Linear Algebra

This section is to do with the numerical analysis of matrices. Given some matrix A, there are various things we might want to do. The most common are: *(a)* solve a system of equations, $A\boldsymbol{x} = \boldsymbol{b}$; *(b)* find the inverse, A^{-1}, of A; *(c)* find the determinant, det A, of A; and *(d)* find some or all of the eigenvalues and/or eigenvectors of A, that is the numbers λ and non-zero vectors $\boldsymbol{x}$ such that $A\boldsymbol{x} = \lambda\boldsymbol{x}$.

For obvious reasons, we will mostly use diddy examples where the matrices are, say, 3×3. In real life, matrices can be any size, and you will indeed find 3×3 matrices in use, but also 100×100 and even 10000×10000. [Examples of such applications later!]

For *big* matrices, the diddy things you may have learned in earlier modules often do not scale very well. The storage needed may be a problem [10000×10000 128-bit 'real' numbers is already 1.6 gigabytes, not these days a problem in itself, but you may need several of them, and copies of parts of them, and you want to be manipulating it all in RAM rather than on disc]. The time needed may be a problem [if you multiply two 10000×10000 matrices, then the result has 10^8 elements, each obtained by doing 10^4 multiplications and additions, which will take around 20 minutes on a 100GHz computer with 5Gb of RAM, and *much* longer on the (current) average home PC]. If you are doing so many operations, then rounding errors may be a problem, especially as many matrices, even small simple ones, are ill-behaved.

Gaussian elimination

This is the systematisation of the 'elimination' process used to solve simple simultaneous equations. For example, consider the equations

$$\begin{pmatrix} 1 & 1 & -1 \\ 2 & -1 & 3 \\ -1 & -2 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -3 \\ -7 \end{pmatrix}.$$

[Equivalently,

$$\begin{aligned} x+y-z &= 4 \\ 2x-y+3z &= -3 \\ -x-2y+2z &= -7\ ]. \end{aligned}$$

We use one equation, *eg* the first, to eliminate x from the others, giving a reduced set of equations in which there are fewer unknowns. Repeating, we get, eventually, down to [with any luck at all] one equation in one unknown, which we can solve and then work back. In this case, we get

$$-3y+5z = -11; \text{ and } -y+z = -3;$$

then, eliminating y,

$$2z = -2,$$

so that, successively, $z = -1$, $-3y-5 = -11$ so $y = 2$ and $x+2-(-1) = 4$ so $x = 1$.

How do we systematise that process? We use a *tableau*.

	1	1	−1	4	… (1)
	2	−1	3	−3	… (2)
	−1	−2	2	−7	… (3)
(2)−2×(1)	0	−3	5	−11	… (4)
(3)+(1)	0	−1	1	−3	… (5)
(4)−3×(5)	0	0	2	−2	… (6)
(6)÷ 2	0	0	1	−1	… (7)
(5)−(7)	0	−1	0	−2	… (8)
−(8)	0	1	0	2	… (9)
(1)−(9)+(7)	1	0	0	1	… (10)

Equations (7), (9) and (10) [the 'back substitution' phase] tell us that $z = -1$, $y = 2$ and $x = 1$, as previously.

Note that there is still a certain amount of freedom in deciding which equations to use to eliminate which variables. When working *exactly* and with small numbers of equations, the choice scarcely matters. But with large numbers of equations, the exact values soon become too large to hold, and so we have to use 'real' arithmetic, introducing rounding errors. When we derived equation (6), we multiplied (5) by 3; so any rounding errors in (5) would also have been multiplied by 3. Doing that *once* doesn't matter; do it 100 times, and the errors soon become serious.

So, in real life, Gaussian elimination is always performed with *pivoting*. The pivot is the element used to eliminate a variable. We have a choice of pivots; any non-zero element in the relevant column. If we choose as pivot the numerically largest element, then all the multipliers are numerically $\leqslant 1$, so the rounding errors stay small.

The other thing to note is that we use equations only to eliminate variables from those still 'active'. *Eg*, we did not use (5) to eliminate y from (1); there is no need, we can leave it until we know y, which saves a significant amount of work in a large tableau.

Matrix inversion

We can use a very similar tableau method to find A^{-1}. The trick is to start with a unit matrix on the right, and use the same sorts of 'row operations' to finish with a unit matrix on the left. For example:

1	1	–1	1	0	0	
2	–1	3	0	1	0	
–1	–2	2	0	0	1	
0	–3	5	–2	1	0	
0	–1	1	1	0	1	
0	0	2	–5	1	–3	
0	0	1	–5/2	1/2	–3/2	*
0	–1	0	7/2	–1/2	5/2	
0	1	0	–7/2	1/2	–5/2	*
1	0	0	2	0	1	*

So, from the lines (*), we can read off:

$$\begin{pmatrix} 1 & 1 & -1 \\ 2 & -1 & 3 \\ -1 & -2 & 2 \end{pmatrix}^{-1} = \begin{pmatrix} 2 & 0 & 1 \\ -\frac{7}{2} & \frac{1}{2} & -\frac{5}{2} \\ -\frac{5}{2} & \frac{1}{2} & -\frac{3}{2} \end{pmatrix}.$$

[For those who know other ways, for large matrices this is *much*, nay *incredibly*, more efficient than Cramer's Rule or thinggies involving determinants.]

Note that finding A^{-1} by this method is already equivalent to solving $A\boldsymbol{x} = \boldsymbol{b}$ for lots of $\boldsymbol{b}$'s, so solving $A\boldsymbol{x} = \boldsymbol{b}$ by finding $\boldsymbol{x} = A^{-1}\boldsymbol{b}$ is not helpful. Unless, that is, we are going to have to solve $A\boldsymbol{x} = \boldsymbol{b}$ for lots of $\boldsymbol{b}$'s and the same A.

Determinants

Row operations of the form

new row = old row − some multiple of another row

do not change the determinant of a matrix. So we can use the same 'tableau' method to get lots of zeros into the matrix before trying to work out its determinant. Specifically, if you get rid of all the elements below the leading diagonal, then the determinant is then just the product of all the elements down the diagonal.

Details left as an exercise!

Iteration

Although Gaussian elimination is the method of choice for many applications, and [with a few tweaks] it remains the most efficient general way of finding accurate solutions of matrix equations, it is a lot of work. [Left as an exercise: How many multiplications in general will be needed if A is 10000×10000? How long will this take on your PC?]

There are many cases in practice for which an approximate solution can be written down 'easily', and we can then hope that a simple iteration will enable us to improve the approximation. In particular, this will lead us to the Gauss–Jacobi and Gauss–Seidel methods.