Ordinary Differential Equations

Boundary Value Problems

Dynamics problems are usually 'initial value' problems: we start the system off from *this* position with *that* velocity [often zero]. More static problems are often 'boundary value' problems: you know what is happening 'at the outside', and need to know what is happening on the inside.

BVPs can occur as part of the solution of partial differential equations [PDEs]; we look at these later. Here we look at them in their own right.

Example: Suppose

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = y,$$

given that $y = 1$ when $x = 0$ and when $x = 1$. [So *we* know that the general solution is

$$y = A\mathrm{e}^{x} + B\mathrm{e}^{-x},$$

so that the boundary conditions give

$$A + B = 1, \quad A\mathrm{e} + B/\mathrm{e} = 1,$$

which we can solve for A and B to find the particular solution, and so the value of y for any given value of x.]

As previously, we can re-write the equation as a pair of first-order equations:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = z; \quad \frac{\mathrm{d}z}{\mathrm{d}x} = y.$$

Now, if we had initial values for y and z, we could solve numerically, *e.g.* by using Runge–Kutta. But we only have the initial value for y. So we *guess* an initial value for z, say $z = 0$. Now we can solve. There are no new principles, and the exact solution is

$$y_1 = \tfrac{1}{2}e^x + \tfrac{1}{2}e^{-x};\ z_1 = \tfrac{1}{2}e^x - \tfrac{1}{2}e^{-x}.$$

Unsurprisingly, this gives us the wrong value for y when $x = 1$; in fact, $y_1(1) \approx 1.543$ instead of 1. So we try again with a different initial value of z. Any other value would do [say $z = 1$]; as our value of $y(1)$ is too large, we should probably try a smaller value of z, say $z = -1$. Again, there are no new principles, and the exact solution is [even simpler]

$$y_2 = e^{-x};\ z_2 = -e^{-x},$$

again giving the wrong value when $x = 1$: $y_2(1) \approx 0.368$.

In this case, the given ODE was linear, so we can interpolate linearly between these two solutions: the general solution is

$$y(x) = \alpha \times y_1(x) + (1-\alpha) \times y_2(x)$$

and we find α by taking $x = 1$:

$$1 \approx \alpha \times 1.543 + (1-\alpha) \times 0.368.$$

[Of course, in a real-life problem, we would find y_1 and y_2 entirely numerically, and build up tables of the values of y and z.]

What if the ODE is not linear? In this case, we cannot simply interpolate linearly between two solutions, we have to choose yet another initial value for z, solve, and hope to get closer to the right value for y when $x = 1$. In fact, effectively the value of $y(1)$ is a [rather difficult] function of the initial z, and we are trying to solve the equation [in this case] $y(1) = 1$ for that initial z. So we have, in general, to use the methods developed for solving equations.

Note that Newton–Raphson is not so easy, as we do not know the derivative of $y(1)$ with respect to the initial z. Note also that simply to work out $y(1)$ involves solving the ODE, so we do not want to do that too often. There is an efficient variation of NR called the 'secant' method, which essentially means replacing the derivative in NR by an estimate of it from the most recent two function values. But whatever you do, we have the problems discussed earlier in the module: 'safe' methods are inefficient [which really matters here], and efficient methods can sometimes blow up; compounded with that are all the usual potential problems of stability and stiffness when solving the ODE. BVPs are often rather nasty.

The methods just discussed are generically called 'shooting' methods. Baasically, given an initial value for y but not for $\frac{\mathrm{d}y}{\mathrm{d}x}$, we guess the initial $\frac{\mathrm{d}y}{\mathrm{d}x}$ and 'shoot' at the other boundary. If we 'overshoot' [our final value is too large], then we 'lower our aim', reduce $\frac{\mathrm{d}y}{\mathrm{d}x}$ and try again, and similarly if we 'undershoot', until eventually we 'hit the target'. It helps if, as is quite often the case in practical problems, we have a reasonably good idea of the initial aim—for example, in cases where the problem is a small perturbation from a previously solved case.

Until fairly modern times, if a BVP could not be solved analytically, shooting was the only practicable numeric solution. But now that we can solve large systems of linear equations on the computer, a different approach is possible.

Finite-difference methods for BVPs

The basic idea is to replace the derivatives in the ODE by differences. Suppose, for definiteness, we use a 'strip width' of 0.1 in our previous example. Then we have 11 x-values, $x_i = 0.1\times i$ for $0 \leqslant i \leqslant 10$, and correspondingly 11 y-values, of which y_0 and y_{10} are given by the boundary values, but the other y_i are unknown. So we need 9 equations in these 9 unknowns. But we can approximate:

$$y''(x_i) \approx (y'(x_{i+\frac{1}{2}}) - y'(x_{i+\frac{1}{2}}))/h \approx (y_{i+1} - 2y_i + y_{i-1})/h^2$$

[with similar results for other derivatives if we needed them], where h is the strip-width, here 0.1. So the ODE $y'' = f(x,y)$ becomes

$$y_{i+1} - 2y_i + y_{i-1} \approx h^2 f(x_i, y_i),$$

or, in our example, $y'' = y$,

$$y_{i+1} - 2y_i + y_{i-1} \approx 0.01 y_i,$$

for $i = 1, 2, \ldots, 9$. Just what we needed, 9 equations for the 9 unknowns.

> Note that numerical differentiation is normally 'difficult' because of the cancellation error, as discussed previously. But in this case we are not using the y_i to estimate derivatives of y; rather, the ODE tells us the derivatives, and we are using these to estimate the y_i. So this process works quite well in practice.

In the present case, as the equations are linear, we have a 9×9 matrix to invert to solve the equations exactly. More generally, the equations would be non-linear, but this will occur on the right-hand side only, and as these are all small because of the factor h^2, an iteration usually works well. That is, we guess the y_i, plug these into the RHS, use these to solve for [better] y_i, and keep going.

For our example, we can solve using Maple. We have to invert a 9×9 matrix, most of whose rows look like

$$\ldots \quad 0 \quad 0 \quad 1 \quad -2 \quad 1 \quad 0 \quad 0 \quad \ldots,$$

and multiply that by the vector of [guessed] right-hand-sides to produce the better values of y_i for the next iteration. The results are shown by the following table:

x_i	y_i [FD solution]	y_i [exact solution]
0.0	1.00000	1.00000
0.1	0.95875	0.95872
0.2	0.92708	0.92703
0.3	0.90469	0.90461
0.4	0.89134	0.89126
0.5	0.88690	0.88682
0.6	0.89134	0.89126
0.7	0.90469	0.90461
0.8	0.92708	0.92703
0.9	0.95875	0.95872
1.0	1.00000	1.00000

It doesn't always work this well!

If we need to improve the accuracy, we can *(a)* reduce the strip width, bearing in mind that [eg] 100×100 matrices can be inverted reasonably routinely, and *(b)* use better difference formulas for the derivatives [but this usually needs some special treatment at the boundaries, else the formula can use y-values outside the boundaries].

Note that in many physical problems, the ODE was derived from a 'finite-difference' approximation, *e.g.* considering heat flow between two x-values separated by a distance h and letting $h \to 0$. So there is often a sense in which this process for solving BVPs is closer to the original problem than the ODE is; which may be one reason why it often works so well.