Iterative solutions

If a matrix is *very* big, then finding its exact inverse, or using a tableau method to solve equations, takes a long time. It is not unusual for a major construct—a bridge, a car, the UK economy—to have several thousand components, each of which may have to satisfy several equations to be 'in equilibrium'; or, if you are dealing with a complicated problem in computational fluid dynamics, you may need to consider the conditions at every point in a 3D grid—*e.g.* the air flow round an aircraft may need to be measured every metre or so [more near the wings/engines] or about 100 000 grid points for a jumbo jet. With so many equations, you really need professional help; and even with that help, you may need to use a super-computer for several days. But there are a few cases where you can make progress in solving the equations without finding the inverse.

- Example: Consider the equations

$$\begin{aligned} 3x + y &= 5, \\ x - 4y &= 6. \end{aligned}$$

Solve the first equation for x in terms of y, and the second for y in terms of x:

$$\begin{aligned} x &= \tfrac{5}{3} - \tfrac{1}{3}y, \\ y &= -\tfrac{3}{2} + \tfrac{1}{4}x. \end{aligned}$$

Now assume any values we like for x and y, for example $x = y = 0$, and substitute in on the RHS to get new values:

$$x = \tfrac{5}{3}, \quad y = -\tfrac{3}{2}.$$

Substitute these values in:

$$x = \tfrac{5}{3} + \tfrac{1}{3}.\tfrac{3}{2} = 2\tfrac{1}{6}, \quad y = -\tfrac{3}{2} + \tfrac{1}{4}.\tfrac{5}{3} = -1\tfrac{1}{12}.$$

Substitute again to get $x = \frac{5}{3} + \frac{1}{3}.\frac{13}{12} = 2\frac{1}{36} \approx 2.0278$, $y = -\frac{3}{2} + \frac{1}{4}.\frac{13}{6} = -\frac{23}{24} \approx -0.9583$, and again to get $x \approx 1.986$, $y \approx -0.9931$, and then $x \approx 1.9977$, $y \approx -1.0035$, and so on. We see that the values are converging nicely to the actual solution $x = 2$, $y = -1$. This method is called the Gauss–Jacobi iterative method, or just Jacobi iteration.

Clearly this method is overkill for two equations in two unknowns. The advantage is that ***if*** *it is working at all* then we can expect it to work, say to 4 decimal places, in, say, 10 iterations. For, say, 10 000 equations, that is roughly the amount of work needed to eliminate 10 of the 10 000 variables, so we've saved the work [*i.e.* over 99% of the total!] needed to eliminate the remaining 9 990.

When does the method not work? Try, for example, swapping the above equations around, so that

$$x = 6 + 4y, \quad y = 5 - 3x.$$

Now if we start with $x = y = 0$, we find successively $x = 6, y = -3$, then $x = -6, y = -13$, then $x = -46, y = 13$, then $x = 58, y = 143, \ldots$, which is easy but not getting anywhere near to the solution. Even if we start with $x = 2.01, y = -1.01$, we get $x = 1.96, y = -1.03$, then $x = 1.88, y = -0.88$ then $x = 2.48, y = -0.64, \ldots$.

The problem is that any errors in x or y are being magnified by the process, whereas in the previous version they were being made smaller. You **must not** use Jacobi iteration *unless* the total of the absolute values of the coefficients on the RHS is less than 1. Sometimes, you *could* be lucky, and the errors could cancel out; but in the real world you won't be.

Equivalently, for Jacobi iteration to work, you need the coefficients down the main diagonal of the matrix to be larger than the sum of all the other coefficients in their rows, ignoring signs. Such a matrix is *diagonally dominant*.

- *Example*: Use an iterative method to solve the matrix equation

$$\begin{pmatrix} 10 & -1 & 1 \\ -1 & 20 & 3 \\ 1 & 3 & -30 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 12 \\ -8 \\ 15 \end{pmatrix}$$

correct to 3 places of decimals.

The 3×3 matrix is diagonally dominant as $10 > 1+1$, $20 > 1+3$ and $30 > 1+3$. The equation for x is $10x-y+z = 12$, or $x = 1.2+\frac{1}{10}y-\frac{1}{10}z$. Similarly, $-x+20y+3z = -8$, or $y = -0.4+\frac{1}{20}x-\frac{3}{20}z$; and $x+3y-30z = 15$, or $z = -0.5+\frac{1}{30}+\frac{1}{10}y$.

Starting from $x = y = z = 0$, the first iteration gives $x = 1.2$, $y = -0.4$, $z = -0.5$. Substituting these values in gives $x = 1.2-0.04+0.05 = 1.21$, $y = -0.4+0.06+0.075 = -0.265$ and $z = -0.5+0.04-0.04 = -0.5$. The next iteration gives, to 4dp, $x = 1.2235$, $y = -0.2645$ and $z = -0.4862$; and the next gives $x = 1.2222$, $y = -0.2659$ and $z = -0.4857$; and the next $x = 1.2220$, $y = -0.2660$ and $z = -0.4859$. So, to 3dp, $x = 1.222$, $y = -0.266$ and $z = -0.486$. [The previous values given were in fact correct to 4dp.]

Computationally, you may find it easier to substitute in the new values of x, y and z as soon as you have them instead of waiting for the next iteration. It is usually slightly more accurate, and it saves having to store the old values [if you are using a computer program]; otherwise, it makes very little difference. This modification to Jacobi iteration is usually called the Gauss–Seidel iterative method.

You might think that if you wrote down a 100×100 matrix, let alone $10\,000 \times 10\,000$, filled with more-or-less random elements, then it is *incredibly* unlikely that in each row one element should be so much larger than all the others in the same row that the matrix is diagonally dominant. And you would be right. But luckily many large matrices are far from random.

In many typical applications, the element m_{ij} of a square matrix M describes the 'interaction' between object i and object j. This could be, for example, the force exerted by component i on component j in some structure [such as a bridge or an aeroplane]; or the number of times web page i links to page j; or the length of the road between town i and town j; or the number of people who live in area i and work in area j; or the mass of isotope i that is produced when isotope j decays radioactively for one second; and so on. In such cases, *(a)* it is very common that most of the interactions are zero [most web pages do not link to any other specified page, there is no direct connexion between the left front wheel of your car and the rear window]; *(b)* and otherwise that many are very small [not many people live in Nottingham and work in Bristol]; and *(c)* in some applications, the 'self-influence' is likely to be the largest [most people living in Nottingham will work in Nottingham, though a reasonable number will commute to Derby or even London].

Property *(c)* is what tends to make a matrix diagonally dominant. It is much more likely to happen if *(a)* or *(b)* applies; this in turn is really a property of the system being studied, but it's very common physically, either because components are often connected to only a few other components, or because 'influence' is affected by distance. For numerical work, it's particularly important if *(a)* applies. This is because the resulting matrix may be very large but consist almost entirely of zero elements—a *sparse* matrix. Inside a computer program, you may be able to store only the non-zero elements, and thus to process a matrix that is nominally much larger than available computer storage would allow. There are special computer techniques for this; well beyond the scope of this module!