Outline

- Error Controlling Methods
- Stiffness in Equations
- Multistep Methods

Error Controlling Methods

Overview

So far we have only discussed asymptotic error results: we know that the error approaches 0 at a given rate $p$ for certain time discretizations. However, in practice, we usually want to do something different: we would like to bound the error by a constant tolerance (specified by your boss) and then find a time step that satisfies this error bound.

There are lots of sophisticated algorithms for handling this issue: many of them exploit knowledge of the underlying ODE system. We will focus on the simplest one, which are known as embedded Runge Kutta methods:

1. Pick a time step that will satisfy all necessary stability conditions.
2. Do one step with a specially chosen $p$th order Runge Kutta method.
3. Do one step with a specially chosen $p+1$th order Runge Kutta method.
4. Note that both steps are valid approximations (since we met the stability requirement). We can bound the error by finding the difference between the two methods. The logic here is tricky: Let $w_{1}^{p+1}$ be the $p+1$ order approximation and $w_{1}^{p}$ be the $p$ order approximation. Then

   $$|w^{p} - y(t_{1})| = |w_{1}^{p} - w_{1}^{p+1} + w_{1}^{p+1} - y(t_{1})|$$

   $$\leq \left| w_{1}^{p} - w_{1}^{p+1} \right| + \left| w_{1}^{p+1} - w_{1}^{p+1} \right|$$

   $$\approx \left| w_{1}^{p} - w_{1}^{p+1} \right|$$

   for sufficiently small $\Delta t$, we can ignore the next higher order approximation.

5. We now have a decent error bound. If the error is too low, then we should use what we have computed but pick a larger time step in the future. If the error is too high, then we should lower the time step and try again.
Example

The Runge-Kutta-Fehlberg method is probably the most commonly used adaptive ODE solver. It is sometimes just called ode45 after its MATLAB function name. It requires a total of six stages: five for the fourth order part (almost optimal) and one more stage for the fifth order part. This is where the term *embedded* comes from: for a particular choice of stages in a method, one can write down a \( p \)th order method that uses all but the last stage and a \( p + 1 \)th order method that uses all the stages.

For comparison: if we tied together RK3 and RK4, we would have to evaluate six total stages since they are not embedded (but they do share the first stage). Runge Kutta Fehlberg has the same number of stage evaluations but yields a fifth order solution. This method was devised by the NASA engineer Erwin Fehlberg in the late 1960s and was their workhorse for a long time (it may still be, but at least in the 1970s this was the best adaptive algorithm).

Example

For the sake of a simple implementation, consider the Bogacki-Shampine method (known as ode23 in MATLAB):

\[
\begin{align*}
k_1 &= f(t_0, w_0) \\
k_2 &= f(t_0 + \frac{1}{2} \Delta t, w_0 + \frac{1}{2} \Delta tk_1) \\
k_3 &= f(t_0 + \frac{3}{4} \Delta t, w_0 + \frac{3}{4} \Delta tk_2) \\
w_1 &= w_0 + \frac{2}{9} \Delta tk_1 + \frac{1}{3} \Delta tk_2 + \frac{4}{9} \Delta tk_3 \\
k_4 &= f(t_0 + \Delta t, w_1) \\
z_1 &= w_1 + \frac{7}{24} \Delta tk_1 + \frac{1}{4} \Delta tk_2 + \frac{1}{3} \Delta tk_3 + \frac{1}{8} \Delta tk_4
\end{align*}
\]

Where the error in \( w_1 \) is \( O(\Delta t^3) \) globally and the error in \( z_1 \) is \( O(\Delta t^2) \) globally. Let’s implement this.
Stiffness in Equations

Definition

Stiffness is notoriously difficult to define. I will go with Bill Henshaw’s definition:

A stiff problem is one that
1. has widely different time scales
2. where the time step restriction arising from an explicit scheme is much smaller than dictated by accuracy alone.

Example

Consider a stiff, highly damped spring:

\[ y'' + \frac{2}{\epsilon} y' + \frac{1}{\epsilon} y = 0, y(0) = 0, \epsilon y'(0) = 1 \]

where \( \epsilon > 0 \) is small. The exact solution to this equation is, approximately, for \( \epsilon \ll 1 \) (so, neglecting \( O(\epsilon^2) \) terms):

\[ y(t) \approx \frac{1}{2} (\exp(-t/2) - \exp(-2t/\epsilon)). \]

What are the accuracy constraints on this method for using forward Euler? What about the stability constraints? How does the use of an implicit method help?
Multistep Methods

Introduction

Edsberg does not supply much material on this subject so I will primarily use material from Ackleh, Allen, Hearfott, and Seshaiyer and also Sauer’s book in this section.

Runge Kutta methods involved evaluating the right hand side function \( f(t, y) \) several times for each time step. Multistep methods use previously calculated values of \( f(t, y) \) instead; this saves some computational effort and is preferable for some problems. The primary drawback is that the stability regions of explicit multistep methods shrink as the order increases (while they tend to grow for Runge Kutta methods).

As motivation, consider a very stiff problem where we might want a higher order implicit method. One way to do this is to integrate both sides of the ODE:

\[
\int_{t_0}^{t_1} y'(t) \, dt = \int_{t_0}^{t_1} f(t, y) \, dt \Rightarrow y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(t, y) \, dt.
\]

The approximation comes from using a quadrature rule to get rid of the integral term:

\[
\int_{t_0}^{t_1} f(t, y) \, dt = \frac{\Delta t}{2} (f(t_1, y_1) + f(t_0, y_0)) + O(\Delta t^3)
\]

This is equivalent to integrating with the trapezoid rule. Since the error in the Trapezoid rule is \( O(\Delta t^3) \), the single step error is \( O(\Delta t^3) \) and the global error is \( O(\Delta t^2) \). Note: I still expect you to derive an equivalent result with Taylor series on the current homework assignment.
General Formulation

All of these methods may be described with a single, general formula (Ackleh, Allen, Hearfott, and Seshaiyer, Equation 7.71):

\[ L[y(t); \Delta t] = \sum_{j=0}^{k} \alpha_j w(t + j\Delta t) - \Delta t \beta j y'(t + j\Delta t) \]

or, equivalently, with our usual index notation:

\[ L[y(t); \Delta t] = \sum_{j=0}^{k} \alpha_j w_j - \Delta t \beta j f_j \]

For the rule we derived above (AM-3), these coefficients are:

Adams-Bashforth and Adams-Moulton Methods

General Idea

All of these methods involve numerical integration: we use the fundamental theorem of calculus to get rid of the derivative on \( y \) and then approximate the integral in \( f \) with some Newton-Cotes formula.

Unproven result: You should recall from NUMCOMP that, in general, for a closed Newton-Cotes formula, the order of accuracy is (at least) equal to the number of quadrature points plus one.

Unproven result: The extrapolation we use in Adams-Bashforth methods (i.e., we do not require knowledge of \( f(t_n, y_n) \)) does not change the order of convergence. One can show this with Taylor series.

Adams-Bashforth

Adams-Bashforth methods are explicit; \( \beta_k = 0 \) for each rule. One example is \( AB - 2 \):

\[ w_2 = w_1 + \Delta t \left( \frac{3}{2} f_1 - \frac{1}{2} f_0 \right) \]

which may be derived from interpolating \( f \) as a linear function passing between \((t_0, f_0)\) and \((t_1, f_1)\):
another is AB-3:

\[ w_3 = w_2 + \frac{\Delta t}{12} (23f_2 - 16f_1 + 5f_0) \]

**Adams-Moulton**

These are *implicit* methods: \( \beta_k \neq 0 \). AM-2 is the Trapezoid method, which we have already seen. The next one, AM-3, is

\[ y_2 = y_1 + \frac{\Delta t}{12} (5f_2 + 8f_1 - f_0). \]

They are derived in the same way as Adams-Bashforth methods but do not use the last extrapolation step.