July 20, 2013

Chapter 16: Differential Equations

Uri M. Ascher and Chen Greif
Department of Computer Science
The University of British Columbia
{ascher,greif}@cs.ubc.ca

Slides for the book
A First Course in Numerical Methods (published by SIAM, 2011)
`http://www.ec-securehost.com/SIAM/CS07.html`

Goals of this chapter

- To develop useful methods for simulating initial value ordinary differential equations;
- to understand several basic concepts and challenges that are central to a large portion of practical numerical computing today;
- * to get a glimpse at solving problems involving partial differential equations.

Outline

- Differential equations
- Euler's method
- Runge-Kutta methods
- Multistep methods
- Absolute stability and stiffness
- Error control and estimation
- *Partial differential equations

*advanced

Differential equations

- Arise in all branches of science and engineering, economics, finance, computer science.
- Relate physical state to rate of change. e.g., rate of change of particle is velocity

$$\frac{dx}{dt} = v(t) = g(t, x), \quad a < t < b.$$

- Ordinary differential equation (ODE): one independent variable ("**time**").
- Partial differential equation (PDE): several independent variables.

Differential equations

- Arise in all branches of science and engineering, economics, finance, computer science.
- Relate physical state to rate of change. e.g., rate of change of particle is velocity

$$\frac{dx}{dt} = v(t) = g(t, x), \quad a < t < b.$$

- Ordinary differential equation (ODE): one independent variable (“**time**”).
- Partial differential equation (PDE): several independent variables.

Differential equations

- Arise in all branches of science and engineering, economics, finance, computer science.
- Relate physical state to rate of change. e.g., rate of change of particle is velocity

$$\frac{dx}{dt} = v(t) = g(t, x), \quad a < t < b.$$

- Ordinary differential equation (ODE): one independent variable ("**time**").
- Partial differential equation (PDE): several independent variables.

Ordinary differential equations

- ODEs usually arise as a *system* of differential equations.
- Write in standard form:

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \quad a < t < b,$$

where t is the independent variable, and $\mathbf{y} = \mathbf{y}(t)$ is sought.

- Note that $\mathbf{y}(t)$ is only defined *implicitly*, through the differential equation. This makes the task of approximating it more difficult than in previous chapters.
- To hope for a unique solution (trajectory), must have additional side conditions.
- Initial value problem: $\mathbf{y}(a)$ is given.

Ordinary differential equations

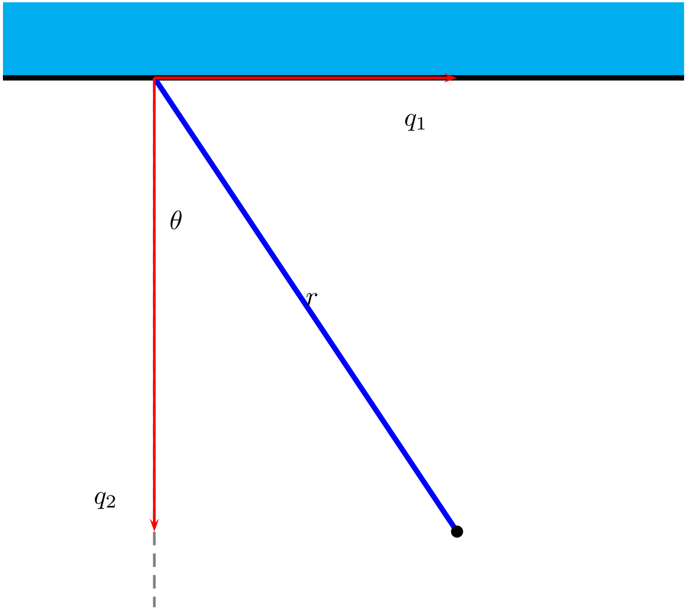
- ODEs usually arise as a *system* of differential equations.
- Write in standard form:

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \quad a < t < b,$$

where t is the independent variable, and $\mathbf{y} = \mathbf{y}(t)$ is sought.

- Note that $\mathbf{y}(t)$ is only defined *implicitly*, through the differential equation. This makes the task of approximating it more difficult than in previous chapters.
- To hope for a unique solution (trajectory), must have additional side conditions.
- Initial value problem: $\mathbf{y}(a)$ is given.

Example: pendulum



Example: pendulum

Newton's law of motion (masses $\times$ accelerations $=$ forces) gives

$$\frac{d^2\theta}{dt^2} \equiv \theta'' = -g\sin(\theta),$$

where g is the scaled constant of gravity, e.g., $g = 9.81$, and t is time.

- Write as first order ODE system: $y_1(t) = \theta(t),\ y_2(t) = \theta'(t)$. Then $y_1' = y_2, \quad y_2' = -g\sin(y_1)$.
- ODE in standard form for the pendulum:

$$\mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ -g\sin(y_1) \end{pmatrix}.$$

- Initial values: $\theta(0)$ and $\theta'(0)$ are given.
- An alternative not considered further in the slides is a boundary value problem, where e.g. $\theta(0)$ and $\theta(\pi)$ are given.

Example: pendulum

Newton's law of motion (masses $\times$ accelerations $=$ forces) gives

$$\frac{d^2\theta}{dt^2} \equiv \theta'' = -g\sin(\theta),$$

where g is the scaled constant of gravity, e.g., $g = 9.81$, and t is time.

- Write as first order ODE system: $y_1(t) = \theta(t),\ y_2(t) = \theta'(t)$. Then $y_1' = y_2, \quad y_2' = -g\sin(y_1)$.
- ODE in standard form for the pendulum:

$$\mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ -g\sin(y_1) \end{pmatrix}.$$

- Initial values: $\theta(0)$ and $\theta'(0)$ are given.
- An alternative not considered further in the slides is a boundary value problem, where e.g. $\theta(0)$ and $\theta(\pi)$ are given.

Partial differential equations

- Before we concentrate on methods for initial value ODEs, here are some PDE prototypes for a more complete general picture.
- Simplest elliptic PDE: Poisson.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = g(x,y).$$

- Simplest parabolic PDE: heat.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}.$$

- Simplest hyperbolic PDE: wave.

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0.$$

Partial differential equations

- Before we concentrate on methods for initial value ODEs, here are some PDE prototypes for a more complete general picture.
- Simplest elliptic PDE: Poisson.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = g(x, y).$$

- Simplest parabolic PDE: heat.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}.$$

- Simplest hyperbolic PDE: wave.

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0.$$

Outline

Forward Euler

- Simplest method(s) for the initial value ODE

$$y' = f(t,y), \quad y(a) = c.$$

 (Consider a scalar ODE for convenience.)
- We use it to demonstrate general concepts:
  - Method derivation
  - Explicit vs. implicit methods
  - Local truncation error and global error
  - Order of accuracy
  - Convergence
  - Absolute stability and stiffness.

Forward Euler: derivation

- Mesh points $t_0 < t_1 < \cdots < t_N$ with $h = t_{i+1} - t_i$. Approximate solution $y_i \approx y(t_i)$.
- Proceed to march from one mesh point to the next (step by step).
- Recall (Chapter 14) the simplest *forward difference*

$$y'(t_i) = \frac{y(t_{i+1}) - y(t_i)}{h} - \frac{h}{2} y''(\xi_i).$$

Therefore,

$$y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2} y''(\xi_i).$$

- So, set

$$\begin{aligned} y_0 &= c, \\ y_{i+1} &= y_i + hf(t_i, y_i), \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

Forward Euler: derivation

- Mesh points $t_0 < t_1 < \cdots < t_N$ with $h = t_{i+1} - t_i$. Approximate solution $y_i \approx y(t_i)$.
- Proceed to march from one mesh point to the next (step by step).
- Recall (Chapter 14) the simplest *forward difference*

$$y'(t_i) = \frac{y(t_{i+1}) - y(t_i)}{h} - \frac{h}{2} y''(\xi_i).$$

Therefore,

$$y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2} y''(\xi_i).$$

- So, set

$$\begin{aligned} y_0 &= c, \\ y_{i+1} &= y_i + hf(t_i, y_i), \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

Forward Euler: derivation

- Mesh points $t_0 < t_1 < \cdots < t_N$ with $h = t_{i+1} - t_i$. Approximate solution $y_i \approx y(t_i)$.
- Proceed to march from one mesh point to the next (step by step).
- Recall (Chapter 14) the simplest *forward difference*

$$y'(t_i) = \frac{y(t_{i+1}) - y(t_i)}{h} - \frac{h}{2} y''(\xi_i).$$

Therefore,

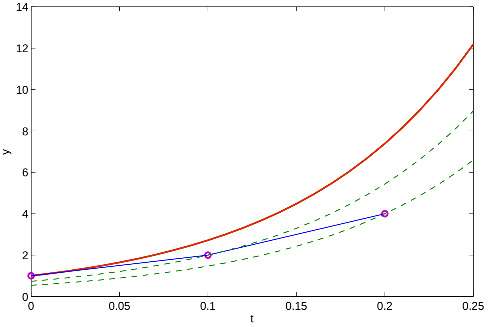
$$y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2} y''(\xi_i).$$

- So, set

$$\begin{aligned} y_0 &= c, \\ y_{i+1} &= y_i + hf(t_i, y_i), \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

Forward Euler: simple example

- For the problem $y' = y,\ y(0) = 1$, the exact solution is $y(t) = e^t$.
- Forward Euler: $y_0 = 1, \quad y_{i+1} = (1+h)y_i,\ i = 0, 1, \ldots.$
- In the "typical figure", red is exact $y(t)$, blue is approximate y_i, $h = 0.1$.
- Measuring obtained error $\max_{1\le n\le 1/h} |y_n - y(t_n)|$, it behaves like $\mathcal{O}(h)$.



Forward Euler: nonlinear example

- A system of 8 ODEs arising in plant physiology. In MATLAB define

```
function f = hires(t,y)
f = y;
f(1) = -1.71*y(1) + .43*y(2) + 8.32*y(3) + .0007;
f(2) = 1.71*y(1) - 8.75*y(2);
f(3) = -10.03*y(3) + .43*y(4) + .035*y(5);
f(4) = 8.32*y(2) + 1.71*y(3) - 1.12*y(4);
f(5) = -1.745*y(5) + .43*y(6) + .43*y(7);
f(6) = -280*y(6)*y(8) + .69*y(4) + 1.71*y(5) - .43*y(6) + .69*y(7);
f(7) = 280*y(6)*y(8) - 1.81*y(7);
f(8) = -280*y(6)*y(8) + 1.81*y(7);
```

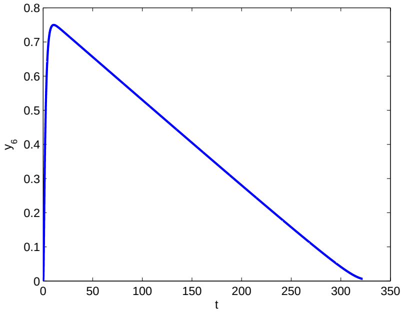
- Integrate from $a = 0$ to $b = 322$ starting from $\mathbf{y}(0) = \mathbf{y}_0 = (1, 0, 0, 0, 0, 0, 0, .0057)^T$.
- Use the script (note: wasteful in terms of storage, but simple)

```
h = .001; t = 0:h:322;
y = y0 * ones(1,length(t));
for i = 1:length(t)-1
    y(:,i+1) = y(:,i) + h*hires(t(i),y(:,i));
end
plot(t,y(6,:))
```

Example: hires

Note the simplicity of the code, as well as the rather small h selected, which yields 322,000 time steps.

Resulting plot of 6th component:



Backward Euler: implicit vs. explicit

- Instead of forward, could use *backward difference*

$$y'(t_{i+1}) = \frac{y(t_{i+1}) - y(t_i)}{h} + \frac{h}{2} y''(\xi_i).$$

Therefore,

$$y(t_{i+1}) = y(t_i) + hf(t_{i+1}, y(t_{i+1})) - \frac{h^2}{2} y''(\xi_i).$$

- So, set

$$\begin{aligned} y_0 &= c, \\ y_{i+1} &= y_i + hf(t_{i+1}, y_{i+1}), \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

- But now, unknown y_{i+1} appears implicitly!
  More complicated and costly to carry out the stepping procedure.
- Forward Euler is an explicit method, backward Euler is an implicit method.

Backward Euler: implicit vs. explicit

- Instead of forward, could use *backward difference*

$$y'(t_{i+1}) = \frac{y(t_{i+1}) - y(t_i)}{h} + \frac{h}{2} y''(\xi_i).$$

Therefore,

$$y(t_{i+1}) = y(t_i) + hf(t_{i+1}, y(t_{i+1})) - \frac{h^2}{2} y''(\xi_i).$$

- So, set

$$\begin{aligned} y_0 &= c, \\ y_{i+1} &= y_i + hf(t_{i+1}, y_{i+1}), \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

- But now, unknown y_{i+1} appears implicitly!
More complicated and costly to carry out the stepping procedure.
- Forward Euler is an explicit method, backward Euler is an implicit method.

Forward Euler outline

- Method derivation
- Explicit vs. implicit methods
- Local truncation error and global error
- Order of accuracy
- Convergence
- Absolute stability and stiffness.

Local truncation error and global error

- Local truncation error, $d_i =$ the amount by which the exact solution fails to satisfy the difference equation, written in divided difference form.
- For forward Euler, $d_i = \frac{h}{2} y''(\xi_i)$.
- The order of accuracy is q if $\max_i |d_i| = \mathcal{O}(h^q)$.
  (The Euler methods are 1st order accurate.)
- Global error

$$e_n = y(t_n) - y_n, \quad n = 0, 1, \ldots, N.$$

- The method converges if $\max_{0 \le n \le N} |e_n| \to 0$ as $h \to 0$.
- For all the methods and problems we consider, there is a constant K s.t.

$$|e_n| \le K \max_i |d_i|, \quad n = 0, 1, \ldots, N.$$

Local truncation error and global error

- Local truncation error, $d_i =$ the amount by which the exact solution fails to satisfy the difference equation, written in divided difference form.
- For forward Euler, $d_i = \frac{h}{2} y''(\xi_i)$.
- The order of accuracy is q if $\max_i |d_i| = \mathcal{O}(h^q)$.
  (The Euler methods are 1st order accurate.)
- Global error

$$e_n = y(t_n) - y_n, \quad n = 0, 1, \ldots, N.$$

- The method converges if $\max_{0 \le n \le N} |e_n| \to 0$ as $h \to 0$.
- For all the methods and problems we consider, there is a constant K s.t.

$$|e_n| \le K \max_i |d_i|, \quad n = 0, 1, \ldots, N.$$

Local truncation error and global error

- Local truncation error, $d_i =$ the amount by which the exact solution fails to satisfy the difference equation, written in divided difference form.
- For forward Euler, $d_i = \frac{h}{2} y''(\xi_i)$.
- The order of accuracy is q if $\max_i |d_i| = \mathcal{O}(h^q)$.
  (The Euler methods are 1st order accurate.)
- Global error

$$e_n = y(t_n) - y_n, \quad n = 0, 1, \ldots, N.$$

- The method converges if $\max_{0 \le n \le N} |e_n| \to 0$ as $h \to 0$.
- For all the methods and problems we consider, there is a constant K s.t.

$$|e_n| \le K \max_i |d_i|, \quad n = 0, 1, \ldots, N.$$

Local truncation error and global error

- Local truncation error, $d_i =$ the amount by which the exact solution fails to satisfy the difference equation, written in divided difference form.
- For forward Euler, $d_i = \frac{h}{2} y''(\xi_i)$.
- The order of accuracy is q if $\max_i |d_i| = \mathcal{O}(h^q)$.
  (The Euler methods are 1st order accurate.)
- Global error

$$e_n = y(t_n) - y_n, \quad n = 0, 1, \ldots, N.$$

- The method converges if $\max_{0 \le n \le N} |e_n| \to 0$ as $h \to 0$.
- For all the methods and problems we consider, there is a constant K s.t.

$$|e_n| \le K \max_i |d_i|, \quad n = 0, 1, \ldots, N.$$

Local truncation error and global error

- Local truncation error, $d_i =$ the amount by which the exact solution fails to satisfy the difference equation, written in divided difference form.
- For forward Euler, $d_i = \frac{h}{2} y''(\xi_i)$.
- The order of accuracy is q if $\max_i |d_i| = \mathcal{O}(h^q)$.
  (The Euler methods are 1st order accurate.)
- Global error

$$e_n = y(t_n) - y_n, \quad n = 0, 1, \ldots, N.$$

- The method converges if $\max_{0 \le n \le N} |e_n| \to 0$ as $h \to 0$.
- For all the methods and problems we consider, there is a constant K s.t.

$$|e_n| \le K \max_i |d_i|, \quad n = 0, 1, \ldots, N.$$

Local truncation error and global error

- Local truncation error, $d_i =$ the amount by which the exact solution fails to satisfy the difference equation, written in divided difference form.
- For forward Euler, $d_i = \frac{h}{2} y''(\xi_i)$.
- The order of accuracy is q if $\max_i |d_i| = \mathcal{O}(h^q)$.
  (The Euler methods are 1st order accurate.)
- Global error

$$e_n = y(t_n) - y_n, \quad n = 0, 1, \ldots, N.$$

- The method converges if $\max_{0 \le n \le N} |e_n| \to 0$ as $h \to 0$.
- For all the methods and problems we consider, there is a constant K s.t.

$$|e_n| \le K \max_i |d_i|, \quad n = 0, 1, \ldots, N.$$

Euler convergence theorem

Let $f(t, y)$ have bounded partial derivatives in a region $\mathcal{D} = \{a \le t \le b,\ |y| < \infty\}$. Note that this implies Lipschitz continuity in y: there exists a constant L such that for all (t, y) and $(t, \hat{y})$ in $\mathcal{D}$ we have

$$|f(t, y) - f(t, \hat{y})| \le L|y - \hat{y}|.$$

Then Euler's method converges and its global error decreases linearly in h. Moreover, assuming further that

$$|y''(t)| \le M, \quad a \le t \le b,$$

the global error satisfies

$$|e_n| \le \frac{Mh}{2L}[e^{L(t_n - a)} - 1], \quad n = 0, 1, \ldots, N.$$

Forward Euler outline

- Method derivation
- Explicit vs. implicit methods
- Local truncation error and global error
- Order of accuracy
- Convergence
- Absolute stability and stiffness

Absolute stability

- Convergence is for $h \to 0$, but in computation the step size is finite and fixed. How is the method expected to perform?
- Consider simplest, test equation

$$y' = \lambda y.$$

Solution: $y(t) = y(0)e^{\lambda t}$. Increases for $\lambda > 0$, decreases for $\lambda < 0$.

- Forward Euler:

$$y_{i+1} = y_i + h\lambda y_i = (1 + h\lambda)y_i = \ldots = (1 + h\lambda)^{i+1} y(0).$$

- So, approximate solution does not grow *only if* $|1 + h\lambda| \le 1$. This is important when $\lambda \le 0$.
- For $\lambda < 0$ must require

$$1 + h\lambda \ge -1, \quad \text{hence } h \le \frac{2}{-\lambda}.$$

Absolute stability and stiffness

- The restriction on the step size is an absolute stability requirement.
- Note: it's a *stability, not accuracy*, requirement.
- If absolute stability requirement is much more restrictive than accuracy requirement, the problem is stiff.
- **Example**

$$y' = -1000(y - \cos(t)) - \sin(t), \quad y(0) = 1,$$

The exact solution is $y(t) = \cos(t)$: varies slowly and smoothly.

- And yet, applying forward Euler, must require

$$h \le \frac{2}{1000} = 0.002,$$

or else roundoff error will rapidly build up, leading to a useless simulation.

Absolute stability and stiffness

- The restriction on the step size is an absolute stability requirement.
- Note: it's a *stability, not accuracy*, requirement.
- If absolute stability requirement is much more restrictive than accuracy requirement, the problem is stiff.
- **Example**

$$y' = -1000(y - \cos(t)) - \sin(t), \quad y(0) = 1,$$

  The exact solution is $y(t) = \cos(t)$: varies slowly and smoothly.

- And yet, applying forward Euler, must require

$$h \leq \frac{2}{1000} = 0.002,$$

  or else roundoff error will rapidly build up, leading to a useless simulation.

Backward Euler and implicit methods

- Apply *backward Euler* to the *test equation*:

$$y_{i+1} = y_i + h\lambda y_{i+1}.$$

Hence

$$y_{i+1} = \frac{1}{1-h\lambda} y_i.$$

- Here $|y_{i+1}| \le |y_i|$ for any $h > 0$ and $\lambda < 0$: no annoying absolute stability restriction.
- For the **example** of previous slide, integrating from 0 to $\pi/2$ using forward Euler with $h \ge .001\pi$ leads to blowup.
  Integrating using backward Euler with $h = .001\pi$ gives error 3.2e-9.
  Using backward Euler with $h = .1\pi$ still results in respectable error 1.7e-5.

Backward Euler and implicit methods

- Apply *backward Euler* to the *test equation*:

$$y_{i+1} = y_i + h\lambda y_{i+1}.$$

Hence

$$y_{i+1} = \frac{1}{1 - h\lambda} y_i.$$

- Here $|y_{i+1}| \leq |y_i|$ for any $h > 0$ and $\lambda < 0$: no annoying absolute stability restriction.
- For the **example** of previous slide, integrating from 0 to $\pi/2$ using forward Euler with $h \geq .001\pi$ leads to blowup.
  Integrating using backward Euler with $h = .001\pi$ gives error 3.2e-9.
  Using backward Euler with $h = .1\pi$ still results in respectable error 1.7e-5.

Stiff problems more generally

- Stiff systems do arise a lot in practice.
- If problem is very stiff, explicit methods are not effective.
- Simplest case of a system: $\mathbf{y}' = A\mathbf{y}$, with A a constant $m \times m$ diagonalizable matrix.
- There is a similarity transformation T so that

$$T^{-1}AT = \mathrm{diag}(\lambda_1, \ldots, \lambda_m).$$

Then for $\mathbf{x} = T^{-1}\mathbf{y}$ obtain m test equations

$$x_j' = \lambda_j x_j, \quad j = 1, \ldots, m.$$

- For forward Euler must require

$$|1 + h\lambda_j| \le 1, \quad j = 1, 2, \ldots, m.$$

- **The big complication**: the eigenvalues λ_j may be complex!

Stiff problems more generally

- Stiff systems do arise a lot in practice.
- If problem is very stiff, explicit methods are not effective.
- Simplest case of a system: $\mathbf{y}' = A\mathbf{y}$, with A a constant $m \times m$ diagonalizable matrix.
- There is a similarity transformation T so that

$$T^{-1}AT = \mathrm{diag}(\lambda_1, \ldots, \lambda_m).$$

Then for $\mathbf{x} = T^{-1}\mathbf{y}$ obtain m test equations

$$x_j' = \lambda_j x_j, \quad j = 1, \ldots, m.$$

- For forward Euler must require

$$|1 + h\lambda_j| \le 1, \quad j = 1, 2, \ldots, m.$$

- **The big complication**: the eigenvalues λ_j may be complex!

Stiff problems more generally

- Stiff systems do arise a lot in practice.
- If problem is very stiff, explicit methods are not effective.
- Simplest case of a system: $\mathbf{y}' = A\mathbf{y}$, with A a constant $m \times m$ diagonalizable matrix.
- There is a similarity transformation T so that

$$T^{-1}AT = \mathrm{diag}(\lambda_1, \ldots, \lambda_m).$$

Then for $\mathbf{x} = T^{-1}\mathbf{y}$ obtain m test equations

$$x_j' = \lambda_j x_j, \quad j = 1, \ldots, m.$$

- For forward Euler must require

$$|1 + h\lambda_j| \le 1, \quad j = 1, 2, \ldots, m.$$

- **The big complication**: the eigenvalues λ_j may be complex!

Outline

Runge-Kutta methods: higher order

- The Euler methods are only first order accurate: want higher accuracy.
- Can write ODE $y' = f(t, y)$ as

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))dt,$$

and discretize integral by a basic quadrature rule.

- Use polynomial interpolation for values of y inside the subinterval (as in quadrature, Chapter 15).
- Implicit trapezoidal method:

$$y_{i+1} = y_i + \frac{h}{2}\Big(f(t_i, y_i) + f(t_{i+1}, y_{i+1})\Big)$$

- Easy to see that the local truncation error is $d_i = \mathcal{O}(h^2)$.
- **But** resulting method is implicit!

Runge-Kutta methods: higher order

- The Euler methods are only first order accurate: want higher accuracy.
- Can write ODE $y' = f(t, y)$ as

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))dt,$$

and discretize integral by a basic quadrature rule.

- Use polynomial interpolation for values of y inside the subinterval (as in quadrature, Chapter 15).
- Implicit trapezoidal method:

$$y_{i+1} = y_i + \frac{h}{2}\Big(f(t_i, y_i) + f(t_{i+1}, y_{i+1})\Big)$$

- Easy to see that the local truncation error is $d_i = \mathcal{O}(h^2)$.
- **But** resulting method is implicit!

Explicit trapezoidal method

- "Bootstrap": Use forward Euler to approximate $f(t_{i+1}, y_{i+1})$ first.

$$\begin{aligned} Y &= y_i + hf(t_i, y_i), \\ y_{i+1} &= y_i + \frac{h}{2}\Big(f(t_i, y_i) + f(t_{i+1}, Y)\Big). \end{aligned}$$

- Obtain explicit trapezoidal, a special case of an explicit Runge-Kutta method.
- Can do the same based on the midpoint quadrature rule:

$$\begin{aligned} Y &= y_i + \frac{h}{2} f(t_i, y_i), \\ y_{i+1} &= y_i + hf(t_i + h/2, Y). \end{aligned}$$

Explicit trapezoidal method

- "Bootstrap": Use forward Euler to approximate $f(t_{i+1}, y_{i+1})$ first.

$$\begin{aligned} Y &= y_i + hf(t_i, y_i), \\ y_{i+1} &= y_i + \frac{h}{2}\Big(f(t_i, y_i) + f(t_{i+1}, Y)\Big). \end{aligned}$$

- Obtain explicit trapezoidal, a special case of an explicit Runge-Kutta method.
- Can do the same based on the midpoint quadrature rule:

$$\begin{aligned} Y &= y_i + \frac{h}{2}f(t_i, y_i), \\ y_{i+1} &= y_i + hf(t_i + h/2, Y). \end{aligned}$$

Explicit trapezoidal & midpoint methods

- Can write explicit trap as
$$\begin{aligned} K_1 &= f(t_i, y_i), \\ K_2 &= f(t_{i+1}, y_i + hK_1), \\ y_{i+1} &= y_i + \frac{h}{2}(K_1 + K_2). \end{aligned}$$
- Can write explicit midpoint as
$$\begin{aligned} K_1 &= f(t_i, y_i), \\ K_2 &= f(t_i + .5h, y_i + .5hK_1), \\ y_{i+1} &= y_i + hK_2. \end{aligned}$$
- Both are explicit 2-stage methods of order 2.

Explicit s-stage RK method

A class of methods is defined by coefficients $a_{j,k},\ b_j,\ \ 1 \le j \le s,\ 1 \le k \le j-1$:

$$\begin{aligned} K_1 &= f(t_i, y_i), \\ K_2 &= f(t_i + hc_2, y_i + ha_{2,1}K_1), \\ \vdots &= \vdots \\ K_j &= f(t_i + hc_j, y_i + h\sum_{k=1}^{j-1} a_{j,k}K_k), \quad 1 \le j \le s \\ y_{i+1} &= y_i + h\sum_{j=1}^{s} b_j K_j. \end{aligned}$$

where $c_j = \sum_{k=1}^{j-1} a_{j,k},\ 1 = \sum_{k=1}^{s} b_k$.

RK in tableau form

Can usefully record the coefficients $a_{j,k}$ and b_k in a tableau

$$\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_s
\end{array}$$

where $c_j = \sum_{k=1}^{s} a_{j,k}$ for $j = 1, 2, \ldots, s$.

The RK method is *explicit* if the matrix A is strictly lower triangular: $a_{j,k} = 0$ if $j \le k$. The method is *implicit* otherwise.

RK in tableau form

Can usefully record the coefficients $a_{j,k}$ and b_k in a tableau

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

where $c_j = \sum_{k=1}^{s} a_{j,k}$ for $j = 1, 2, \ldots, s$.
The RK method is *explicit* if the matrix A is strictly lower triangular: $a_{j,k} = 0$ if $j \leq k$. The method is *implicit* otherwise.

Classical 4th order RK

- The original 4-stage Runge method is explicit:

$$\begin{aligned}
K_1 &= f(t_i, y_i),\\
K_2 &= f(t_i + h/2, y_i + \frac{h}{2}K_1),\\
K_3 &= f(t_i + h/2, y_i + \frac{h}{2}K_2),\\
K_4 &= f(t_{i+1}, y_i + hK_3),\\
y_{i+1} &= y_i + \frac{h}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right).
\end{aligned}$$

- Tableau form

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0\\
1/2 & 1/2 & 0 & 0 & 0\\
1/2 & 0 & 1/2 & 0 & 0\\
1 & 0 & 0 & 1 & 0\\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}$$

Classical 4th order RK

- The original 4-stage Runge method is explicit:

$$\begin{aligned}
K_1 &= f(t_i, y_i),\\
K_2 &= f(t_i + h/2, y_i + \frac{h}{2}K_1),\\
K_3 &= f(t_i + h/2, y_i + \frac{h}{2}K_2),\\
K_4 &= f(t_{i+1}, y_i + hK_3),\\
y_{i+1} &= y_i + \frac{h}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right).
\end{aligned}$$

- Tableau form

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0\\
1/2 & 1/2 & 0 & 0 & 0\\
1/2 & 0 & 1/2 & 0 & 0\\
1 & 0 & 0 & 1 & 0\\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}$$

Example with known solution

$y' = -y^2, \; y(0) = 1 \quad \Longrightarrow \; y(t) = 1/t$.
The table shows absolute errors at $t = 1$ as well as effective convergence rates. To obtain small errors, the higher order methods are more effective.

h	Euler	rate	RK2	rate	RK4	rate
0.2	4.7e-3		3.3e-4		2.0e-7	
0.1	2.3e-3	1.01	7.4e-5	2.15	1.4e-8	3.90
0.05	1.2e-3	1.01	1.8e-5	2.07	8.6e-10	3.98
0.02	4.6e-3	1.00	2.8e-6	2.03	2.2-11	4.00
0.01	2.3e-4	1.00	6.8e-7	2.01	1.4e-12	4.00
0.005	1.2e-4	1.00	1.7e-7	2.01	8.7e-14	4.00
0.002	4.6e-5	1.00	2.7e-8	2.00	1.9e-15	4.19

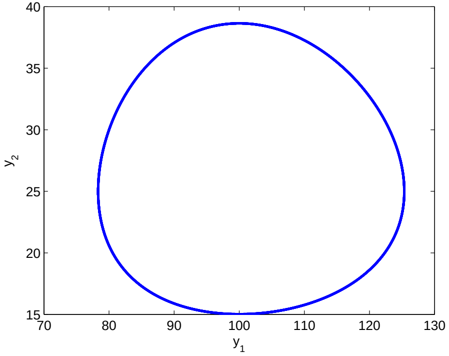
where

$$\text{rate}(h) = \log_2 \left(\frac{e(2h)}{e(h)} \right).$$

Lotka-Volterra predator-prey model

$$\begin{aligned} y_1' &= .25y_1 - .01y_1y_2, \quad y_1(0) = 80, \\ y_2' &= -y_2 + .01y_1y_2, \quad y_2(0) = 30. \end{aligned}$$

Integrating from $a = 0$ to $b = 100$ using RK4 with step size $h = 0.01$:



Outline

- Differential equations
- Euler's method
- Runge-Kutta methods
- Multistep methods
- Absolute stability and stiffness
- Error control and estimation
- *Partial differential equations

Linear multistep methods

- The RK family of methods are one-step, increasing the order of the method by repeated evaluations of f in $[t_i, t_{i+1}]$.
- Another approach for increasing order is to use past values y_{i+1-j} and corresponding $f_{i+1-j} = f(t_{i+1-j}, y_{i+1-j}),\ j = 0, 1, \ldots, s$.
- The general linear multistep formula is

$$\sum_{j=0}^{s} \alpha_j y_{i+1-j} = h \sum_{j=0}^{s} \beta_j f_{i+1-j}.$$

  Here, α_j, β_j are coefficients; set $\alpha_0 = 1$. The method is explicit iff $\beta_0 = 0$.
- One-step $s = 1$ **examples:** forward Euler ($\alpha_1 = -1, \beta_1 = 1$), backward Euler ($\alpha_1 = -1, \beta_0 = 1$), implicit trapezoid ($\alpha_1 = -1, \beta_0 = \beta_1 = .5$).
- However, the explicit RK2 and RK4 are not in the linear multistep form.

Linear multistep methods

- The RK family of methods are one-step, increasing the order of the method by repeated evaluations of f in $[t_i, t_{i+1}]$.
- Another approach for increasing order is to use past values y_{i+1-j} and corresponding $f_{i+1-j} = f(t_{i+1-j}, y_{i+1-j}),\ j = 0, 1, \ldots, s$.
- The general linear multistep formula is

$$\sum_{j=0}^{s} \alpha_j y_{i+1-j} = h \sum_{j=0}^{s} \beta_j f_{i+1-j}.$$

  Here, α_j, β_j are coefficients; set $\alpha_0 = 1$. The method is explicit iff $\beta_0 = 0$.
- One-step $s = 1$ **examples:** forward Euler ($\alpha_1 = -1, \beta_1 = 1$), backward Euler ($\alpha_1 = -1, \beta_0 = 1$), implicit trapezoid ($\alpha_1 = -1, \beta_0 = \beta_1 = .5$).
- However, the explicit RK2 and RK4 are not in the linear multistep form.

Adams methods

- Starting with the integral form

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))dt,$$

integrate polynomial interplant of $f(t, y)$ using values

  - $f_i, f_{i-1}, \ldots, f_{i+1-s}$, which gives an *explicit* Adams-Bashforth method of order s; or
  - $f_{i+1}, f_i, f_{i-1}, \ldots, f_{i+1-s}$, which gives an *implicit* Adams-Moulton method of order $s+1$.

- e.g. *2-step Adams-Bashforth*: $y_{i+1} = y_i + \frac{h}{2}(3f_i - f_{i-1})$.
- These methods are good for **non-stiff** problems, where they are often used in pairs of predictor-corrector. They can be highly efficient (more than RK) if high order accuracy is required for a smooth problem. For everyday use, though, RK is generally considered more versatile.

BDF methods

- These methods are called backward differentiation formulas (BDF), and use f value only at the unknown level $i+1$.
- Starting with the form

$$\sum_{j=0}^{s} \alpha_j y_{i+1-j} = h\beta_0 f_{i+1},$$

  determine the coefficients using polynomial interpolation of $y_{i+1}, y_i, y_{i-1}, \ldots, y_{i+1-s}$, which gives an *implicit* method of order s.
- The *1-step BDF* is backward Euler. The *2-step BDF* is $y_{i+1} = \frac{4}{3}y_i - \frac{1}{3}y_{i-1} + \frac{2h}{3}f_{i+1}$.
- These methods are particularly good for **stiff** problems, considered next.

Outline

- Differential equations
- Euler's method
- Runge-Kutta methods
- Multistep methods
- Absolute stability and stiffness
- Error control and estimation
- *Partial differential equations

Absolute stability and stiffness

- Convergence is for $h \to 0$, but in computation the step size is finite and fixed. How is the method expected to perform?
- Consider simple test equation for **complex** scalar λ (representing an eigenvalue of an ODE system Jacobian matrix)

$$y' = \lambda y.$$

Solution: $y(t) = y(0)e^{\lambda t}$. So $|y(t)| = |y(0)|e^{\mathbf{R}(\lambda)t}$. Magnitude increases for $\mathbf{R}(\lambda) > 0$, decreases for $\mathbf{R}(\lambda) < 0$.

- Forward Euler:

$$y_{i+1} = y_i + h\lambda y_i = (1 + h\lambda)y_i = \ldots = (1 + h\lambda)^{i+1}y(0).$$

- So, approximate solution does not grow only if $|1 + h\lambda| \leq 1$. Important when $\mathbf{R}(\lambda) \leq 0$.
- This condition means that $z = h\lambda$ must not lie outside the disk of radius 1 centred at $(-1, 0)$ in the complex plane.

Absolute stability and stiffness

- Convergence is for $h \to 0$, but in computation the step size is finite and fixed. How is the method expected to perform?
- Consider simple test equation for **complex** scalar λ (representing an eigenvalue of an ODE system Jacobian matrix)

$$y' = \lambda y.$$

Solution: $y(t) = y(0)e^{\lambda t}$. So $|y(t)| = |y(0)|e^{\mathbf{R}(\lambda)t}$. Magnitude increases for $\mathbf{R}(\lambda) > 0$, decreases for $\mathbf{R}(\lambda) < 0$.

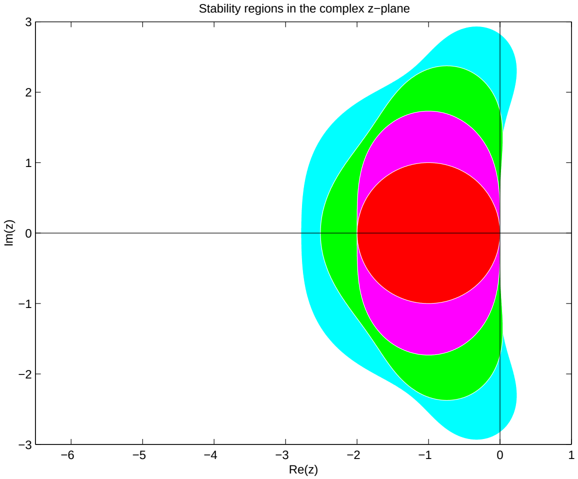
- Forward Euler:

$$y_{i+1} = y_i + h\lambda y_i = (1 + h\lambda)y_i = \ldots = (1 + h\lambda)^{i+1}y(0).$$

- So, approximate solution does not grow only if $|1 + h\lambda| \le 1.$ Important when $\mathbf{R}(\lambda) \le 0$.
- This condition means that $z = h\lambda$ must not lie outside the disk of radius 1 centred at $(-1, 0)$ in the complex plane.

Absolute stability: explicit RK

The plot depicts absolute stability regions $|R(z)| \leq 1$ in the complex $z = h\lambda$-plane for s-stage explicit RK methods of order s, $s = 1, 2, 3, 4$. The region is larger for larger s.



Stiff problem

- The ODE is stiff if an unreasonably small step size h must be used for forward Euler.
- In this case explicit RK or multistep are inadequate – resort to appropriate *implicit* methods.
- Method is A-stable if absolute stability region contains entire left half z-plane. Both backward Euler and implicit trapezoidal are A-stable.
- Method is L-stable if $|R(z)| \rightarrow 0$ as $\mathbf{R}(z) \rightarrow 0$.
  Backward Euler is L-stable, implicit trapezoidal is not.

Stiff system

- For a nonlinear ODE system, must look at the eigenvalues of the Jacobian matrix

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{y}} = \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \cdots & \cdots & \frac{\partial f_1}{\partial y_m} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_m}{\partial y_1} & \cdots & \cdots & \frac{\partial f_m}{\partial y_m} \end{pmatrix}.$$

- Note that since J depends on the unknown $\mathbf{y}(t)$, we may not know good bounds on these eigenvalues in advance.
- Moreover, the stability restriction for an explicit method may dictate different step sizes at different t.
- An implicit method requires the solution of a nonlinear algebraic system at each step; see Chapters 3 and 9. Use some variant of Newton's method (and not a cheap fixed point iteration).

Example: the hires system

Return to the "hires" system, for which $J(\mathbf{y}) =$

$$\begin{pmatrix} -1.71 & .43 & 8.32 & 0 & 0 & 0 & 0 & 0 \\ 1.71 & -8.75 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10.03 & .43 & .035 & 0 & 0 & 0 \\ 0 & 8.32 & 1.71 & -1.12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.745 & .43 & .43 & 0 \\ 0 & 0 & 0 & .69 & 1.71 & 0 & .69 & 0 \\ 0 & 0 & 0 & 0 & 0 & -280y_8 - .43 & 0 & -280y_6 \\ 0 & 0 & 0 & 0 & 0 & 280y_8 & -1.81 & 280y_6 \\ 0 & 0 & 0 & 0 & 0 & -280y_8 & 1.81 & -280y_6 \end{pmatrix}.$$

- At $t = 0$ where $\mathbf{y} = \mathbf{y}_0$, the eigenvalues are approximately

  $$0, -10.4841, -8.2780, -0.2595, -0.5058, -2.6745 \pm 0.1499\imath, -2.3147.$$

  So, $h = 0.1$ is a safe choice for forward Euler.
- At $t \approx 10.7$, the eigenvalues are approximately

  $$-211.7697, -10.4841, -8.2780, -2.3923, -2.1400, -0.4907, -3e{-5}, -3e{-12}.$$

  Here, $h = 0.005$ is required for forward Euler.
- This innocent-looking problem is mildly stiff.

Outline

- Differential equations
- Euler's method
- Runge-Kutta methods
- Multistep methods
- Absolute stability and stiffness
- Error control and estimation
- *Partial differential equations

Error control and estimation

- As in Chapter 15, want to build a function which returns, given $f(t, y)$, initial value information $a, y(a)$, tolerance `tol` and a set of output points $\hat{t}_1, \ldots, \hat{t}_{\hat{N}}$, a corresponding set of solution values accurate at these points to within `tol`.
- Unlike in Chapter 15, however, the error here propagates and accumulates from one time step (or mesh subinterval) to the next, hence the task of estimating the global error is much more awkward.
- Often in practice, a tolerance on the global error is not known, and we only want to control the step size $h_i = t_{i+1} - t_i$ from the current known (t_i, y_i) to the next (t_{i+1}, y_{i+1}).
- Thus, select h_i so that the local error in the ith step

$$l_{i+1} = \bar{y}(t_{i+1}) - y_{i+1},$$

  is below a (local) tolerance h_i`tol`. Here, $\bar{y}(t)$ is the local exact (unknown) ODE solution satisfying $\bar{y}(t_i) = y_i$.
- There are examples where this sort of error control leads to very large global errors, but in many applications good results are obtained very efficiently!

Error control and estimation

- As in Chapter 15, want to build a function which returns, given $f(t,y)$, initial value information $a, y(a)$, tolerance `tol` and a set of output points $\hat{t}_1, \ldots, \hat{t}_{\hat{N}}$, a corresponding set of solution values accurate at these points to within `tol`.
- Unlike in Chapter 15, however, the error here propagates and accumulates from one time step (or mesh subinterval) to the next, hence the task of estimating the global error is much more awkward.
- Often in practice, a tolerance on the global error is not known, and we only want to control the step size $h_i = t_{i+1} - t_i$ from the current known (t_i, y_i) to the next (t_{i+1}, y_{i+1}).
- Thus, select h_i so that the local error in the ith step

$$l_{i+1} = \bar{y}(t_{i+1}) - y_{i+1},$$

is below a (local) tolerance h_i`tol`. Here, $\bar{y}(t)$ is the local exact (unknown) ODE solution satisfying $\bar{y}(t_i) = y_i$.
- There are examples where this sort of error control leads to very large global errors, but in many applications good results are obtained very efficiently!

Error control and estimation

- As in Chapter 15, want to build a function which returns, given $f(t, y)$, initial value information $a, y(a)$, tolerance `tol` and a set of output points $\hat{t}_1, \ldots, \hat{t}_{\hat{N}}$, a corresponding set of solution values accurate at these points to within `tol`.
- Unlike in Chapter 15, however, the error here propagates and accumulates from one time step (or mesh subinterval) to the next, hence the task of estimating the global error is much more awkward.
- Often in practice, a tolerance on the global error is not known, and we only want to control the step size $h_i = t_{i+1} - t_i$ from the current known (t_i, y_i) to the next (t_{i+1}, y_{i+1}).
- Thus, select h_i so that the local error in the ith step

  $$l_{i+1} = \bar{y}(t_{i+1}) - y_{i+1},$$

  is below a (local) tolerance $h_i\texttt{tol}$. Here, $\bar{y}(t)$ is the local exact (unknown) ODE solution satisfying $\bar{y}(t_i) = y_i$.
- There are examples where this sort of error control leads to very large global errors, but in many applications good results are obtained very efficiently!

Using a pair of RK methods

- How can we estimate $\bar{y}(t_{i+1})$ effectively?
- Suppose we use an RK method of order q for y_{i+1}, given y_i.
  Then use another RK method of order $q+1$ to calculate $\hat{y}_{i+1}$, given y_i, and estimate

$$|l_{i+1}| \approx |\hat{y}_{i+1} - y_{i+1}|.$$

- Rationale: by order, $\hat{y}_{i+1}$ is closer to $\bar{y}(t_{i+1})$ than to y_{i+1}.
- So, at step i, with $h = h_i$, if

$$|\hat{y}_{i+1} - y_{i+1}| \le h\texttt{tol}$$

  then accept step: set $y_{i+1} \leftarrow \hat{y}_{i+1}, \; i \leftarrow i+1$.
- If step is not accepted, then set

$$h \leftarrow h \left(\frac{\mu h\, \texttt{tol}}{|\hat{y}_{i+1} - y_{i+1}|} \right)^{\frac{1}{q}},$$

  and repeat. This selection is based on the asymptotic behaviour $l_{i+1} \sim h^{q+1}$.

Using a pair of RK methods

- How can we estimate $\bar{y}(t_{i+1})$ effectively?
- Suppose we use an RK method of order q for y_{i+1}, given y_i.
  Then use another RK method of order $q+1$ to calculate $\hat{y}_{i+1}$, given y_i, and estimate

$$|l_{i+1}| \approx |\hat{y}_{i+1} - y_{i+1}|.$$

- Rationale: by order, $\hat{y}_{i+1}$ is closer to $\bar{y}(t_{i+1})$ than to y_{i+1}.
- So, at step i, with $h = h_i$, if

$$|\hat{y}_{i+1} - y_{i+1}| \le h\,\texttt{tol}$$

  then accept step: set $y_{i+1} \leftarrow \hat{y}_{i+1}, \; i \leftarrow i+1$.
- If step is not accepted, then set

$$h \leftarrow h \left(\frac{\mu h\,\texttt{tol}}{|\hat{y}_{i+1} - y_{i+1}|} \right)^{\frac{1}{q}},$$

  and repeat. This selection is based on the asymptotic behaviour $l_{i+1} \sim h^{q+1}$.

Using a pair of RK methods

- How can we estimate $\bar{y}(t_{i+1})$ effectively?
- Suppose we use an RK method of order q for y_{i+1}, given y_i.
  Then use another RK method of order $q+1$ to calculate $\hat{y}_{i+1}$, given y_i, and estimate

$$|l_{i+1}| \approx |\hat{y}_{i+1} - y_{i+1}|.$$

- Rationale: by order, $\hat{y}_{i+1}$ is closer to $\bar{y}(t_{i+1})$ than to y_{i+1}.
- So, at step i, with $h = h_i$, if

$$|\hat{y}_{i+1} - y_{i+1}| \le h\,\mathtt{tol}$$

  then accept step: set $y_{i+1} \leftarrow \hat{y}_{i+1}, \; i \leftarrow i+1$.
- If step is not accepted, then set

$$h \leftarrow h \left(\frac{\mu h\,\mathtt{tol}}{|\hat{y}_{i+1} - y_{i+1}|} \right)^{\frac{1}{q}},$$

  and repeat. This selection is based on the asymptotic behaviour $l_{i+1} \sim h^{q+1}$.

Adaptive stepping

- In the correction formula for h, there is a safety factor μ, e.g. $\mu = 0.9$.
- In MATLAB the function ode45 integrates *non-stiff* problems efficiently using a pair of explicit RK methods with $q = 4$ (hence the name) which share inner stages, hence are more efficient.
- In MATLAB, for *stiff* problems, use the function ode23s, which is based on a pair of BDF methods. (What is q?)
- Next, we show an example demonstrating great success with this approach.
- However, we caution that there are other examples where a uniform step size is fine and other challenges are more important.

Adaptive stepping

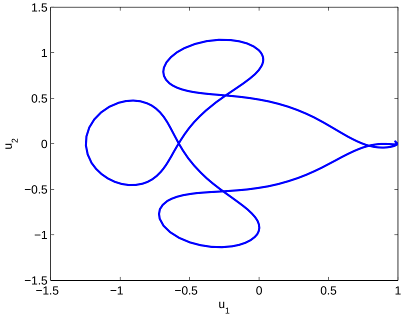
- In the correction formula for h, there is a safety factor μ, e.g. $\mu = 0.9$.
- In MATLAB the function `ode45` integrates *non-stiff* problems efficiently using a pair of explicit RK methods with $q = 4$ (hence the name) which share inner stages, hence are more efficient.
- In MATLAB, for *stiff* problems, use the function `ode23s`, which is based on a pair of BDF methods. (What is q?)
- Next, we show an example demonstrating great success with this approach.
- However, we caution that there are other examples where a uniform step size is fine and other challenges are more important.

Adaptive stepping

- In the correction formula for h, there is a safety factor μ, e.g. $\mu = 0.9$.
- In MATLAB the function `ode45` integrates *non-stiff* problems efficiently using a pair of explicit RK methods with $q = 4$ (hence the name) which share inner stages, hence are more efficient.
- In MATLAB, for *stiff* problems, use the function `ode23s`, which is based on a pair of BDF methods. (What is q?)
- Next, we show an example demonstrating great success with this approach.
- However, we caution that there are other examples where a uniform step size is fine and other challenges are more important.

Example: astronomical orbit

The orbit depicted below obeys an ODE system of size 4: see Section 16.6 for details.
Using RK4 with a uniform step size, about 10,000 steps are needed for a qualitatively correct approximation.
Using `ode45` with default tolerances, 309 steps are required, with $\max h_i = 0.1892,\ \min h_i = 1.5921e-7$.



Outline

- Differential equations
- Euler's method
- Runge-Kutta methods
- Multistep methods
- Absolute stability and stiffness
- Error control and estimation
- *Partial differential equations

Partial differential equations

- The area of numerical methods for PDEs is vast and is generally well beyond a first course, even at a beginning graduate level.
- However, some PDEs do appear in our text (and slides) in context.
- Elliptic ("steady state") PDEs such as the Model Poisson equation require solution of large and often sparse systems of linear equations. They are used repeatedly as examples in Chapter 7.
- The model problem for time-dependent parabolic PDEs is the heat equation, considered in the next example.
- Recall also the nonlinear PDE in the last section of Chapter 14, for which a spectral Fourier method was used.
- We will also see some examples of treating hyperbolic PDEs next.

Partial differential equations

- The area of numerical methods for PDEs is vast and is generally well beyond a first course, even at a beginning graduate level.
- However, some PDEs do appear in our text (and slides) in context.
- Elliptic ("steady state") PDEs such as the Model Poisson equation require solution of large and often sparse systems of linear equations. They are used repeatedly as examples in Chapter 7.
- The model problem for time-dependent parabolic PDEs is the heat equation, considered in the next example.
- Recall also the nonlinear PDE in the last section of Chapter 14, for which a spectral Fourier method was used.
- We will also see some examples of treating hyperbolic PDEs next.

Partial differential equations

- The area of numerical methods for PDEs is vast and is generally well beyond a first course, even at a beginning graduate level.
- However, some PDEs do appear in our text (and slides) in context.
- Elliptic ("steady state") PDEs such as the Model Poisson equation require solution of large and often sparse systems of linear equations. They are used repeatedly as examples in Chapter 7.
- The model problem for time-dependent parabolic PDEs is the heat equation, considered in the next example.
- Recall also the nonlinear PDE in the last section of Chapter 14, for which a spectral Fourier method was used.
- We will also see some examples of treating hyperbolic PDEs next.

Partial differential equations

- The area of numerical methods for PDEs is vast and is generally well beyond a first course, even at a beginning graduate level.
- However, some PDEs do appear in our text (and slides) in context.
- Elliptic ("steady state") PDEs such as the Model Poisson equation require solution of large and often sparse systems of linear equations. They are used repeatedly as examples in Chapter 7.
- The model problem for time-dependent parabolic PDEs is the heat equation, considered in the next example.
- Recall also the nonlinear PDE in the last section of Chapter 14, for which a spectral Fourier method was used.
- We will also see some examples of treating hyperbolic PDEs next.

Partial differential equations

- The area of numerical methods for PDEs is vast and is generally well beyond a first course, even at a beginning graduate level.
- However, some PDEs do appear in our text (and slides) in context.
- Elliptic ("steady state") PDEs such as the Model Poisson equation require solution of large and often sparse systems of linear equations. They are used repeatedly as examples in Chapter 7.
- The model problem for time-dependent parabolic PDEs is the heat equation, considered in the next example.
- Recall also the nonlinear PDE in the last section of Chapter 14, for which a spectral Fourier method was used.
- We will also see some examples of treating hyperbolic PDEs next.

Example: heat equation

- Consider

$$\frac{\partial u}{\partial t} = \left(\frac{\partial^2 u}{\partial {x_1}^2} + \frac{\partial^2 u}{\partial {x_2}^2} \right).$$

Here $0 \le x_1, x_2 \le 1$ are spatial variables and $t \ge 0$ is time.

- Require boundary conditions, e.g.

$$u(t, x_1, 0) = u(t, x_1, 1) = u(t, 0, x_2) = u(t, 1, x_2) = 0, \quad \forall t \ge 0,$$

and initial conditions, e.g.

$$u(0, x_1, x_2) = 25, \quad 0 < x_1, x_2 < 1.$$

- Discretize in space using uniform step Δx as in Chapter 7, obtain mesh function in time satisfying the ODE system

$$\begin{aligned} \frac{du_{i,j}}{dt} &= \frac{1}{\Delta x^2}(u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}),\ 1 \le i, j \le M-1, \\ u_{i,j} &= 0,\ \text{otherwise.} \end{aligned}$$

Example: heat equation cont.

- Reshape $u_{i,j}$'s as a vector $\mathbf{y}(t)$, obtain linear initial value ODE system

$$\mathbf{y}' = A\mathbf{y}. \quad \mathbf{y}(0) = 25 * \mathbf{1}$$

  with A large and sparse.

- The eigenvalues of A are

$$\lambda_{l,m} = \frac{1}{\Delta x^2}[4 - 2(\cos(l\pi\Delta x) + \cos(j\pi\Delta x))], \quad 1 \le l, j \le M-1 \ ,$$

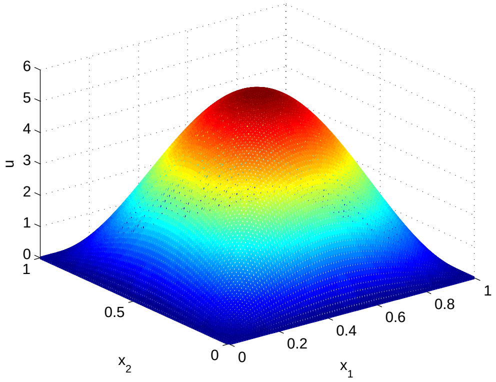
  giving for forward Euler with step size h the absolute stability condition

$$h \le \frac{\Delta x^2}{12}.$$

- This problem is therefore mildly stiff. Need $\mathcal{O}(M^2)$ time steps, each costing $\mathcal{O}(M^2)$ flops, to advance to $t = 0.1$ using explicit method.
- Crank-Nicolson: use implicit trapezoidal. Then no stability restrictions, but take $h = \mathcal{O}(\Delta x)$ for accuracy balancing.
- Must solve linear system at each step. However, so long as this requires less than $\mathcal{O}(M^3)$ flops (preconditioned conjugate gradients requires less), the implicit method wins!

Example: heat equation solution

The solution plot below is at $t = 0.1$.



Example: advection equation

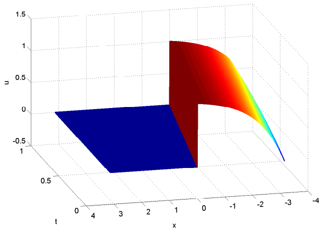
- This is a simple-looking hyperbolic PDE: for $t \geq 0, \ -\infty < x < \infty$,

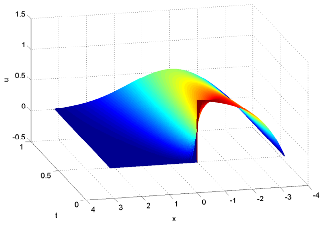
$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x}.$$

  Initial conditions $u(t = 0, x) = u_0(x)$.
- Exact solution is

$$u(t, x) = u_0(t + x),$$

  so no smoothing of initial value function for $t > 0$.
- Compare below solutions for advection (left) and heat (right) for $u_0(x) = (1 - \exp(-x - \pi))/(1 - \exp(-\pi))$ for $x < 0$, $= 0$ otherwise.





Example: classical wave equation

- Consider the PDE

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0, \quad t \geq 0,\ x_0 < x < x_{J+1},$$

under initial conditions $u(0,x) = u_0(x)$, $\frac{\partial u}{\partial t}(0,x) \equiv 0$, and boundary conditions $u(t,x_0) = u(t,x_{J+1}) = 0 \quad \forall t$.

- Exact solution

$$u(t,x) = \frac{1}{2}[u_0(x-ct) + u_0(x+ct)],$$

where $c > 0$ is the speed of sound.

- For numerical method use the *explicit, two-step* leap-frog scheme

$$\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} = c^2 \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}, \quad j = 1, 2, \ldots, J.$$

- Stability restriction coincides with CFL condition

$$c\Delta t \leq \Delta x.$$

Example: classical wave equation

- Consider the PDE

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0, \quad t \geq 0,\ x_0 < x < x_{J+1},$$

under initial conditions $u(0,x) = u_0(x)$, $\frac{\partial u}{\partial t}(0,x) \equiv 0$, and boundary conditions $u(t,x_0) = u(t,x_{J+1}) = 0 \qquad \forall t$.

- Exact solution

$$u(t,x) = \frac{1}{2}[u_0(x-ct) + u_0(x+ct)],$$

where $c > 0$ is the speed of sound.

- For numerical method use the *explicit, two-step* leap-frog scheme

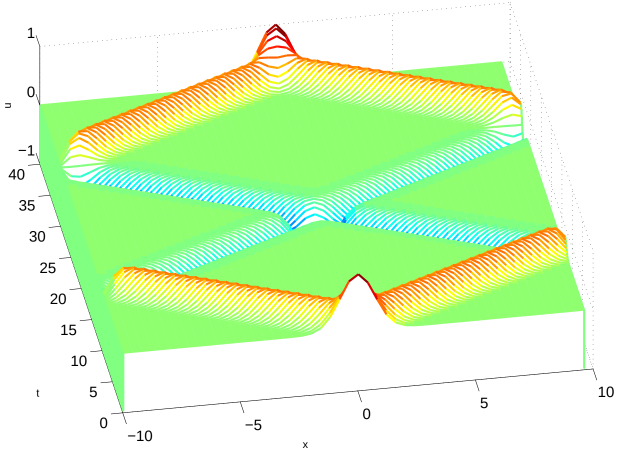
$$\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} = c^2 \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}, \quad j = 1, 2, \ldots, J.$$

- Stability restriction coincides with CFL condition

$$c\Delta t \leq \Delta x.$$

Example: classical wave equation

The "Waterfall" solution below is for $c = 1$, $u_0(x) = \exp(-\alpha x^2)$, $-10 \le x \le 10$. We set $\Delta x = .04$, $\Delta t = .02$.


1
0
−1
u
40
35
30
25
20
15
10
5
0
t
−10
−5
0
5
10
x