

## Lecture 11: Time stepping methods

COMP5930M Scientific Computation

# Today

Time stepping for PDE models

Classes of methods

Stability

Implications for nonlinear systems

Time stepping process

Next

# Time stepping



- ▶ In the Method of Lines framework we define a semi-discrete form of our partial differential equation (PDE) as a coupled system of ordinary differential equations (ODEs)

For example, a typical discrete 1D PDE at node  $i$  is written as:

$$\dot{\mathbf{u}}_i = \mathbf{f}(\mathbf{u}_{i-1}, \mathbf{u}_i, \mathbf{u}_{i+1})$$

- ▶ In this form, many standard numerical methods for ODEs can be applied

## First-order accurate methods

- ▶ Forward (explicit) Euler:

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} = \mathbf{f}(\mathbf{u}^k)$$

- ▶ Backward (implicit) Euler:

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} = \mathbf{f}(\mathbf{u}^{k+1})$$

Both are  $\mathcal{O}(\Delta t)$  accurate (termed, first order)

# Implicit or Explicit methods?

## Efficiency trade-offs

- ▶ Explicit - many short, cheap timesteps
- ▶ Implicit - fewer long, expensive timesteps

## Accuracy requirements

- ▶ Explicit timestep often limited for stability, e.g.  $\Delta t < Ch^2$
- ▶ Implicit timestep often limited for accuracy, by  $\Delta t < Ch$

## Higher accuracy in time



- ▶ For efficient computation of time-dependent PDE problems we should balance the errors in time and space
- ▶ FDM applied to many standard second-order (in space) PDE models leads to a spatial error of  $\mathcal{O}(h^2)$
- ▶ After applying an approximation in time:
  - ▶ An  $\mathcal{O}(\Delta t)$  scheme leads to a total error  $E \propto \mathcal{O}(\Delta t, h^2)$
  - ▶ An  $\mathcal{O}(\Delta t^2)$  scheme leads to a total error  $E \propto \mathcal{O}(\Delta t^2, h^2)$

Need to find time-stepping method at least  $\mathcal{O}(\Delta t^2)$  accurate

## The $\theta$ -method



Idea: Linear combination of the implicit and explicit Euler methods

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} = \theta \mathbf{f}(\mathbf{u}^{k+1}) + (1 - \theta) \mathbf{f}(\mathbf{u}^k)$$

for  $0 \leq \theta \leq 1$

- ▶  $\theta = 0$  gives explicit Forward Euler,  $\mathcal{O}(\Delta t)$  accurate
- ▶  $\theta = 1$  gives implicit Backward Euler,  $\mathcal{O}(\Delta t)$  accurate
- ▶  $\theta = \frac{1}{2}$  gives an implicit,  $\mathcal{O}(\Delta t^2)$  accurate scheme
  - known as the trapezoidal rule or Crank-Nicolson method<sup>1</sup>

---

<sup>1</sup>Named after Phyllis Nicolson, a physicist at University of Leeds

## Time stepping families

- ▶ It is possible to define methods of arbitrarily high order
- ▶ Runge-Kutta family

- ▶ One-step methods:

$\mathbf{u}^{k+1}$  depends on  $\mathbf{u}^k$  only

but  $\mathbf{f}(\mathbf{u})$  is evaluated multiple times per step (stages)

- ▶ Commonly explicit methods (RK4), implicit also possible



## Time stepping families

- ▶ It is possible to define methods of arbitrarily high order
- ▶ Runge-Kutta family

- ▶ One-step methods:

$\mathbf{u}^{k+1}$  depends on  $\mathbf{u}^k$  only

but  $\mathbf{f}(\mathbf{u})$  is evaluated multiple times per step (stages)

- ▶ Commonly explicit methods (RK4), implicit also possible
- ▶ Backward differentiation formulae (BDF)
  - ▶ Multi-step methods:

$\mathbf{u}^{k+1}$  depends on  $\mathbf{u}^k, \mathbf{u}^{k-1}, \dots, \mathbf{u}^{k-m+1}$

for a general  $m$ -step method

- ▶ Implicit methods, balance between stability and cost

## Time stepping families



- ▶ It is possible to define methods of arbitrarily high order

- ▶ **Runge-Kutta family**

- ▶ One-step methods:

$\mathbf{u}^{k+1}$  depends on  $\mathbf{u}^k$  only

but  $\mathbf{f}(\mathbf{u})$  is evaluated multiple times per step (stages)



- ▶ Commonly explicit methods (RK4), implicit also possible

- ▶ **Backward differentiation formulae (BDF)**

- ▶ Multi-step methods:

$\mathbf{u}^{k+1}$  depends on  $\mathbf{u}^k, \mathbf{u}^{k-1}, \dots, \mathbf{u}^{k-m+1}$

for a general  $m$ -step method

- ▶ **Implicit methods, balance between stability and cost**

## Runge-Kutta methods

- ▶ Simplest example is Explicit Euler

$$\begin{aligned}\mathbf{k}_1 &= \mathbf{f}(\mathbf{u}^k) && \text{(stage 1)} \\ \mathbf{u}^{k+1} &= \mathbf{u}^k + \Delta t \mathbf{k}_1\end{aligned}$$

- ▶ A second-order (explicit) scheme

$$\begin{aligned}\mathbf{k}_1 &= \mathbf{f}(\mathbf{u}^k) && \text{(stage 1)} \\ \mathbf{k}_2 &= \mathbf{f}\left(\mathbf{u}^k + \frac{\Delta t}{2} \mathbf{k}_1\right) && \text{(stage 2)} \\ \mathbf{u}^{k+1} &= \mathbf{u}^k + \Delta t \mathbf{k}_2\end{aligned}$$

## General Runge-Kutta method with $s$ stages

For the general problem  $\mathbf{u}'(t) = \mathbf{f}(t, \mathbf{u}(t))$ :

$$\mathbf{k}_i = \mathbf{f}\left(t_k + c_i \Delta t, \mathbf{u}^k + \Delta t \sum_{j=1}^s a_{ij} \mathbf{k}_j\right), \quad i = 1, \dots, s \quad (\text{stage } i)$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \Delta t \sum_{i=1}^s b_i \mathbf{k}_i \quad (\text{update})$$

- ▶ If  $a_{ij} = 0$  for all  $i \leq j$  method is explicit, otherwise implicit
- ▶ Implicit R-K leads to nonlinear system of equations to solve for  $\mathbf{k}_1, \dots, \mathbf{k}_s$  at each time step  $\Rightarrow$  full system is of size  $n \times s$ !

## Backward Differentiation Method (BDF- $m$ )

- **Idea:** Use  $m$  previous solutions  $\mathbf{u}^k, \dots, \mathbf{u}^{k-m+1}$  to construct Lagrange interpolating polynomial  $\mathbf{p}_m(t)$  of degree  $m$  s.t.

$$\mathbf{p}_m(t_{k-i+1}) = \mathbf{u}^{k-m+1}, \quad i = 0, \dots, m$$

and solve  $\mathbf{p}'_m(t_k + \Delta t) = \mathbf{f}(\mathbf{u}^{k+1})$  to find  $\mathbf{u}^{k+1}$ .

- Simplest example is Implicit Euler (BDF-1)

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} = \mathbf{f}(\mathbf{u}^{k+1})$$

## Backward Differentiation Method (BDF- $m$ )

- **Idea:** Use  $m$  previous solutions  $\mathbf{u}^k, \dots, \mathbf{u}^{k-m+1}$  to construct Lagrange interpolating polynomial  $\mathbf{p}_m(t)$  of degree  $m$  s.t.

$$\mathbf{p}_m(t_{k-i+1}) = \mathbf{u}^{k-m+1}, \quad i = 0, \dots, m$$

and solve  $\mathbf{p}'_m(t_k + \Delta t) = \mathbf{f}(\mathbf{u}^{k+1})$  to find  $\mathbf{u}^{k+1}$ .

- Simplest example is Implicit Euler (BDF-1)

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} = \mathbf{f}(\mathbf{u}^{k+1})$$

- The second order scheme (BDF-2)

$$\frac{3\mathbf{u}^{k+1} - 4\mathbf{u}^k + \mathbf{u}^{k-1}}{2\Delta t} = \mathbf{f}(\mathbf{u}^{k+1})$$

## Backward Differentiation Method (BDF- $m$ )

- ▶ **Idea:** Use  $m$  previous solutions  $\mathbf{u}^k, \dots, \mathbf{u}^{k-m+1}$  to construct Lagrange interpolating polynomial  $\mathbf{p}_m(t)$  of degree  $m$  s.t.

$$\mathbf{p}_m(t_{k-i+1}) = \mathbf{u}^{k-m+1}, \quad i = 0, \dots, m$$

and solve  $\mathbf{p}'_m(t_k + \Delta t) = \mathbf{f}(\mathbf{u}^{k+1})$  to find  $\mathbf{u}^{k+1}$ .

- ▶ Simplest example is **Implicit Euler (BDF-1)**

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} = \mathbf{f}(\mathbf{u}^{k+1})$$

- ▶ **The second order scheme (BDF-2)**

$$\frac{3\mathbf{u}^{k+1} - 4\mathbf{u}^k + \mathbf{u}^{k-1}}{2\Delta t} = \mathbf{f}(\mathbf{u}^{k+1})$$

## Stability of a time-stepping method

- ▶ There are many different definitions of (numerical) stability
- ▶ Time-stepping method is absolutely stable (**A-stable**) if the numerical solution applied to the linear test problem:

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

approaches zero,  $u_k \rightarrow 0$  as  $k \rightarrow \infty$ , whenever  $\operatorname{Re}(\lambda) < 0$  and for any fixed step size  $\Delta t > 0$ .



## Stability of a time-stepping method

- ▶ There are many different definitions of (numerical) stability
- ▶ Time-stepping method is absolutely stable (**A-stable**) if the numerical solution applied to the linear test problem:

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

approaches zero,  $u_k \rightarrow 0$  as  $k \rightarrow \infty$ , whenever  $\operatorname{Re}(\lambda) < 0$  and for any fixed step size  $\Delta t > 0$ .

- ▶ Exact solution is  $u(t) = \exp(\lambda t)$ , which goes to 0 as  $t \rightarrow \infty$  if and only if  $\operatorname{Re}(\lambda) < 0$

## Stability of a time-stepping method

- ▶ There are many different definitions of (numerical) stability
- ▶ Time-stepping method is absolutely **stable (A-stable)** if the numerical solution applied to the linear test problem:

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

approaches zero,  $u_k \rightarrow 0$  as  $k \rightarrow \infty$ , whenever  $\operatorname{Re}(\lambda) < 0$  and for any fixed step size  $\Delta t > 0$ .

- ▶ Exact solution is  $u(t) = \exp(\lambda t)$ , which goes to 0 as  $t \rightarrow \infty$  if and only if  $\operatorname{Re}(\lambda) < 0$

## Example: Stability of Euler methods

Explicit Euler:

$$u^{k+1} = u^k + \lambda \Delta t u^k = (1 + \lambda \Delta t) u^k = R(z) u^k$$

where  $R(z) = R(\lambda \Delta t)$  is called the **stability function**.

The method is stable for  $|R(z)| < 1$ .

## Example: Stability of Euler methods

Explicit Euler:

$$u^{k+1} = u^k + \lambda \Delta t u^k = (1 + \lambda \Delta t) u^k = R(z) u^k$$

where  $R(z) = R(\lambda \Delta t)$  is called the **stability function**.

The method is stable for  $|R(z)| < 1$ .

Stability region:  $|1 + z| < 1$

$\Rightarrow$  disc of radius 1 centered at  $z = -1$

## Example: Stability of Euler methods

Explicit Euler:

$$u^{k+1} = u^k + \lambda \Delta t u^k = (1 + \lambda \Delta t) u^k = R(z) u^k$$

where  $R(z) = R(\lambda \Delta t)$  is called the **stability function**.

The method is stable for  $|R(z)| < 1$ .

Stability region:  $|1 + z| < 1$

$\Rightarrow$  disc of radius 1 centered at  $z = -1$

## Example: Stability of Euler methods

Implicit Euler:

$$u^{k+1} = u^k + \lambda \Delta t u^{k+1}$$

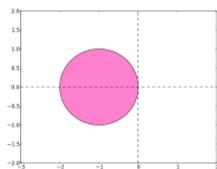
so that the stability function is  $R(z) = 1/(1 - z)$ .

Stability region:  $|1 - z| > 1$

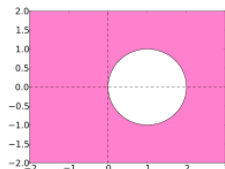
$\Rightarrow$  complex plane except for a disc of radius 1 centered at  $z = 1$

## Stability regions of different methods

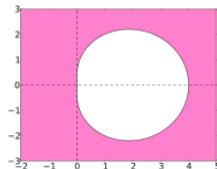
We can illustrate A-stability by coloring the points  $\lambda\Delta t \in \mathbb{C}$  for which different methods are stable for the problem  $u'(t) = \lambda u(t)$ :



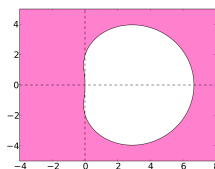
Forward Euler



Backward Euler



BDF-2



BDF-3

## Properties of BDF methods

- ▶ BDF- $m$  methods can be defined for any  $m = 1, 2, 3, \dots$
- ▶ BDF-1 and BDF-2 are A-stable
- ▶ BDF-3 almost A-stable (except small region near  $Im$  axis)
- ▶ BDF- $m$  for  $m \geq 6$  is unstable, do not use



## Application to nonlinear systems

When choosing the time stepping method, we should consider:

- ▶ Computational complexity
  - ▶ Explicit Runge-Kutta method: no nonlinear equation solves
- ▶ Stability and accuracy

## Application to nonlinear systems

When choosing the time stepping method, we should consider:

- ▶ Computational complexity
  - ▶ Explicit Runge-Kutta method: no nonlinear equation solves
  - ▶ Implicit Runge-Kutta ( $s$  stages): solve nonlinear system of  $s \times n$  equations on each time step
- ▶ Stability and accuracy

## Application to nonlinear systems

When choosing the time stepping method, we should consider:

- ▶ Computational complexity
  - ▶ Explicit Runge-Kutta method: no nonlinear equation solves
  - ▶ Implicit Runge-Kutta ( $s$  stages): solve nonlinear system of  $s \times n$  equations on each time step
  - ▶ BDF method: solve nonlinear system of  $n$  equations on each time step, but need to store  $m$  previous solutions
- ▶ Stability and accuracy

## Application to nonlinear systems

When choosing the time stepping method, we should consider:

- ▶ **Computational complexity**
  - ▶ Explicit Runge-Kutta method: no nonlinear equation solves
  - ▶ Implicit Runge-Kutta ( $s$  stages): solve nonlinear system of  $s \times n$  equations on each time step
  - ▶ BDF method: solve nonlinear system of  $n$  equations on each time step, but need to store  $m$  previous solutions
- ▶ **Stability and accuracy**
  - ▶ First-order methods not accurate enough

## Application to nonlinear systems

When choosing the time stepping method, we should consider:

- ▶ Computational complexity
  - ▶ Explicit Runge-Kutta method: no nonlinear equation solves
  - ▶ Implicit Runge-Kutta ( $s$  stages): solve nonlinear system of  $s \times n$  equations on each time step
- ▶ Stability and accuracy
  - ▶ First-order methods not accurate enough
  - ▶ Some problems are stiff and explicit methods require  $\Delta t \ll 1$

## Application to nonlinear systems

When choosing the time stepping method, we should consider:

- ▶ Computational complexity
  - ▶ Explicit Runge-Kutta method: no nonlinear equation solves
  - ▶ Implicit Runge-Kutta ( $s$  stages): solve nonlinear system of  $s \times n$  equations on each time step
  - ▶ BDF method: solve nonlinear system of  $n$  equations on each time step, but need to store  $m$  previous solutions
- ▶ Stability and accuracy
  - ▶ First-order methods not accurate enough
  - ▶ Some problems are stiff and explicit methods require  $\Delta t \ll 1$
  - ▶ Some A-stable methods have too much damping that eliminates high-frequency oscillations in the solution

## Application to nonlinear systems

When choosing the time stepping method, we should consider:

- ▶ Computational complexity
  - ▶ Explicit Runge-Kutta method: no nonlinear equation solves
  - ▶ Implicit Runge-Kutta ( $s$  stages): solve nonlinear system of  $s \times n$  equations on each time step
  - ▶ BDF method: solve nonlinear system of  $n$  equations on each time step, but need to store  $m$  previous solutions
- ▶ Stability and accuracy
  - ▶ First-order methods not accurate enough
  - ▶ Some problems are stiff and explicit methods require  $\Delta t \ll 1$
  - ▶ Some A-stable methods have too much damping that eliminates high-frequency oscillations in the solution

## The time stepping process

1.  $t = t^0$ ,  $\mathbf{u}^0 = \mathbf{U}(x, t^0)$
  2. for  $k = 0, 1, \dots$ 
    - 2.1 set initial guess  $\mathbf{u}_0^{k+1} = \mathbf{u}^k$
    - 2.2  $\mathbf{u}^{k+1} = \text{Newton}(\mathbf{F}(\mathbf{u}^{k+1}), \mathbf{u}_0^{k+1}, \Delta t, Tol)$
    - 2.3  $t = t + \Delta t$
- ▶ For small  $\Delta t$  the initial guess  $\mathbf{u}_0^{k+1}$  will be accurate
  - ▶ For very large  $\Delta t$  we may have problems with convergence
  - ▶  $Tol$  should be chosen sufficiently small so as not to compromise the accuracy of the PDE model



## Summary

- ▶ High-order time stepping methods balance the discretisation error in time and space and outperform first order methods
- ▶ Explicit Runge-Kutta methods usually not appropriate for PDEs due to stringent stability requirements ( $\Delta t \ll 1$ )
- ▶ Implicit BDF methods offer a good balance between stability and computational cost
- ▶ Time-stepping method should be chosen based on the physics of the specific problem being solved

## Next time

- ▶ Lecture
  - ▶ Approximation of 2-d nonlinear PDEs
  - ▶ Sparse system structure and efficient algorithms