Unit VII
Differential Equations II

Unit VII
Differential Equations II

In the last unit, we learned some of the basic principles and techniques of solving differential equations. These are useful for quick illustrative solutions to differential equations that are not too challenging, the sort of situation in which a ten minute effort with Excel to modest accuracy will suffice. However, more serious problems deserve more serious algorithms. In this unit, we will investigate two more serious algorithms, the first a more general utility method, and the second a method that is particularly applicable to one dimensional quantum mechanics problems.

Guidebook Entry VII.1: Runge Kutta Method

In general, when we solve a differential equation, we are updating a starting value of a function based on the derivative of the function. If the differential equation is of second order (which is the dominant case for physics examples), then we simply apply this updating twice in series. So, we lose little generality by looking at least initially at first order differential equations.

Let's consider a differential equation for position as a function of time that looks like this

$$x'(t) = f.$$

I have been careful not to specify what the function f depends on. It may depend on x, or t, or potentially even more complicated things if this is a second order equation. But let's ignore that for the time being, and imagine that f depends only on t. If that is true, we can find x(t) by directly integrating the function f. Write this solution formally below.

If this is an interesting case, then f probably cannot be integrated analytically. If this is true, then we will have to apply a numerical technique to the integration of f. Imagine that we are going to integrate a single small step of f, and write this as a single step of size h using a Simpson's rule integration. That is, write x(h) as an expression involving x(0), f(0), f(h/2) and f(h).

This is indeed a good method to use if f depends only on t. However, often f is related to a force law, and depends not on t alone, but also on the position x. Explain why this makes evaluation of the Simpson integration difficult.

Now let's consider f to be a function of both time and distance, which we will write as f(t,x). Use a simple Euler method and f_o to estimate x(h/2):

Write a formal expression (i.e. there is nothing to calculate since we don't know f) for $f_{h/2}$ using this estimate of x(h/2):

Use your expression for $f_{h/2}$ to make an estimate of x(h) (this is essentially a half-step Euler method extrapolation).

Now, combine all of these estimates of f into the Simpson formula you wrote on page 1 to give a single step of h in x(t):

Let's see how this works in practice. For simplicity, consider the first order differential equation

$$x'(t) = -x(t)$$

which might in fact be the velocity of an object subject to a drag force, where x is the velocity, or it might be radioactive decay where x represents the number of radioactive nuclei present. In any event, we expect a decreasing exponential as the solution. Choose an initial value $x(0) = 1$, and use Excel to integrate this function out to 5 or 10 in steps of $h = 0.1$. Compare your results to those of the simple Euler method, and the half-step Euler method (either using your old sheet, or making new entries for those methods).

Describe your results.

The formula you derived still has a small error term in third order, which can be corrected using the following algorithm:

$$f_0 = f(t_o, x_o)$$
$$f_1 = f(t_0 + \tfrac{h}{2}, x_0 + \tfrac{h}{2} f_0)$$
$$f_2 = f(t_0 + h, x_0 - hf_0 + 2hf_1)$$

and then

$$x(h) = x(0) + \tfrac{h}{6}(f_0 + 4f_1 + f_2)$$

which is truly correct through terms of the order h^3. This is known as the third order Runge-Kutta formula. A fourth-order algorithm involves only slightly more calculation, and is generally viewed to be the best compromise in balancing calculational effort with accuracy per step. The fourth-order formula is often simply referred to as "the" Runge-Kutta method, and looks like

$$f_0 = f(t_0, x_0)$$
$$f_1 = f(t_0 + \tfrac{h}{2}, x_0 + \tfrac{h}{2} f_0)$$
$$f_2 = f(t_0 + \tfrac{h}{2}, x_0 + \tfrac{h}{2} f_1)$$
$$f_3 = f(t_0 + h, x_0 + hf_2)$$

and then

$$x(h) = x(0) + \tfrac{h}{6}(f_0 + 2f_1 + 2f_2 + f_3).$$

Notice that both the third- and fourth-order Runge-Kutta methods reduce to Simpson's rule for differential equations where f is a function of time only.

The following method, which is described in several books; I first learned about it in William Thompson's book. Since it works *so* well for one dimensional quantum mechanics problems, it is one that every computational physicist should know about.

Guidebook Entry VII.2: Waves, Quantum Mechanics and Numerov's Algorithm

There is a particularly nice algorithm for equations of the form

$$y''(x) + K(x)y = S(x)$$

which is given implicitly by the formula

$$\left(1 + \frac{h^2}{12}K_1\right)y_1 - 2\left(1 - \frac{5h^2}{12}K_0\right)y_0 + \left(1 + \frac{h^2}{12}K_{-1}\right)y_{-1} = \frac{h^2}{12}\left(S_1 + 10S_0 + S_{-1}\right)$$

where this gives the step from y_o. The way this is written allows one to move either to smaller values of x, or larger values of x, that is, either to the left or the right. Notice that this requires two previous values to make a step, so one must use a simple Euler method to make that first step. Alternatively, if one knows a symmetry property of the function (e.g. it is a function like the cosine), one can choose y_{-1} and y_o to be symmetrically located about the origin and have equal values. The error term in this algorithm is of the order of h^5, or one power higher than the fourth order Runge-Kutta.

Let's imagine that we are applying this to the differential equation for the sine and cosine functions:

$$y'' = -y.$$

What are the values for K(x) and S(x)?

Rearrange the Numerov algorithm for application to this differential equation, and write the result below.

We will soon make a spreadsheet that contains this algorithm, but first we have to have a way to get initial values. Let's imagine that we are looking for a sine solution. What is the value of this function at x = 0?

What is the slope of this function at x = 0?

We could just make a simple Euler step to get our starting values. This would introduce a small phase shift into our result. We can avoid this by choosing two values symmetrically located about the origin. What are these values, if the x's are to be separated by 0.1? (This result gives no phase shift, but instead introduces a small amplitude error. This turns out to be unimportant for quantum examples.)

Apply the Numerov algorithm in your spreadsheet to this differential equation. You may wish to compare this to a real sine function. Look at ratios rather than differences to see if my claim of amplitude error is correct. Describe what you observe.

Finally, let's apply this to the quantum wave example of a finite square well. Here is a problem that can be solved analytically, although piecewise. The differential equation looks like

$$-\frac{\hbar^2}{2m}y''(x) + \left(V(x) - E\right)y = 0$$

where E is a constant and

$$V(x) = 10 \text{ for } |x| > 1 \text{ otherwise } V(x) = 0.$$

For convenience, you may choose

$$\frac{\hbar^2}{2m} = 1.$$

What are the terms S and K from the Numerov algorithm for this differential equation?

E is a constant that expresses the energy of the system. Put this in a cell in your spreadsheet so that we might easily vary it. Choose an even symmetry (equal values for your starting points located symmetrically about x = 0) for your wave function--y = 1 is just fine. Complete a spreadsheet that integrates this function from your starting points out to 4.

Those of you who have used taken the Modern Physics course probably recall a program called quantum well. You have just produced a spreadsheet that does the same thing. The game is to choose a value for E that makes the wave function asymptotically approach zero in the "forbidden" regions, i.e. for $x > 1$. How describe what happens as you vary E. How many even solutions can you find?

The numerical differential equation solver in Mathematica, which we used last week, uses a fancy version of a Runge-Kutta algorithm which is known as adaptive. Just as we learned about in the integration section, selection of step size can be a tricky procedure. Make the step size too large, and you can become terribly inaccurate where the function is wiggling around quickly (and the higher derivative terms are large). Be too cautious and make the step size overly small, and the stepping simply takes too much time. Mathematica is clever, and investigates the function and decides on its own what sort of step size to use, and then can vary that step size as it moves along.

Another peril in the differential equation solving business is that of stiff differential equations. These are second-order differential equations that can allow increasing exponential sorts of solutions that are in principle ruled out by initial conditions, but in fact creep in because of numerical errors. Mathematica claims to be tough on these as well as adaptive.

For an optional extra exercise, you may try the following to see if Mathematica can do better than Numerov on a case that will test its skill on the the former problem through its adaptive techniques.

Guidebook Entry VII.3: Life on the [Potential} Edge: a Numerical Contest (Optional)

Consider a quantum mechanical potential that is given by a linear function. For those of you who have taken Modern Physics, or seen a little quantum mechanics elsewhere, this means that the particle has increasingly positive kinetic energy on one side (meaning oscillatory wave function solutions) on one side, and progressively more and more negative kinetic energy on the other side (which means exponentially decreasing solutions).

Use simple particular values for your functions and initial conditions. In particular, use

$$V(x) = x \text{ , and } E = 1.$$

Start out your solution "even" at x = 0, and step solutions into the oscillatory region (negative x's). Compare the performance of Mathematica and your Numerov algorithm; you will have to make your best guess for a step size for Numerov.