Notes: ODEs

Logistics

1. I will be away next week looking for housing. Jon Weare will give the lecture that day.

2. This week and next week are ODEs; then two weeks of Monte Carlo, one week of review, and we’re done! The final will be May 7, 7:10-9:00 in room 1302.

3. I will give practice exams and solutions. There will be some theoretical questions and some simple computations done by hand. Open anything you want, except your neighbor.

4. I still owe you a cleaned-up interpolation / FFT chapter. The current chapter is better, but quirky. I will deviate from it some again. Will post my (terse) notes for this lecture.

5. Ethan requests for your HW submissions that you submit one document that includes all plots, code listings, and answers in some logical format. Keeping a half dozen printouts per person straight takes a lot of time. If I need separate code for an autograder, I will prompt you for it.

Goals

Here’s what I think you should get out of our discussion:

1. How to convert ODEs into a standard form for a solver.

2. A detailed understanding of the most basic methods (forward Euler, backward Euler, trap rule).


4. The meaning of stiffness and stability regions.

5. The relation between error analysis and sensitivity analysis for ODEs.

6. The ideas behind Runge-Kutta and multistep methods.
7. What standard methods are available and where to find (and how to use) software implementations.

8. How adaptivity and error estimation work.

9. The idea of an implicit solver and connections to nonlinear equation solvers.

ODE basics

1. Why solve ODEs numerically rather than symbolically? In general, most ODEs have no closed form solution in terms of elementary functions. It’s a sad fact that’s often left out of introductory ODE classes.

2. The standard form for an ODE will be the system

\[ y'(t) = f(y, t). \]

If \( f \) does not depend on \( t \), the system is autonomous. Will often assume autonomous systems for simplicity.

3. What about a second order system, e.g.

\[ y''(t) = f(y, t)? \]

Introduce an auxiliary variable:

\[
\begin{bmatrix}
  y' \\
  v
\end{bmatrix} =
\begin{bmatrix}
  v \\
  f(y, t)
\end{bmatrix}.
\]

Standard trick, works with general higher-order systems.

4. Sufficient condition for existence and uniqueness: \( f(u, t) \) is globally uniformly Lipschitz in the first argument for all \( t \) in some interval, i.e. there is some \( C \) such that

\[ \| f(u, t) - f(v, t) \| \leq C \| u - v \|. \]

Not necessary – but you should worry if a solution goes through some region where \( f \) doesn’t look nicely Lipschitz. Bad idea to compute a solution when one doesn’t exist!

Will care about Lipschitz condition when analyzing methods, too.
5. Suppose we have an autonomous system, globally Lipschitz $f$. Also suppose that $y(t)$ and $\hat{y}(t)$ are different trajectories from different initial conditions. Define $e(t) = \hat{y}(t) - y(t)$. Then

$$|e'(t)| = |f(\hat{y}(t)) - f(y(t))| \leq C|e(t)|.$$ 

Can solve this differential inequality to find

$$|e(t)| \leq |e(0)| \exp( Ct ).$$

This is a special case of Gronwall’s inequality.

The constant $C$ says something about the stability of the ODE – how quickly trajectories want to diverge from each other.

6. Aside: implicitly defined equations are trickier, particularly if they involve algebraic constraints. There is a whole different lore for solving such differential-algebraic equations; see the textbook by Ascher and Petzold, for example.

**Forward Euler**

Consider the autonomous system $y' = f(y)$. Will analyze solution by forward Euler:

1. Basic idea (differential form): use the difference formula

$$y'(t_n) \approx \frac{y(t_{n+1}) - y(t_n)}{h}$$

    to get

$$\hat{y}_{n+1} - \hat{y}_n = \frac{f(\hat{y}_n)}{h}.$$ 

2. Basic idea (integral form): write differential equation as

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(y(s)) \, ds$$

    Approximate the integral by left-hand rule:

$$\hat{y}_{n+1} = \hat{y}_n + hf(\hat{y}_n)$$
3. Example (useful for later): \( f(y) = \lambda y \). Exact solution is 
\[ y(t) = y_0 \exp(\lambda t). \]
Forward Euler gives
\[ \hat{y}_n = y_{n-1} + h\lambda y_{n-1} = (1 - h\lambda)y_{n-1} \]
or
\[ \hat{y}(t_n) = \hat{y}_0(1 + h\lambda)^{t_n/h}. \]
This has the right behavior as \( h \to 0 \), since
\[ \lim_{h \to 0} (1 + h\lambda)^{1/h} = \exp(\lambda). \]

**Consistency + stability = convergence**

1. Suppose we write the discrete difference operator for an ODE solver applied to a function \( x \) abstractly as \( N_h x \). For example, for Euler’s method, we would have
\[ N_h x(t_{n+1}) = \frac{x(t_{n+1}) - x(t_n)}{h} - f(x(t_n)). \]
The approximate solution \( \hat{y} \) satisfies \( N_h \hat{y} = 0 \).

2. **Consistency** for an ODE method means that when you plug the *true* solution \( y \) into the approximate equations, you get a small residual, i.e. \( N_h y = O(h^p) \) for some \( p > 0 \). For example, for Euler’s method, we have
\[ N_h y(t_{n+1}) = \frac{y(t_{n+1}) - y(t_n)}{h} - f(y(t_n)) = \frac{h}{2} y''(\xi) = O(h). \]
The residual \( N_h y(t_{n+1}) \) is also called the *local truncation error*.

3. **Stability** informally means that errors don’t blow up as they propagate through the recurrence. Formally, it means that on some time interval \([0, T]\), there is some \( K \) so that if \( h = T/N \) is small enough, for any \( 1 \leq n \leq N \),
\[ |x_n - z_n| \leq K \{ |x_0 - z_0| + \max_{1 \leq j \leq N} |N_h x(t_j) - N_h z(t_j)| \}. \]
This statement says that if the residual errors for $x$ and $z$ are close to each other, then $x$ and $z$ cannot be too far apart. In effect, this is a bound on the inverse of the difference operator.

4. If a method is both consistent and stable, then for small enough $h$,

$$\left|\hat{y}(t_n) - y(t_n)\right| \leq K\{|\hat{y}_0 - y_0| + \max_j |N_h\hat{y}(t_j) - N_hy(t_j)|\} = K\{|\hat{y}_0 - y_0| + O(h^p)\}.$$  

If the initial conditions are exact, then the error is $O(h^p)$, the same as the residual error. Thus, consistency + stability implies convergence of the method. Of course, if $K$ is large, the global error may be much worse than the residual error.

5. What about stability of Euler’s method? Suppose $x(t)$ and $z(t)$ are functions defined on mesh points. By the definition of $N_h$,

$$x_{n+1} = x_n + hf(x_n) + hN_hx(t_{n+1}) \tag{1}$$  
$$z_{n+1} = z_n + hf(z_n) + hN_hz(t_{n+1}) \tag{2}.$$  

Define $d_n = |x_n - z_n|$ and $\theta = \max_{1 \leq j \leq N} |N_hx(t_{j+1}) - N_hz(t_{j+1})|$. Subtract (1)-(2) to get

$$d_{n+1} \leq d_n + h|f(x_n) - f(z_n)| + h\theta.$$  

If we assume $f$ is Lipschitz with constant $C$, then

$$d_{n+1} \leq (1 + Ch)d_n + h\theta.$$  

It is easy to solve this difference inequality (similar to Gronwall!):

$$d_1 \leq (1 + Ch)d_0 + h\theta,$$

$$d_2 \leq (1 + Ch)^2d_0 + (1 + Ch)h\theta + h\theta,$$

$$d_n \leq (1 + Ch)^nd_0 + h\theta \sum_{j=0}^{n-1} (1 + Ch)^j.$$  

Using the geometric sum formula, we have

$$d_n \leq (1 + Ch)^nd_0 + h\theta \frac{(1 + Ch)^n - 1}{(1 + Ch) - 1} = (1 + Ch)^nd_0 + \frac{1}{c}((1 + Ch)^n - 1)\theta,$$
and using the inequality $1 + Ch \leq \exp(Ch)$, we have

$$d_n \leq \exp(CT_n)d_0 + \frac{1}{C} (\exp(CT_n) - 1) \theta.$$ 

Therefore, Euler’s method is stable with constant

$$K = \max \left\{ \exp(CT), \frac{1}{C}(\exp(CT) - 1) \right\}.$$ 

**An informal approach**

1. Here’s another way to think about the error in forward Euler. Suppose forward Euler gives us values $\hat{y}_1, \hat{y}_2, \ldots$. Let us define another sequence $\tilde{y}_n$ by $\tilde{y}_n = z(t_n)$, where $z$ is a solution to the initial value problem

$$z' = f(z), \quad z(t_{n-1}) = \hat{y}_{n-1}.$$ 

That is, $\tilde{y}_n$ is what we would get if we followed the exact trajectory described by the differential equation for one step, starting at $\hat{y}_{n-1}$. The local error is the difference $\hat{y}_n - \tilde{y}_n$. For forward Euler, we know that this difference should be $O(h^2)$.

2. Given $\hat{y}_{n-1}$ and $y(t_{n-1})$, we know from our earlier analysis of diverging trajectories for ODEs that

$$|\tilde{y}_n - y(t_n)| \leq |\hat{y}_{n-1} - y(t_{n-1})|e^{Ch}.$$ 

By the triangle inequality,

$$|\hat{y}_n - y(t_n)| \leq |\hat{y}_{n-1} - y(t_{n-1})|e^{Ch} + \xi_n,$$

where $\xi_n = |\hat{y}_n - \tilde{y}_n|$ is the local error.

3. As in the previous section, we can use a geometric sum and simplify to get

$$|\hat{y}_n - y(t_n)| \leq e^{Ct_n}|\hat{y}_0 - y_0| + \frac{1}{C} (e^{Ct_n} - 1) \max_{1 \leq j \leq N} \frac{\xi_j}{h}.$$ 

That is, the local errors due to numerical approximation cannot get arbitrarily magnified as we propagate our numerical solution along.
Stiffness

1. Suppose we apply Euler’s method to $y' = -1000y$ with step size $h = 0.01$. Then we get

$$y_n = (-9)^n y_0$$

which blows up pretty quickly – even though the true solution decays. Something is amiss!

2. More generally, consider the model problem

$$y' = -\lambda y.$$ 

When the real part of $\lambda$ is less than zero, solutions to this differential equation asymptotically decay to zero as $t \to \infty$. Euler’s method applied to this equation gives us

$$y_n = (1 + h\lambda)^n y_0,$$

which decays when $|1 + h\lambda| < 1$. The unit disk about $-1$ in which $h\lambda$ must lie for us to get decay is called the region of absolute stability for the method.

If $\lambda$ is very negative, for example, then we must take very small step sizes in order for the numerical solution to have the right qualitative behavior.

3. A stiff differential equation is characterized by multiple time scales: some directions have fast decay that we don’t really care to track in detail, while other directions correspond to much slower motion. If we care about the dynamics along a slow manifold and not the fast dynamics that pull us toward that manifold, then we want a method that (unlike Euler) can take ignore the details of the fast dynamics in order to take large time steps that are consistent with the slow dynamics.

4. As described above, we essentially want to find a slow manifold – without taking lots of short time steps to reach it. We should suspect that means we will need to solve a nonlinear equation that tells us how to get to the slow manifold. Indeed, this is the idea behind implicit methods, which we describe next.
Backward Euler and the trapezoidal rule

1. We found forward Euler by writing the integral equation

\[ y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(y(s)) \, ds \]

and discretizing the integral with the left-hand rule. If we use the right-hand rule, we get the backward Euler formula:

\[ \hat{y}_{n+1} = \hat{y}_n + hf(\hat{y}_{n+1}). \]

If we use the trapezoidal rule for integration, we get the trapezoidal rule for time stepping:

\[ \hat{y}_{n+1} = \hat{y}_n + \frac{h}{2} (f(\hat{y}_n) + f(\hat{y}_{n+1})). \]

2. Both the backward Euler formula and the trapezoidal rule are \textit{implicit}, and in general we would need to compute the new value \( y_{n+1} \) by solving nonlinear equation. The standard approach to this is a Newton iteration (or a modified Newton iteration). Because they solve a nonlinear equation involving \( f \), most implicit time stepping software likes you to be able to compute \( f' \) on demand – though the software may compute it for you by finite differences if you don’t provide a routine.

3. What happens when we apply backward Euler or the trapezoidal rule to the model problem \( y' = \lambda y \)?

Backward Euler gives:

\[
\begin{align*}
y_{n+1} &= y_n + h\lambda y_{n+1} \\
y_{n+1} &= (1 - h\lambda)^{-1} y_n \\
y_n &= (1 - h\lambda)^{-n} y_0
\end{align*}
\]

Therefore, the region for absolute stability is \(|1 - h\lambda| > 1\) – the area outside the circle of radius one about 1. This region includes the entire left half plane, so the numerical solution decays whenever the true solution decays; this property is known as \textit{A-stability}. The numerical solution also decays whenever the true solution is highly oscillatory, or whenever it grows quickly. That is, backward Euler exhibits \textit{numerical dissipation}. 
4. The trapezoidal rule gives

\[ y_{n+1} = y_n + \frac{h\lambda}{2} (y_n + y_{n+1}) \]

\[ y_n = \left( \frac{1 + h\lambda/2}{1 - h\lambda/2} \right)^n y_0. \]

Therefore, the region of absolute stability is \(|1 + h\lambda/2| < |1 - h\lambda/2|\); that is, the points where \(h\lambda/2\) is closer to \(-1\) than to \(1\). This is precisely the left half plane, i.e. the points \(\Re(h\lambda) < 0\). So the numerical approximation for the model problem decays precisely if the true solution does. Also, because \((1 + h\lambda/2)/(1 - h\lambda/2)\) maps the imaginary axis to the unit circle, the numerical solution to the model problem is purely oscillatory precisely if the true solution is purely oscillatory.