Numerical Methods II
Absolute Stability and Stiffness

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

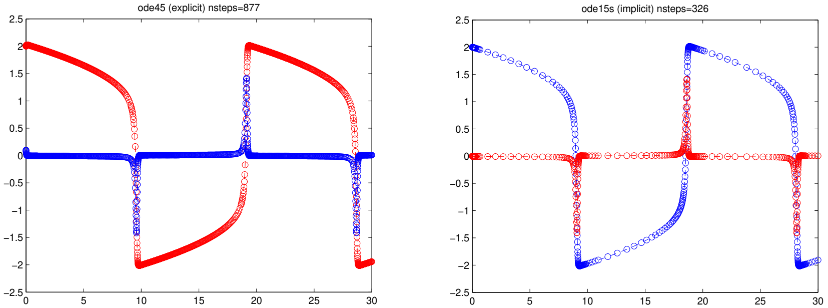
[1]MATH-GA.2020-001 / CSCI-GA.2421-001, Spring 2023

Feb 14, 2023

Outline

Long Time (In)Stability

Stiff van der Pol system



A stiff problem is one where Δt has to be small even though the solution is smooth and a large Δt is OK for accuracy.

Stiff example

- In section 7.1 LeVeque discusses

$$x'(t) = \lambda\,(x - \cos t) - \sin t.$$

  with solution $x(t) = \cos t$ if $x(0) = 1$.

- If $\lambda = 0$ then this is very simple to solve using Euler's method, for example, $\Delta t = 10^{-3}$ up to time $T = 2$ gives error $\sim 10^{-3}$.
- For $\lambda = -10$, one gets an even smaller error with the same time step size.

Instability

But for $\lambda = -2100$, results for $\Delta t > 2/2100 = 0.000954$ are completely useless: **method is unstable**.

Table 7.1. *Errors in the computed solution using Euler's method for Example 7.3, for different values of the time step k. Note the dramatic change in behavior of the error for $k < 0.000952$.*

k	Error
0.001000	0.145252E+77
0.000976	0.588105E+36
0.000950	0.321089E-06
0.000800	0.792298E-07
0.000400	0.396033E-07

Conditional Stability

- Consider the model problem for $\lambda < 0$:

$$x'(t) = \lambda x(t)$$
$$x(0) = 1,$$

with an exact solution that **decays exponentially**, $x(t) = e^{\lambda t}$.

- Applying Euler's method to this model equation gives:

$$x^{(k+1)} = x^{(k)} + \lambda x^{(k)} \Delta t = (1 + \lambda \Delta t)\, x^{(k)} \quad \Rightarrow$$

$$x^{(k)} = (1 + \lambda \Delta t)^k$$

- The numerical solution will **decay** if the time step satisfies the **stability criterion**

$$|1 + \lambda \Delta t| \leq 1 \quad \Rightarrow \quad \Delta t < -\frac{2}{\lambda}.$$

- Otherwise, the numerical solution will eventually blow up!

Unconditional Stability

- The above analysis shows that **forward Euler is conditionally stable**, meaning it is stable if $\Delta t < 2/|\lambda|$.
- Let us examine the stability for the model equation $x'(t) = \lambda x(t)$ for **backward Euler**:

$$x^{(k+1)} = x^{(k)} + \lambda x^{(k+1)} \Delta t \quad \Rightarrow \quad x^{(k+1)} = x^{(k)} / (1 - \lambda \Delta t)$$

$$x^{(k)} = x^{(0)} / (1 - \lambda \Delta t)^k$$

- We see that the implicit **backward Euler is unconditionally stable**, since for any time step

$$|1 - \lambda \Delta t| > 1.$$

Stiff Equations

Stiff Equations

- For a real "non-linear" problem, $x'(t) = f\left[x(t), t\right]$, the role of λ is played by
$$\lambda \longleftrightarrow \frac{\partial f}{\partial x}.$$
- Consider the following model equation:
$$x'(t) = \lambda\left[x(t) - g(t)\right] + g'(t),$$
where $g(t)$ is a nice (regular) function evolving on a time scale of order 1, and $\lambda \lll -1$ is a large negative number.
- The exact solution consists of a fast-decaying "irrelevant" component and a slowly-evolving "relevant" component:
$$x(t) = \left[x(0) - g(0)\right] e^{\lambda t} + g(t).$$
- Using Euler's method requires a time step $\Delta t < 2/\left|\lambda\right| \lll 1$, i.e., many time steps in order to see the relevant component of the solution.

Stiff Systems

- An ODE or a system of ODEs is called **stiff** if the solution evolves on widely-separated timescales and the fast time scale decays (dies out) quickly.
- We can make this precise for linear systems of ODEs, $\mathbf{x}(t) \in \mathbb{R}^n$:

$$\mathbf{x}'(t) = \mathbf{A}\left[\mathbf{x}(t)\right].$$

- Assume that $\mathbf{A}$ has an eigenvalue decomposition, with potentially **complex eigenvalues**:

$$\mathbf{A} = \mathbf{X}\boldsymbol{\Lambda}\mathbf{X}^{-1},$$

  and express $\mathbf{x}(t)$ in the basis formed by the eigenvectors $\mathbf{x}_i$:

$$\mathbf{y}(t) = \mathbf{X}^{-1}\left[\mathbf{x}(t)\right].$$

contd.

$$\mathbf{x}'(t) = \mathbf{A}\left[\mathbf{x}(t)\right] = \mathbf{X}\boldsymbol{\Lambda}\left[\mathbf{X}^{-1}\mathbf{x}(t)\right] = \mathbf{X}\boldsymbol{\Lambda}\left[\mathbf{y}(t)\right] \quad \Rightarrow$$

$$\mathbf{y}'(t) = \boldsymbol{\Lambda}\left[\mathbf{y}(t)\right]$$

- The different y variables are now **uncoupled**: each of the n ODEs is independent of the others:

$$y_i = y_i(0)e^{\lambda_i t}.$$

- Assume for now that all eigenvalues are **real and negative**, $\boldsymbol{\lambda} < 0$, so each component of the solution decays:

$$\mathbf{x}(t) = \sum_{i=1}^{n} y_i(0)e^{\lambda_i t}\mathbf{x}_i \quad \to \quad 0 \text{ as } t \to \infty.$$

- For the forward Euler's method, we require

$$\Delta t < \frac{2}{\max_i \left|\mathrm{Re}(\lambda_i)\right|}.$$

Stiffness

- A system is **stiff** if there is a strong **separation of time scales**:

$$r = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|} \gg 1.$$

- For non-linear problems $\mathbf{A}$ is replaced by the Jacobian $\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{f}(\mathbf{x}, t)$, i.e., what matters are the **eigenvalues of the Jacobian**.
- In general, the Jacobian will have **complex eigenvalues**, so absolute value above means complex modulus.
- For a more in-depth discussion of stiffness, see Section 8.2 in the book of LeVeque.

Absolute Stability

Absolute Stability

- We see now that for systems we need to allow λ to be a **complex number** but we can still look at scalar equations.
- A method is called **absolutely stable** if for $\text{Re}(\lambda) < 0$ the numerical solution of the **scalar model equation**

$$x'(t) = \lambda x(t)$$

  decays to zero, like the actual solution.
- We call the **region of absolute stability** the set of complex numbers

$$z = \lambda \Delta t$$

  for which the numerical solution decays to zero.
- For systems of ODEs **all scaled eigenvalues of the Jacobian** $\lambda_i \Delta t$ **should be in the stability region.**

Stability regions

- For Euler's method, the stability condition is

$$|1+\lambda \Delta t| = |1+z| = |z-(-1)| \le 1 \quad \Rightarrow$$

which means that z must be in a unit disk in the complex plane centered at $(-1,0)$:

$$z \in \mathcal{C}_1(-1,0).$$

- A general one-step method of order p applied to the **model equation** $x' = \lambda x$ where $\lambda \in \mathbb{C}$ gives:

$$x^{n+1} = R(z = \lambda \Delta t) x^n.$$

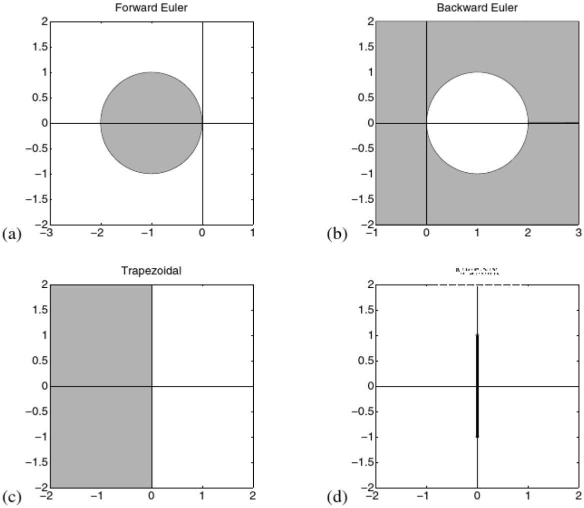
$$R(z) = e^z + O\left(z^{p+1}\right) \text{ for small } |z|.$$

- The **region of absolute stability** is the set

$$\mathcal{S} = \{z \in \mathbb{C} : |R(z)| \le 1\}.$$

Simple Schemes

Forward/backward Euler, **implicit trapezoidal**, and leapfrog schemes



A-Stable methods

- A method is **A-stable** if its stability region contains the entire left half plane.
- The backward Euler and the implicit midpoint scheme are both A-stable, but they are also both implicit and thus expensive in practice!
- Theorem: **No explicit one-step method can be A-stable** (discuss in class why).
- Theorem: All explicit RK methods with r stages and of order r have the same stability region (discuss why).

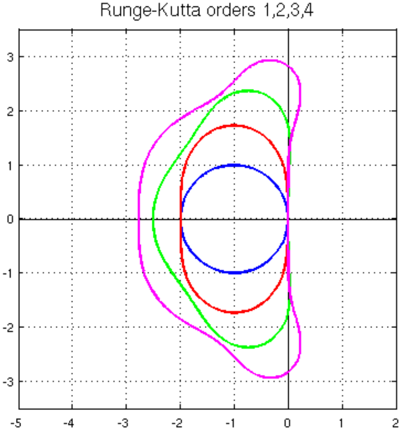
One-Step Methods

- Any r-stage **explicit** RK method will produce $R(z)$ that is a **polynomial** of degree r.
- Any r-stage **implicit** RK method has **rational** $R(z)$ (ratio of polynomials).
  The degree of the denominator cannot be larger than the **number of linear systems that are solved** per time step.
- RK methods give polynomial or rational approximations $R(z) \approx e^z$.
- A 4-stage explicit RK method therefore has

$$R(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4$$

Explicit RK Methods

Stability regions for **all** *r*-stage explicit RK methods



One needs at least 3 stages to be stable for purely imaginary eigenvalues (hyperbolic PDEs later on).

Transients, damping and oscillations

Stiff equation example from LeVeque with implicit trapezoidal (left) vs. backward Euler (right)

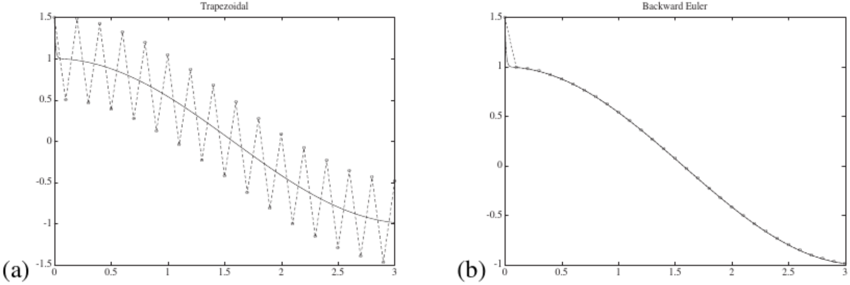


Figure 8.4. *Comparison of* (a) *trapezoidal method and* (b) *backward Euler on a stiff problem with an initial transient (Case* 2 *of Example* 8.3*).*

L-stable methods

- We can explain this by noting that for large $|z| = |\lambda \Delta t| \gg 1$ we have:

$$R(z) = \begin{cases} \frac{1}{1-z} \approx 0 & \text{Backward Euler} \\ \frac{1+z/2}{1-z/2} \approx -1 & \text{Implicit trapezoidal} \end{cases}$$

- So **backward Euler damps transients/errors** like $|\lambda \Delta t|^{-k}$ after k iterations, while implicit trapezoidal/midpoint just multiplies them by $\approx (-1)^k$ without damping.
- A method is **L-stable** if it is A-stable and it damps fast components of the solution

$$\lim_{z \to -\infty} |R(z)| = 0.$$

- TR-BDF2 (see RK lecture) is L-stable and second order.
- **Just because a method is stable doesn't mean it is accurate**. A higher-order method does not necessarily give a more accurate solution if the time step is not asymptotically small.

Implicit RK Methods

- An implicit RK method of maximum order per number of function evaluations must generate a **Pade approximation**, e.g.,

$$e^z \approx \begin{cases} \frac{1+z/2}{1-z/2} & \text{Implicit trapezoidal} \\ \frac{1+z/3}{1-2z/3+z^2/6} & \text{Fully implicit RK2} \end{cases}$$

- The diagonally implicit RK2 (DIRK2) method with tableau

$$\mathbf{c} = [\gamma, 1-\gamma], \; \mathbf{b} = [1/2, 1/2], \; \mathbf{A} = \begin{bmatrix} \gamma & \\ 1-2\gamma & \gamma \end{bmatrix},$$

is third-order accurate and A-stable for $\gamma = \frac{1}{2} + \frac{\sqrt{3}}{6}$, but is only L-stable for $\gamma = 1 \pm \sqrt{2}/2$ and second-order.

Implicit Methods

- **Implicit methods are** generally **more stable** than explicit methods, and solving stiff problems generally requires using an implicit method.
- Beware of **order reduction:** (DI)RK methods of order larger than 2 can exhibit reduced order of accuracy (usually down to 2nd order) for very stiff problems even though they are stable (concept of **stage order** becomes important also).
- The price to pay is solving a system of non-linear equations at every time step (linear if the ODE is linear):
  This is best done using **Newton-Raphson**'s method, where the solution at the previous time step is used as an initial guess.
- For PDEs, the linear systems become large and implicit methods can become very expensive...

Implicit-Explicit Methods

- When solving PDEs, we will often be faced with problems of the form

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t) + \mathbf{g}(\mathbf{x}, t) = \text{stiff+non-stiff}$$

  where the stiffness comes only from $\mathbf{f}$.
- These problems are treated using **implicit-explicit (IMEX)** or **semi-implicit** schemes, which only treat $\mathbf{f}(\mathbf{x})$ implicitly (see HW4 for KdV equation).
- A very simple example of a second-order scheme is to treat $\mathbf{g}(\mathbf{x})$ using the **Adams-Bashforth** multistep method and treat $\mathbf{f}(\mathbf{x})$ using the implicit trapezoidal rule (**Crank-Nicolson** method), the **ABCN** scheme:

$$\begin{aligned} x^{(k+1)} = x^{(k)} + & \frac{\Delta t}{2}\left[\mathbf{f}\left(x^{(k)},\ t^{(k)}\right) + f\left(x^{(k+1)},\ t^{(k+1)}\right)\right] \\ & +\Delta t\left[\frac{3}{2}g\left(x^{(k)},\ t^{(k)}\right) - \frac{1}{2}g\left(x^{(k-1)},\ t^{(k-1)}\right)\right]. \end{aligned}$$

Conclusions

Which Method is Best?

- As expected, there is no universally "best" method for integrating ordinary differential equations: It depends on the problem:
  - How stiff is your problem (may demand implicit method), and does this change with time?
  - How many variables are there, and how long do you need to integrate for?
  - How accurately do you need the solution, and how sensitive is the solution to perturbations (chaos).
  - How well-behaved or not is the function $f(x, t)$ (e.g., sharp jumps or discontinuities, large derivatives, etc.).
  - How costly is the function $f(x, t)$ and its derivatives (Jacobian) to evaluate.
  - Is this really ODEs or a something coming from a PDE integration (next lecture)?

Conclusions/Summary

- Time stepping methods for ODEs are **convergent if and only if they are consistent and stable**.
- We distinguish methods based on their **order of accuracy** and on whether they are **explicit** (forward Euler, Heun, RK4, Adams-Bashforth), or **implicit** (backward Euler, Crank-Nicolson), and whether they are **adaptive**.
- **Runge-Kutta methods** require more evaluations of f but are more robust, especially if adaptive (e.g., they can deal with sharp changes in f). Generally the recommended first-try ($ode45$ or $ode23$ in MATLAB).
- **Multi-step methods** offer high-order accuracy and require few evaluations of f per time step. They are not very robust however. Recommended for well-behaved non-stiff problems ($ode113$).
- For **stiff problems** an **implicit method** is necessary, and it requires solving (linear or nonlinear) systems of equations, which may be complicated (evaluating Jacobian matrices) or costly ($ode15s$).