

# Lecture 4: Timestepping and ODEs

1. Introduction to ODEs
2. Explicit time-stepping methods
  - ▶ Forward Euler, Runge-Kutta, Adams-Bashforth
  - ▶ stability, consistency, convergence, stiffness
3. Implicit time-stepping methods

## Last Day Review

Recall from last day, we can use Taylor expansions of a smooth function  $u(x)$ :

$$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(x) + \frac{h^4}{24}u''''(x) + \dots$$

Using these and the *method of undetermined coefficients*, we can derive approximations to derivatives:

**Forward difference:**

$$u'(x) = \frac{u(x+h) - u(x)}{h} + O(h)$$

**Backward difference:**

$$u'(x) = \frac{u(x) - u(x-h)}{h} + O(h)$$

**Centered difference:**

$$u'(x) = \frac{u(x+h) - u(x-h)}{2h} + O(h^2)$$

# Ordinary differential equations: ODE IVPs

**ODE:** ordinary differential eqn (i.e., just 1 independent variable)

**IVP:** initial-value problem

First-order IVP in standard form:

$$\begin{aligned}u' &= f(t, u), \quad t > 0 \\u(0) &= \eta \quad (\text{initial data})\end{aligned}$$

We're looking for the solution, a function  $u(t)$  for  $t > 0$ .

Looks restrictive but this form is quite general.

# Ordinary differential equations: ODE IVPs

1. It could describe a system of  $N$  ODEs.

►  $\mathbf{u}(t)$  is an  $N$ -vector, so is  $\boldsymbol{\eta}$ .

$$\mathbf{u}' = f(t, \mathbf{u}), \quad t > 0$$

$$\mathbf{u}(0) = \boldsymbol{\eta} \quad (\text{initial data})$$

# Ordinary differential equations: ODE IVPs

2. Equivalence to higher-order system, e.g. harmonic oscillator

$$w'' = -w', w(0) = 1, w'(0) = 0$$

Define  $u_1 = w$ ,  $u_2 = w'$ , then we have

$$u_1' = u_2,$$

$$u_2' = -u_2.$$

with initial conditions  $u_1(0) = 1$ ,  $u_2(0) = 0$ .

In this example ODE the variable  $t$  does not appear explicitly  
( $f(t, u)$  is just  $f(u)$ ). Such an ODE is said to be **autonomous**.

## Need for numerical methods

Above simple examples are linear and can be solved analytically. Most **nonlinear ODEs**, however, cannot; we need *numerics*.

van der Pol equation, a nonlinear example

$$w'' + C(w^2 - 1)w' + w = 0,$$

for fixed  $C > 0$ . For  $|w| > 1$ , a damped oscillator; for  $|w| < 1$ , negatively damped. Big solutions decay; small solutions grow. As  $t \rightarrow \infty$  we have convergence to a **limit cycle** of size  $O(1)$ .

Equivalent first-order system:

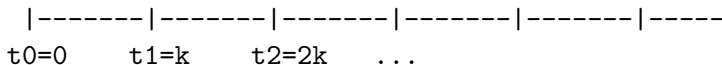
$$\begin{aligned}u_1' &= u_2, \\u_2' &= -u_1 - C[(u_1)^2 - 1]u_2.\end{aligned}$$

[m01\_vanderpol.m]



# Time discretization

A grid in time:  $t_n = nk$ ,  $k > 0$  a fixed **time step**. Often “ $\Delta t$ ” or “ $\tau$ ” used instead.



(often in practice  $k$  not fixed, and is changed automatically).

The exact solution (usually unknown) is  $u(t)$  defined for all continuous values of  $t$ . We will find a numerical solution at these discrete points  $v_n \approx u(t_n)$ .

# Forward Euler method

The simplest method is the forward Euler method:

$$v^0 = \eta \text{ (initial condition)}$$

$$v^1 = v^0 + kf(v^0)$$

$$v^2 = v^1 + kf(v^1)$$

... etc.

# Forward Euler method

We are *time-stepping* with the formula

$$v^{n+1} = v^n + kf(t_n, v^n)$$

And note this is the “forward difference” we saw earlier. Matlab codes: [m02\_euler.m], [m03\_euler\_graph.m].

We often want methods which are more accurate, or more stable, or have other features.

## Methods with smaller errors

- ▶ Under some assumptions (smooth enough solution) the forward Euler method has a (global) error which behaves like  $Ck$  (notation:  $error = O(k)$ ).
- ▶ Roughly speaking if we want 6 digits of accuracy, we probably need about one million steps. Often we want more accuracy for less work. Money for nuthin'!
- ▶ Compared to forward Euler, we often want methods which are more accurate, or more stable, or have other features. There are two main classes of methods: Runge–Kutta (“1-step”); Adams–[others] (“multistep”)

# Adams-Bashforth multistep formulas

Notation: define  $f^n = f(t_n, v^n)$ .

$v^{n+1} = v^n + kf^n$ : *forward Euler* again, accuracy  $O(k)$

$v^{n+1} = v^n + \frac{k}{2}(3f^n - f^{n-1})$ : accuracy  $O(k^2)$

$v^{n+1} = v^n + \frac{k}{12}(23f^n - 16f^{n-1} + 5f^{n-2})$ : accuracy  $O(k^3)$

$v^{n+1} = v^n + \frac{k}{24}(55f^n - 59f^{n-1} + 37f^{n-2} - 9f^{n-3})$ : accuracy  $O(k^4)$

...

Adams-Bashforth uses previous values, not just the current one.  
[modify our code]. Tricky to start up because extra values are needed.

# Runge-Kutta methods

Use temporary intermediate values (called *stage values*).

1. Forward Euler (again)  $O(k)$

2. “Modified Euler”  $O(k^2)$

$$a = kf(t_n, v^n),$$

$$b = kf(t_n + k/2, v^n + a/2),$$

$$v^{n+1} = v^n + b.$$

### 3. “The Fourth-order Runge-Kutta Method” $O(k^4)$

$$a = kf(t_n, v^n),$$

$$b = kf(t_n + k/2, v^n + a/2),$$

$$c = kf(t_n + k/2, v^n + b/2),$$

$$d = kf(t_n + k, v^n + c),$$

$$v^{n+1} = v^n + \frac{1}{6}(a + 2b + 2c + d).$$

If you could take just one formula to a desert island, this is it.

Higher-order RK formulas get very complicated. The coefficients of Runge-Kutta methods can be stored in a “Butcher Tableau”.

# IVP codes in higher-level software

- ▶ ODE codes are among the most successful algorithms (and software) in all of scientific computing. The key is that they are **adaptive** – varying step size, and sometimes order, automatically.
- ▶ Matlab has many codes, but in my experience, most people use ode45 and ode15s. These differ in numerics but are basically the same in syntax. For info see:

L. F. Shampine and M. W. Reichel, The Matlab ODE suite, SIAM J. Sci. Comput. 18 (1997), 1-22.



# Analysis

We would like to understand how the error of these various methods behaves.

Some concepts we will need:

- ▶ consistency and local truncation error
- ▶ stability
- ▶ convergence

# The local truncation error

- ▶ Substitute the true solution of the differential equation into the discrete problem. It will not satisfy it exactly: the discrepancy is the called the *local truncation error (LTE)*.

**symbol:**  $\tau$

- ▶ Use Taylor expansions (very similar to the derivation of forward difference, etc). Exact solution unknown, assumed smooth.
- ▶ Express LTE in “big Oh” notation in  $k$ .

# Consistency

LTE to zero as  $k \rightarrow 0$ .

example: 2nd-order Adams-Bashforth

$$v^{n+1} = v^n + \frac{k}{2}(3f^n - f^{n-1})$$

## example: 2nd-order Adams-Bashforth

$$v^{n+1} = v^n + \frac{k}{2}(3f^n - f^{n-1})$$

To determine the LTE, it is important to write this in a form similar to the DE:  $u' = f(t, u)$

$$\frac{v^{n+1} - v^n}{k} = \frac{1}{2}(3f^n - f^{n-1})$$

$$\frac{v^{n+1} - v^n}{k} = \frac{1}{2}(3f^n - f^{n-1})$$

Now expand each with Taylor:  $u(t_n + k)$  and  $f(t_n - k)$  (remember  $u' = f$ ):

$$u(t + k) = u + ku' + (1/2)k^2u'' + (1/6)k^3u''' + \dots$$

$$f(t - k) = u' - ku'' + (1/2)k^2u''' - \dots$$

$$\tau = LHS - RHS...$$

Substitute in, find the amount by which the exact solution  $u(t_{n+1})$  fails to satisfy the discrete equation:

$$\tau = \frac{u(t_n + k) - u(t_n)}{k} \left( \frac{1}{2}(3f^n - f^{n-1}) \right) = \dots = -(5/12)k^2 u''' + \dots$$

So the local truncation error is  $\tau = O(k^2)$ . And as this goes to zero as  $k \rightarrow 0$ , we can say this method is *consistent*.

# Runge–Kutta

Doing this sort of Taylor analysis for RK leads to *order conditions*:

*“These computations, which are not reproduced in Kutta’s 1905 paper (they are, however, in Heun 1900), are very tedious. And they grow enormously with higher-orders. We shall see. . . using an appropriate notation, they can become very elegant.” [Hairer, Wanner, Norsett]*

## Aside: LTE and consistency for ODEs (optional)

There are two basic contradictory definitions of LTE in the literature (oops). Depends on where you apply the Taylor expansions. E.g., for forward Euler:

1. As we did above: apply the Taylor analysis to  $\frac{u^{n+1}-u^n}{k} = f(u^n)$ . This gives LTE of  $O(k)$ .
2. Apply the Taylor analysis to  $u^{n+1} = u^n + kf(u^n)$ . This is sometimes called the *one-step error*: under this, LTE is  $O(k^2)$ . But we apply  $O(1/k)$  steps for an estimated/expected global error of  $O(k)$ .



# Stable and unstable methods

- ▶ Having a small LTE isn't sufficient to tell us that the sequence actually converges! Recall that the method will be used repeatedly: an unstable method is one that where each error (from the LTE, rounding error or whatever) builds up (e.g., over time).
- ▶ A stable method is one where small errors are “damped out”.

## Stable and unstable methods

- ▶ Here's an example of an unstable formula with order 3, try it and see how the computed solutions blow up as  $k \rightarrow 0$ .

$$v^{n+1} = -4v^n + 5v^{n-1} + k(4f^n + 2f^{n-1})$$

[m05\_unstab.m: an example of a multistep method that is consistent... but unstable.]

[m05\_unstab.m: also an example of a non-self-starting method: use forward Euler or exact solution, either way still unstable.]

# Convergence

Convergence says the actual overall global error (say at  $t = T_f$  some final time) decreases as  $k \rightarrow 0$ . This is the property we want.

Intuitively, global convergence requires both consistency and stability. And in many cases this is enough. . .

# Fundamental Theorem

Fundamental theorem of finite difference methods (or “of numerical analysis”?)

$$\text{Consistency} + \text{Stability} \iff \text{Convergence}$$

We will see this again-and-again. The notion of stability might change, depending on the situation (generally stability harder than consistency).

For multistep methods, this is the **Dahlquist Equivalence Theorem**.

# Absolute Stability for Runge–Kutta methods

- ▶ Dahlquist test problem:  $u' = \lambda u$ ,  $u(0) = 1$ .
- ▶ Here  $\lambda$  is a complex number. Exact solution is  $u(t) = \exp(\lambda t)$ . Real part of  $\lambda$  negative gives decay in the exact solution.
- ▶ We apply the numerical method to this test problem and:

---

If the solution grows without bound, we say **unstable**, otherwise, we say the numerical method is **stable**.

---

## Example: forward Euler

$$v^{n+1} = v^n + k\lambda v^n,$$

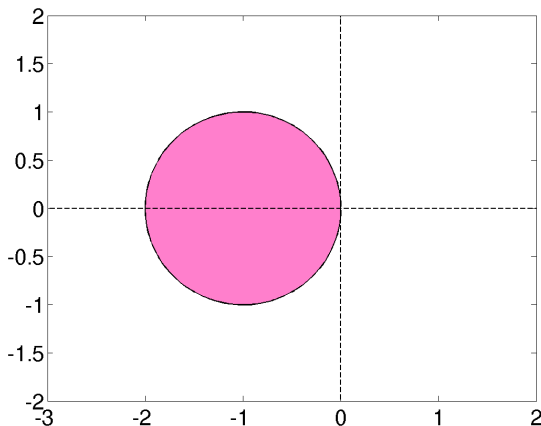
$$v^{n+1} = (1 + k\lambda)v^n.$$

But we can apply this all the way back to  $t_0$ :

$$v^{n+1} = (1 + k\lambda)^{n+1}v^0 = (1 + k\lambda)^{n+1}.$$

Thus for forward Euler, the soln will blow-up as  $n \rightarrow \infty$  unless  $|1 + k\lambda| \leq 1$ . If  $\lambda$  is real, this means

Or in the complex plane, we have a **absolute stability region** for forward Euler  $z = k\lambda$  consisting of a unit disc centered at  $z = -1$ .



# Absolute stability

---

For a given  $\lambda$ , and a given numerical method, we need to choose  $k$  such that  $k\lambda$  lives inside the stability region.

---

- ▶ Different RK and MS methods have different absolute stability regions.



## Why is this a reasonable test problem?

- ▶ This analysis is not much use if it only applies to this one problem. . .
- ▶ Works well if (in theory) we can linearize our problem, and apply an eigenvalue decomposition. Each component of the diagonalized linear system will then behave like the Dahlquist test problem.

# Stiffness

- ▶ A stiff ODE is one with widely varying time scales.
- ▶ More precisely, an ODE with solution of interest  $u(t)$  is stiff when there are time scales present in the equation that are much shorter than that of  $u(t)$  itself.

Neither is an ideal definition.

- ▶ A quote from HW: “Stiff equations are problems for which explicit methods don’t work”.
- ▶ a stiff *problem* is one where implicit methods work better.
- ▶ C.W. Gear, 1982:

*Although it is common to talk about “stiff differential equations,” an equation per se is not stiff, a particular initial value problem for that equation may be stiff, in some regions, but the size of these regions depend on the initial values and the error tolerance.*

## Why it matters:

- ▶ regardless of stiffness, you'll get convergence as  $k \rightarrow 0$  (Dahlquist's original theory).
- ▶ But the result might be rubbish except when  $k$  is very small, because of modes  $k\lambda$  that aren't in the stability region.

## Example

ODE

$$u' = -\sin(t)$$

with IC  $u(0) = 1$  has solution  $u(t) = \cos(t)$ .

Change ODE to

$$u' = -100(u(t) - \cos(t)) - \sin(t),$$

then this *still* has solution  $u(t) = \cos(t)$ .

**But** the numerics are much different!

- ▶ [m07\_stiff\_fe\_be.m: a convergence study showing forward Euler convergences for very small  $k$  but is unstable for larger  $k$ .]
- ▶ We can also linearize around the soln: let  $u(t) = \cos(t) + w(t)$  and we get an ODE for  $w(t)$  of  $w' = -100w$  which does indeed have a very different *time scale* than  $\cos(t)$ .

# Introduction to implicit methods

(Part of) the cure for stiffness? Use implicit methods. E.g., backward Euler method:

$$v^{n+1} = v^n + kf^{n+1} = v^n + kf(t_{n+1}, v^{n+1}).$$

- ▶ Note  $v^{n+1}$ , *the thing we want to solve for*, is inside the function  $f \rightarrow$  implicit.
- ▶ [m07\_stiff\_fe\_be.m: implement backward Euler and note no instability]

# A-stability

Recall the exact solution of  $u' = \lambda u$  is stable (the solution itself, nothing about numerical methods) if  $\operatorname{Re}(\lambda) < 0$ . Seems desirable to have numerical methods that are also stable in this way.

Want their linear stability regions to include the left-hand of the complex plane.



**Defn** [Dahlquist 1963]: a method, whose stability region includes the left-hand complex plane is called **A-stable**.

---

We also expect these to work well for stiff problems b/c  $k\lambda$  (for any  $\operatorname{Re}(\lambda) < 0$ ) will be in the stability region.

---

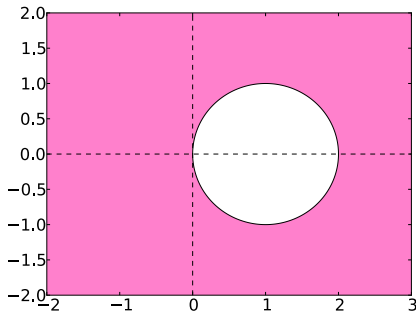
Dahlquist in 1979:

*I didn't like all these "strong", "perfect", "absolute", "generalized", "super", "hyper", "complete" and so on in mathematical definitions, I wanted something neutral; and having been impressed by David Young's "Property A", I choose the term "A-stable".*

## Example, backward Euler

$$v^{n+1} = v^n + kf(t_n + k, v^{n+1}) = v^n + kf^{n+1}$$

Using Dahlquist test problem, can show it is stable in the exterior of a unit disc centered at 1 in the *right-half* complex plane.



Hairer & Wanner: “The backward Euler method is a *very* stable method.”

(There are also implicit multistep and implicit Runge-Kutta.)

### Solving the system

Implicit methods have both  $v^{n+1}$  and  $f^{n+1}$ : we need to solve a linear or nonlinear system to advance the step. Linear algebra if linear, something like Newton’s method if nonlinear. We do not discuss this issue further.