# Numerical methods

## Lectures 1 and 2: Explicit methods for numerically solving ODEs

Oscar Benjamin and Lucia Marucci



## Solving ODEs numerically: first-order

Out in industry, most (non-trivial) ODEs are solved numerically. This is relatively easy to do provided the ODE is *nice* (said to be *non-stiff*).

Computers are inherently discrete devices (being digital), thus numerical solutions are *discrete approximations* to the actual solution.

- What does this actually mean?
- Have the solution at particular data points *not* continuously*

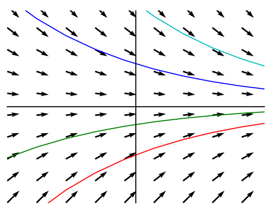
Solution will only be an *approximation*. Key question: how good is the approximation?
Simplest method of solving ODEs: *Euler's method*

- Also least accurate method available



## General solutions and initial conditions

Consider $\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = -x$ which has solutions $x = x_0 \mathrm{e}^{-t}$. Analytically we find the general solution and use initial conditions to find the constants.

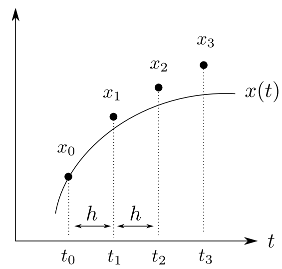


Numerically we must *begin* with an initial condition.



## Numerical solutions are discrete

We represent the continuous solutions $x(t)$ using a discrete set of values $x_0$, $x_1$, $x_2$ ... which are estimates of the true solution at discrete times $t_1$, $t_2$ ...



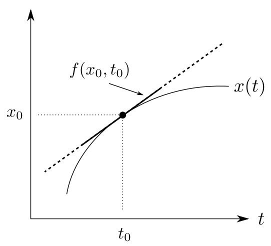

Hopefully $x_n$ will be close to $x(t_n)$...



## Initial value problem

We want to solve

$$\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = f(x,t), \quad x(t_0) = x_0$$

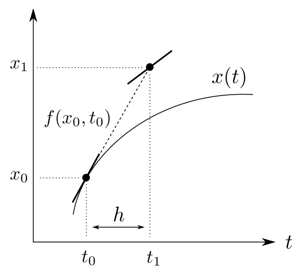



We have the first point $(x_0, t_0)$ so how do we find $x_1$?



## Euler's method in pictures

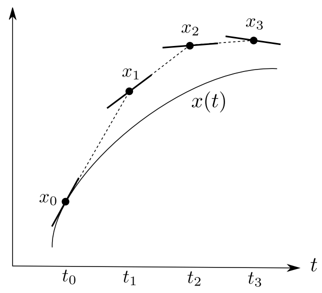
$$x_1 = x_0 + hf(x_0, t_0)$$




Assume the solution continues in a straight line with the same gradient that it must have at $t_0$.

## Euler's method in pictures

$$x_{n+1} = x_n + hf(x_n, t_n)$$





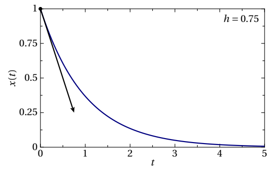

## Solving ODEs numerically: first-order

Consider the general first-order ODE

$$\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = f(x, t)$$

Formally, we can write Euler's method as generating a *sequence* of data points $\{x_n\}$ such that

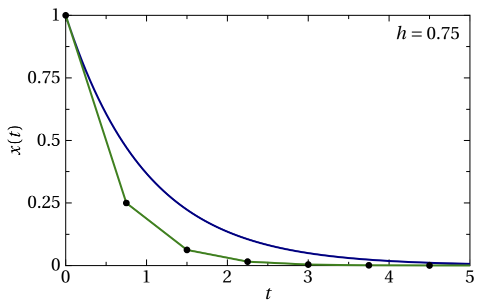
$$x_{n+1} = x_n + h\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = x_n + hf(x_n, t_n)$$






## Solving ODEs numerically: first-order

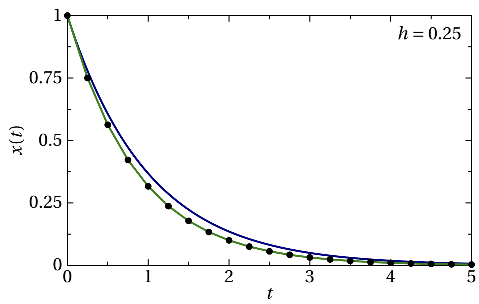
$$\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = -x$$






## Solving ODEs numerically: first-order

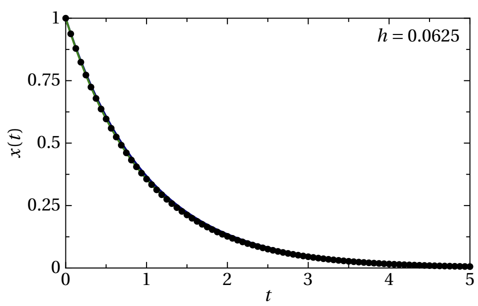
$$\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = -x$$






## Solving ODEs numerically: first-order

$$\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = -x$$






## Solving ODEs numerically: first-order

**Euler's method**

$$x_{n+1} = x_n + h\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = x_n + hf(x_n, t_n)$$

**Key idea**

As $h \to 0$ the error between the *numerical approximation* and the *true solution* goes to zero.

## Solving ODEs numerically: first-order

**Example**

$$\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = -x = f(x,t)$$

Starting from the point $x(0) = x_0 = 1$ with $h = 0.5$

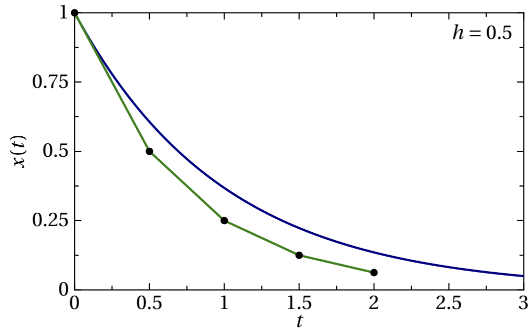
$$x_1 = \underbrace{1}_{x_0} + \underbrace{0.5}_{h} \times f(\underbrace{1}_{x_0}, \underbrace{0}_{t_0}) = 1 - 0.5 = 0.5$$
$$x_2 = 0.5 + 0.5 \times f(0.5, 0.5) = 0.5 - 0.25 = 0.25$$
$$x_3 = 0.25 + 0.5 \times f(0.25, 1) = 0.25 - 0.125 = 0.125$$
$$x_4 = 0.125 + 0.5 \times f(0.125, 1.5) = 0.125 - 0.0625 = 0.0625$$

Thus we have a *discrete approximation* of the solution to the ODE above

$$x \approx \{\{0, x_0\}, \{0.5, x_1\}, \{1, x_2\}, \{1.5, x_3\}, \{2, x_4\}\}$$
$$= \{\{0, 1\}, \{0.5, 0.5\}, \{1, 0.25\}, \{1.5, 0.125\}, \{2, 0.0625\}\}$$



## Solving ODEs numerically: first-order






## Solving ODEs numerically: higher-order

Euler's method also works for higher-order ODEs in *state-space form*!

$$\frac{\mathrm{d}\,\vec{y}}{\mathrm{d}\,t} = \vec{f}(\vec{y}, t) \qquad \Rightarrow \qquad \vec{y}_{n+1} = \vec{y}_n + h\vec{f}(\vec{y}_n, t_n)$$

**Example**

$$\frac{\mathrm{d}^2\,x}{\mathrm{d}\,t^2} + 0.1\frac{\mathrm{d}\,x}{\mathrm{d}\,t} + x = \sin(t) \qquad \text{with} \qquad x(0) = 1,\ \dot{x}(0) = 0$$

Rewrite in *state-space form*; $y_0 = x$ and $y_1 = \frac{\mathrm{d}\,x}{\mathrm{d}\,t}$

$$\frac{\mathrm{d}\,y_0}{\mathrm{d}\,t} = y_1$$
$$\frac{\mathrm{d}\,y_1}{\mathrm{d}\,t} = -0.1y_1 - y_0 + \sin(t)$$

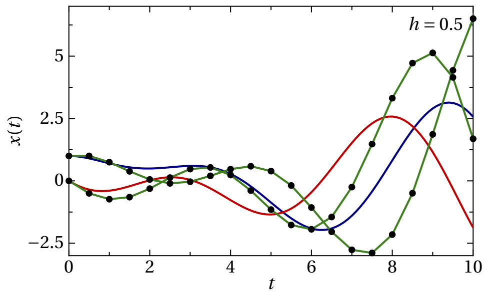
Applying Euler's method yields

$$y_{0,n+1} = y_{0,n} + hy_{1,n}, \qquad y_{1,n+1} = y_{1,n} - h\,(0.1y_{1,n} + y_{0,n} - \sin(t_n))$$



## Solving ODEs numerically: higher-order

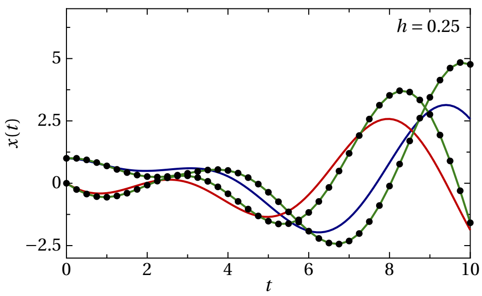
$$\frac{\mathrm{d}^2\,x}{\mathrm{d}\,t^2} + 0.1\frac{\mathrm{d}\,x}{\mathrm{d}\,t} + x = \sin(t) \qquad \text{with} \qquad x(0) = 1,\ \dot{x}(0) = 0$$






## Solving ODEs numerically: higher-order

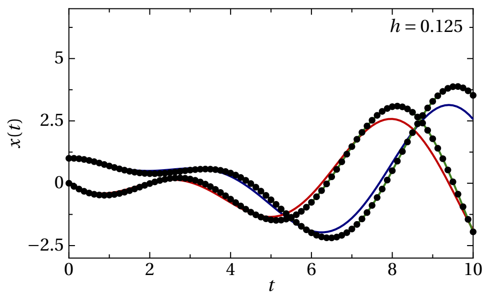
$$\frac{\mathrm{d}^2\,x}{\mathrm{d}\,t^2} + 0.1\frac{\mathrm{d}\,x}{\mathrm{d}\,t} + x = \sin(t) \qquad \text{with} \qquad x(0) = 1,\ \dot{x}(0) = 0$$





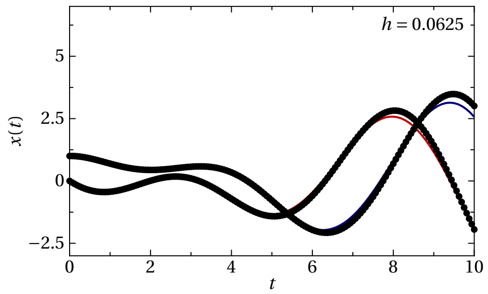

## Solving ODEs numerically: higher-order

$$\frac{\mathrm{d}^2\,x}{\mathrm{d}\,t^2} + 0.1\frac{\mathrm{d}\,x}{\mathrm{d}\,t} + x = \sin(t) \qquad \text{with} \qquad x(0) = 1,\ \dot{x}(0) = 0$$



## Solving ODEs numerically: higher-order

$$\frac{\mathrm{d}^2 x}{\mathrm{d}t^2} + 0.1\frac{\mathrm{d}x}{\mathrm{d}t} + x = \sin(t) \qquad \text{with} \qquad x(0) = 1,\ \dot{x}(0) = 0$$






## Solving ODEs numerically: errors

A question you should *always* ask of any numerical method is *"how large are the errors?"* — there is always a difference between the *true solution* and the *approximate numerical solution*

"If you don't care about the error, then the answer is 42"

Generally it is difficult to give precise estimates without knowing the true solution. . .

Instead we use *asymptotic estimates*

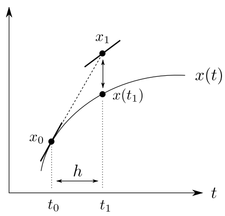
- Try to capture the behaviour of the method as $h \to 0$
- Typically uses big-O notation



## Local error

Local error:

$$\text{error} = x_1 - x(t_1)$$

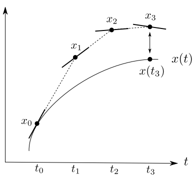



The error is the difference between our estimate and the true solution. The *local error* is the error after *one step*.
The error gets smaller as the stepsize $h$ gets smaller.



## Global error

Global error after $n$ steps. $\quad \text{error} = x_n - x(t_n)$



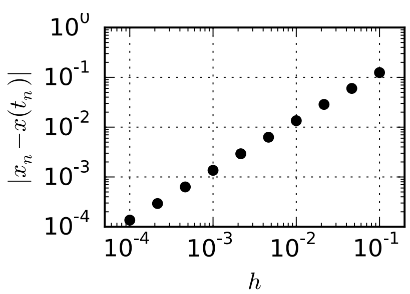

The global error after $n$ steps is the difference between our estimate $x_n$ and the true solution $x(t_n)$.
Hopefully the global error gets smaller when the local error gets smaller - but this is not always the case (e.g. for stiff problems).



Global error using Euler's method to solve for $x(1)$ given

$$\frac{\mathrm{d}x}{\mathrm{d}t} = x, \quad x(0) = 1$$

(True answer is e.)

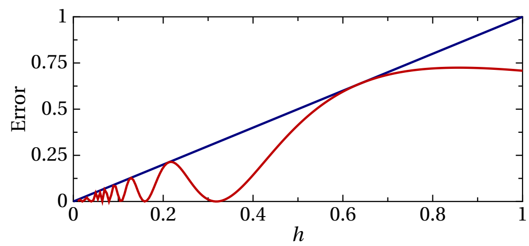





## Solving ODEs numerically: big-O notation

*Big-O notation* tells us how fast the error decays; generally the faster it decays the more accurate the method.

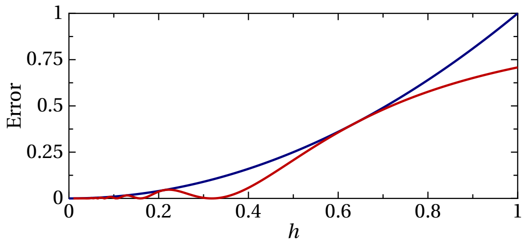
For Euler's method the (global) error decays at a rate $O(h)$. This means that, at worst, it decays linearly.




Roughly speaking, $O(h)$ error means that *halving* the step size $h$ will *halve* the error

## Solving ODEs numerically: big-O notation

Better numerical methods for solving ODEs have higher convergence rates for the error.
For example, the *explicit midpoint rule* for integrating ODEs is $O(h^2)$ accurate.

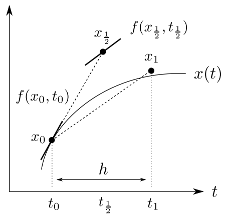



Roughly speaking, $O(h^2)$ error means that *halving* the step size $h$ will *quarter* the error



## Explicit midpoint method

$$x_{n+\frac{1}{2}} = x_n + f(x_n, t_n)\frac{h}{2}, \qquad x_{n+1} = x_n + f(x_{n+\frac{1}{2}}, t_{n+\frac{1}{2}})h$$




Estimate the midpoint (using a half-size Euler step) and then calculate the gradient $f(x_{n+\frac{1}{2}}, t_{n+\frac{1}{2}})$ at the midpoint and use that to step from $x_n$ to $x_{n+1}$.



## Solving ODEs numerically: high-order explicit methods

We can carry on to generate higher-order explicit methods in a similar manner. The most common one is *Runge-Kutta*

$$\begin{aligned}
k_1 &= hf(x_n, t_n)\\
k_2 &= hf(x_n + k_1/2, t_n + h/2)\\
k_3 &= hf(x_n + k_2/2, t_n + h/2)\\
k_4 &= hf(x_n + k_3, t_n + h)\\
x_{n+1} &= x_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

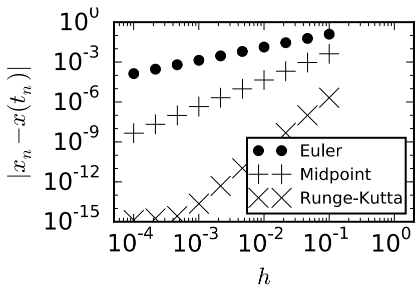
which is $O(h^4)$ accurate. This method is the one you should try first!



Global error using different methods to solve for $x(1)$ given

$$\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = x, \quad x(0) = 1$$

(True answer is $\mathrm{e}$.)






## Solving ODEs numerically: big-O notation

Formally, if we have a function $f$ which represents the error,

$$f(h) = O(g(h)) \qquad \text{as} \quad h \to 0$$

if and only if there exists a constant $M$ such that

$$|f(h)| \leq M|g(h)| \qquad \text{for all} \quad h < \delta$$

for any (arbitrary) choice of $\delta$.

The function $g$ can be any function which tends to zero as $h \to 0$:

- $g(h) = h$ — Euler's method
- $g(h) = h^2$ — Trapezium rule
- $g(h) = h^4$ — Runge-Kutta

Most of the time we use *Runge-Kutta*!



## Solving ODEs numerically: error analysis

Euler's method is written as

$$x(t+h) = x(t) + h\frac{\mathrm{d}\,x}{\mathrm{d}\,t}$$

We can calculate the error by comparing this with an exact expression for $x(t+h)$; we get an exact expression from a Taylor series

$$x(t+h) = x(t) + h\frac{\mathrm{d}\,x}{\mathrm{d}\,t} + \frac{h^2}{2!}\frac{\mathrm{d}^2\,x}{\mathrm{d}\,t^2} + \frac{h^3}{3!}\frac{\mathrm{d}^3\,x}{\mathrm{d}\,t^3} + O(h^4)$$

The error at each step is the difference between these two expressions; thus the error is

$$\text{error at each step} = \frac{h^2}{2!}\frac{\mathrm{d}^2\,x}{\mathrm{d}\,t^2} + \frac{h^3}{3!}\frac{\mathrm{d}^3\,x}{\mathrm{d}\,t^3} + O(h^4)$$

As $h$ gets *smaller*, the largest term will be the quadratic term $h^2$, thus the error at each step decays $O(h^2)$.

# Solving ODEs numerically: error analysis

The error rate that is typically quoted is the *global error*, which is the total error in getting to a particular point in time.

To find the *global error*, consider integrating the equations until $t = T$.

With a step size of $h$, it takes $T/h$ steps to get to time $t = T$. If the error in each step is $O(h^2)$ after $T/h$ steps the error will be $O(h)$.

Thus, the *global error* for Euler's method is $O(h)$.