Notes in Lieu of Class Lecture on Chapter I.7 on Tuesday, Feb 17, 2015

Why do we employ numerical methods to study first order ODEs? First of all, as we have seen, often we may not be able to find a formula solution of an ODE. Second, the graphical method that we have developed – that is, vector fields, only gives us a general notion of how "all" the solution curves of the ODE behave. However, we may be seeking the behavior of a specific solution having a certain initial value. Numerical methods enable us to give very accurate information on a specific solution to an IVP – but as we shall see, with an important caveat.

The simplest numerical method is the *Euler Method*. Given $y'=f(t, y)$, $y(t_0)=y_0$, that is, an IVP, we may not have a formula for the solution $y = \phi(t)$ that goes through the given point. But we do know its slope at the point (t_0, y_0) – namely, $\phi'(t_0) = f(t_0, \phi(t_0)) = f(t_0,y_0)$. For a small value $h > 0$, we can estimate the value of $\phi(t)$ at $t = t_1 = t_0 + h$, using the point-slope formula for the tangent line to $\phi(t)$ at (t_0,y_0) – namely

$$\phi(t_1) \approx y_1 = y_0 + hf(t_0,y_0).$$

Repeating this calculation as we step by h units to the right, we obtain a series of points that approximate the solution curve $y=\phi(t)$ at $t_n=t_0+nh$, that is

$$y_{n+1}=y_n+hf(t_n, y_n),\ n \geq 0.$$

Euler's Method is described in both DEwM, Chapter 8 and the NODE notes, I.7. You might try problems 3 & 4 in the latter reference to solidfy your understanding of the Euler Method.

Euler's Method is a coarse method that rarely results in estimates of accuracy greater than one or two decimal places. Over the years, numerical analysts have discovered many more sophisticated methods that yield much greater accuracy. These include the Improved Euler Method, Implicit Euler, the Runge-Kutta Method and others described in the above two references. Matlab's numerical solver, **ode45**, is one such highly sophisticated method. It is extraordinarily accurate. You can find a great deal of information about it in Chapter 8 of DEwM.

Here are some of the highlights:

- Since numerical methods yield only approximations to a solution, there is an inherent *error*, that is, the discrepancy between the actual value of the solution and the value computed by the numerical method. In fact, the error has two components – the so-called *formula error* intrinsic in the fact that the method's formula is only an approximation; and the *round-off error* caused by the fact that your computing device only maintains a certain number of decimal places. In practice, the latter is insignificant compared to the former.
- The increment *h* is called the *step* and its value the *step size*. Clearly, the smaller the step size, the more computation is required, but also the approximation is more accurate. Generally, the formula error will be proportional to a power of *h*, said power being called the *order* of the method. Euler is a first order method; **ode45** is a combination of fourth and fifth order methods.
- The step size is constant in the Euler Method, and in each computation only the previous step is used. Many methods employ variable step size (where, amazingly enough, the machine chooses the size on its own) and multi-step methods (meaning that many of the previous steps are used in a single computation). Generally, these are more accurate than single, fixed step size methods, but they are also far more computationally intensive. **ode45** is of that ilk.
- One can distinguish between local and global error—roughly the error you make in one step versus the error you make over the whole interval in which the solution is computed. You can read more about that in DEwM.
- You can control the error to some extent with certain Matlab settings – once again, see DEwM.
- We have seen that numerical methods, e.g., **ode45**, crank out a series of points that approximate points on the actual solution curve. **ode45** also generates a graph through these points. The simplest way to do that is merely to connect the successive points by straight lines. Matlab does something more sophisticated using an "interpolation method." For more on that, look at Sections 8.2.1 & 9.6.1 in DEwM. Section 9.6 contains more

useful information on **ode45**, as well as hints that might prove useful in working the Matlab problems.

Now comes a very important point that is only glossed over in the NODE notes – namely *stability* and *reliability*. In Section 8.4 of DEwM, an example is presented that reveals a rather drastic inaccuracy for the numerical solution that is computed there. This eventuality is bound up with the notion of *stability,* which is discussed in Section 5.3 of DEwM. An IVP is called *stable* if the long-term behavior of the solution is fairly insensitive to small changes in the initial data as *t* increases. It is called *unstable* if it is highly sensitive, as *t* increases, to small changes in initial data. Examples of both behaviors are exhibited in Section 5.3. The point is that numerical solutions of unstable IVPs may be highly unreliable. It is important to keep this in mind when employing numerical methods. Applied to stable IVPs, numerical methods are phenomenally accurate. Applied to unstable IVPs, numerical methods may be wildly inaccurate.

A result (Theorem 5.2, p. 60) is given in Section 5.3 of DEwM, which guarantees the stability of an IVP. Roughly speaking, it says that you need $\partial f/\partial y < 0$ on the open region for which the *existence and uniqueness theorem* applies to your IVP in order to guarantee stability.(Note that for the example in Section 8.4, this test fails, resulting in the unfortunate inaccuracy in the numerical solution.)

Finally, you are encouraged to read Chapter 8 carefully. You should look at I.7 in the NODE notes as well, but it is imperative that you consult Chapter 8. In particular, the material in Section 8.1 should prove useful in working problems in PSC.