Scientific Computing: Ordinary Differential Equations

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

[1]Course MATH-GA.2043 or CSCI-GA.2112, Fall 2019

Nov 19th and Dec 5th, 2019

Outline

Outline

Initial Value Problems

- We want to numerically approximate the solution to the **ordinary differential equation**

$$\frac{dx}{dt} = x'(t) = \dot{x}(t) = f\left(x(t), t\right),$$

with **initial condition** $x(t = 0) = x(0) = x_0$.

- This means that we want to generate an approximation to the **trajectory** $x(t)$, for example, a sequence $x(t_k = k\Delta t)$ for $k = 1, 2, \ldots, N = T/\Delta t$, where Δt is the **time step** used to discretize time.
- If f is independent of t we call the system **autonomous**.
- Note that second-order equations can be written as a **system** of first-order equations:

$$\frac{d^2x}{dt^2} = \ddot{x}(t) = f\left[x(t), t\right] \quad \equiv \quad \begin{cases} \dot{x}(t) = & v(t) \\ \dot{v}(t) = & f\left[x(t), t\right] \end{cases}$$

Relation to Numerical Integration

- If f is independent of x then the problem is equivalent to numerical integration

$$x(t) = x_0 + \int_0^t f(s)ds.$$

- More generally, we cannot compute the integral because it depends on the unknown answer $x(t)$:

$$x(t) = x_0 + \int_0^t f\left(x(s), s\right) ds.$$

- Numerical methods are based on approximations of $f\left(x(s), s\right)$ into the "future" based on knowledge of $x(t)$ in the "past" and "present".

Convergence

- Consider a trajectory numerically discretized as a sequence that **approximates** the exact solution at a **discrete** set of points:

$$x^{(k)} \approx x(t_k = k\Delta t), \quad k = 1, \ldots, T/\Delta t.$$

- A method is said to **converge with order** $p > 0$, or to have **order of accuracy** p, if for any finite T for which the ODE has a solution,

$$\left| x^{(k)} - x(k\Delta t) \right| = O(\Delta t^p) \text{ for all } 0 \leq k \leq T/\Delta t.$$

- All methods are recursions that compute a new $x^{(k+1)}$ from previous $x^{(k)}$ by evaluating $f(x)$ several times. For example, **one-step methods** have the form

$$x^{(k+1)} = G\left(x^{(k)}, \Delta t;\, f\right).$$

Consistency

- The **local trunction error** (LTE) of a method is the amount by which the *exact* solution does not satisfy the numerical scheme at the end of the time step if started from the correct solution $x^{(k)} = x\left(k\Delta t\right)$:

$$e_k = x\left[\left(k+1\right)\Delta t\right] - G\left[x(k\Delta t), \Delta t;\, f\right],$$

- A method is **consistent with order** $q > 1$ if $|e_k| = O(\Delta t^q)$.
- The **global truncation error** is the actual error

$$E_{t=k\Delta t} = \left| x^{(k)} - x(t = k\Delta t) \right|.$$

- Numerical analysis question: Can the global error be bounded by $O\left(\Delta t^p\right)$ if the local one is $O(\Delta t^q)$?

Propagation of errors

- *Crude* estimate: If one makes an error $O(\Delta t^q)$ at each time step, the global error after $T/\Delta t$ time steps can become on the order of

$$\left|x^{(k)} - x(k\Delta t)\right| = O\left(\Delta t^q \cdot \frac{T}{\Delta t}\right) = O\left(\Delta t^{q-1}\right) = O(\Delta t^p),$$

  and we must have $p = q - 1 > 0$ for convergence.
- This result is often the right one, but it has a hidden assumption that *errors made at previous steps do not grow* but rather stay of the same order so they can be added.
- In practice, errors made in previous time steps will either grow or shrink with time. If they grow "too fast" we are in trouble.
- So we arrive for the first time at a recurring theme: **Convergence requires stability in addition to consistency**. We discuss stability later on, after we give some basic methods for solving ODEs.

Outline

Euler's Method

- Assume that we have our approximation $x^{(k)}$ and want to move by one time step:

$$x^{(k+1)} \approx x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left(x(s), s\right) ds.$$

- The simplest possible thing is to use a piecewise constant approximation:

$$f\left(x(s), s\right) \approx f(x^{(k)}) = f^{(k)},$$

which gives the **forward Euler method**

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

- This method requires only one function evaluation per time step.

Euler's Method

$$\text{Scheme: } x^{(k+1)} - x^{(k)} - f^{(k)}\Delta t = 0$$

- The local trunction error is easy to find using a Taylor series expansion:

$$\begin{aligned} e_k &= x\left[(k+1)\,\Delta t\right] - x\left(k\Delta t\right) - f\left[x\left(k\Delta t\right)\right]\Delta t = \\ &= x\left[(k+1)\,\Delta t\right] - x\left(k\Delta t\right) - \left[x'\left(k\Delta t\right)\right]\Delta t = \frac{x''(\xi)}{2}\Delta t^2, \end{aligned}$$

  for some $k\Delta t \le \xi \le (k+1)\,\Delta t$.
- Therefore the LTE is $O(\Delta t^2)$, $q = 2$.
- The global truncation error, however, is of order $O(\Delta t)$, $p = q + 1$, so this is a **first-order accurate** method.

Backward Euler

$$x^{(k+1)} \approx x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds.$$

- How about we use a piecewise constant-approximation, but based on the end-point:

$$f\left[x(s), s\right] \approx f(x^{(k+1)}) = f^{(k+1)},$$

which gives the first-order **backward Euler method**

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

- This **implicit method** requires **solving a non-linear equation** at every time step, which is expensive and hard.
We will understand why implicit methods are needed when we examine absolute stability later on.

Runge-Kutta Methods

- **Runge-Kutta methods** are a powerful class of **one-step methods** similar to Euler's method, but more accurate.
- As an example, consider using a trapezoidal rule to approximate the integral

$$x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds \approx x^{(k)} + \frac{\Delta t}{2}\left[f\left(k\Delta t\right) + f\left(\left(k+1\right)\Delta t\right)\right],$$

$$x^{(k+1)} = x^{(k)} + \frac{\Delta t}{2}\left[f\left(x^{(k)},\ t^{(k)}\right) + f\left(x^{(k+1)},\ t^{(k+1)}\right)\right]$$

  which requires solving a nonlinear equation for $x^{(k+1)}$.
- This is the simplest **implicit Runge-Kutta method**, usually called the **implicit trapezoidal method**.
- The local truncation error is $O(\Delta t^3)$, so the global error is **second-order accurate** $O(\Delta t^2)$.

LTE: θ-method (after Iserles)

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\left[\theta \mathbf{f}\left(t_n, \mathbf{y}_n\right) + (1-\theta)\,\mathbf{f}\left(t_{n+1}, \mathbf{y}_{n+1}\right)\right]$$

$\theta = 1$ is Forward Euler, $\theta = 0$ is Backward Euler, $\theta = 1/2$ is Implicit Trapezoidal

$$\begin{aligned}
&\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h[\theta \boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + (1-\theta)\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))] \\
&= \boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h[\theta \boldsymbol{y}'(t_n) + (1-\theta)\boldsymbol{y}'(t_{n+1})] \\
&= \left[\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \tfrac{1}{2}h^2\boldsymbol{y}''(t_n) + \tfrac{1}{6}h^3\boldsymbol{y}'''(t_n)\right] - \boldsymbol{y}(t_n) \\
&\quad - h\left\{\theta \boldsymbol{y}'(t_n) + (1-\theta)\left[\boldsymbol{y}'(t_n) + h\boldsymbol{y}''(t_n) + \tfrac{1}{2}h^2\boldsymbol{y}'''(t_n)\right]\right\} + \mathcal{O}\left(h^4\right) \\
&= \left(\theta - \tfrac{1}{2}\right) h^2 \boldsymbol{y}''(t_n) + \left(\tfrac{1}{2}\theta - \tfrac{1}{3}\right) h^3 \boldsymbol{y}'''(t_n) + \mathcal{O}\left(h^4\right).
\end{aligned}$$

Midpoint/Trapezoidal Methods

- Schemes that **treat beginning and end of time step in a symmetric fashion** will lead to a **cancellation of first-order error terms** in Taylor series and will thus be **second order** (Lesson: second order is easy).
- In addition to trapezoidal one can do **implicit midpoint** scheme:

$$x^{(k+1)} = x^{(k)} + \frac{\Delta t}{2} f\left(\frac{x^{(k)} + x^{(k+1)}}{2},\ t^{(k)} + \frac{\Delta t}{2}\right)$$

  Observe this is the same as trapezoidal for linear problems (why?).
- In an explicit method, we would approximate $x^\star \approx x^{(k+1)}$ first using Euler's method, to get the simplest **explicit Runge-Kutta method**, usually called **Heun's or explicit trapezoidal method**

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

Explicit Midpoint

- **Explicit midpoint** rule

$$x^{\left(k+\frac{1}{2},\star\right)} = x^{(k)} + f\left(x^{(k)},\ t^{(k)}\right)\frac{\Delta t}{2}$$
$$x^{(k+1)} = x^{(k)} + \frac{\Delta t}{2} f\left(x^{\left(k+\frac{1}{2},\star\right)},\ t^{(k)} + \frac{\Delta t}{2}\right).$$

- Explicit midpoint/trapezoidal are a representative of a powerful class of second-order methods called **predictor-corrector methods**: Euler (forward or backward) method is the predictor, and then (implicit or explicit) trapezoidal/midpoint method is the corrector.
- One can also consider these as examples of **multi-stage one-step** methods: the predictor is the first stage, the corrector the second.

LTE: explicit midpoint (LeVeque)

$$\tau^n = \frac{1}{k}(u(t_{n+1}) - u(t_n)) - f\left(u(t_n) + \frac{1}{2}kf(u(t_n))\right). \tag{5.31}$$

Note that

$$f\left(u(t_n) + \frac{1}{2}kf(u(t_n))\right) = f\left(u(t_n) + \frac{1}{2}ku'(t_n)\right)$$
$$= f(u(t_n)) + \frac{1}{2}ku'(t_n)f'(u(t_n)) + \frac{1}{8}k^2(u'(t_n))^2 f''(u(t_n)) + \cdots.$$

Since $f(u(t_n)) = u'(t_n)$ and differentiating gives $f'(u)u' = u''$, we obtain

$$f\left(u(t_n) + \frac{1}{2}kf(u(t_n))\right) = u'(t_n) + \frac{1}{2}ku''(t_n) + O(k^2).$$

Using this in (5.31) gives

$$\tau^n = \frac{1}{k}\left(ku'(t_n) + \frac{1}{2}k^2u''(t_n) + O(k^3)\right)$$
$$- \left(u'(t_n) + \frac{1}{2}ku''(t_n) + O(k^2)\right)$$

Higher-Order Runge-Kutta Methods

- The idea in RK methods is to evaluate the function $f(x,t)$ several times and then take a time-step based on an average of the values.
- In practice, this is done by performing the calculation in **stages**: Calculate an intermediate approximation $x^\star$, evaluate $f(x^\star)$, and go to the next stage.
- The most celebrated Runge-Kutta methods is a **four-stage** fourth-order accurate RK4 method based on **Simpson's rule** for the integral:

$$x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds$$

$$\approx x^{(k)} + \frac{\Delta t}{6}\left[f(x^{(k)}) + 4f(x^{(k+1/2)}) + f(x^{(k+1)})\right]$$

$$= x^{(k)} + \frac{\Delta t}{6}\left[f^{(k)} + 4f^{(k+1/2)} + f^{(k+1)}\right],$$

and we approximate $4f^{(k+1/2)} = 2f^{(k+1/2;1)} + 2f^{(k+1/2;2)}$.

RK4 Method

$$f^{(k)} = f\left(x^{(k)}\right), \quad x^{(k+1/2;1)}, = x^{(k)} + \frac{\Delta t}{2} f^{(k)}$$

$$f^{(k+1/2;1)} = f\left(x^{(k+1/2;1)},\ t^{(k)} + \Delta t/2\right)$$

$$x^{(k+1/2;2)} = x^{(k)} + \frac{\Delta t}{2} f^{(k+1/2;1)}$$

$$f^{(k+1/2;2)} = f\left(x^{(k+1/2;2)},\ t^{(k)} + \Delta t/2\right)$$

$$x^{(k+1;1)} = x^{(k)} + \Delta t\, f^{(k+1/2;2)}$$

$$f^{(k+1)} = f\left(x^{(k+1;1)},\ t^{(k)} + \Delta t\right)$$

$$x^{(k+1)} = x^{(k)} + \frac{\Delta t}{6}\left[f^{(k)} + 2f^{(k+1/2;1)} + 2f^{(k+1/2;2)} + f^{(k+1)}\right]$$

Intro to multistep Methods

$$x^{(k+1)} \approx x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds.$$

- Euler's method was based on a piecewise constant approximation (extrapolation) of $f(s) \equiv f\left[x(s), s\right]$.
- If we instead integrate the linear extrapolation

$$f(s) \approx f\left(x^{(k)}, t^{(k)}\right) + \frac{f\left(x^{(k)}, t^{(k)}\right) - f\left(x^{(k-1)}, t^{(k-1)}\right)}{\Delta t}(s - t_k),$$

  we get the second-order **two-step Adams-Bashforth** method

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

- This is an example of a **multi-step method**, which requires keeping previous values of f.

Outline

In MATLAB

- In MATLAB, there are several functions whose names begin with

$$[\mathbf{t}, \mathbf{x}] = ode(f,\ [t_0, t_e],\ x_0,\ odeset(\dots)).$$

- *ode*23 is a second-order **adaptive explicit** Runge-Kutta method, while *ode*45 is a fourth-order version (try it first).
- *ode*23*tb* is a second-order **implicit** RK method.
- *ode*113 is a **variable-order explicit** multi-step method that can provide very high accuracy.
- *ode*15*s* is a **variable-order implicit** multi-step method.
- For implicit methods the Jacobian can be provided using the *odeset* routine – very important!

Rigid body motion

```
function dy = rigid(t,y)
dy = zeros(3,1);    % a column vector
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
%-----------

opts = odeset('RelTol',1e-3,'AbsTol',[1e-4 1e-4 1e-5]);
[T,Y] = ode45(@rigid, [0 12], [0 1 1], opts);

plot(T,Y(:,1),'o-r', T,Y(:,2),'s-b', T,Y(:,3),'d-g');
xlabel('t'); ylabel('y'); title('RelTol=1e-3');
```

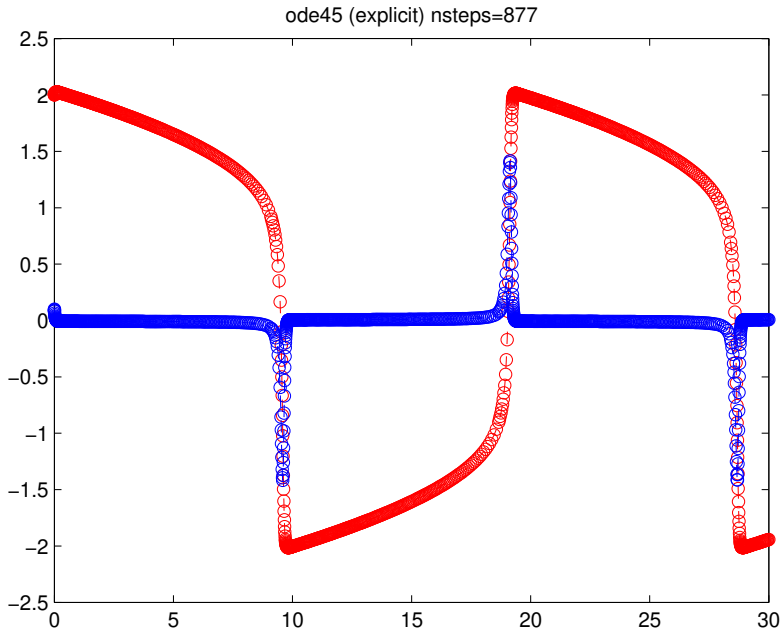
van der Pol equation

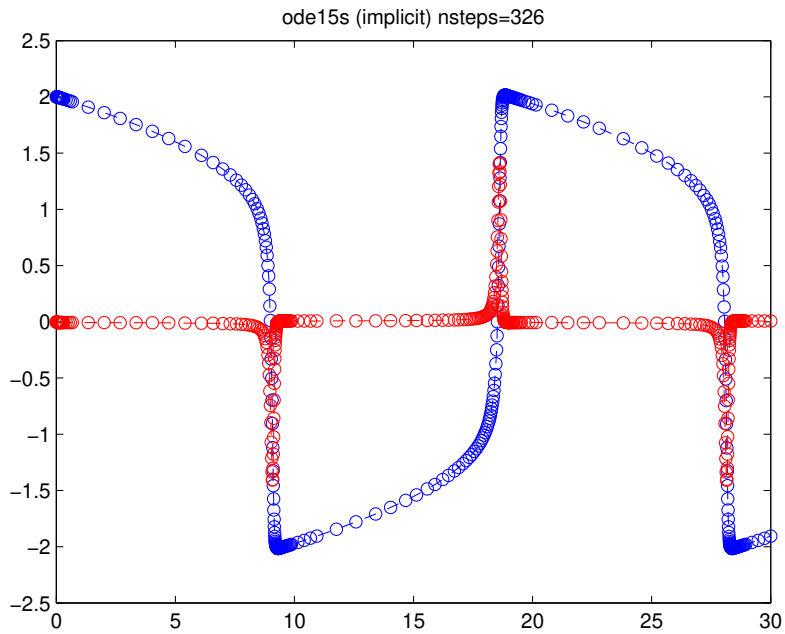
```
r=10; % Try r=100
f = @(t,y) [y(2); r*(1 - y(1)^2)*y(2) - y(1)];

figure(2); clf
[T,Y] = ode45(f,[0 3*r],[2 1]);
plot(T,Y(:,1),'o--r', T,Y(:,2)/r,'o--b')
title(['ode45 (explicit) nsteps=', int2str(size(T,1))]);

figure(3); clf
[T,Y] = ode15s(f,[0 3*r],[2 0]);
plot(T,Y(:,1),'o--b', T,Y(:,2)/r,'o--r')
title(['ode15s (implicit) nsteps=', int2str(size(T,1))]);
```

Stiff van der Pol system ($r = 10$)





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.

Outline

Zero Stability

- Consistency is not enough — we must also examine **error propagation** from one time step to another.
- A method is called **zero stable** if for all sufficiently small but finite Δt, introducing perturbations at each step (e.g., roundoff errors, errors in evaluating f) with magnitude less than some small ϵ perturbs the solution by at most $O(\epsilon)$.
- This simply means that errors do not increase but rather decrease from step to step.
- Denote our numerical approximation with time step size τ:

$$\mathbf{x}^{(k)} \approx \mathbf{x}\left(k\Delta t\right).$$

- A method is **convergent** if applying it to any system of ODEs where $\mathbf{f}$ is Lipshitz over a **finite time interval** $T > 0$ during which the ODE has a solution, the numerical approximation converges to that solution,

$$\lim_{\Delta t \to 0} \mathbf{x}^{(N=T/\Delta t)} = \mathbf{x}\left(T\right).$$

Convergence

- A central theorem in numerical methods for differential equations is that
  Any consistent method is convergent if and only if it is zero stable, or

$$\text{consistency} + \text{(zero) stability} = \text{convergence}.$$

- Note that we haven't given a precise definition to **zero stability** because in some sense it is defined as: the extra conditions that are needed to make a consistent method convergent.
- **Convergence is a statement about a limit, and does not imply a method will give reasonable answers for finite $\Delta t > 0$.**
  For that we will later introduce **absolute stability**.
- It can be shown that **one-step methods are zero-stable** if f is well-behaved (Lipschitz continuous w.r.t. second argument).

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

Instability

For $\lambda = -2100$, results for $\Delta t > 2/2100 = 0.000954$ **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 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**:

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

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.

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

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

- 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:

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

Stiffness

- If we solve the original system using Euler's method, the time step must be smaller than the smallest stability limit,

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

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

$$r = \frac{\max_i |\mathrm{Re}(\lambda_i)|}{\min_i |\mathrm{Re}(\lambda_i)|} \gg 1.$$

- For non-linear problems $\mathbf{A}$ is replaced by the Jacobian $\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{f}(\mathbf{x}, t)$.
- In general, the Jacobian will have **complex eigenvalues** as well.

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 $\mathrm{Re}(\lambda) < 0$ 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

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

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:

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

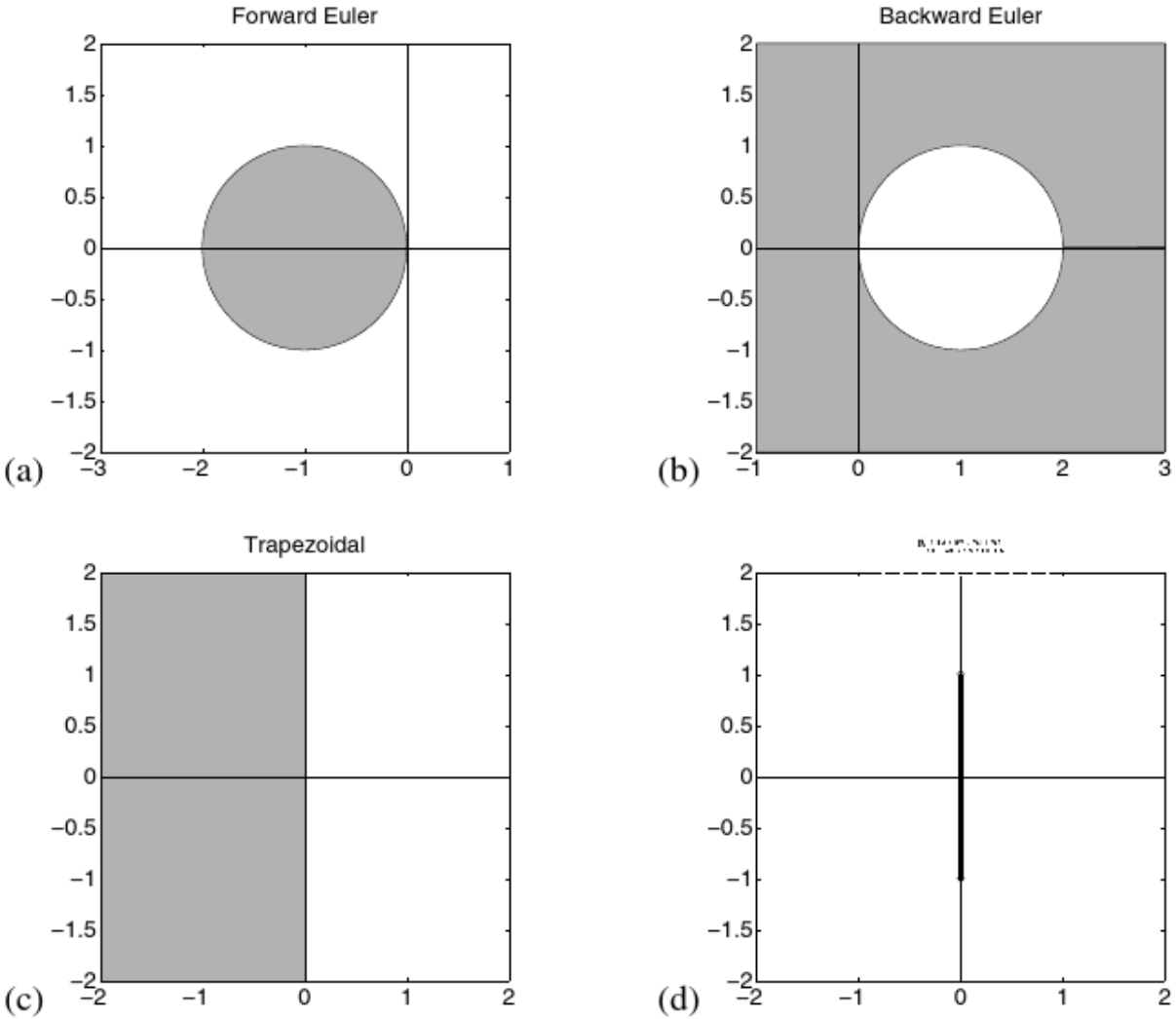
$$R(z) = e^z + O\left(z^{p+1}\right) \text{ for small } |z|.$$

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

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

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).

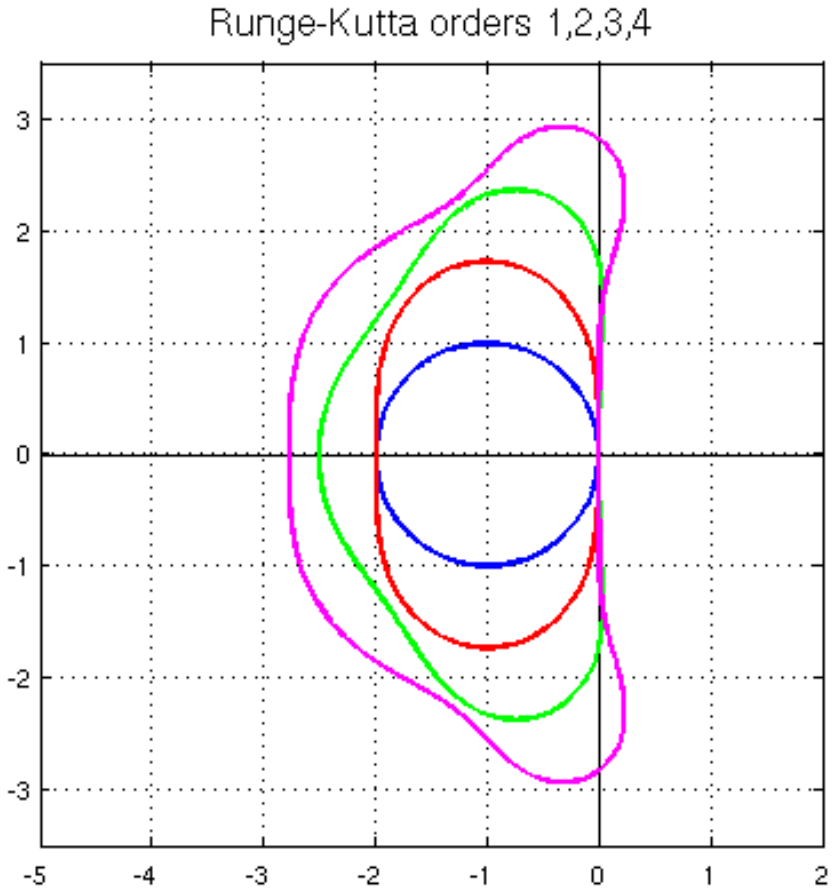
One-Step Methods

- 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)

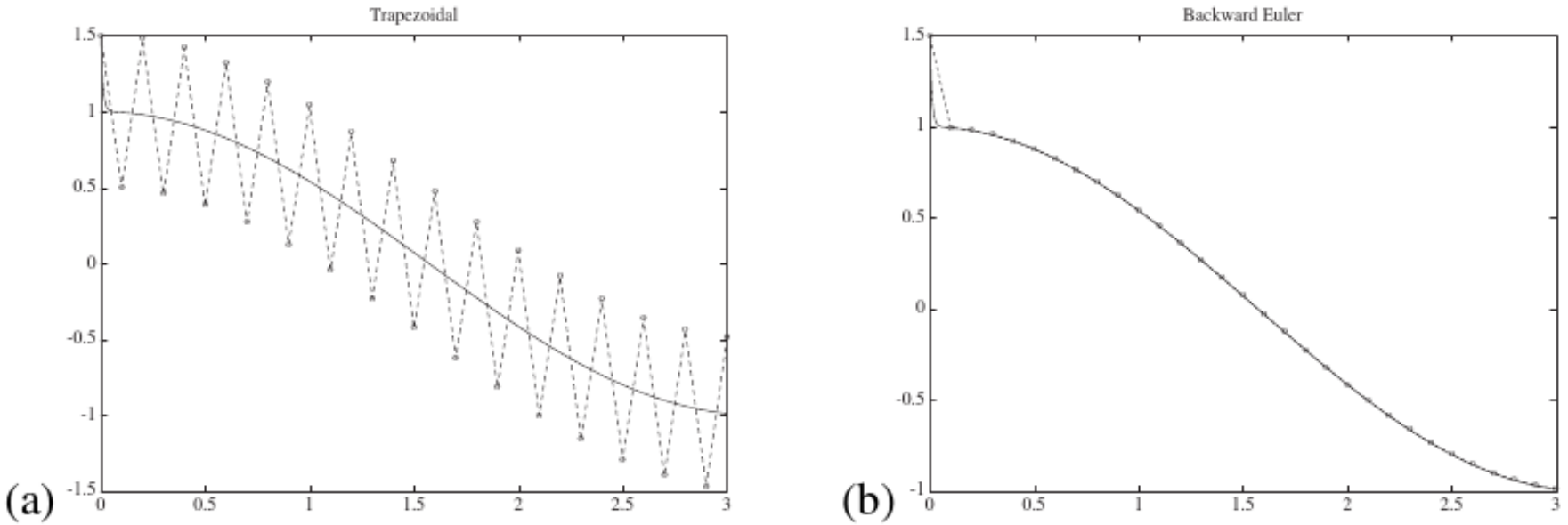


Figure 8.4. *Comparison of* (a) *trapezoidal method and* (b) *backward Euler on a stiff problem with an initial transient (Case 2 of Example 8.3).*

Implicit Methods

- **Implicit methods are** generally **more stable** than explicit methods, and solving stiff problems generally requires using an implicit method.
- 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:

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

Outline

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 (*ode*45 or *ode*23 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 (*ode*113).
- 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 (*ode*15*s*).