Process Model Formulation and Solution, 3E4

Section F, Ordinary Differential Equations

Instructor: Kevin Dunn dunnkg@mcmaster.ca

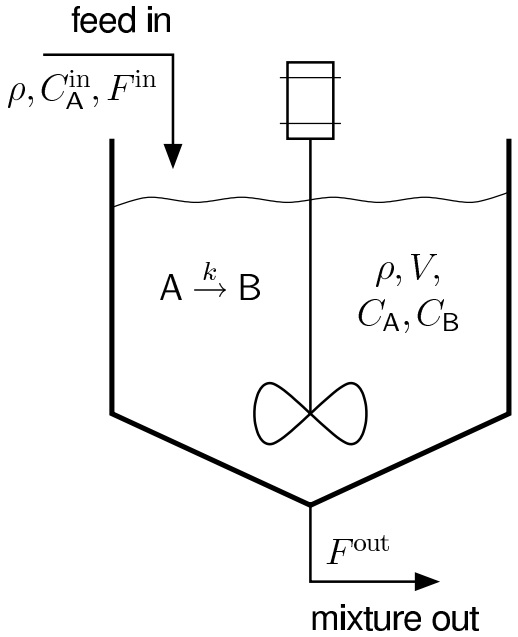
McMaster University

Department of Chemical Engineering

15 November 2010

Why solve ordinary differential equations?

Consider the modelling of an isothermal CSTR, with a single inlet stream.



Assumptions:

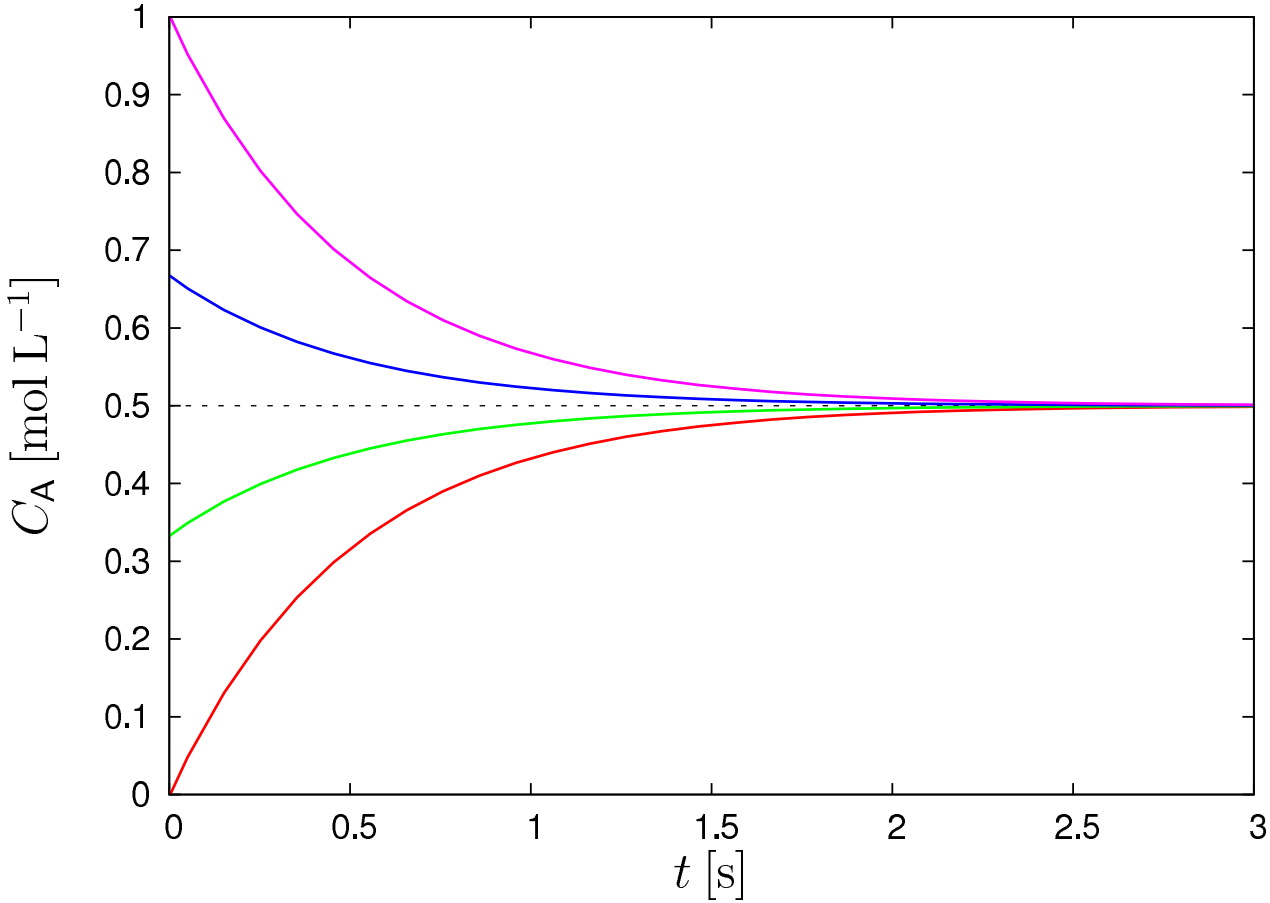
- **A1:** Perfect mixing
- **A2:** Equal inflow and outflow
- **A3:** Constant liquid density
- **A4:** Single first-order reaction

Modelling goals:

- Calculate the concentration C_A in the reactor at a given time $t \geq 0$

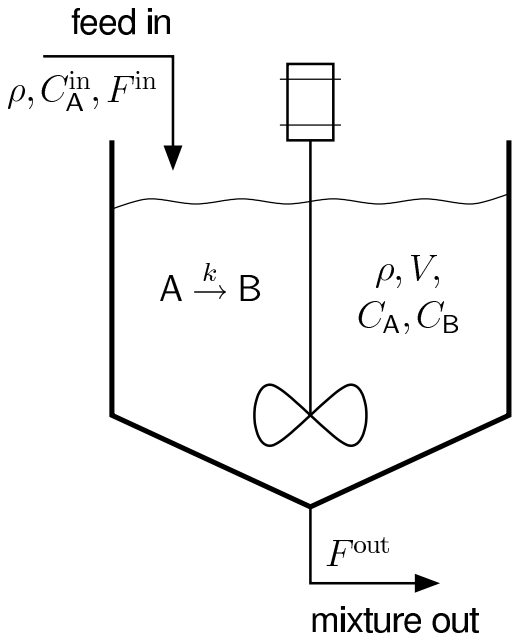
Why solve ordinary differential equations?

F^{in}	C_{A}^{in}	V	k
$1\ \text{L s}^{-1}$	$1\ \text{mol L}^{-1}$	$1\ \text{L}$	$1\ \text{s}^{-1}$



Why solve ordinary differential equations?

Consider the modelling of a CSTR, with a single inlet stream.



Assumptions:

- **A1:** Perfect mixing
- **A2:** Outflow $F^{\text{out}}(t) = \alpha\sqrt{V(t)}$
- **A3:** Constant liquid density
- **A4:** Single first-order reaction

Modeling Goals:

- Calculate the volume V and the concentration C_A in the reactor at a given time $t \geq 0$

Outline and recommended readings

Motivation

Differential equations: concepts and classification

Numerical solution of ODEs
- Euler's method
- Taylor series methods
- Runge-Kutta methods
- Heun's predictor-corrector method

(Strongly) recommended readings:

- Chapters PT7, 25.1-25.4, and 26.1 in: S. C. Chapra, and R. P. Canale, *"Numerical Methods for Engineers"*, McGraw Hill, 5th/6th edition

Differential equations: concepts and classification

We consider ordinary differential equations (ODE) of the following form:

$$\begin{cases} x_1'(t) = f_1(t, x_1, \ldots, x_n) \\ \quad\vdots \\ x_n'(t) = f_n(t, x_1, \ldots, x_n) \end{cases} \qquad \text{or} \qquad \mathbf{x}'(t) = \mathbf{f}(t, \mathbf{x}(t))$$

- t, independent variable; $\mathbf{x}$, dependent variable(s)
- Same number of dependent variables and equations

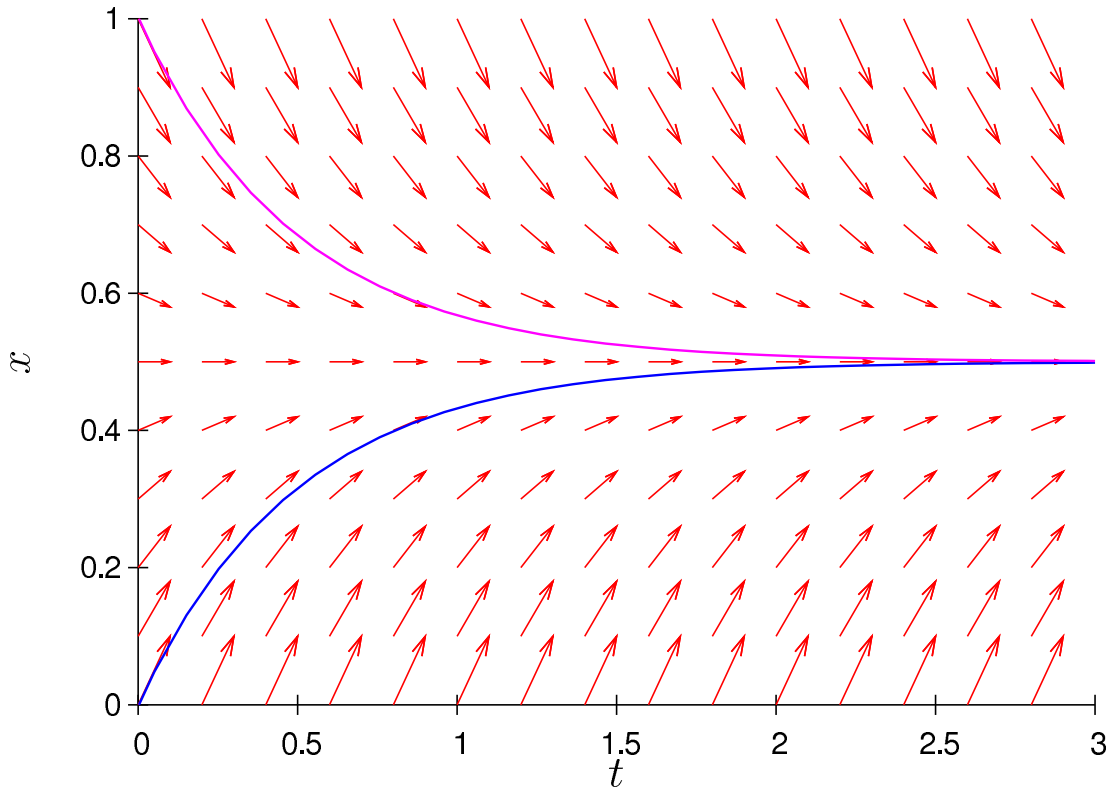
Scalar vs. vectorial ODEs:

- **Scalar ODE:** single variable/equation, $x(t) \in \mathbb{R}$
- **Vectorial ODEs:** multiple variables/equations, $\mathbf{x}(t) \in \mathbb{R}^n$
- **Examples:** Revisit CSTR case studies

Need an equal number of initial conditions as dependent variables!

Differential equations: concepts and classification

Geometrical interpretation: Scalar ODEs $\boxed{x'(t) = 1 - 2x(t)}$



- Infinitely many curves satisfy the differential equation!
- Extra condition needed to uniquely specify the solution: $\boxed{x(0) = x_0}$

Differential equations: concepts and classification

Initial Value Problems (IVPs): ⟵ our focus

- All initial conditions specified at the same time
- **Example:**

$$\begin{cases} x_1'(t) = 2x_1(t) - x_2(t); & x_1(0) = a \\ x_2'(t) = -x_1(t) + 3x_2(t); & x_2(0) = b \end{cases}$$

Boundary Value Problems (BVPs):

- Not all initial conditions specified at the same time
  - Two-point boundary value problem: Split boundary conditions
  - Also, multi-point boundary value problem
- **Example:**

$$\begin{cases} x_1'(t) = 2x_1(t) - x_2(t); & x_1(0) = a \\ x_2'(t) = -x_1(t) + 3x_2(t); & x_2(1) = b \end{cases}$$

Differential equations: concepts and classification

linear ordinary differential equations:

$$\begin{cases} x_1'(t) = a_{11}(t)x_1(t) + \cdots + a_{1n}(t)x_n(t) + b_1(t) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ x_n'(t) = a_{n1}(t)x_1(t) + \cdots + a_{nn}(t)x_n(t) + b_n(t) \end{cases} \quad \text{or} \quad \mathbf{x}'(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{b}(t)$$

- a_{ij} and b_j may depend on time (differentiable functions in t)
- Analytical solutions are known for linear ODEs
- Solution are guaranteed to exist at all times

Nonlinear ordinary differential equations:

$$\begin{cases} x_1'(t) = f_1(t, x_1(t), \ldots, x_n(t)) \\ \qquad\qquad \vdots \\ x_n'(t) = f_n(t, x_1(t), \ldots, x_n(t)) \end{cases} \quad \text{or} \quad \mathbf{x}'(t) = \mathbf{f}(t, \mathbf{x}(t))$$

- General, nonlinear functions $f_1, \ldots, f_n$ (differentiable in $t, x_1, \ldots, x_n$)
- No analytical solution in general!
- Solution may not exist at all time instants

Differential equations: concepts and classification

Example: Linear *vs.* nonlinear ODEs

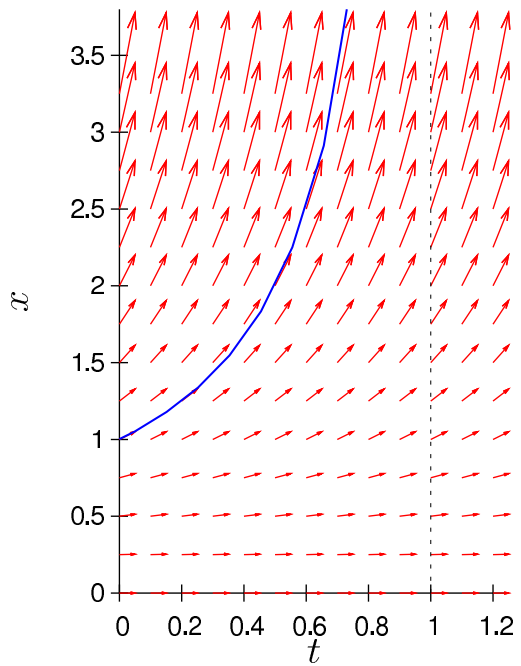
Which of the following ODEs are linear? Which ones are nonlinear?

$$\begin{cases} x_1'(t) = 2x_1(t) - t\,x_2(t) \\ x_2'(t) = -x_1(t) + 3x_2(t) \end{cases} \qquad \begin{cases} x_1'(t) = 2\mathrm{e}^{-t}[1 - x_2(t)] \\ x_2'(t) = x_1(t)[3x_2(t) - 1] \end{cases}$$

Example 2: A "nasty" nonlinear ODE

Consider the scalar ODE: $x'(t) = x^2(t)$, with the initial condition $x(0) = x_0$

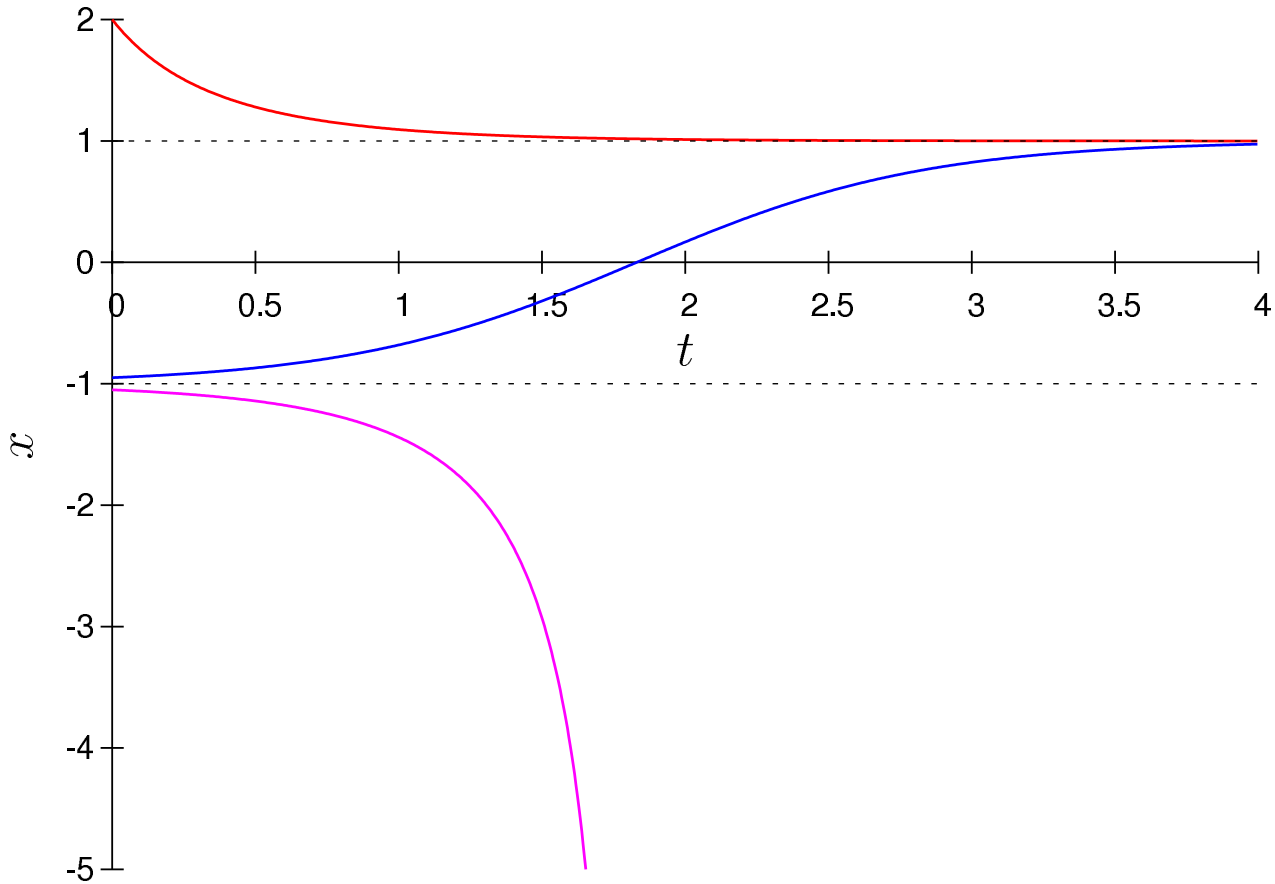
- Is $x(t) = \dfrac{x_0}{1 - x_0 t}$ a solution?
- What's happening for $x_0 = 0$?
- What's happening when $x_0 > 0$?



Differential equations: concepts and classification

Example 3: asymptotic behaviour and stability

Consider the scalar ODE: $x'(t) = 1 - x^2(t)$. Observe and discuss the solutions obtained from various initial conditions at $t = 0$.



Differential equations: concepts and classification

First- and higher-order ODEs:

- 1st-Order ODE: $x'(t) = f(t, x(t))$
- nth-Order ODE: $\boxed{x^{(n)}(t) = f(t, x(t), x'(t), \ldots, x^{(n-1)}(t))}$

The good news!

A nth-order ODE is equivalent to a system of n 1st-order ODEs: **how?**

- The solution to a nth-order ODE is uniquely specified by exactly n initial conditions
  - e.g., $x(0), x'(0), \ldots, x^{n-1}(0)$

Workshop:

Reformulate the following 2nd-order ODE as a set of 1st-order ODEs:

$$x''(t) + x(t)[3x'(t) - 1] = \mathrm{e}^{-t}$$

Numerical solution of ODEs

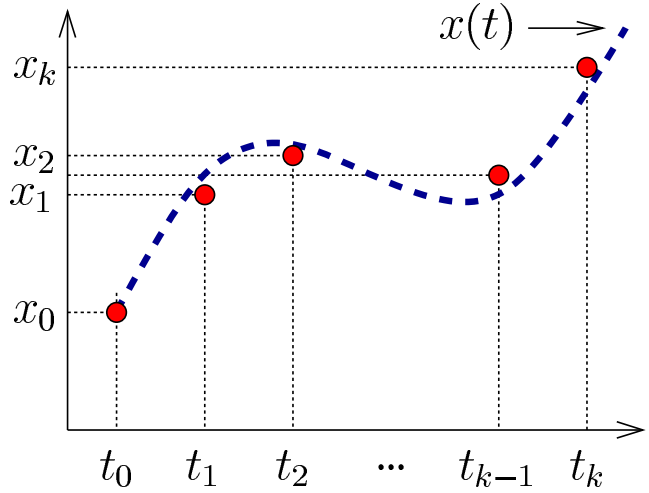
Exact solution to an initial value problem in ODEs:

- x satisfies the initial condition $x(t_0) = x_0$
- x satisfies the differential relation $x'(t) = f(t, x(t))$ at **any** $t \geq t_0$

The numerical solution problem

Given k time instants $t_1, t_2, \ldots, t_k$,
calculate values $x_1, x_2, \ldots, x_k$ such that:

$$x_1 \approx x(t_1),\ x_2 \approx x(t_2), \ldots, x_k \approx x(t_k)$$



All numerical solution methods for ODEs are iterative!

- Given solution estimates $x_1, x_2, \ldots, x_k$ at previous points $t_1, t_2, \ldots, t_k$, calculate an estimate x_{k+1} at next point t_{k+1}
  - **One-step methods:** estimate x_{k+1} by using x_k only
  - **Multi-step methods:** estimate x_{k+1} by using $x_k, x_{k-1}, \ldots$

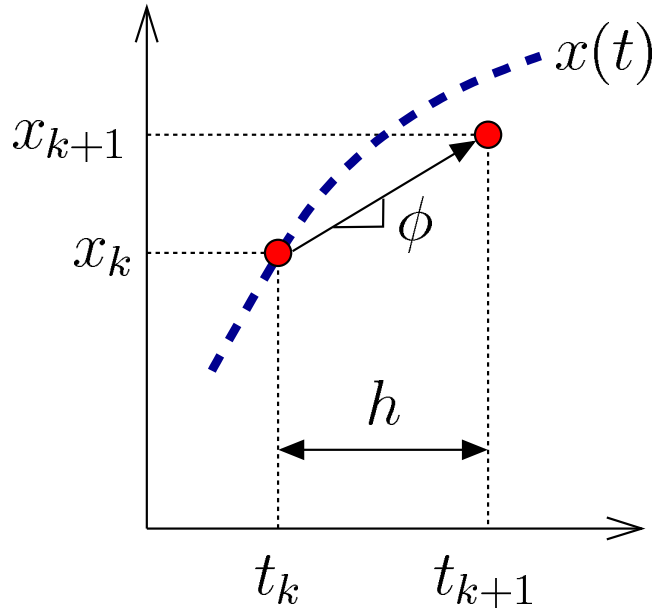
One-step solution methods: principles

Main concept:

Given $x_k \approx x(t_k)$, calculate $x_{k+1} \approx x(t_{k+1})$ as:

$$x_{k+1} = x_k + \phi\, h$$

- ϕ, slope
- $h = t_{k+1} - t_k$, stepsize



- The slope ϕ is used to extrapolate to a new value
- The formula can be applied step-by-step to approximate the solution:
  1. **Initialize:** $k \leftarrow 0$, $x_0 = x(t_0)$
  2. **Repeat:** $k \leftarrow k+1$, $t_{k+1} = t_k + h$, $x_{k+1} = x_k + \phi\, h$; **Until:** $t_{k+1} \geq t_f$
- One-step methods differ in the manner the slope is estimated
- Applicable to both scalar and vectorial ODEs

Euler's method: basics (a.k.a. Euler-Cauchy, or Point-Slope)

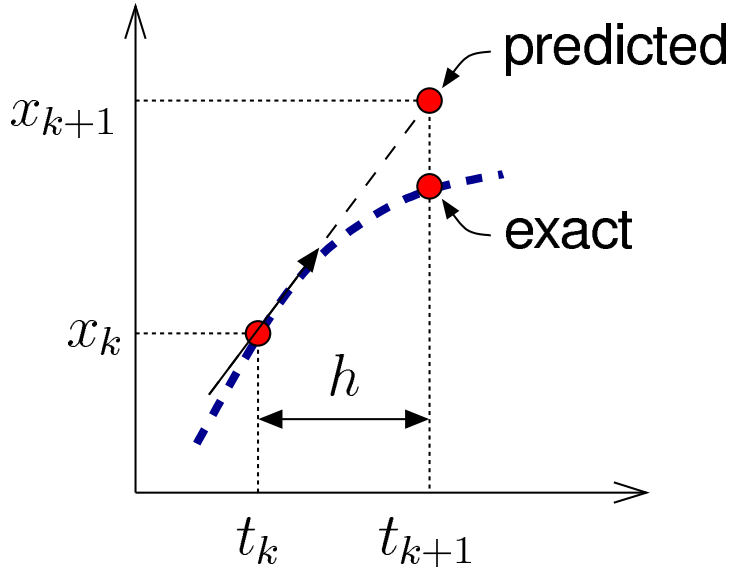
Main concept:

Use the time derivative as the slope estimate:

$$\phi = x'(t_k) = f(t_k, x_k)$$

Calculate $x_{k+1} \approx x(t_{k+1})$ as:

$$\boxed{x_{k+1} = x_k + f(t_k, x_k)\, h}$$



(Local) error analysis:

- The error is: $e_{k+1} = x(t_{k+1}) - x_{k+1} = x(t_{k+1}) - x_k - f(t_k, x_k)\, h$
- If $x_k = x(t_k)$, then $\boxed{e_{k+1} = O(h^2)}$ **why?**

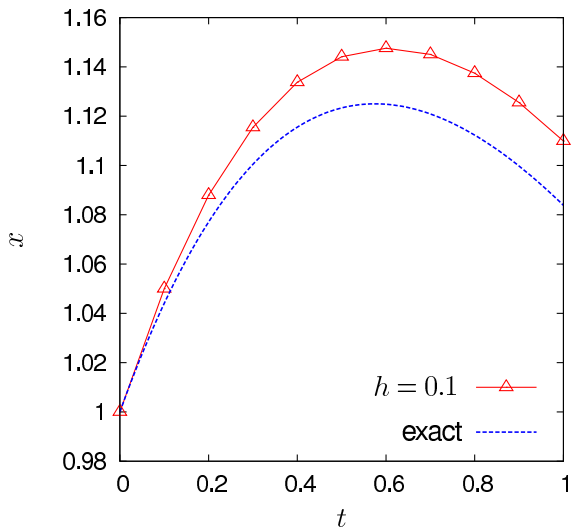
Euler's method: application

Use Euler's method to solve the scalar ODE $x'(t) = -\frac{1}{2}x(t) + \exp(-t)$ for $t \in [0, 1]$, with initial condition $x(0) = 1$ and various stepsizes $h = 10^{-1}$, 10^{-2}, 10^{-3} and 10^{-4}

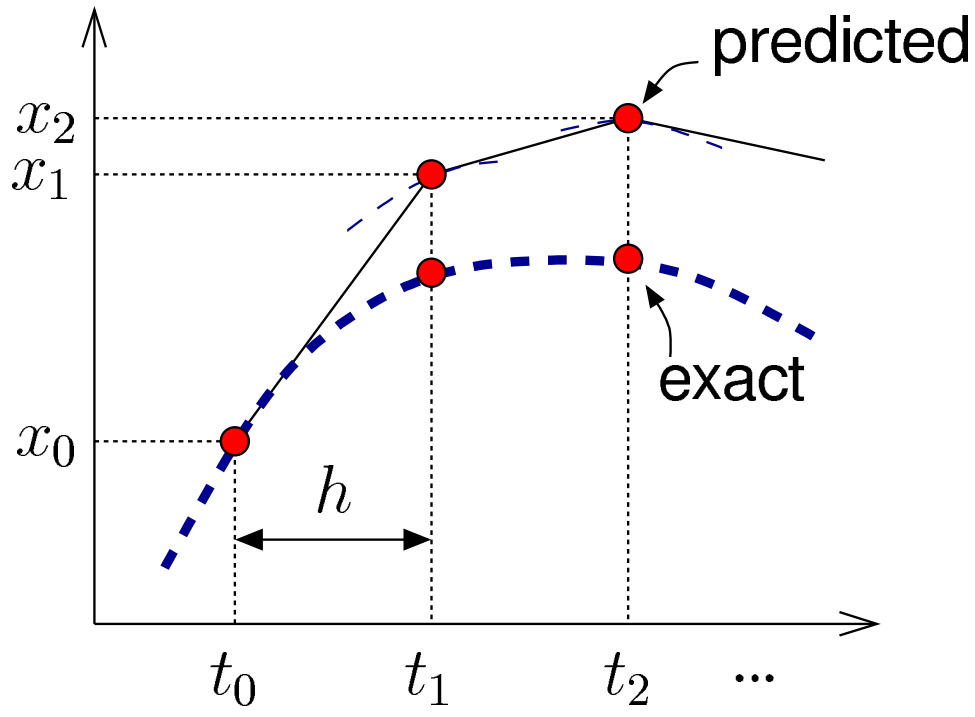
Exact solution: $\boxed{x(t) = 3\exp(-\frac{1}{2}t) - 2\exp(-t)}$

h	x_1	e_1	x_N	e_N
1e-1	1.05	-5.99e-3	1.10991	-2.61e-2
1e-2	1.005	-6.22e-5	1.08634	-2.51e-3
1e-3	1.0005	-6.25e-7	1.08408	-2.50e-4
1e-4	1.00005	-6.25e-9	1.08386	-2.50e-5
			$x(1) = 1.08383$	

Error on $x(1)$ is not $O(h^2)$!



Euler's method: revisited



- **Global error analysis:** The global error grows after each step!
- The number of steps to go from t_0 to t_f is $N \approx \frac{t_f - t_0}{h}$

$$e_{\text{tot}} = \sum_{k=1}^{N} e_k = N \times O(h^2) = O(h)$$: **why?**

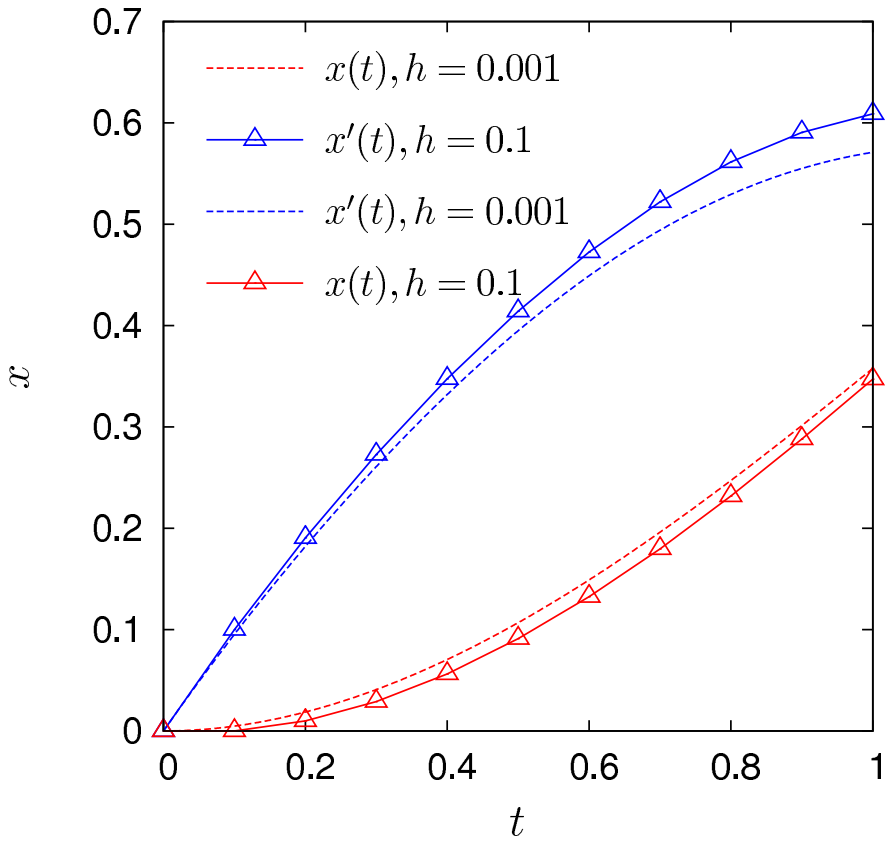
Euler's method: coding

Write a function that solves the differential equation $x'(t) = f(t, x(t))$ for $t \in [t_0, t_f]$, with initial condition $x(t_0) = x_0$, using N steps

Euler's method: application to vectorial ODEs

Use Euler's method to solve the high-order ODE $x''(t) + x(t)[3x'(t) - 1] = \exp(-t)$ for $t \in [0, 1]$, with initial conditions $x(0) = 0$, $x'(0) = 0$, and stepsize $h = 10^{-1}$

t_k	$x_{1,k}$	$x_{2,k}$
0	0	0
0.1000	0	0.1000
0.2000	0.0100	0.1905
0.3000	0.0290	0.2728
0.4000	0.0563	0.3474
0.5000	0.0911	0.4142
0.6000	0.1325	0.4726
0.7000	0.1797	0.5220
0.8000	0.2319	0.5615
0.9000	0.2881	0.5905
1.0000	0.3471	0.6090



Taylor series method: principles

Principle: Try to get more accurate estimates with larger/fewer stepsizes!

- Include higher-order terms of the Taylor series expansion of $x(t)$:
$$x(t+h) \approx x(t) + x'(t)h + x''(t)\frac{h^2}{2} + \cdots + x^{(j)}(t)\frac{h^j}{j!}$$
- $k = 1$: [Euler's method] local error $O(h^2)$; global error $O(h)$
$$x'(t_k) \approx f(t_k, x_k),$$
$$x_{k+1} = x_k + f(t_k, x_k)h$$
- $k = 2$: local error $O(h^3)$; global error $O(h^2)$
$$x''(t_k) \approx \frac{\partial f(t_k, x_k)}{\partial t} + \frac{\partial f(t_k, x_k)}{\partial x} f(t_k, x_k),$$
$$x_{k+1} = x_k + f(t_k, x_k)h + \left[\frac{\partial f(t_k, x_k)}{\partial t} + \frac{\partial f(t_k, x_k)}{\partial x} f(t_k, x_k)\right]\frac{h^2}{2}$$
- $k = 3$: ...

Problem: Higher-order derivatives become increasingly more complicated

Taylor series method: application

Use the Taylor series method (order 2) to solve the scalar ODE $x'(t) = -\frac{1}{2}x(t) + \exp(-t)$ for $t \in [0, 1]$, with initial condition $x(0) = 1$ and stepsize $h = 10^{-1}$

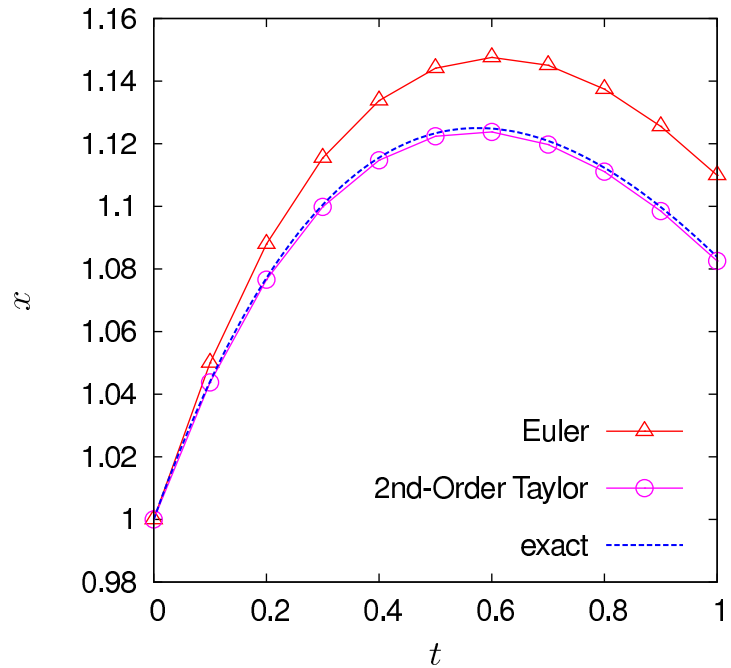
Exact solution: $\boxed{x(t) = 3\exp(-\frac{1}{2}t) - 2\exp(-t)}$

- **Euler's method** $(k = 1)$:

$$x_{k+1} = x_k + \left[-\frac{x_k}{2} + \exp(-t_k)\right] h$$

- **Taylor series method** $(k = 2)$:

$$x_{k+1} = x_k + ?$$



Runge-Kutta methods in a nutshell

$$\boxed{x'(t) = f(t, x(t)); \quad x(t_0) = x_0}$$

Procedure for all Runge-Kutta methods

- Given an estimate x_k of the solution at time t_k,
- Calculate the next estimate $x_{k+1} = x_k + \phi\, h$, with:

$$\phi = a_1 K_1 + a_2 K_2 + \cdots + a_N K_N, \qquad N\text{: Order of the method}$$

and,

$$\begin{aligned}
K_1 &= f(t_k, x_k) \\
K_2 &= f(t_k + p_1 h, x_k + q_{1,1}\, K_1\, h) \\
K_3 &= f(t_k + p_2 h, x_k + q_{2,1}\, K_1\, h + q_{2,2}\, K_2\, h) \\
&\vdots \\
K_N &= f(t_k + p_{N-1} h, x_k + q_{N-1,1}\, K_1\, h + q_{N-1,2}\, K_2\, h + \cdots + q_{N-1,N-1}\, K_{N-1}\, h)
\end{aligned}$$

Runge-Kutta methods in a nutshell

- The K_i's are computed recursively
- N function evaluations are required at each time step
  - Euler's method if $N = 1$
- **Parameters:** p_k, $q_{k,i}$, a_i

Butcher tableau:

$$\begin{array}{c|cccc} p_1 & q_{1,1} & & & \\ p_2 & q_{2,1} & q_{2,2} & & \\ \vdots & \vdots & \vdots & \ddots & \\ p_{N-1} & q_{N-1,1} & q_{N-1,2} & \cdots & q_{N-1,N-1} \\ \hline a_1 & a_2 & \cdots & a_{N-1} & a_N \end{array}$$

- Consistent Runge-Kutta scheme: $\boxed{p_k = \sum_i q_{k,i}}$

Selection of the parameters p_k, $q_{k,i}$, a_i

Runge-Kutta of order N to be as accurate as a Taylor Series of order N

Second-order Runge-Kutta method: derivation

Formula:

$$x_{k+1} = x_k + \left[a_1 \underbrace{f(t_k, x_k)}_{K_1} + a_2 \underbrace{f(t_k + p_1 h, x_k + q_{1,1} f(t_k, x_k) h)}_{K_2} \right] h$$

- 4 coefficients: $p_1, q_{1,1}, a_1, a_2$
- Express equivalence conditions with 2nd-order Taylor series method:

$$\boxed{a_1 + a_2 = 1, \quad a_2\, p_1 = \frac{1}{2}, \quad a_2\, q_{1,1} = \frac{1}{2}}$$

: why?

  - These conditions ensure: Local error of $O(h^3)$: global error of $O(h^2)$
  - Consistency condition trivially satisfied: $p_1 = q_{1,1}$
  - 4 parameters $\leftrightarrow$ 3 conditions: 1 degree of freedom!

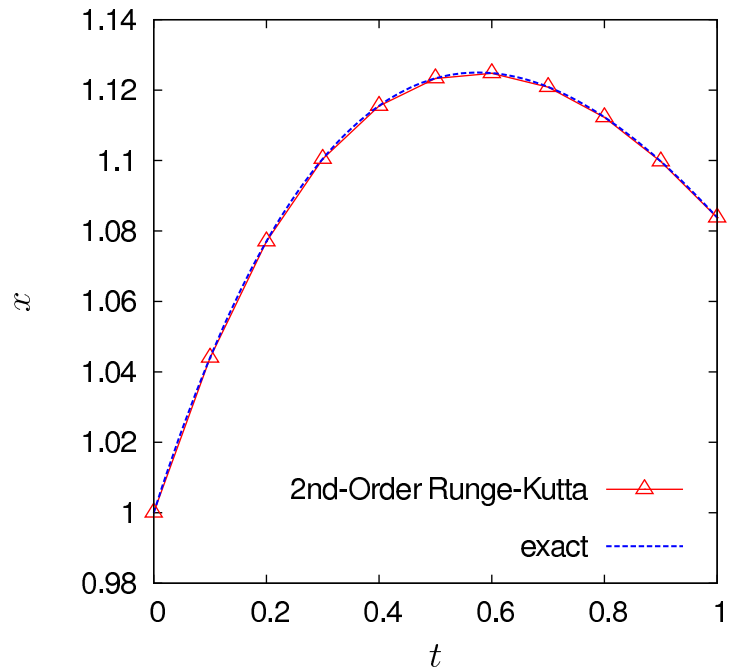
Example: what is the formula for a Runge-Kutta method of order 2 with $a_1 = \frac{1}{2}$?

Second-order Runge-Kutta method: application

Use a second-order Runge-Kutta method to solve the scalar ODE $x'(t) = -\frac{1}{2}x(t) + \exp(-t)$ for $t \in [0, 1]$, with initial condition $x(0) = 1$ and stepsize $h = 10^{-1}$

Exact solution: $\boxed{x(t) = 3\exp(-\frac{1}{2}t) - 2\exp(-t)}$

h	x_k	$\lvert x(t_k) - x_k \rvert$
0.0	1.0000	0
0.1	1.0440	2.16e-05
0.2	1.0770	3.72e-05
0.3	1.1004	4.77e-05
0.4	1.1155	5.39e-05
0.5	1.1233	5.66e-05
0.6	1.1248	5.63e-05
0.7	1.1208	5.35e-05
0.8	1.1123	4.88e-05
0.9	1.0997	4.25e-05
1.0	1.0838	3.49e-05



Fourth-order Runge-Kutta method: derivation

- **Exact same idea:** Express equivalence conditions with the 4th-order Taylor series method!

Classical 4th-order Runge-Kutta formula:

$$\boxed{x_{k+1} = x_k + \frac{1}{6}\left[K_1 + 2K_2 + 2K_3 + K_4\right] h}$$

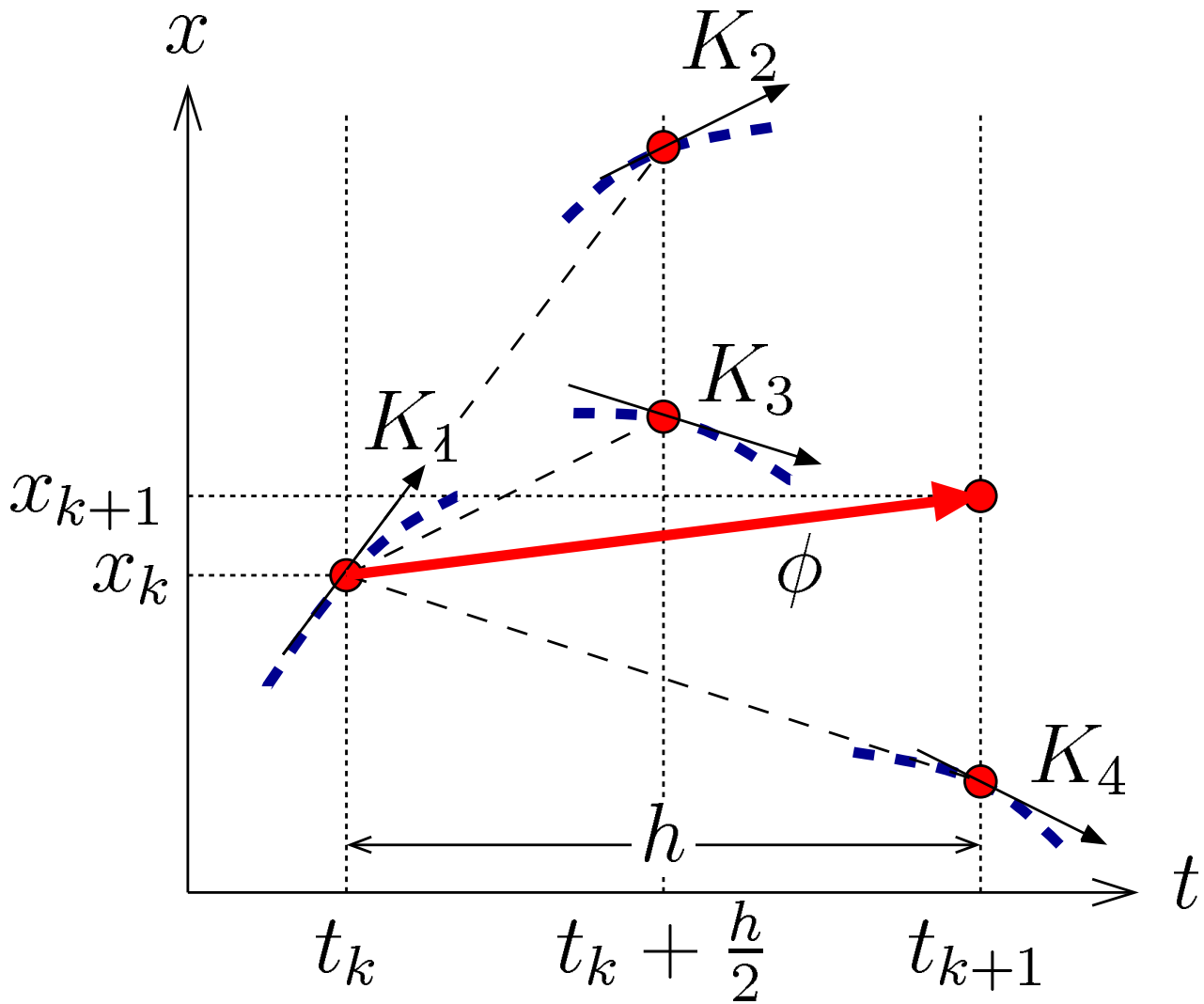
with:
$$K_1 = f\left(t_k, x_k\right)$$
$$K_2 = f\left(t_k + \frac{1}{2}h, x_k + \frac{1}{2}h\,K_1\right)$$
$$K_3 = f\left(t_k + \frac{1}{2}h, x_k + \frac{1}{2}h\,K_2\right)$$
$$K_4 = f\left(t_k + h, x_k + h\,K_3\right)$$

Butcher tableau:

$\frac{1}{2}$	$\frac{1}{2}$		
$\frac{1}{2}$	0	$\frac{1}{2}$	
1	0	0	1
$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

- Local error: $O(h^5)$: global error: $O(h^4)$

Classical Runge-Kutta method: graphical illustration

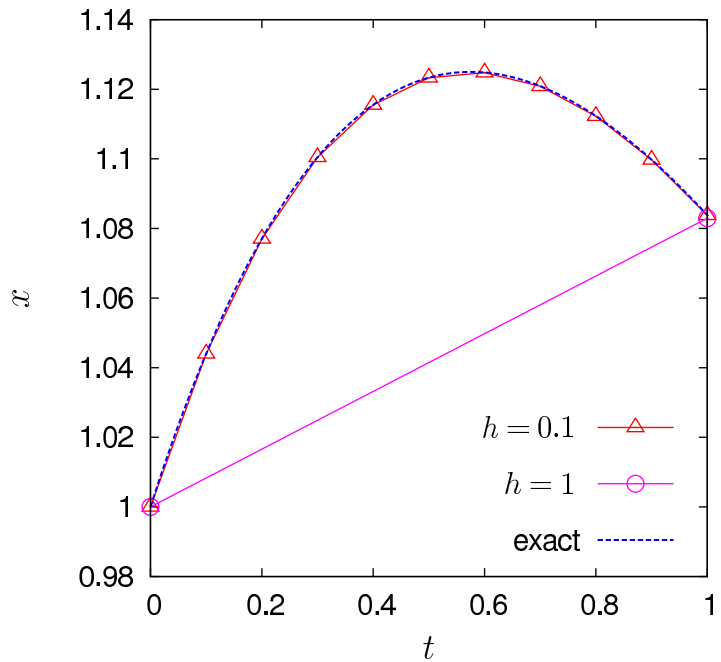


Classical Runge-Kutta method: application

Use the classical Runge-Kutta method to solve the scalar ODE $x'(t) = -\frac{1}{2}x(t) + \exp(-t)$ for $t \in [0, 1]$, with initial condition $x(0) = 1$ and stepsizes $h = 0.1$ and $h = 1$

Exact solution: $\boxed{x(t) = 3\exp(-\frac{1}{2}t) - 2\exp(-t)}$

h	x_k	e_k	x_k	e_k
0.0	1.0000	0	1.0000	0
0.1	1.0440	1.10e-08		
0.2	1.0770	2.01e-08		
0.3	1.1004	2.74e-08		
0.4	1.1155	3.33e-08		
0.5	1.1233	3.79e-08		
0.6	1.1248	4.14e-08		
0.7	1.1208	4.40e-08		
0.8	1.1123	4.57e-08		
0.9	1.0997	4.67e-08		
1.0	1.0838	4.71e-08	1.0829	8.89e-04

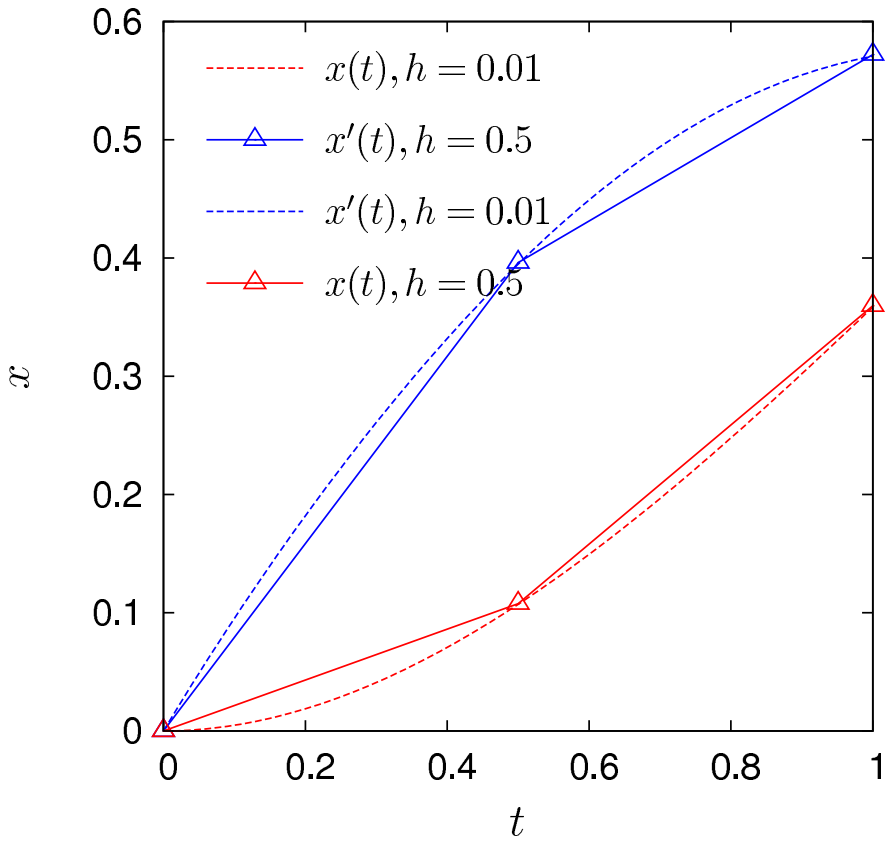


Fourth-Order Runge-Kutta method: vectorial ODEs

Use the classical Runge-Kutta method to compute the solution of the second-order ODE $x''(t) + x(t)[3x'(t) - 1] = \exp(-t)$ at $t = 1$, for the initial conditions $x(0) = 0$, $x'(0) = 0$, and using a single step $h = 1$.

t_k	$x_{1,k}$	$x_{2,k}$
0	0	0
0.5000	0.10765	0.39613
1.0000	0.35958	0.57207

- Exact same procedure as with scalar ODEs!
- Excellent accuracy, even for large steps



Heun's predictor-corrector method: basics

Predictor/corrector basics:

- **Predictor step:**

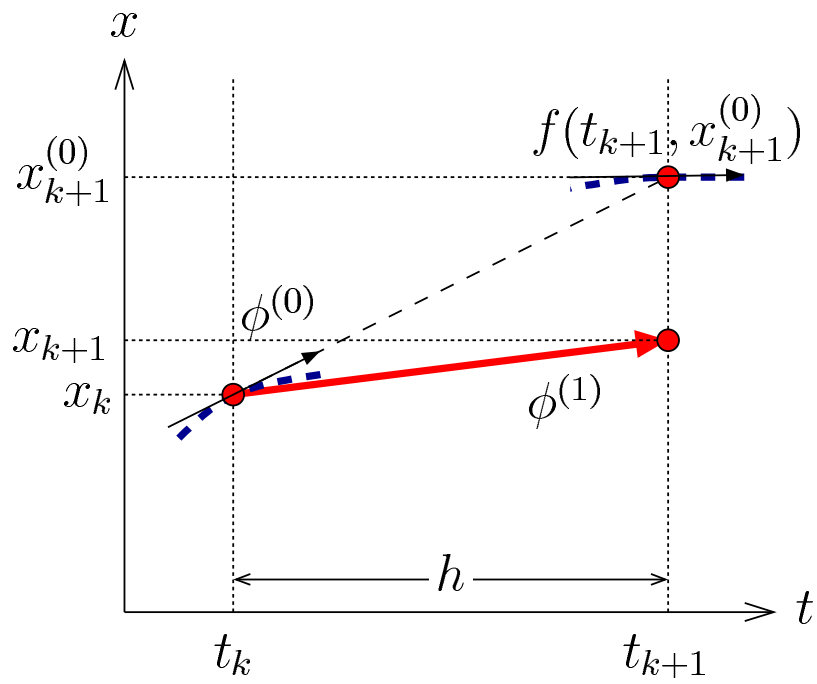
$$\phi^{(0)} = f(t_k, x_k)$$

$$x_{k+1}^{(0)} = x_k + h\,\phi^{(0)}$$

- **Corrector step:**

$$\phi^{(1)} = \frac{1}{2}\left[\phi^{(0)} + f(t_{k+1}, x_{k+1}^{(0)})\right]$$

$$x_{k+1} = x_k + h\,\phi^{(1)}$$



- Same formula as the 2nd-order Runge-Kutta method with parameters:

$$\boxed{a_1 = a_2 = \frac{1}{2}, \quad p_1 = q_{1,1} = 1} \qquad : \textbf{check!}$$

- **Error analysis:** local error is $O(h^3)$; global error is $O(h^2)$

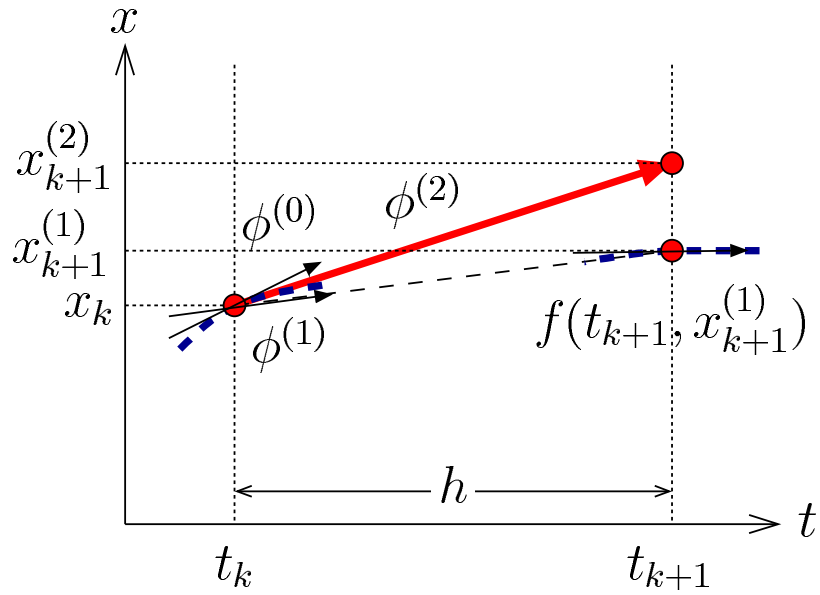
Heun's predictor-corrector method: iterative corrector

Iterative corrector step:

$$\phi^{(j+1)} = \frac{1}{2}\left[\phi^{(0)} + f\left(t_{k+1}, x_{k+1}^{(j)}\right)\right]$$

$$x_{k+1}^{(j+1)} = x_k + h\,\phi^{(j+1)}$$

- May not improve things up for a large step h
- Iterates may even diverge!



0. **Initialization:**
   Set $j \leftarrow 0$; Choose $\epsilon^{\text{tol}} > 0$
1. **Predictor step:**

$$\phi^{(0)} = f\left(t_k, x_k\right)$$

$$x_{k+1}^{(0)} = x_k + h\,\phi^{(0)}$$

2. **Corrector step:** **repeat:**

$$\phi^{(j+1)} = \frac{1}{2}\left[\phi^{(0)} + f\left(t_{k+1}, x_{k+1}^{(j)}\right)\right]$$

$$x_{k+1}^{(j+1)} = x_k + h\,\phi^{(j+1)}$$

UNTIL: $|x_{k+1}^{(j+1)} - x_{k+1}^{(j)}| < \epsilon^{\text{tol}}$

Heun's predictor-corrector method: application

Use Heun's *predictor-corrector* method with tolerance $\epsilon^{\text{tol}} = 0.1$ to compute the solution of the scalar ODE $x'(t) = -\frac{1}{2}x(t) + 4\exp(0.8t)$ at $t = 1$, for the initial conditions $x(0) = 2$ and using a single step $h = 1$.

Exact solution:

$$\boxed{x(t) = \frac{40}{13}\exp(0.8t) - \frac{14}{13}\exp(-0.5t)}$$

j	$\phi^{(j)}$	$x_1^{(j)}$	$\lvert x_1^{(j)} - x_1^{(j-1)}\rvert$	e_1 [%]
0	3.00000	5.00000		
1	4.70108	6.70108	1.701	8.18%
2	4.27581	6.27581	0.425	1.31%
3	4.38213	6.38213	0.106	3.03%
4	4.35555	6.35555	0.027	2.60%
		$x(1) = 6.19463$		

Numerical solution of ODEs: final words

Adaptive step-size control:

- Give the ability to vary the stepsize over the integration horizon
- Typically based on the local truncation error
  - E.g., adaptive Runge-Kutta methods

Stability:

When the effects of local errors do not accumulate catastrophically

- Unrelated to accuracy: A stable method can be very inaccurate!
- Determined by: 1) differential equations, 2) solution method, 3) stepsize h

Stiffness:

When some terms in the solution vary much more rapidly than others

- Special care needed in the numerical integration to guarantee stability
- Usual solution methods (Runge-Kutta, etc.) often impractical!
- Implicit solution techniques and multistep methods needed