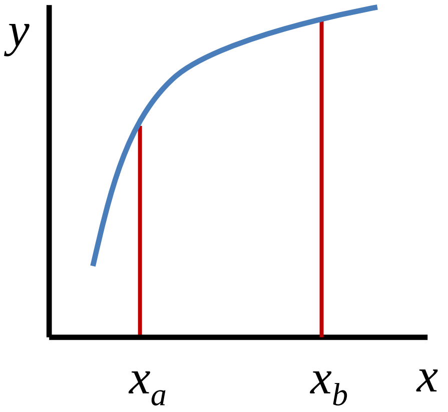
Ordinary Differential Equations (ODE)

Wouter den Otter

MSM/CTW & CBP/TNW

APiE 2013

the problem



Solve $\frac{dy}{dx} = f(x, y)$

over the interval $x_a \le x \le x_b$,

with boundary condition $y(x_a) = y_a$.

Solution: $y(x) = Y(x, x_a, y_a)$

discretization

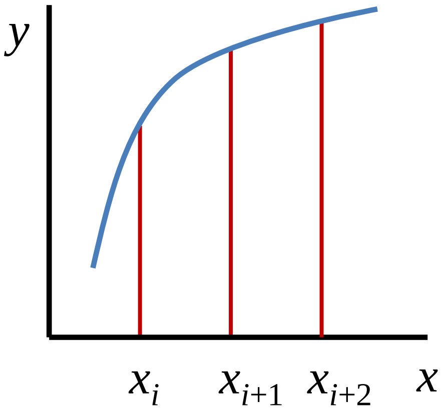
Divide the *x* range into *N* steps of width *h*:

$$x_i = x_a + ih \qquad\qquad h = \frac{x_b - x_a}{N}$$

Exact solution: $y_i = y(x_i)$

Numerical *approximation*: $z_i \approx y(x_i)$

Euler method



$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} y'(x)\,dx$$

$$= y(x_i) + \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx$$

$$= y(x_i) + hf(x_i, y(x_i)) + \tfrac{1}{2} h^2 y''(\xi) \quad \text{with } x_i \le \xi \le x_{i+1}$$

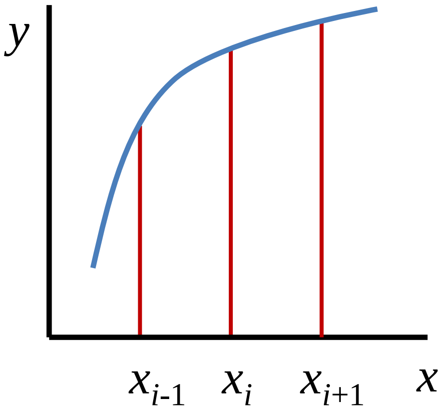
$$\boxed{z_{i+1} = z_i + hf(x_i, z_i)}$$

$$z_{i+2} = z_{i+1} + hf(x_{i+1}, z_{i+1})$$

local truncation error: $O(h^2)$

etc

midpoint rule



$$y(x_{i+1}) = y(x_{i-1}) + \int_{x_{i-1}}^{x_{i+1}} y'(x)\,dx$$

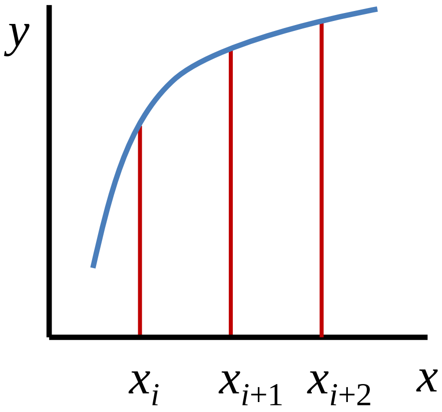
$$= y(x_{i-1}) + \int_{x_{i-1}}^{x_{i+1}} f\left(x, y(x)\right)dx$$

$$= y(x_{i-1}) + 2hf\left(x_i, y(x_i)\right) + \tfrac{1}{3}h^3 y'''(\xi) \quad \text{with } x_{i-1} \le \xi \le x_{i+1}$$

$$\boxed{z_{i+1} = z_{i-1} + 2hf\left(x_i, z_i\right)}$$

local truncation error: $O\left(h^3\right)$

trapezium rule (1)



$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} y'(x)\,dx$$

$$= y(x_i) + \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx$$

$$= y(x_i) + \tfrac{1}{2} h\left[f(x_i, y(x_i)) + f(x_{i+1}, y(x_{i+1}))\right] - \tfrac{1}{12} h^3 y'''(\xi)$$

with $x_i \le \xi \le x_{i+1}$

$$\boxed{z_{i+1} = z_i + \tfrac{1}{2} h\left[f(x_i, z_i) + f(x_{i+1}, z_{i+1})\right]}$$

local truncation error: $O(h^3)$ *But …*

trapezium rule (2)

- analytically solve z_{i+1} from $z_{i+1} = z_i + \frac{1}{2}h\left[f\left(x_i, z_i\right) + f\left(x_{i+1}, z_{i+1}\right)\right]$

'implicit' method

Or

- predictor $z_{i+1}^{(p)} = z_i + hf\left(x_i, z_i\right)$

  corrector $z_{i+1}^{(c)} = z_i + \frac{1}{2}h\left[f\left(x_i, z_i\right) + f\left(x_{i+1}, z_{i+1}^{(p)}\right)\right]$

- iterative $z_{i+1}^{(n+1)} = z_i + \frac{1}{2}h\left[f\left(x_i, z_i\right) + f\left(x_{i+1}, z_{i+1}^{(n)}\right)\right]$

multi-step: Adams - Bashforth - Moulton

predictor

$$z_{i+1}^{(p)} = z_i + \frac{1}{12} h\left[23 f\left(x_i, z_i\right) - 16 f\left(x_{i-1}, z_{i-1}\right) + 5 f\left(x_{i-2}, z_{i-2}\right)\right]$$

corrector

$$z_{i+1}^{(c)} = z_i + \frac{1}{12} h\left[5 f\left(x_{i+1}, z_{i+1}^{(p)}\right) + 8 f\left(x_i, z_i\right) - f\left(x_{i-1}, z_{i-1}\right)\right]$$

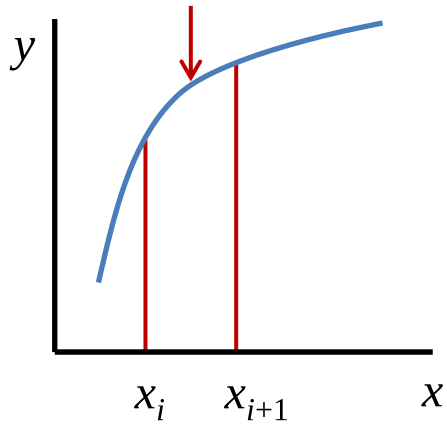
local truncation error: $O\left(h^4\right)$

sub-step: Runge-Kutta methods (1)

$$k_1 = hf\left(x_i, z_i\right)$$

$$k_2 = hf\left(x_i + \tfrac{1}{2}h, z_i + \tfrac{1}{2}k_1\right)$$

$$z_{i+1} = z_i + k_2$$



2^{nd} order RK or midpoint method

extra *f*-evalutations worth the effort if you can increase the step

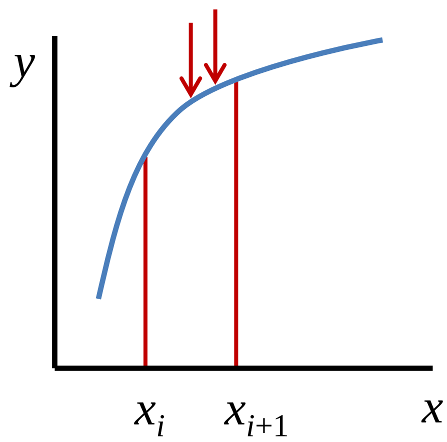
sub-step: Runge-Kutta methods (1)

$$k_1 = hf\left(x_i, z_i\right)$$

$$k_2 = hf\left(x_i + \tfrac{1}{2}h, z_i + \tfrac{1}{2}k_1\right)$$

$$k_3 = hf\left(x_i + \tfrac{3}{4}h, z_i + \tfrac{3}{4}k_1\right)$$

$$z_{i+1} = z_i + \tfrac{1}{9}\left(2k_1 + 3k_2 + 4k_3\right)$$



coefficients are determined by minimizing local truncation error, in this case to $O\left(h^4\right)$.

error estimates

- local truncation error: $y_i \to z_{i+1} = y_{i+1} + r_{i+1}(h)$

  typically, there exists an *n* such that $|r_{i+1}(h)| \le \alpha h^{n+1}$ for all *i*.

- global truncation error: $y_a \to z_b = z_{a+L} = y_{a+L} + R(h)$

  naïve guess: $|R(h)| = \sum_1^N |r_i(h)| \approx \frac{L}{h} \cdot \alpha h^{n+1} = \alpha L h^n$

  methods with this property are called ‘order *n*’.

  in practice: can be better, can also be *much* worse.

Richardson extrapolation

Suppose $y_a \to z_b = y_b + R(h)$ with $R(h) = \beta h$,

then
$$y_b = z_N + \beta h$$
$$y_b = z_{2N} + \beta h/2$$

hence $\frac{1}{2}\beta h = z_{2N} - z_N$

and $$z_{2N}^{(1)} = z_{2N} + \left(z_{2N} - z_N\right)$$

stiff ODEs (1)

Evolution on two widely separate scales

1. $y' = k\left(y - g(x)\right)$ with large negative k

$$y(x) \approx g(x) + \left[y(x_a) - g(x_a)\right]\exp\left[k(x - x_a)\right]$$

2. $$\begin{aligned} u' &= +998u + 1998v \\ v' &= -999u - 1999v \end{aligned} \quad \& \quad \begin{aligned} u(0) &= 1 \\ v(0) &= 0 \end{aligned} \quad \rightarrow \quad \begin{aligned} u &= 2e^{-x} - e^{-1000x} \\ v &= -e^{-x} + e^{-1000x} \end{aligned}$$

Fast scale sets small integration step, wasteful on slow scale.

stiff ODEs (2)

Consider $y' = -ky$ with $k > 0$.

- forward Euler (explicit)

$$z_{i+1} = z_i + hf(z_i) = z_i(1 - hk)$$

$$z_{i+1} = z_0(1 - hk)^{i+1}$$

only stable for $0 < h < 2/k$

- backward Euler (implicit)

$$z_{i+1} = z_i + hf(z_{i+1}) = z_i - hkz_{i+1}$$

$$z_{i+1} = \frac{z_0}{(1 + hk)^{i+1}}$$

stable for all h

(but not necessarily accurate)

stiff ODEs (3)

semi-implicit backward Euler method

$$z_{i+1} = z_i + hf\left(z_{i+i}\right)$$

$$= z_i + h\left[f\left(z_i\right) + f'\left(z_i\right)\cdot\left(z_{i+1} - z_i\right)\right]$$

$$z_{i+1} = z_i + \frac{h}{1 - hf'\left(z_i\right)} f\left(z_i\right)$$

(semi-implicit) backward Euler often works for stiff equations

stability

- conditional stability:
  for many algorithms, $y' = -ky$ with $k > 0$
  will convergence only within a limited range of kh.

- asymptotic stability:
  sensitivity of z_b to a small change in z_a;
  in poor methods, any deviation may diverge for $h \to 0$.

general advise

Always be wary of your results.

Try several algorithms.

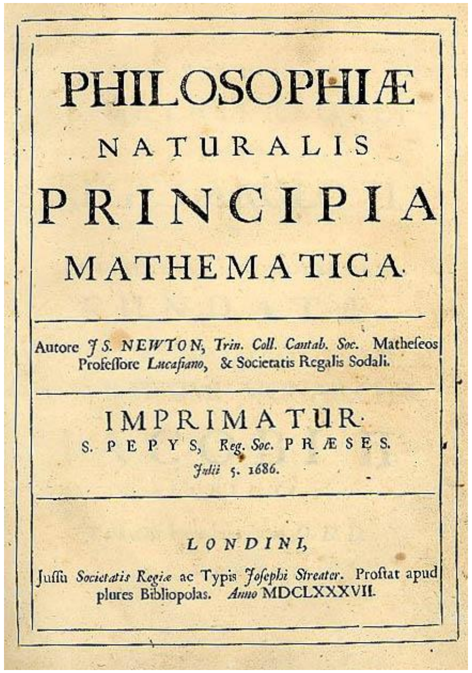
Study the step size dependence.

some options

forward z_{i+1} from derivative at i	backward z_{i+1} from derivative at i+1
sub-step z_{i+1} from values at i+¾, i+½, …, i	multi-step z_{i+1} from values at i, i-1, …
one evaluation z_{i+1} from one evaluation	multiple evaluations predictor-corrector or iterative
variable step adjust h to local steepness	fixed step obligatory in multi-step

classical mechanics

the problem



Newton

$$\ddot{x}(t) = F(x(t))/m$$

one second order ODE

or two coupled first order ODEs

$$\dot{x}(t) = v(t)$$

$$\dot{v}(t) = F(x(t))/m$$

units

- computers do not know units, only numbers.
  - units are the programmer's responsibility
  - stick to one convention, e.g. S.I.

- dimensionless parameters are often useful,
  - to reduce number of 'independent' parameters
  - to map problems

Euler

$$\dot{x}(t) = v(t) \qquad \longrightarrow \qquad x_{i+1} = x_i + v_i \Delta t$$

$$\dot{v}(t) = F(x(t))/m \qquad \qquad v_{i+1} = v_i + F(x_i)\Delta t/m$$

Global error is first order.
typically performs *very* badly.

Euler - Cromer

$$\dot{x}(t) = v(t)$$

$$\dot{v}(t) = F(x(t))/m$$

$$\longrightarrow$$

$$v_{i+1} = v_i + F(x_i)\Delta t/m$$

$$x_{i+1} = x_i + v_{i+1}\Delta t$$

Global error first order.
typically performs rather well (!?)

(Störmer-) Verlet

$$
\begin{aligned}
x(t+\Delta t) &= x(t) + \dot{x}(t)(+\Delta t) + \tfrac{1}{2}\ddot{x}(t)(+\Delta t)^2 + \tfrac{1}{6}\dddot{x}(t)(+\Delta t)^3 + O(\Delta t^4) \\
x(t-\Delta t) &= x(t) + \dot{x}(t)(-\Delta t) + \tfrac{1}{2}\ddot{x}(t)(-\Delta t)^2 + \tfrac{1}{6}\dddot{x}(t)(-\Delta t)^3 + O(\Delta t^4)
\end{aligned}
$$

$+$

$$
x(t+\Delta t) + x(t-\Delta t) = 2x(t) + 0 + \ddot{x}(t)(\Delta t)^2 + 0 + O(\Delta t^4)
$$

$$
\boxed{x_{i+1} = 2x_i - x_{i-1} + F(x_i)(\Delta t)^2 / m}
$$

global error is second order.
shares with Newton: time reversible and 'symplectic'.
performs very well.

leap frog (Verlet)

$$x_{i+1} = 2x_i - x_{i-1} + F(x_i)(\Delta t)^2 / m \longrightarrow \boxed{\begin{aligned} w_{i+1} &= w_i + F(x_i)\Delta t / m \\ x_{i+1} &= x_i + w_{i+1}\Delta t \end{aligned}}$$

The x_i are still second order, time reversible and symplectic.
The w_i are first order approximations for $v\left(\left(i-\frac{1}{2}\right)\Delta t\right)$.
Performs very well.

note similarity to Euler-Cromer.

Runge-Kutta

$$\dot{x}(t) = v(t)$$

$$\dot{v}(t) = F\left(x(t)\right)/m$$

$$\longrightarrow \quad \mathbf{u} = \begin{pmatrix} x \\ v \end{pmatrix} \quad \& \quad \dot{\mathbf{u}} = \mathbf{f}(\mathbf{u})$$

solve in matlab using ode23 or ode45.

problem for this week

the harmonic oscillator

$$m\ddot{x} = -\frac{d}{dx}\left(\tfrac{1}{2}kx^2\right)$$

$$m = 2\text{kg}$$

$$k = 5\,\text{N/m}$$

solve in matlab
using Euler, Verlet and Runge-Kutta (ode45).

detailed questions on blackboard.