Lecture 5

Austin Baird
UNC Department of Mathematics
UNC Department of Biology

February 13, 2014

Summary

- Last week we learned:
  - Numerical integration.
  - Rectangle rule.
  - Trapizoid rule
  - Review of matlab solvers used in solving non-linear ODEs
- Today we will:
  - Runge-Kutta.
  - Theory and implementation
  - Error analysis of ode solvers

Problem formulation

We want to be able to evaluate equations of the form:

$$\frac{dy}{dt} = f(t, y)$$
$$y(t_0) = y_0$$

Here t_0 is initial time, and y_0 is the initial value. We've done this once before with eulers method, but is that good enough?

Error and Doing a Bit Better

Recall from our previous homework that Eulers method is a first order method: $err = \mathcal{M}h$. Or that error is a linear function of step size $h = \Delta t$. Also recall Eulers method:

$$y(t_{n+1}) \approx y_n + hf(t_n, y_n)$$

or

$$y_{n+1} = y_n + hf(t_n, y_n)$$

here we approximate the solution, $y(t_{n+1})$ of the n+1 time step with previously computed data: y_n and the derivative: $f(t_n, y_n)$

Error and Doing a Bit Better

Because Euler's method is a first order method, if we want four digits of accuracy, then we have to select a time step:

$$\Delta t = 0.0001$$

Say we want to compute the solution of ten seconds:

$$N_{timesteps} = \frac{10}{0.0001} = 100000$$

One hundred thousand time steps! Seems a bit steep and can take a significant amount of time to compute in practice.

Error and Doing a Bit Better

Now say that we have a second order method:

$$err = \mathcal{M}h^2$$

Then we can compute the time step needed to get four digits of accuracy:

$$\sqrt{0.0001} = 0.01 = \Delta t$$

We can now see that it will only take 1000 times steps to compute the solution over 10 seconds. So much better!!

log-log plots

We can now extract the order of a given method by doing the following:

If we have the error = *err* and the time step: Δt, then we have the following relation (which can be seen in graph form):

$$err = \mathcal{M}h^2$$

This means that the error is a quadratic function of time step $h = \Delta t$. We can also take the log, *ln*, of the data and get:

$$log(err) = 2log(h) + log(\mathcal{M})$$

This shows now that we have a linear function with slope 2. If we do this to our data and take the slope of the line, then we can get the order of the method!

Example of a Second Order Method

Now we want to use the value of the derivative $f(t, y)$ at an intermediate time step: $\frac{t}{2}$. We want to use the value of the derivative between t_n and t_{n+1}. We will first introduce the Midpoint Method or also known as Two step Runge-Kutta:

$$
\begin{aligned}
k_1 &= hf(t_n, y_n) \\
k_2 &= hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\
y_{n+1} &= y_n + k_2
\end{aligned}
$$

This is whats known as a Multi-step Method and it gives us second order accuracy (We need to show this numerically!).

Matlab Implimentation

What do we need to compute this? In general all the tools we used in Eulers method apply here, with one additional computational step.

- Discretize the domain, $t = [t_0, t_1, \ldots, t_n]$
- Use the function: $f(t, y(t))$
- First we compute k_1
- $\gg k_1 = hf(t_n, y_n)$
- Now use this value in the next computational step:
- $\gg k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$
- Then we can compute our new approximation:
- $\gg y_{n+1} = y_n + k_2$
- Repeat at each time step!

Homework 5

We know that Euler method is 1st order, we want to verify that this new method is second order.

- Alter your Euler code so that it deals with time derivatives (this should be just changing x's to t's)
- Write a function which will solve $\frac{dy}{dt} = f(t, y)$ Using the two step Runge-Kutta method (introduced in these lecture notes).
- Write a separate script which does the following:
  - A plot which graphs error v Δt.
  - A log-log plot of err v Δt. (you can google log-log plot matlab, or you can just take $\gg ln(err), \gg ln(\Delta t)$, and graph.
  - A print out of the slope of the line that you just graphed (this should be close to 2, second order method!)
  - A plot of the graph of (eulers method - midpoint method) v time
  - A print out of the max error over the entire time simulation between the two methods. (i.e. max(euler-midpoint))