3 Numerical Solutions of Differential Equations Fall 2003

References: Stewart ***Calculus - Early Transcendentals*** (4th edition), Sec. 9.2
Edwards and Penney ***Differential Equations*** (2nd edition), Sec. 2.4

Note: In this discussion we revert to the ordinary concept of a function rather than the more specialized notion discussed in Section 2.

Suppose y is a function of time t, $y = f(t)$, that satisfies the differential equation

$$\frac{dy}{dt} = F(y, t), \qquad \text{where } F(y,t) \text{ is a known function.}$$

For many differential equations the solution $y = f(t)$ cannot be expressed in terms of familiar functions. A simple example of such a differential equation is $\frac{dy}{dt} = y^2 + t$.
When $y = f(t)$ cannot be expressed in terms of familiar functions, we would like to develop a scheme for computing an *approximate* numerical value of y for any specified numerical value of t. We could then plot a graph of y as a function of t, and explore the properties of the function.

Euler's Method

To determine a unique solution, we need a starting point (the value of y at some initial value of t). Let the initial value of t be t_o, and let the value of y at this time be y_o. That is, $y_o = f(t_o)$, where the values y_o and t_o are specified at the beginning. Then we can get an approximate value for y at a slightly later time, say, $t_1 = t_o + \Delta t$, as follows:

During the time interval Δt, the *change* Δy in y is given approximately by

$$\Delta y \cong \left(\frac{dy}{dt}\right)_{t\,=\,t_o} \Delta t \ .$$

(This becomes exact in the limit as $\Delta t \to 0$.) Let y_1 be the value of y at time t_1; then

$$\Delta y \ = \ y_1 - y_o \ \cong \ \left(\frac{dy}{dt}\right)_{t\,=\,t_o} \Delta t \ .$$

Also, from the differential equation, $\left(\frac{dy}{dt}\right)_{t\,=\,t_o} = F(y_o, t_o)$, so

$$y_1 \cong y_o + F(y_o, t_o)\,\Delta t.$$

Because y_o and t_o are known, we can evaluate this and get an approximate value for y_1. In the above example, where $F(y, t) = y^2 + t$, we obtain

$$y_1 \cong y_o + (y_o{}^2 + t_o)\,\Delta t.$$

Using this approximate value for y_1, we can now repeat this whole process to get the approximate value y_2 at the slightly later time $t_2 = t_1 + \Delta t$:

$$y_2 \cong y_1 + F(y_1, t_1)\,\Delta t.$$

(We are assuming that the time intervals between successive points are all equal to Δt.)

In this way we construct a sequence of solution points (y_i, t_i). When we connect these points with line segments, the result should be some approximation of the curve representing the actual solution $y = f(t)$. Intuitition suggests that the smaller Δt is, the more precise our approximation will be.

We can implement this scheme using Maple. In the above example, let $y_o = 0$ and $t_o = 0$. It's customary in the literature to denote the step size Δt as h. Then $t_n = nh$. Suppose $h = 0.1$. If we take a total of ten steps (so n goes from 0 to 9), the maximum value of t is 1. For our example, with $F(y, t) = y^2 + t$, a possible Maple scheme is:

```
restart;
y[0] := 0;
for n from 0 to 9  do
y[n + 1] := y[n] + (y[n]^2 + n*0.1)*0.1;
end do;
```

(Note the use of square brackets, y[n], to denote the subscripted variable y_n.)

To make it easier to experiment with various values of h, and various ranges of values of t, you may want to use something like

```
restart;
h := 0.1;     nmax := 9;     y[0] := 0;
for n from  0  to nmax  do
y[n + 1] := y[n] + (y[n]^2  +  n*h)*h;
end do;
```

Or if you want to make a table of y's and the corresponding t's, you can end the fourth and fifth lines with colons and add the following line between them. Try it!

```
print([n*h, y[n]]);
```

This procedure is called *Euler's method*. Intuitively, the precision should improve if we decrease the step size h. In fact, it can be shown that for a given differential equation, the cumulative error in the approximation at any point is proportional to h. Hence this is called a *first-order* method; decreasing the step size h by a factor of 1/2 decreases the cumulative error by the same factor.

Improved Euler Method

There are various ways of refining this process to obtain better precision with only modest increase in computation. One of the simplest refinements is the following. Instead of computing y_{n+1} by evaluating $F(y, t)$ at the point $[y_n, t_n]$, we first use this to get an *estimate* of y_{n+1}, which we may call $y_{\text{est.}}$ We then use this to get an approximate value of $F(y, t)$ at time $t_n + h$, and then compute y_{n+1} by using the *average* of $F(y_n,t_n)$ and $F(y_{\text{est}}, t_{n+1})$. Geometrically, this amounts to computing the next point on the curve not by using the slope at the previous point, but by first estimating the slope at the next point, and then computing the next point by using the *average* slope in the interval.

More explicitly

$$y_{\text{est}} = y_n + F(y_n, t_n)\,h,$$

$$\left(\frac{dy}{dt}\right)_{\text{ave}} = \frac{1}{2}\big[F(y_n, t_n) + F(y_{\text{est}}, t_{n+1})\big],$$

and finally

$$y_{n+1} = y_n + \frac{1}{2}\big[F(y_n, t_n) + F(y_{\text{est}}, t_{n+1})\big]\,h.$$

Using the average slope of $y(t)$ in each interval instead of the slope at the beginning improves the precision of the method greatly. It can be shown that in this scheme the accumulated error in the computation, for any given equation, is proportional to h^2, and this is called a *second-order* method. Changing h by a factor of 1/2 changes the cumulative error by 1/4.

Runga-Cutta Method

The Runga-Cutta method is a further elaboration of the basic idea of the improved Euler method. It uses estimates of y and dy/dt at several points in each interval during the computation of y_{n+1} from y_n. For a given step size and range of values of t, it requires five to 10 times as many computations as the Euler method, but the cumulative error can be shown to be proportional to h^4. Decreasing h by 1/2 decreases the cumulative error by 1/16. This method is very widely used. One of the methods Maple uses is a version of the "fourth-order Runga-Cutta" method. Maple can also vary the step size, using larger steps in regions where the function is changing slowly, hence economizing on computations.

Higher Order Equations

All the methods described above can be adapted to numerical solution of equations containing higher-order derivatives. In mechanics, where Newton's second law contains a second derivative, we often encounter equations containing y, $\frac{dy}{dt}$, and $\frac{d^2y}{dt^2}$.

For such cases, let $v = \frac{dy}{dt}$; then $\frac{d^2y}{dt^2} = \frac{dv}{dt}$. This process converts the single second-order equation for y into two coupled first-order equations for y and v.

Similarly, suppose there are two variables, say x and y, and two coupled second-order equations. (A familiar example is a trajectory problem with air resistance, where x and y are the coordinates of the particle, both functions of t.) We can define

$$vx = \frac{dx}{dt} \qquad \text{and} \qquad vy = \frac{dy}{dt}.$$

This converts the pair of second-order equations into a set of four coupled first-order equations, which can be solved with an elaboration of any of the methods described above. Maple does this simply and painlessly.