$. The problem is that the constant, $c = 1000$, can matter (depending on how big n is), as can the extra calculations from the lower order term(s), in this case $1000n$.

A note on terminology: ***flops*** stands for both floating point operations (the number of operations required) and floating point operations per second, the speed of calculation.

1.5 Notation and dimensions

I'll try to use capital letters for matrices, A, and lower-case for vectors, x. Then x_i is the ith element of x, A_{ij} is the ith row, jth column element, and $A_{\cdot j}$ is the jth column and $A_{i\cdot}$ the ith row. By default, we'll consider a vector, x, to be a one-column matrix, and $x^\top$ to be a one-row matrix. Some of the textbook resources also use a_{ij} for A_{ij} and a_j for the jth column.

Throughout, we'll need to be careful that the matrices involved in an operation are conformable: for $A + B$ both matrices need to be of the same dimension, while for AB the number of columns of A must match the number of rows of B. Note that this allows for B to be a column vector, with only one column, Ab. Just checking dimensions is a good way to catch many errors. Example: is $\text{Cov}(Ax) = A\text{Cov}(x)A^\top$ or $\text{Cov}(Ax) = A^\top\text{Cov}(x)A$? Well, if A is $m \times n$, it must be the former, as the latter is not conformable.

The **inner product** of two vectors is $\sum_i x_i y_i = x^\top y \equiv \langle x, y \rangle \equiv x \cdot y$.

The **outer product** is $xy^\top$, which comes from all pairwise products of the elements.

When the indices of summation should be obvious, I'll sometimes leave them implicit. Ask me

if it's not clear.

1.6 Norms

$\|x\|_p = (\sum_i |x_i|^p)^{1/p}$ and the standard (Euclidean) norm is $\|x\|_2 = \sqrt{\sum x_i^2} = \sqrt{x^\top x}$, just the length of the vector in Euclidean space, which we'll refer to as $\|x\|$, unless noted otherwise. One commonly used norm for a matrix is the Frobenius norm, $\|A\|_F = (\sum_{i,j} a_{ij}^2)^{1/2}$.

In this Unit, we'll make use of the **induced matrix norm**, which is defined relative to a corresponding vector norm, $\|\cdot\|$, as:

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

So we have

$$\|A\|_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sup_{\|x\|_2 = 1} \|Ax\|_2$$

A property of any legitimate matrix norm (including the induced norm) is that $\|AB\| \leq \|A\|\|B\|$. Recall that norms must obey the triangle inequality, $\|A + B\| \leq \|A\| + \|B\|$.

A normalized vector is one with "length", i.e., Euclidean norm, of one. We can easily normalize a vector: $\tilde{x} = x/\|x\|$

The angle between two vectors is

$$\theta = \cos^{-1}\left(\frac{\langle x, y\rangle}{\sqrt{\langle x, x\rangle\langle y, y\rangle}}\right)$$

1.7 Orthogonality

Two vectors are orthogonal if $x^\top y = 0$, in which case we say $x \perp y$. An **orthogonal matrix** is a matrix in which all of the columns are orthogonal to each other and normalized. Orthogonal matrices can be shown to have full rank. Furthermore if A is orthogonal, $A^\top A = I$, so $A^{-1} = A^\top$. Given all this, the determinant of orthogonal A is either 1 or -1. Finally the product of two orthogonal matrices, A and B, is also orthogonal since $(AB)^\top AB = B^\top A^\top AB = B^\top B = I$.

Permutations Sometimes we make use of matrices that permute two rows (or two columns) of another matrix when multiplied. Such a matrix is known as an elementary permutation matrix and is an orthogonal matrix with a determinant of -1. You can multiply such matrices to get more general permutation matrices that are also orthogonal. If you premultiply by P, you permute rows, and if you postmultiply by P you permute columns. Note that on a computer, you wouldn't need to actually do the multiply (and if you did, you should use a sparse matrix routine), but rather one can

often just rework index values that indicate where relevant pieces of the matrix are stored (more in the next section).

1.8 Some vector and matrix properties

$AB \neq BA$ but $A + B = B + A$ and $A(BC) = (AB)C$.

In R, recall the syntax is

```
A + B
A %*% B # matrix multiplication
A * B # Hadamard (direct) product
```

You don't need the spaces, but they're nice for code readability.

1.9 Trace and determinant of square matrices

The trace of a matrix is the sum of the diagonal elements. For square matrices, $\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B)$, $\text{tr}(A) = \text{tr}(A^\top)$.

We also have $\text{tr}(ABC) = \text{tr}(CAB) = \text{tr}(BCA)$ - basically you can move a matrix from the beginning to the end or end to beginning, provided they are conformable for this operation. This is helpful for a couple reasons:

1. We can find the ordering that reduces computation the most if the individual matrices are not square.

2. $x^\top Ax = \text{tr}(x^\top Ax)$ since the quadratic form, $x^\top Ax$, is a scalar, and this is equal to $\text{tr}(xx^\top A)$ where $xx^\top A$ is a matrix. It can be helpful to be able to go back and forth between a scalar and a trace in some statistical calculations.

For square matrices, the determinant exists and we have $|AB| = |A||B|$ and therefore, $|A^{-1}| = 1/|A|$ since $|I| = |AA^{-1}| = 1$. Also $|A| = |A^\top|$, which can be seen using the QR decomposition for A and understanding properties of determinants of triangular matrices (in this case R) and orthogonal matrices (in this case Q).

For square, invertible matrices, we have that $(A^{-1})^\top = (A^\top)^{-1}$. Why? Since we have $(AB)^\top = B^\top A^\top$, we have:

$$A^\top (A^{-1})^\top = (A^{-1}A)^\top = I$$

so $(A^\top)^{-1} = (A^{-1})^\top$.

Other matrix multiplications The Hadamard or direct product is simply multiplication of the correspoding elements of two matrices by each other. In R this is simply `A * B`.
Challenge: How can I find tr(AB) without using `A %*% B` ?

The Kronecker product is the product of each element of one matrix with the entire other matrix"

$$A \otimes B = \begin{pmatrix} A_{11}B & \cdots & A_{1m}B \\ \vdots & \ddots & \vdots \\ A_{n1}B & \cdots & A_{nm}B \end{pmatrix}$$

The inverse of a Kronecker product is the Kronecker product of the inverses,

$$B^{-1} \otimes A^{-1}$$

which is obviously quite a bit faster because the inverse (i.e., solving a system of equations) in this special case is $O(n^3 + m^3)$ rather than the naive approach being $O((nm)^3)$.

1.10 Matrix decompositions

A matrix decomposition is a re-expression of a matrix, A, in terms of a product of two or three other, simpler matrices, where the decomposition shows structure or relationships present in the original matrix, A. The "simpler" matrices may be simpler in various ways, including

- having fewer rows or columns;
- being diagonal, triangular or sparse in some way,
- being orthogonal matrices.

In addition, once you have a decomposition, computation is generally easier, because of the special structure of the simpler matrices.

We'll see this in great detail in Section 3.

2 Statistical interpretations of matrix invertibility, rank, etc.

2.1 Linear independence, rank, and basis vectors

A set of vectors, $v_1, \ldots v_n$, is linearly independent (LIN) when none of the vectors can be represented as a linear combination, $\sum c_i v_i$, of the others for scalars, $c_1, \ldots, c_n$. If we have vectors of