Circuit simulation: transient analysis

M. B. Patil
www.ee.iitb.ac.in/~sequel

Department of Electrical Engineering
Indian Institute of Technology Bombay

Outline

* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Methods for transient analysis

Consider the system of ODE's given by,

$$\begin{aligned}
\frac{dx_1}{dt} &= f_1(t, x_1, x_2, \ldots, x_N), \\
\frac{dx_2}{dt} &= f_1(t, x_1, x_2, \ldots, x_N), \\
&\vdots \\
\frac{dx_N}{dt} &= f_1(t, x_1, x_2, \ldots, x_N),
\end{aligned}$$

with the initial values at $t = t_0$ specified as $x_1(t_0) = x_1^0$, $x_2(t_0) = x_2^0$, etc.

Methods for transient analysis

Consider the system of ODE's given by,

$$\begin{aligned}
\frac{dx_1}{dt} &= f_1(t, x_1, x_2, \ldots, x_N), \\
\frac{dx_2}{dt} &= f_1(t, x_1, x_2, \ldots, x_N), \\
&\vdots \\
\frac{dx_N}{dt} &= f_1(t, x_1, x_2, \ldots, x_N),
\end{aligned}$$

with the initial values at $t = t_0$ specified as $x_1(t_0) = x_1^0$, $x_2(t_0) = x_2^0$, etc.

The equations can be written in a concise vector form:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0 .$$

Methods for transient analysis

Consider the system of ODE's given by,

$$\begin{aligned}
\frac{dx_1}{dt} &= f_1(t, x_1, x_2, \ldots, x_N), \\
\frac{dx_2}{dt} &= f_1(t, x_1, x_2, \ldots, x_N), \\
&\vdots \\
\frac{dx_N}{dt} &= f_1(t, x_1, x_2, \ldots, x_N),
\end{aligned}$$

with the initial values at $t = t_0$ specified as $x_1(t_0) = x_1^0$, $x_2(t_0) = x_2^0$, etc.

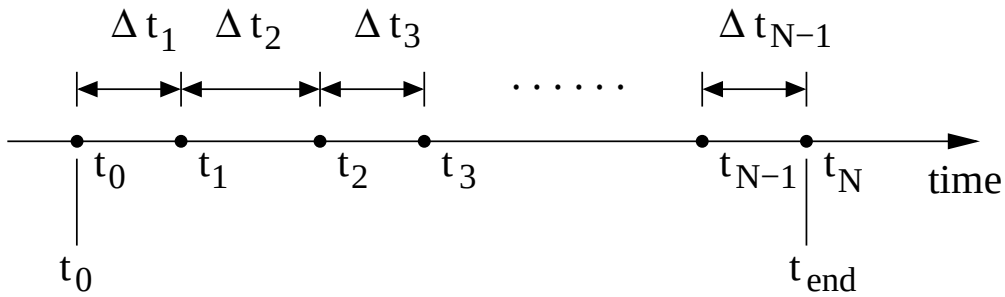
The equations can be written in a concise vector form:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0 .$$

We will consider the special case of a *single* ODE:

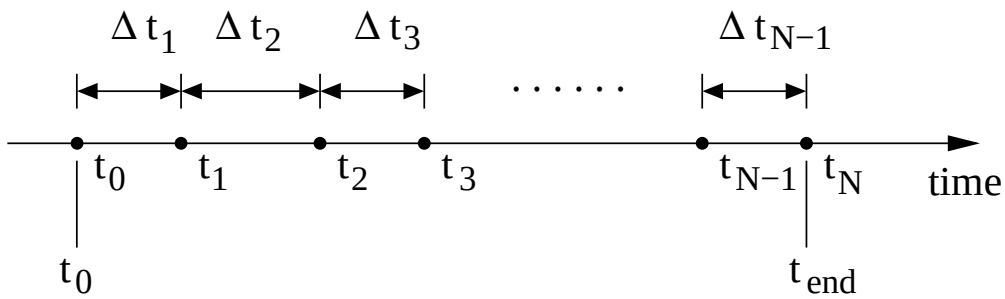
$$\frac{dx}{dt} = f(t, x), \quad x(t_0) = x_0 .$$

Discretization of time



* Denote the exact solution of $\dot{x} = f(t, x)$, $x(t_0) = x_0$ on $[t_0, t_{\text{end}}]$, by $x(t)$, and the numerical solution by the sequence $\{x_n\}$, where x_n is the numerical solution computed for $t = t_n$.

Discretization of time



* Denote the exact solution of $\dot{x} = f(t, x)$, $x(t_0) = x_0$ on $[t_0, t_{\text{end}}]$, by $x(t)$, and the numerical solution by the sequence $\{x_n\}$, where x_n is the numerical solution computed for $t = t_n$.
* The primary objective of a numerical method is to obtain $\{x_n\}$ such that $|x(t_n) - x_n|$ is "small" for all n.

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$, is said to be well-posed [1] if

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$, is said to be well-posed [1] if

(a) a unique solution $x(t)$ exists, and

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$, is said to be well-posed [1] if

(a) a unique solution $x(t)$ exists, and

(b) For any $\epsilon > 0$, and

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$, is said to be well-posed [1] if

(a) a unique solution $x(t)$ exists, and

(b) For any $\epsilon > 0$, and

  (i) some ϵ_0 s.t. $|\epsilon_0| < \epsilon$, and

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$,
is said to be well-posed [1] if

(a) a unique solution $x(t)$ exists, and

(b) For any $\epsilon > 0$, and

  (i) some ϵ_0 s.t. $|\epsilon_0| < \epsilon$, and

  (ii) a function $\delta(t)$ which is continuous on $[a, b]$, with $|\delta(t)| < \epsilon$ on $[a, b]$,

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$,
is said to be well-posed [1] if

(a) a unique solution $x(t)$ exists, and

(b) For any $\epsilon > 0$, and

(i) some ϵ_0 s.t. $|\epsilon_0| < \epsilon$, and

(ii) a function $\delta(t)$ which is continuous on $[a, b]$, with $|\delta(t)| < \epsilon$ on $[a, b]$,

there exists a positive constant k such that the perturbed problem,

$$\dot{z} = f(t, z) + \delta(t), \quad a \leq t \leq b, \quad z(a) = \alpha + \epsilon_0 \,,$$

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$,
is said to be well-posed [1] if

(a) a unique solution $x(t)$ exists, and

(b) For any $\epsilon > 0$, and

(i) some ϵ_0 s.t. $|\epsilon_0| < \epsilon$, and

(ii) a function $\delta(t)$ which is continuous on $[a, b]$, with $|\delta(t)| < \epsilon$ on $[a, b]$,

there exists a positive constant k such that the perturbed problem,

$$\dot{z} = f(t, z) + \delta(t), \quad a \leq t \leq b, \quad z(a) = \alpha + \epsilon_0 \,,$$

has a unique solution, with

$$|z(t) - x(t)| < k\,\epsilon \;\text{ for }\; a \leq t \leq b \,.$$

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$,
is said to be well-posed [1] if

(a) a unique solution $x(t)$ exists, and

(b) For any $\epsilon > 0$, and

(i) some ϵ_0 s.t. $|\epsilon_0| < \epsilon$, and

(ii) a function $\delta(t)$ which is continuous on $[a, b]$, with $|\delta(t)| < \epsilon$ on $[a, b]$,

there exists a positive constant k such that the perturbed problem,

$$\dot{z} = f(t, z) + \delta(t), \quad a \leq t \leq b, \quad z(a) = \alpha + \epsilon_0 \,,$$

has a unique solution, with

$$|z(t) - x(t)| < k\,\epsilon \;\text{ for }\; a \leq t \leq b \,.$$

* In other words, if the original problem is perturbed, the solution is perturbed in a bounded manner.

What is a "well-posed" problem?

The initial value problem $\dot{x} = f(t, x)$, $a \leq t \leq b$, $x(a) = \alpha$,
is said to be well-posed [1] if

(a) a unique solution $x(t)$ exists, and

(b) For any $\epsilon > 0$, and

(i) some ϵ_0 s.t. $|\epsilon_0| < \epsilon$, and

(ii) a function $\delta(t)$ which is continuous on $[a, b]$, with $|\delta(t)| < \epsilon$ on $[a, b]$,

there exists a positive constant k such that the perturbed problem,

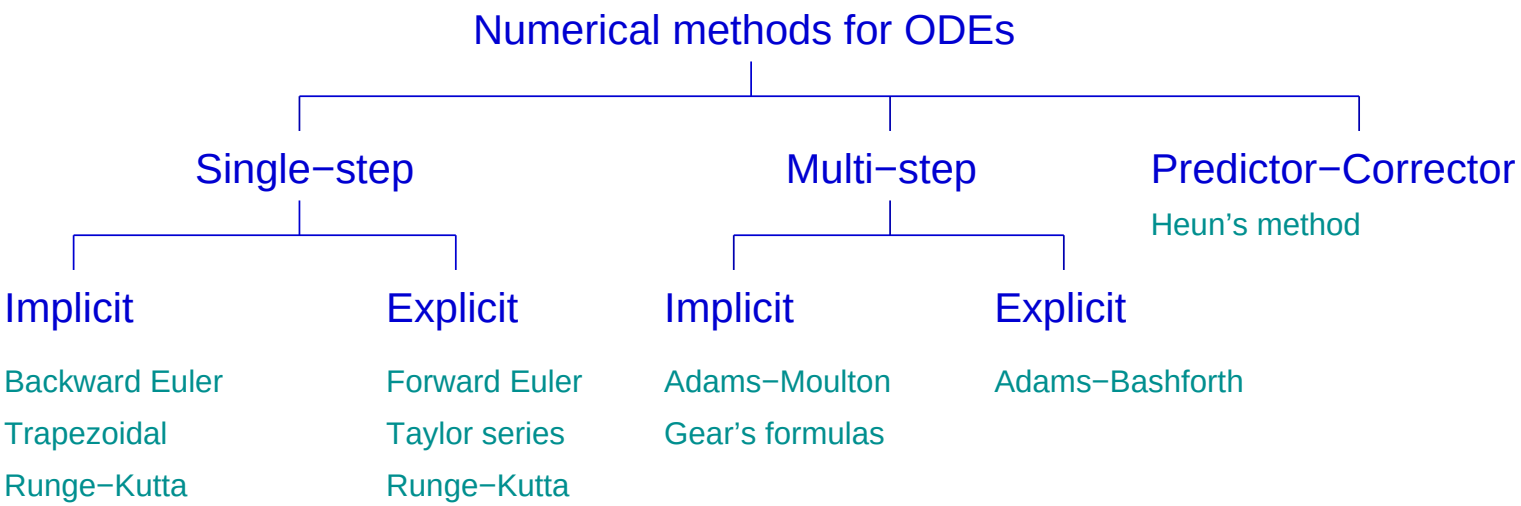
$$\dot{z} = f(t, z) + \delta(t), \quad a \leq t \leq b, \quad z(a) = \alpha + \epsilon_0 \,,$$

has a unique solution, with

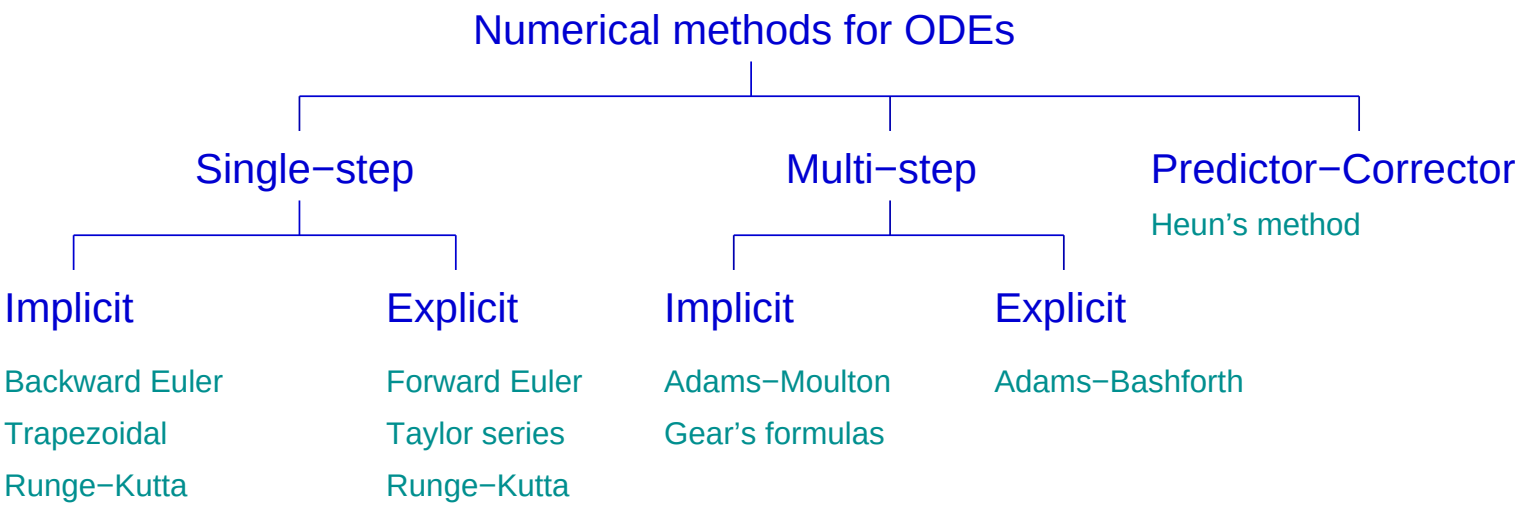
$$|z(t) - x(t)| < k\,\epsilon \;\text{ for }\; a \leq t \leq b \,.$$

* In other words, if the original problem is perturbed, the solution is perturbed in a bounded manner.
* Numerical methods are expected to work well only for well-posed problems because the problem being solved by these methods is generally a perturbed version of the original problem (due to round-off errors, for example).

Broad classification of methods

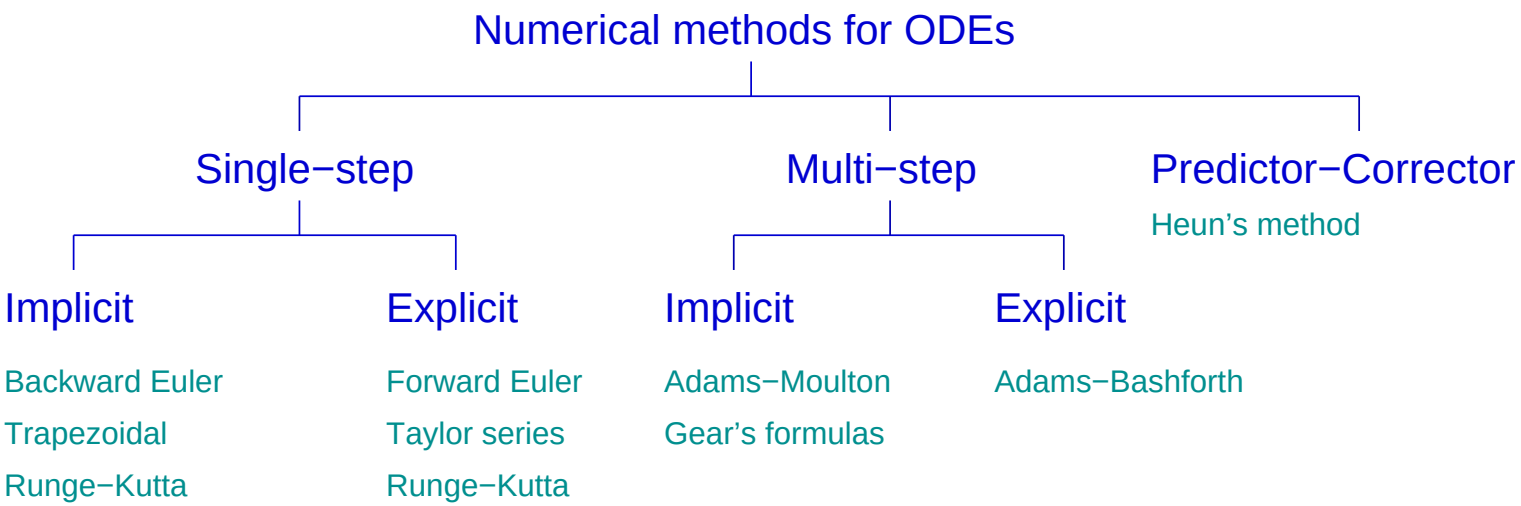


Broad classification of methods



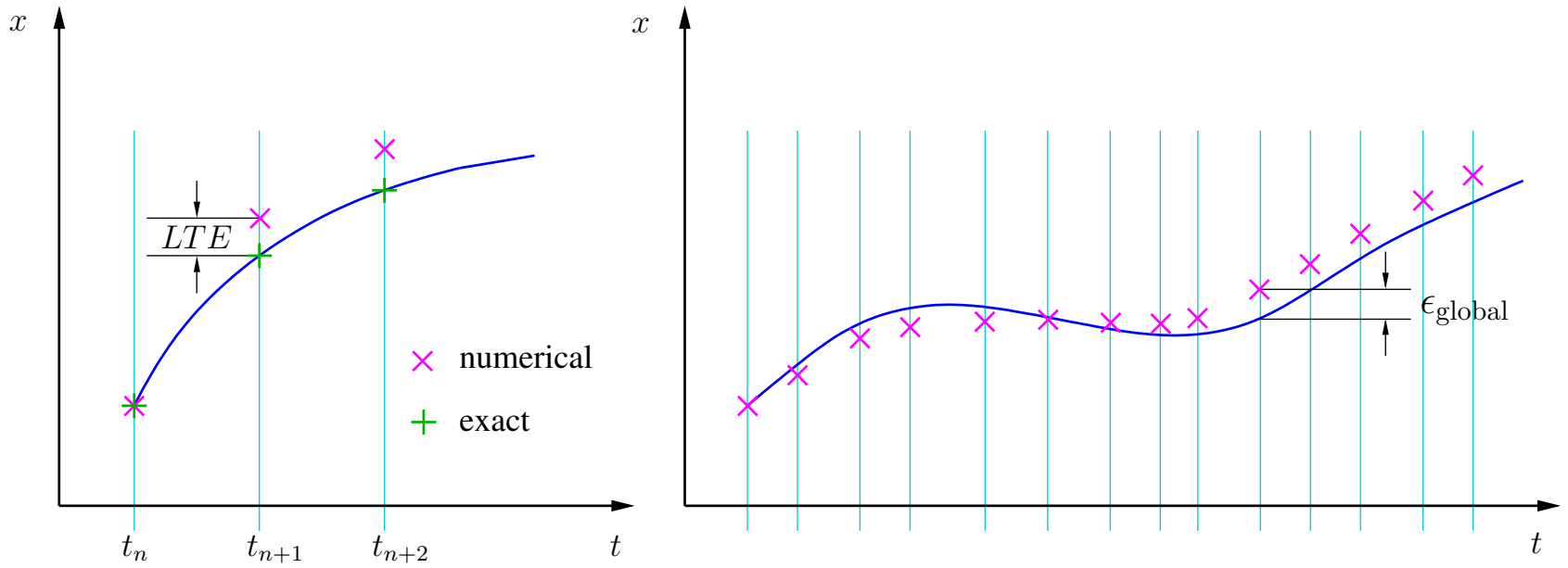
* Of these methods, only a small subset is useful for circuit simulation.

Broad classification of methods



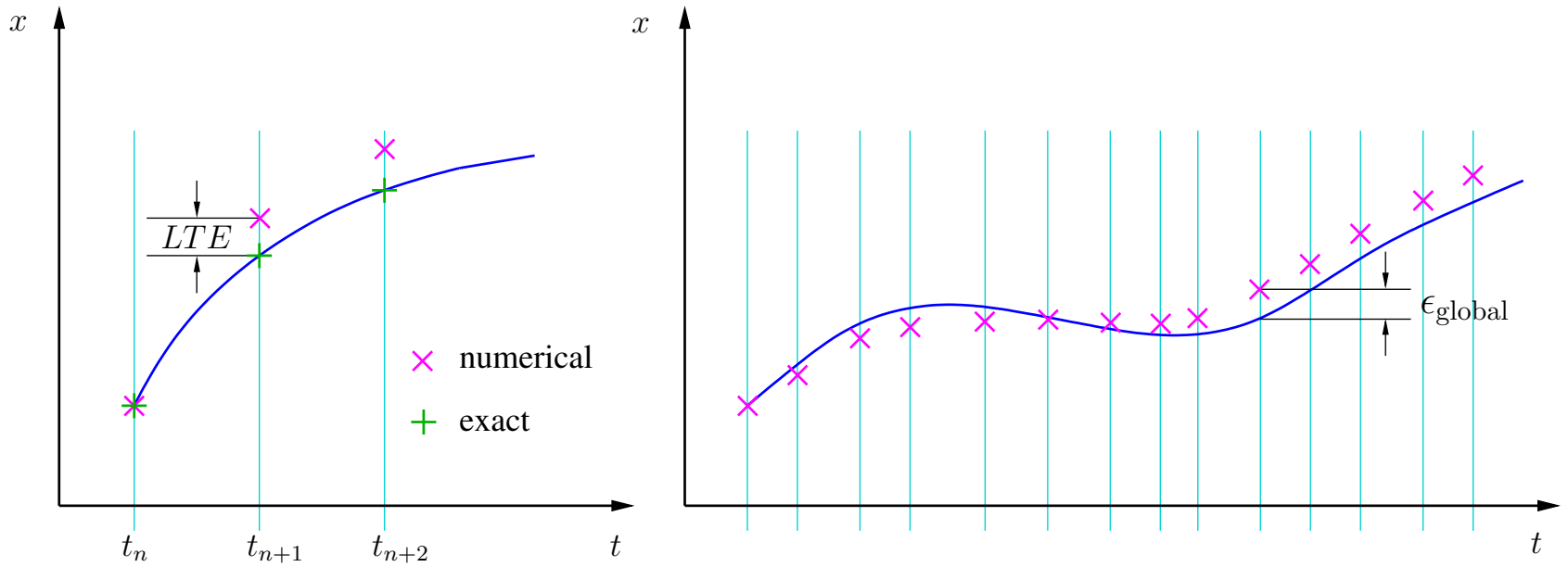
* Of these methods, only a small subset is useful for circuit simulation.
* Other classifications are possible, based on stability and order.

Local Truncation Error (LTE) and Global Error



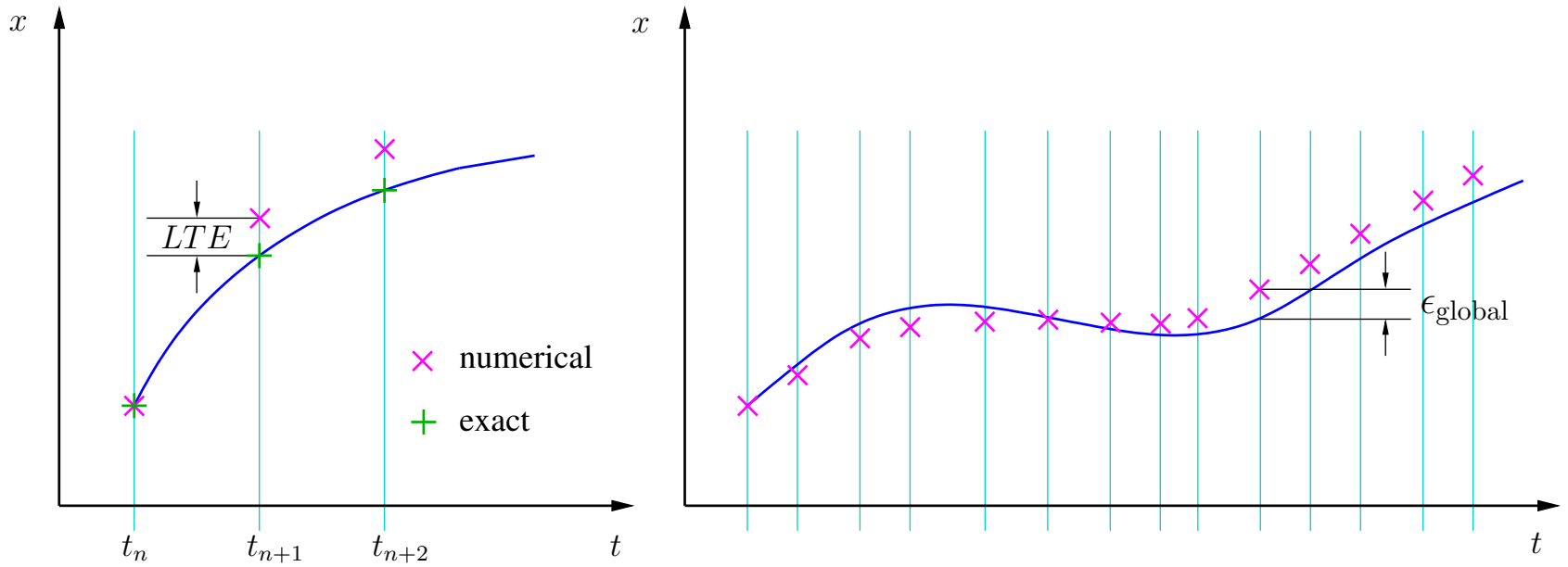
* The local truncation error is due to the approximations made in the algorithm. It is *local* since the starting point is assumed to be exact.

Local Truncation Error (LTE) and Global Error



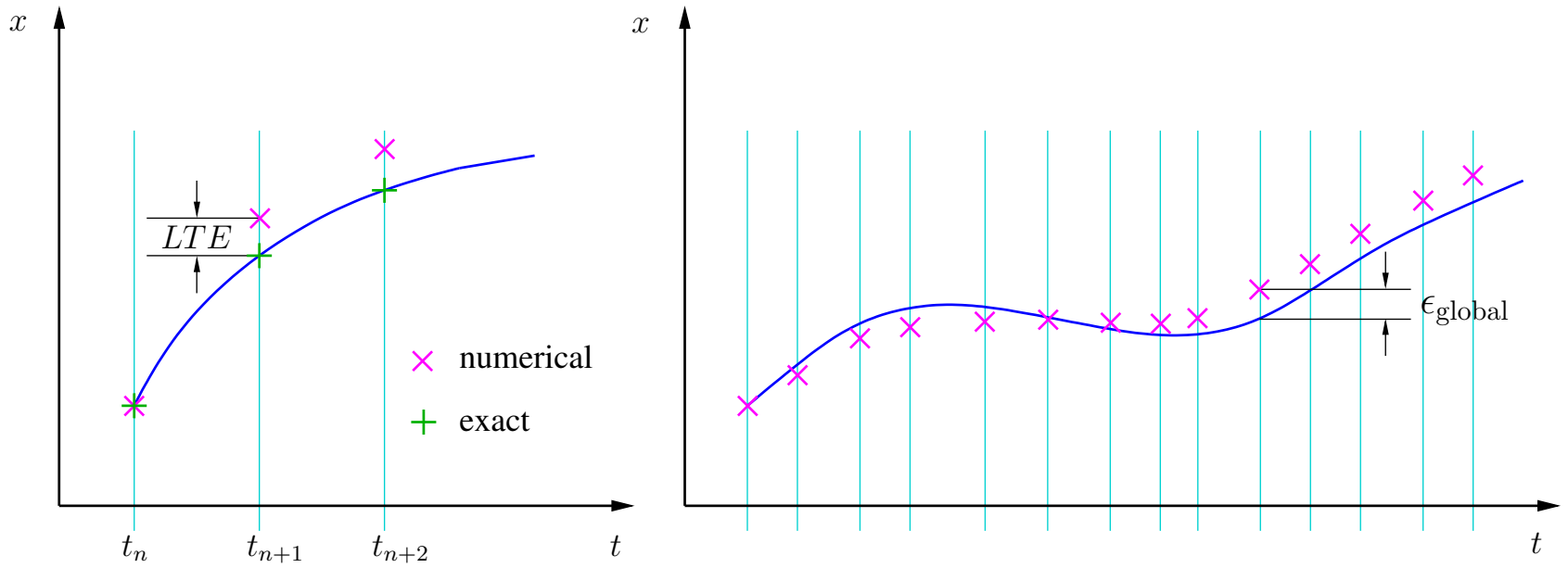
* The local truncation error is due to the approximations made in the algorithm. It is *local* since the starting point is assumed to be exact.
* The global error is due to all previous local errors, but it is *not* a simple accumulation of the local errors.

Local Truncation Error (LTE) and Global Error



* The local truncation error is due to the approximations made in the algorithm. It is *local* since the starting point is assumed to be exact.
* The global error is due to all previous local errors, but it is *not* a simple accumulation of the local errors.
* Other sources of error: (a) round-off error due to finite precision (b) In case of implicit methods, the equations are not solved exactly but to a certain tolerance.

Local Truncation Error (LTE) and Global Error



* The local truncation error is due to the approximations made in the algorithm. It is *local* since the starting point is assumed to be exact.
* The global error is due to all previous local errors, but it is *not* a simple accumulation of the local errors.
* Other sources of error: (a) round-off error due to finite precision (b) In case of implicit methods, the equations are not solved exactly but to a certain tolerance.
* If the LTE is $O(h^{k+1})$, the method is said to be of order k.

Numerical methods for solving ODEs: issues of interest

* Is it one-step or multi-step?

Numerical methods for solving ODEs: issues of interest

* Is it one-step or multi-step?
* How is it derived?

Numerical methods for solving ODEs: issues of interest

* Is it one-step or multi-step?
* How is it derived?
* What is its order (accuracy)?

Numerical methods for solving ODEs: issues of interest

* Is it one-step or multi-step?
* How is it derived?
* What is its order (accuracy)?
* What are its stability properties? (Will the method allow relatively large time steps?)

Numerical methods for solving ODEs: issues of interest

* Is it one-step or multi-step?
* How is it derived?
* What is its order (accuracy)?
* What are its stability properties? (Will the method allow relatively large time steps?)
* How is it implemented (for a single ODE and for a system of ODEs)?

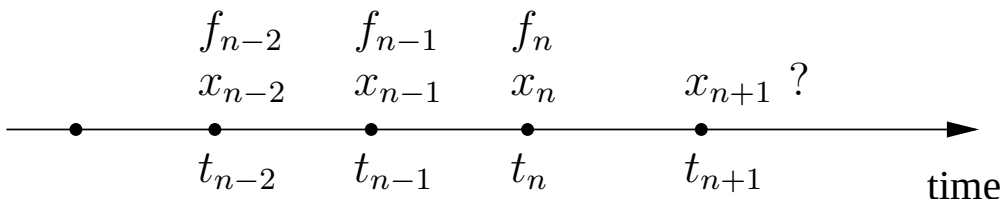
Numerical methods for solving ODEs: issues of interest

* Is it one-step or multi-step?
* How is it derived?
* What is its order (accuracy)?
* What are its stability properties? (Will the method allow relatively large time steps?)
* How is it implemented (for a single ODE and for a system of ODEs)?
* What is the computational effort per time step?

Numerical methods for solving ODEs: issues of interest

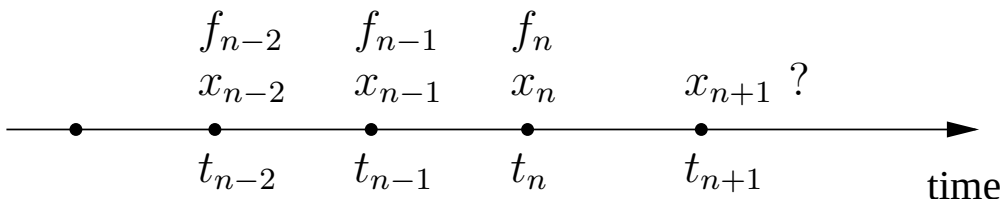
* Is it one-step or multi-step?
* How is it derived?
* What is its order (accuracy)?
* What are its stability properties? (Will the method allow relatively large time steps?)
* How is it implemented (for a single ODE and for a system of ODEs)?
* What is the computational effort per time step?
* What is the memory requirement?

Problem description



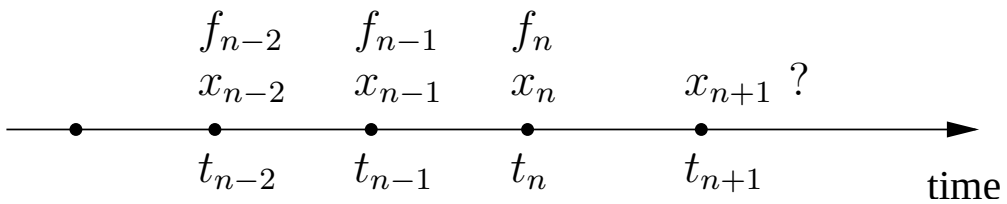
* The ODE to be solved is $\dot{x} = f(t, x)$, with $x(0) = x_0$. The numerical solution up to t_n is available, and that for t_{n+1} is to be computed.

Problem description



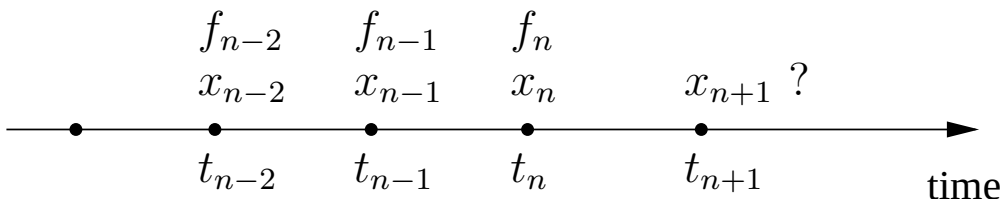
* The ODE to be solved is $\dot{x} = f(t, x)$, with $x(0) = x_0$. The numerical solution up to t_n is available, and that for t_{n+1} is to be computed.
* The past function values f_n, f_{n-1}, .. are also available.

Problem description



* The ODE to be solved is $\dot{x} = f(t, x)$, with $x(0) = x_0$. The numerical solution up to t_n is available, and that for t_{n+1} is to be computed.
* The past function values f_n, f_{n-1}, .. are also available.
* Single-step methods: Only the information at t_n is used.

Problem description



* The ODE to be solved is $\dot{x} = f(t, x)$, with $x(0) = x_0$. The numerical solution up to t_n is available, and that for t_{n+1} is to be computed.
* The past function values f_n, f_{n-1}, .. are also available.
* Single-step methods: Only the information at t_n is used.
* Multi-step methods: The information at t_n and some others (t_{n-1}, t_{n-2}, ..) is also used.

Outline

* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Taylor's theorem

If a function $x(t)$ and its first $(n+1)$ derivatives are continuous on an interval containing t_n and t_{n+1} $(= t_n + h)$, then the value of the function at t_{n+1} is given by,

Taylor's theorem

If a function $x(t)$ and its first $(n+1)$ derivatives are continuous on an interval containing t_n and t_{n+1} $(= t_n + h)$, then the value of the function at t_{n+1} is given by,

$$x(t_{n+1}) = x(t_n) + x'(t_n)\,h + \frac{x''(t_n)}{2\,!}\,h^2 + \cdots + \frac{x^{(k)}(t_n)}{k\,!}\,h^k + \frac{x^{(k+1)}(\xi)}{(k+1)\,!}\,h^{k+1} \quad (1)$$

for some ξ between t_n and t_{n+1}.

Taylor's theorem

If a function $x(t)$ and its first $(n+1)$ derivatives are continuous on an interval containing t_n and t_{n+1} $(= t_n + h)$, then the value of the function at t_{n+1} is given by,

$$x(t_{n+1}) = x(t_n) + x'(t_n)\,h + \frac{x''(t_n)}{2!}\,h^2 + \cdots + \frac{x^{(k)}(t_n)}{k!}\,h^k + \frac{x^{(k+1)}(\xi)}{(k+1)!}\,h^{k+1} \quad (1)$$

for some ξ between t_n and t_{n+1}.

* In other words, for the conditions specified on $x(t)$, it is possible to find ξ, $t_n < \xi < t_{n+1}$, such that Eq. 1 is satisfied *exactly*.

Taylor's theorem

If a function $x(t)$ and its first $(n+1)$ derivatives are continuous on an interval containing t_n and t_{n+1} $(= t_n + h)$, then the value of the function at t_{n+1} is given by,

$$x(t_{n+1}) = x(t_n) + x'(t_n)\,h + \frac{x''(t_n)}{2\,!}\,h^2 + \cdots + \frac{x^{(k)}(t_n)}{k\,!}\,h^k + \frac{x^{(k+1)}(\xi)}{(k+1)\,!}\,h^{k+1} \quad (1)$$

for some ξ between t_n and t_{n+1}.

* In other words, for the conditions specified on $x(t)$, it is possible to find ξ, $t_n < \xi < t_{n+1}$, such that Eq. 1 is satisfied *exactly*.
* As $h \to 0$, $\xi \to t_n$, and defining $C = x^{(k+1)}(t_n)/(k+1)\,!$, the last term in Eq. 1 approaches Ch^{k+1}.

Taylor's theorem

If a function $x(t)$ and its first $(n+1)$ derivatives are continuous on an interval containing t_n and t_{n+1} $(= t_n + h)$, then the value of the function at t_{n+1} is given by,

$$x(t_{n+1}) = x(t_n) + x'(t_n)\,h + \frac{x''(t_n)}{2\,!}\,h^2 + \cdots + \frac{x^{(k)}(t_n)}{k\,!}\,h^k + \frac{x^{(k+1)}(\xi)}{(k+1)\,!}\,h^{k+1} \quad (1)$$

for some ξ between t_n and t_{n+1}.

* In other words, for the conditions specified on $x(t)$, it is possible to find ξ, $t_n < \xi < t_{n+1}$, such that Eq. 1 is satisfied *exactly*.
* As $h \to 0$, $\xi \to t_n$, and defining $C = x^{(k+1)}(t_n)/(k+1)\,!$, the last term in Eq. 1 approaches Ch^{k+1}.
* We can rewrite Taylor's theorem as,

$$x(t_{n+1}) = x(t_n) + x'(t_n)\,h + \frac{x''(t_n)}{2\,!}\,h^2 + \cdots + \frac{x^{(k)}(t_n)}{k\,!}\,h^k + O(h^{k+1})\,. \quad (2)$$

Taylor series methods for solving $\dot{x} = f(t, x)$

Taylor's theorem:

$$
\begin{aligned}
x(t_{n+1}) &= x(t_n) + x'(t_n)\, h + O(h^2)\,, \\
x(t_{n+1}) &= x(t_n) + x'(t_n)\, h + \frac{x''(t_n)}{2\,!}\, h^2 + O(h^3)\,, \text{etc.}
\end{aligned}
$$

Taylor series methods for solving $\dot{x} = f(t, x)$

Taylor's theorem:

$$x(t_{n+1}) = x(t_n) + x'(t_n)\, h + O(h^2)\,,$$

$$x(t_{n+1}) = x(t_n) + x'(t_n)\, h + \frac{x''(t_n)}{2\,!}\, h^2 + O(h^3)\,, \text{etc.}$$

We want to apply this to $\dfrac{dx}{dt} = f(t, x)\,.$

Taylor series methods for solving $\dot{x} = f(t, x)$

Taylor's theorem:

$$x(t_{n+1}) = x(t_n) + x'(t_n)\, h + O(h^2)\,,$$

$$x(t_{n+1}) = x(t_n) + x'(t_n)\, h + \frac{x''(t_n)}{2\,!}\, h^2 + O(h^3)\,, \text{etc.}$$

We want to apply this to $\dfrac{dx}{dt} = f(t, x)\,.$

In the Taylor series method of order k, the first k derivative terms are retained.

Taylor series methods for solving $\dot{x} = f(t, x)$

Taylor's theorem:

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + x'(t_n)\, h + O(h^2)\,, \\ x(t_{n+1}) &= x(t_n) + x'(t_n)\, h + \frac{x''(t_n)}{2\,!}\, h^2 + O(h^3)\,, \text{etc.} \end{aligned}$$

We want to apply this to $\dfrac{dx}{dt} = f(t, x)$.

In the Taylor series method of order k, the first k derivative terms are retained.

$$\begin{aligned} x_{n+1} &= x_n + h f^n\,, \\ x_{n+1} &= x_n + h f^n + \frac{h^2}{2}\left(f_t^n + f^n f_x^n\right)\,, \end{aligned}$$

where $f^n = f(t_n, x_n)$, $f_t^n = \dfrac{\partial f}{\partial t}(t_n, x_n)$, and $f_x^n = \dfrac{\partial f}{\partial x}(t_n, x_n)$.

Taylor series method for solving $\dot{x} = f(t, x)$

The derivatives can be computed as,

$$x'(t_n) \quad = \quad f(t_n, x(t_n)) \,,$$

Taylor series method for solving $\dot{x} = f(t, x)$

The derivatives can be computed as,

$$\begin{aligned} x'(t_n) &= f(t_n, x(t_n)) \,, \\ x''(t_n) &= \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} \frac{dx}{dt} = f_t + f\, f_x \,, \end{aligned}$$

Taylor series method for solving $\dot{x} = f(t, x)$

The derivatives can be computed as,

$$
\begin{aligned}
x'(t_n) &= f(t_n, x(t_n)) \,, \\
x''(t_n) &= \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} \frac{dx}{dt} = f_t + f\, f_x \,, \\
x^{(3)}(t_n) &= \frac{\partial}{\partial t} \left[f_t + f\, f_x \right] + \frac{\partial}{\partial x} \left[f_t + f\, f_x \right] f \\
&= \left[f_{tt} + f\, f_{xt} + f_t\, f_x \right] + \left[f_{tx} + f_x\, f_x + f\, f_{xx} \right] f \\
&= f_{tt} + 2\, f\, f_{xt} + f_t\, f_x + f\, f_x^2 + f^2\, f_{xx} \,,
\end{aligned}
$$

Taylor series method for solving $\dot{x} = f(t, x)$

The derivatives can be computed as,

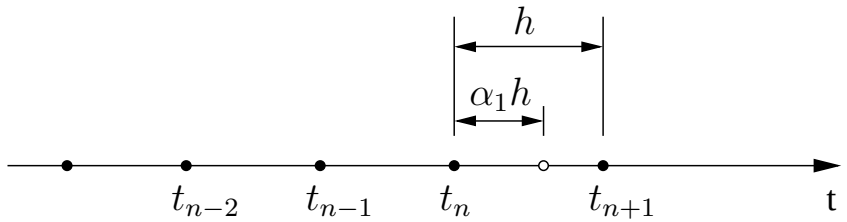
$$
\begin{aligned}
x'(t_n) &= f(t_n, x(t_n)) \,, \\
x''(t_n) &= \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} \frac{dx}{dt} = f_t + f\, f_x \,, \\
x^{(3)}(t_n) &= \frac{\partial}{\partial t} \left[f_t + f\, f_x \right] + \frac{\partial}{\partial x} \left[f_t + f\, f_x \right] f \\
&= \left[f_{tt} + f\, f_{xt} + f_t\, f_x \right] + \left[f_{tx} + f_x\, f_x + f\, f_{xx} \right] f \\
&= f_{tt} + 2\, f\, f_{xt} + f_t\, f_x + f\, f_x^2 + f^2\, f_{xx} \,,
\end{aligned}
$$

Note that computation of the derivatives becomes expensive as the order increases $\rightarrow$ Runge-Kutta methods.

Outline

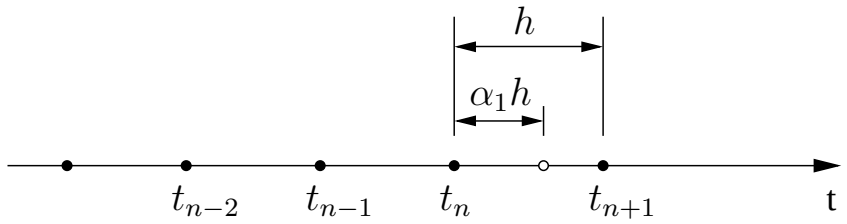
* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Runge-Kutta method for solving $\dot{x} = f(t, x)$



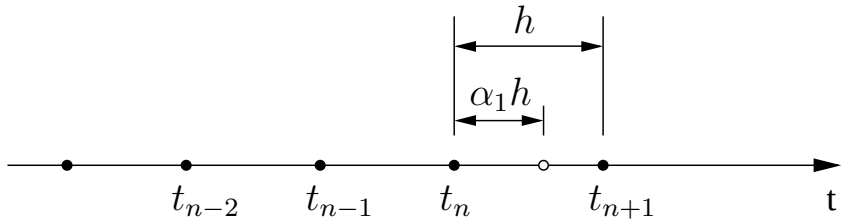
* Basic idea: Instead of evaluating higher-order derivatives (as required in Taylor series method), evaluate the function $f(t, x)$ at some intermediate points such that the resulting formula is equivalent to a Taylor series formula.

Runge-Kutta method for solving $\dot{x} = f(t, x)$



* Basic idea: Instead of evaluating higher-order derivatives (as required in Taylor series method), evaluate the function $f(t, x)$ at some intermediate points such that the resulting formula is equivalent to a Taylor series formula.
* Note that this is still a single-step method since we are using information only at t_n (and not t_{n-1}, t_{n-2}, etc.).

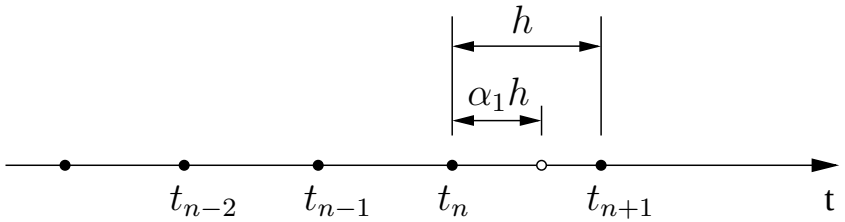
Runge-Kutta method for solving $\dot{x} = f(t, x)$



Consider the algorithm given by,

$$
\begin{aligned}
f_0 &= f(t_n, x_n)\,, \\
f_1 &= f(t_n + \alpha_1\, h, x_n + h\,\beta_{1,0}\, f_0)\,, \quad (\alpha_1 < 1)\,, \\
x_{n+1} &= x_n + h[\gamma_0\, f_0 + \gamma_1\, f_1]\,.
\end{aligned}
$$

Runge-Kutta method for solving $\dot{x} = f(t, x)$

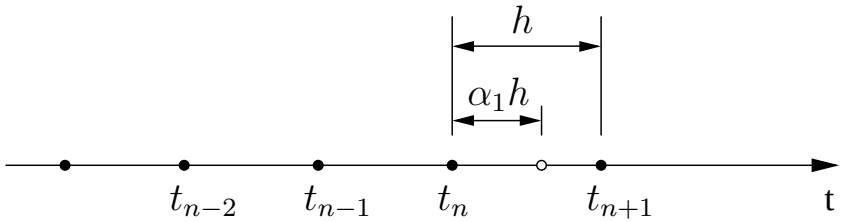


Consider the algorithm given by,

$$\begin{aligned} f_0 &= f(t_n, x_n)\,, \\ f_1 &= f(t_n + \alpha_1\, h, x_n + h\,\beta_{1,0}\, f_0)\,, \quad (\alpha_1 < 1)\,, \\ x_{n+1} &= x_n + h[\gamma_0\, f_0 + \gamma_1\, f_1]\,. \end{aligned}$$

* Only function evaluations are involved (and not derivative computation).

Runge-Kutta method for solving $\dot{x} = f(t, x)$

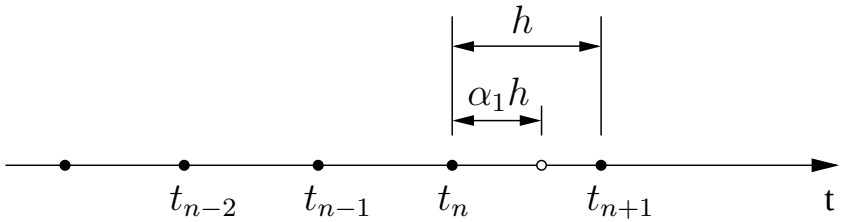


Consider the algorithm given by,

$$\begin{aligned} f_0 &= f(t_n, x_n)\,, \\ f_1 &= f(t_n + \alpha_1\, h, x_n + h\,\beta_{1,0}\, f_0)\,, \quad (\alpha_1 < 1)\,, \\ x_{n+1} &= x_n + h[\gamma_0\, f_0 + \gamma_1\, f_1]\,. \end{aligned}$$

* Only function evaluations are involved (and not derivative computation).
* The algorithm looks very different than the Taylor series method, but let us take a closer look.

Runge-Kutta method for solving $\dot{x} = f(t, x)$



Consider the algorithm given by,

$$\begin{aligned} f_0 &= f(t_n, x_n)\,, \\ f_1 &= f(t_n + \alpha_1\, h, x_n + h\,\beta_{1,0}\, f_0)\,, \quad (\alpha_1 < 1)\,, \\ x_{n+1} &= x_n + h[\gamma_0\, f_0 + \gamma_1\, f_1]\,. \end{aligned}$$

* Only function evaluations are involved (and not derivative computation).
* The algorithm looks very different than the Taylor series method, but let us take a closer look.
* The reason for using subscripts for α and β will become clear later.

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* Taylor series for a function of two variables:

$$f(t\,, x) = f(t_n, x_n) + f_t(t_n, x_n)(t - t_n) + f_x(t_n, x_n)(x - x_n)$$
$$+ \frac{1}{2\,!}\left[f_{tt}(t_n, x_n)(t - t_n)^2 + f_{tx}(t_n, x_n)(t - t_n)(x - x_n) + f_{xx}(t_n, x_n)(x - x_n)^2\right]$$
$$+ \text{Higher-order terms}\,.$$

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* Taylor series for a function of two variables:

$$f(t\,, x) = f(t_n, x_n) + f_t(t_n, x_n)(t - t_n) + f_x(t_n, x_n)(x - x_n)$$
$$+ \frac{1}{2\,!}\left[f_{tt}(t_n, x_n)(t - t_n)^2 + f_{tx}(t_n, x_n)(t - t_n)(x - x_n) + f_{xx}(t_n, x_n)(x - x_n)^2\right]$$
$$+ \text{Higher-order terms}\,.$$

* Substituting $t = t_n + \alpha_1\, h$ and $x = x_n + h\,\beta_{1,0}\, f_0$, we get

$$\begin{aligned} x_{n+1} &= x_n + h[\gamma_0\, f_0 + \gamma_1\, f(t_n + \alpha_1\, h, x_n + h\,\beta_{1,0}\, f_0)] \\ &= x_n + \gamma_0 hf + \gamma_1 hf + \alpha_1\gamma_1 h^2 f_t + \beta_{1,0}\gamma_1 h^2 f f_x + O(h^3)\,. \end{aligned}$$

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* Taylor series for a function of two variables:

$$f(t\,, x) = f(t_n, x_n) + f_t(t_n, x_n)(t - t_n) + f_x(t_n, x_n)(x - x_n)$$
$$+ \frac{1}{2\,!}\left[f_{tt}(t_n, x_n)(t - t_n)^2 + f_{tx}(t_n, x_n)(t - t_n)(x - x_n) + f_{xx}(t_n, x_n)(x - x_n)^2\right]$$
$$+ \text{Higher-order terms}\,.$$

* Substituting $t = t_n + \alpha_1\, h$ and $x = x_n + h\,\beta_{1,0}\, f_0$, we get

$$\begin{aligned} x_{n+1} &= x_n + h[\gamma_0\, f_0 + \gamma_1\, f(t_n + \alpha_1\, h, x_n + h\,\beta_{1,0}\, f_0)] \\ &= x_n + \gamma_0 hf + \gamma_1 hf + \alpha_1\gamma_1 h^2 f_t + \beta_{1,0}\gamma_1 h^2 f f_x + O(h^3)\,. \end{aligned}$$

* Compare with the second-order Taylor series method,

$$x_{n+1} = x_n + h\,f + \frac{h^2}{2}\,(f_t + f\,f_x)\,.$$

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* Taylor series for a function of two variables:

$$f(t\,, x) = f(t_n, x_n) + f_t(t_n, x_n)(t - t_n) + f_x(t_n, x_n)(x - x_n)$$
$$+ \frac{1}{2\,!}\left[f_{tt}(t_n, x_n)(t - t_n)^2 + f_{tx}(t_n, x_n)(t - t_n)(x - x_n) + f_{xx}(t_n, x_n)(x - x_n)^2\right]$$
$$+ \text{Higher-order terms}\,.$$

* Substituting $t = t_n + \alpha_1\, h$ and $x = x_n + h\,\beta_{1,0}\, f_0$, we get

$$\begin{aligned} x_{n+1} &= x_n + h[\gamma_0\, f_0 + \gamma_1\, f(t_n + \alpha_1\, h, x_n + h\,\beta_{1,0}\, f_0)] \\ &= x_n + \gamma_0 hf + \gamma_1 hf + \alpha_1\gamma_1 h^2 f_t + \beta_{1,0}\gamma_1 h^2 f f_x + O(h^3)\,. \end{aligned}$$

* Compare with the second-order Taylor series method,

$$x_{n+1} = x_n + h\,f + \frac{h^2}{2}\,(f_t + f\,f_x)\,.$$

* With the conditions,

$$\begin{aligned} \gamma_0 + \gamma_1 &= 1\,, \\ \alpha_1\gamma_1 &= 1/2\,, \\ \beta_{1,0}\gamma_1 &= 1/2\,, \end{aligned}$$

the two algorithms are the same to $O(h^2)$.

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* There are four parameters for this method $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ and only three constraints:

$$\begin{aligned} \gamma_0 + \gamma_1 &= 1\,, \\ \alpha_1 \gamma_1 &= 1/2\,, \\ \beta_{1,0} \gamma_1 &= 1/2\,. \end{aligned}$$

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* There are four parameters for this method $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ and only three constraints:

$$\begin{aligned} \gamma_0 + \gamma_1 &= 1\,, \\ \alpha_1 \gamma_1 &= 1/2\,, \\ \beta_{1,0} \gamma_1 &= 1/2\,. \end{aligned}$$

* It is therefore not one method, but a *family* of methods.

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* There are four parameters for this method $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ and only three constraints:

$$\begin{aligned} \gamma_0 + \gamma_1 &= 1\,, \\ \alpha_1 \gamma_1 &= 1/2\,, \\ \beta_{1,0} \gamma_1 &= 1/2\,. \end{aligned}$$

* It is therefore not one method, but a *family* of methods.
* We can treat one of them (say, γ_0) as a "free" parameter. Assigning a value to the free parameter then defines the algorithm completely.

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* There are four parameters for this method $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ and only three constraints:

$$\begin{aligned} \gamma_0 + \gamma_1 &= 1\,, \\ \alpha_1 \gamma_1 &= 1/2\,, \\ \beta_{1,0} \gamma_1 &= 1/2\,. \end{aligned}$$

* It is therefore not one method, but a *family* of methods.
* We can treat one of them (say, γ_0) as a "free" parameter. Assigning a value to the free parameter then defines the algorithm completely.
* The parameters $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ are chosen so that the LTE is small for problems that are typically encountered.

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* There are four parameters for this method $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ and only three constraints:

$$\begin{aligned} \gamma_0 + \gamma_1 &= 1\,, \\ \alpha_1 \gamma_1 &= 1/2\,, \\ \beta_{1,0} \gamma_1 &= 1/2\,. \end{aligned}$$

* It is therefore not one method, but a *family* of methods.
* We can treat one of them (say, γ_0) as a "free" parameter. Assigning a value to the free parameter then defines the algorithm completely.
* The parameters $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ are chosen so that the LTE is small for problems that are typically encountered.
* For example, if γ_0 is chosen to be $1/4$, we get $\alpha_1 = 2/3, \beta_{1,0} = 2/3, \gamma_1 = 3/4$.

Runge-Kutta method for solving $\dot{x} = f(t, x)$

* There are four parameters for this method $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ and only three constraints:

$$\begin{aligned} \gamma_0 + \gamma_1 &= 1\,, \\ \alpha_1 \gamma_1 &= 1/2\,, \\ \beta_{1,0} \gamma_1 &= 1/2\,. \end{aligned}$$

* It is therefore not one method, but a *family* of methods.
* We can treat one of them (say, γ_0) as a "free" parameter. Assigning a value to the free parameter then defines the algorithm completely.
* The parameters $(\alpha_1, \beta_{1,0}, \gamma_0, \gamma_1)$ are chosen so that the LTE is small for problems that are typically encountered.
* For example, if γ_0 is chosen to be $1/4$, we get $\alpha_1 = 2/3, \beta_{1,0} = 2/3, \gamma_1 = 3/4$.
* The corresponding algorithm is,

$$\begin{aligned} f_0 &= f(t_n, x_n)\,, \\ f_1 &= f(t_n + \tfrac{2}{3}\,h, x_n + \tfrac{2}{3}\,h f_0)\,, \\ x_{n+1} &= x_n + h\,[\tfrac{1}{4}\,f_0 + \tfrac{3}{4}\,f_1]\,. \end{aligned}$$

Butcher array representation of RK methods [4]

	f_0	f_1	$\cdots$	f_{s-1}	f_s	
α_0	$\beta_{0,0}$	$\beta_{0,1}$		$\beta_{0,s-1}$	$\beta_{0,s}$	X_0
α_1	$\beta_{1,0}$	$\beta_{1,1}$		$\beta_{1,s-1}$	$\beta_{1,s}$	X_1
$\vdots$						$\vdots$
α_s	$\beta_{s,0}$	$\beta_{s,1}$		$\beta_{s,s-1}$	$\beta_{s,s}$	X_s
	γ_0	γ_1	$\cdots$	γ_{s-1}	γ_s	

Interpretation: For $i = 0, 1, \cdots, s$,

$$\begin{aligned} T_i &= t_n + \alpha_i h\,, \\ X_i &= x_n + h \sum_{j=0}^{s} \beta_{i,j} f_j\,, \\ f_i &= f(T_i, X_i)\,. \end{aligned}$$

Finally,

$$x_{n+1} = x_n + h \sum_{i=0}^{s} \gamma_i f_i\,.$$

Explicit RK methods

* When the β matrix in the Butcher array is lower-triangular, the RK method is *explicit*, i.e., the computation of f_i involves only f_1, f_2, $\cdots$, f_{i-1}.

Explicit RK methods

* When the β matrix in the Butcher array is lower-triangular, the RK method is *explicit*, i.e., the computation of f_i involves only f_1, f_2, $\cdots$, f_{i-1}.
* We can compute $f_0 = f(t_n, x_n)$, then f_1 (using f_0), followed by f_2 (using f_0 and f_1), and so on.

Explicit RK methods

* When the β matrix in the Butcher array is lower-triangular, the RK method is *explicit*, i.e., the computation of f_i involves only f_1, f_2, $\cdots$, f_{i-1}.
* We can compute $f_0 = f(t_n, x_n)$, then f_1 (using f_0), followed by f_2 (using f_0 and f_1), and so on.

Examples: second-order formulas [4]

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

$\alpha_1 = \frac{1}{2}$

$$\begin{array}{c|cc} 0 & & \\ \frac{2}{3} & \frac{2}{3} & \\ \hline & \frac{1}{4} & \frac{3}{4} \end{array}$$

$\alpha_1 = \frac{2}{3}$

(Heun form)

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

$\alpha_1 = 1$

(Improved Euler)

Explicit RK methods

Examples: third-order formulas [4]

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{3}{8} & \frac{2}{3} & \frac{1}{6} \end{array}$$

$\alpha_1 = \frac{1}{2}$, $\alpha_2 = 1$

(Classic form)

$$\begin{array}{c|ccc} 0 & & & \\ \frac{2}{3} & \frac{2}{3} & & \\ \frac{2}{3} & 0 & \frac{2}{3} & \\ \hline & \frac{1}{4} & \frac{3}{8} & \frac{3}{8} \end{array}$$

$\alpha_1 = \alpha_2 = \frac{2}{3}$

(Nystrom form)

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{3} & \frac{1}{3} & & \\ \frac{2}{3} & 0 & \frac{2}{3} & \\ \hline & \frac{1}{4} & 0 & \frac{3}{4} \end{array}$$

$\alpha_1 = \frac{1}{3}$, $\alpha_2 = \frac{2}{3}$

(Heun form)

Explicit RK methods

Examples: fourth-order formulas [4]

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

$\alpha_1 = \alpha_2 = \frac{1}{2}$ (Classic form)

0				
$\frac{1}{3}$	$\frac{1}{3}$			
$\frac{2}{3}$	$-\frac{1}{3}$	1		
1	1	-1	1	
	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$

$\alpha_1 = \frac{1}{3}$, $\alpha_2 = \frac{2}{3}$ (Kutta form)

Implicit RK methods

* When there are non-zero entries on the diagonal or in the upper triangle of the β matrix of the Butcher array, the corresponding RK method is an *implicit* method.

Implicit RK methods

* When there are non-zero entries on the diagonal or in the upper triangle of the β matrix of the Butcher array, the corresponding RK method is an *implicit* method.
* In this case, computation of f_i may involve f_i, f_{i+1}, etc., thus ruling out a simple successive computation of f_0, f_1, f_2,$\cdots$ (which is possible for explicit RK methods).

Implicit RK methods

* When there are non-zero entries on the diagonal or in the upper triangle of the β matrix of the Butcher array, the corresponding RK method is an *implicit* method.
* In this case, computation of f_i may involve f_i, f_{i+1}, etc., thus ruling out a simple successive computation of f_0, f_1, f_2,$\cdots$ (which is possible for explicit RK methods).
* Computation of f_i would then require an iterative procedure, which makes it expensive.

Implicit RK methods

* When there are non-zero entries on the diagonal or in the upper triangle of the β matrix of the Butcher array, the corresponding RK method is an *implicit* method.
* In this case, computation of f_i may involve f_i, f_{i+1}, etc., thus ruling out a simple successive computation of f_0, f_1, f_2,$\cdots$ (which is possible for explicit RK methods).
* Computation of f_i would then require an iterative procedure, which makes it expensive.

However, implicit methods have some advantages:

Implicit RK methods

* When there are non-zero entries on the diagonal or in the upper triangle of the β matrix of the Butcher array, the corresponding RK method is an *implicit* method.
* In this case, computation of f_i may involve f_i, f_{i+1}, etc., thus ruling out a simple successive computation of f_0, f_1, f_2,$\cdots$ (which is possible for explicit RK methods).
* Computation of f_i would then require an iterative procedure, which makes it expensive.

However, implicit methods have some advantages:

* An implicit RK formula may allow a higher order as compared to an explicit RK formula with the same number of stages.

Implicit RK methods

* When there are non-zero entries on the diagonal or in the upper triangle of the β matrix of the Butcher array, the corresponding RK method is an *implicit* method.
* In this case, computation of f_i may involve f_i, f_{i+1}, etc., thus ruling out a simple successive computation of f_0, f_1, f_2,$\cdots$ (which is possible for explicit RK methods).
* Computation of f_i would then require an iterative procedure, which makes it expensive.

However, implicit methods have some advantages:

* An implicit RK formula may allow a higher order as compared to an explicit RK formula with the same number of stages.
* Implicit formulas generally have better stability properties.

Implicit RK methods [4]

Examples:

$(3-\sqrt{3})/6$	$1/4$	$(3-2\sqrt{3})/12$
$(3+\sqrt{3})/6$	$(3+2\sqrt{3})/12$	$1/4$
	$1/2$	$1/2$

Fourth-order Gauss implicit method

Implicit RK methods [4]

Examples:

$(3-\sqrt{3})/6$	$1/4$	$(3-2\sqrt{3})/12$
$(3+\sqrt{3})/6$	$(3+2\sqrt{3})/12$	$1/4$
	$1/2$	$1/2$

Fourth-order Gauss implicit method

0	0	0	0	0
$(5-\sqrt{5})/10$	$(5+\sqrt{5})/60$	$1/6$	$(15-7\sqrt{5})/60$	0
$(5+\sqrt{5})/10$	$(5-\sqrt{5})/60$	$(15+7\sqrt{5})/60$	$1/6$	0
1	$1/6$	$(5-\sqrt{5})/12$	$(5+\sqrt{5})/12$	0
	$1/12$	$5/12$	$5/12$	$1/12$

Sixth-order Lobatto implicit method

RK method: system of equations

The RK methods (both explicit and implicit) can be used to solve a system of ODEs,

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}) , \quad \mathbf{x}(t_0) = \mathbf{x}_0 .$$

RK method: system of equations

The RK methods (both explicit and implicit) can be used to solve a system of ODEs,

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}) , \quad \mathbf{x}(t_0) = \mathbf{x}_0 .$$

The computation involves the following:

For $i = 0, 1, \cdots, s$,

$$\begin{aligned} T_i &= t_n + \alpha_i h , \\ \mathbf{X_i} &= \mathbf{x_n} + h \sum_{j=0}^{s} \beta_{i,j} \, \mathbf{f_j} , \\ \mathbf{f_j} &= \mathbf{f}\left(T_i, \mathbf{X_i}\right) , \end{aligned}$$

and finally,

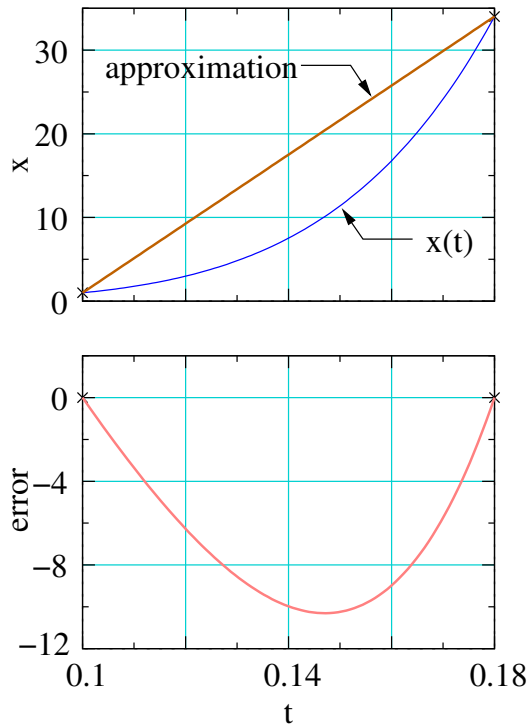
$$\mathbf{x_{n+1}} = \mathbf{x_n} + h \sum_{i=0}^{s} \gamma_i \, \mathbf{f_i} .$$

Outline

* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

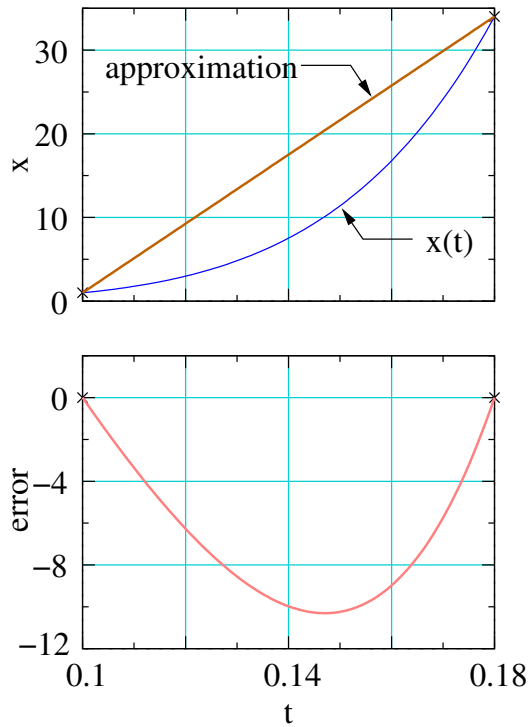
Multi-step methods for solving $\dot{x} = f(t, x)$

Consider fitting the function $x(t)$ with a staight line.



Multi-step methods for solving $\dot{x} = f(t, x)$

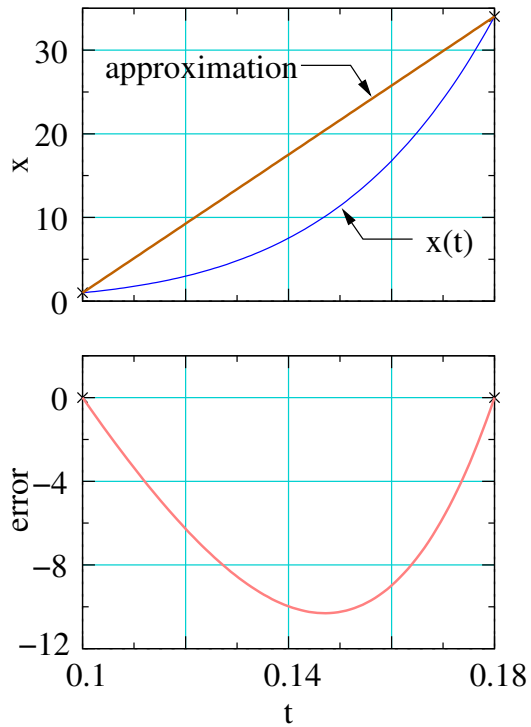
Consider fitting the function $x(t)$ with a staight line.



The fit can be improved in two ways:

Multi-step methods for solving $\dot{x} = f(t, x)$

Consider fitting the function $x(t)$ with a staight line.

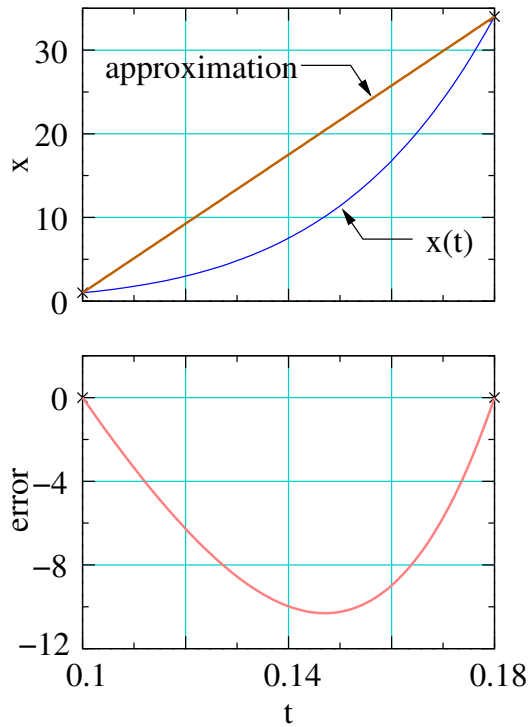


The fit can be improved in two ways:

* Reduce the time step.

Multi-step methods for solving $\dot{x} = f(t, x)$

Consider fitting the function $x(t)$ with a staight line.

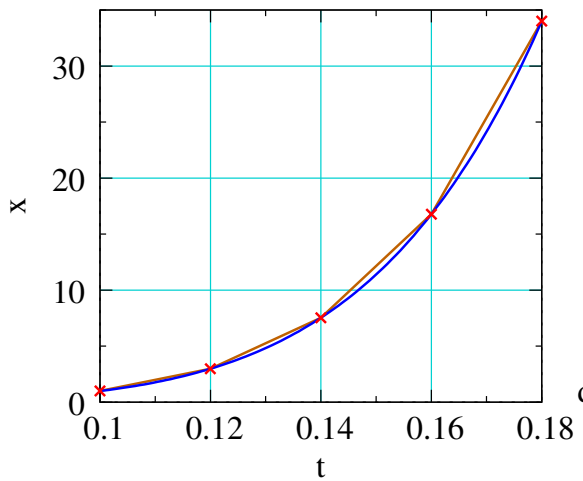


The fit can be improved in two ways:

* Reduce the time step.
* Use a higher-order polynomial.

Multi-step methods for solving $\dot{x} = f(t, x)$

Use of a smaller time step:

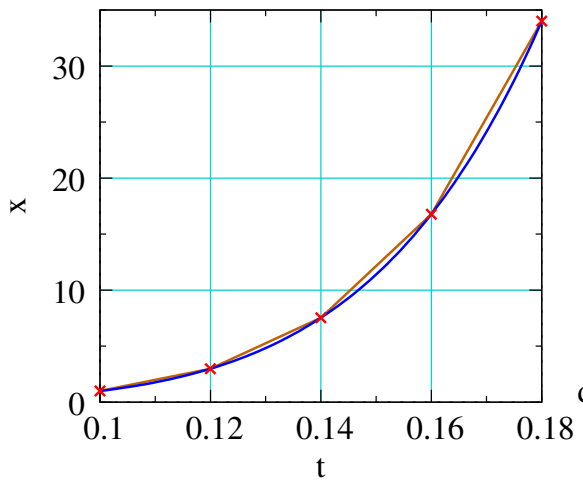


solid line: x(t)

dashed line: approximation

Multi-step methods for solving $\dot{x} = f(t, x)$

Use of a smaller time step:



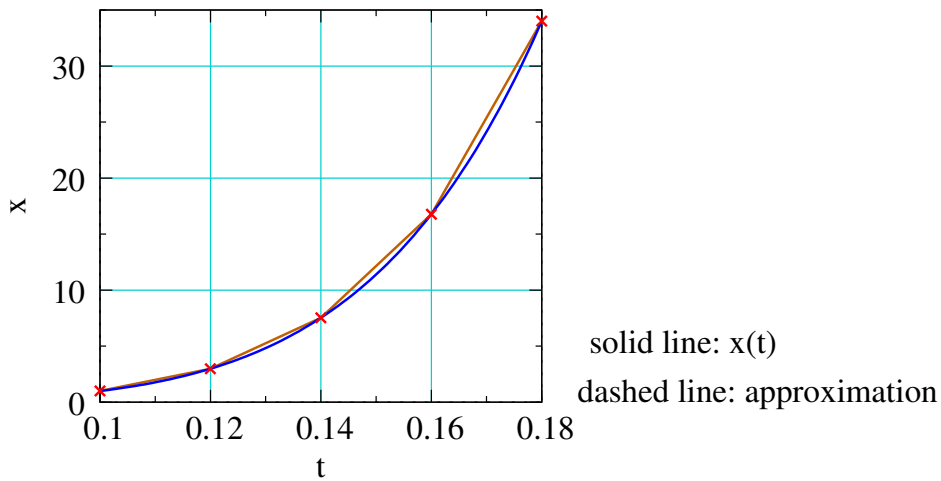
solid line: x(t)

dashed line: approximation

* The approximation is better when the step size is reduced.

Multi-step methods for solving $\dot{x} = f(t, x)$

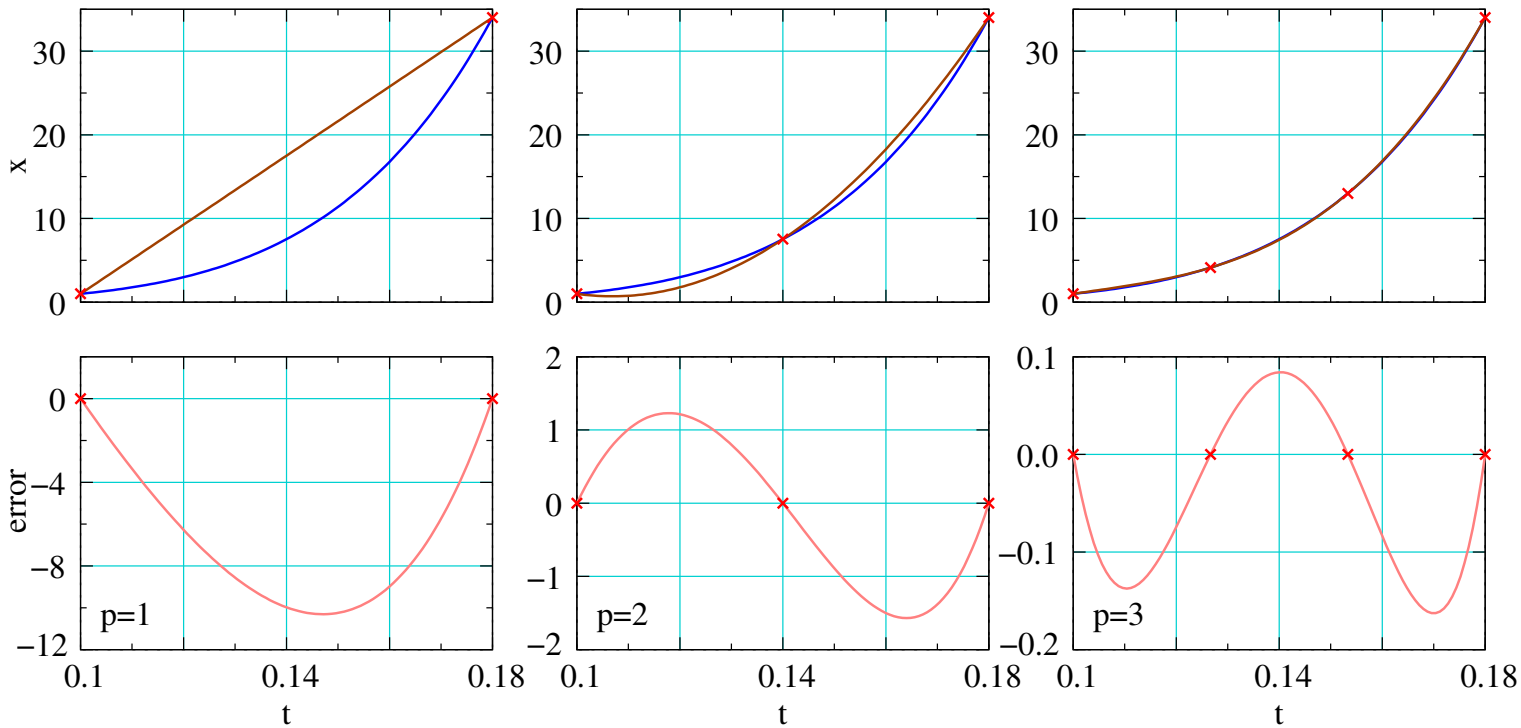
Use of a smaller time step:



* The approximation is better when the step size is reduced.
* A larger number of time steps $\Rightarrow$ slower simulation

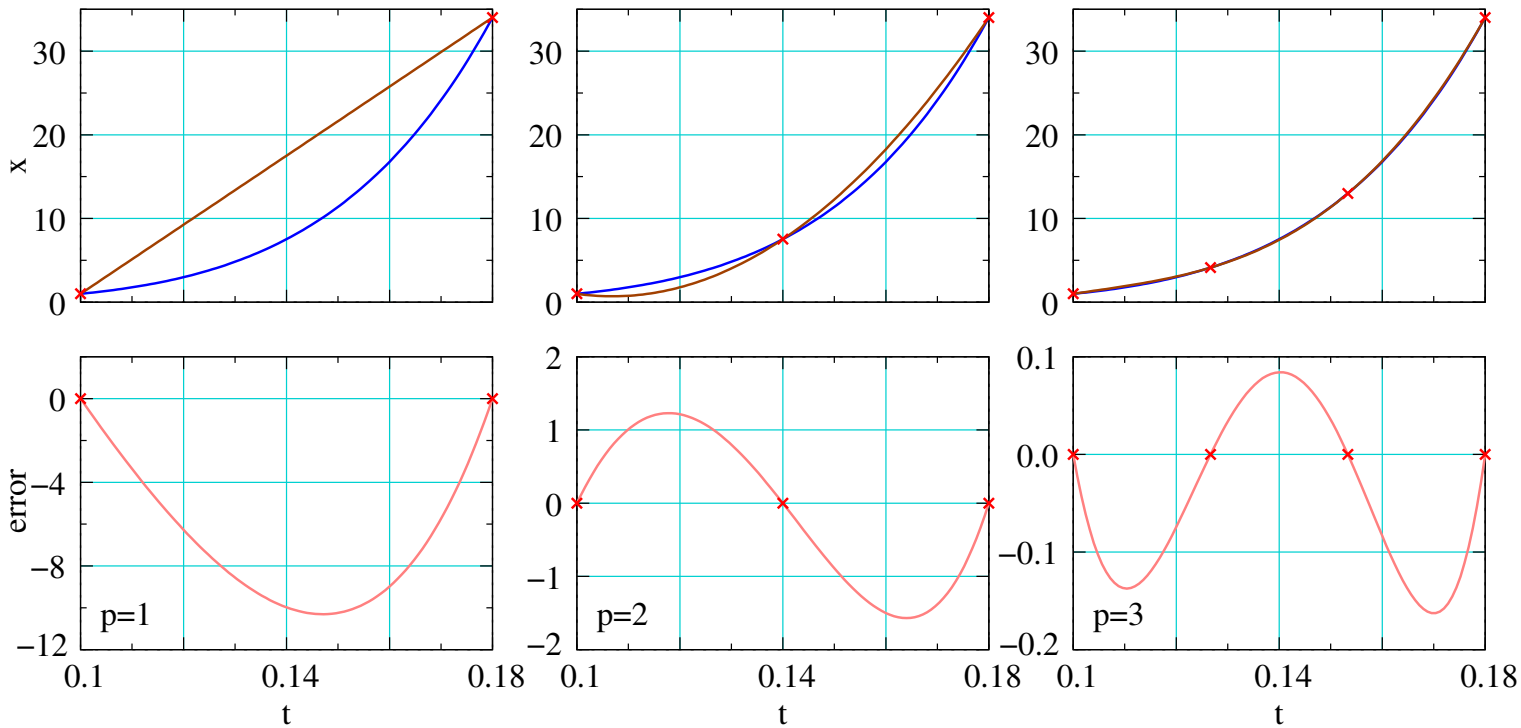
Multi-step methods for solving $\dot{x} = f(t, x)$

Use of a higher-order polynomial:



Multi-step methods for solving $\dot{x} = f(t, x)$

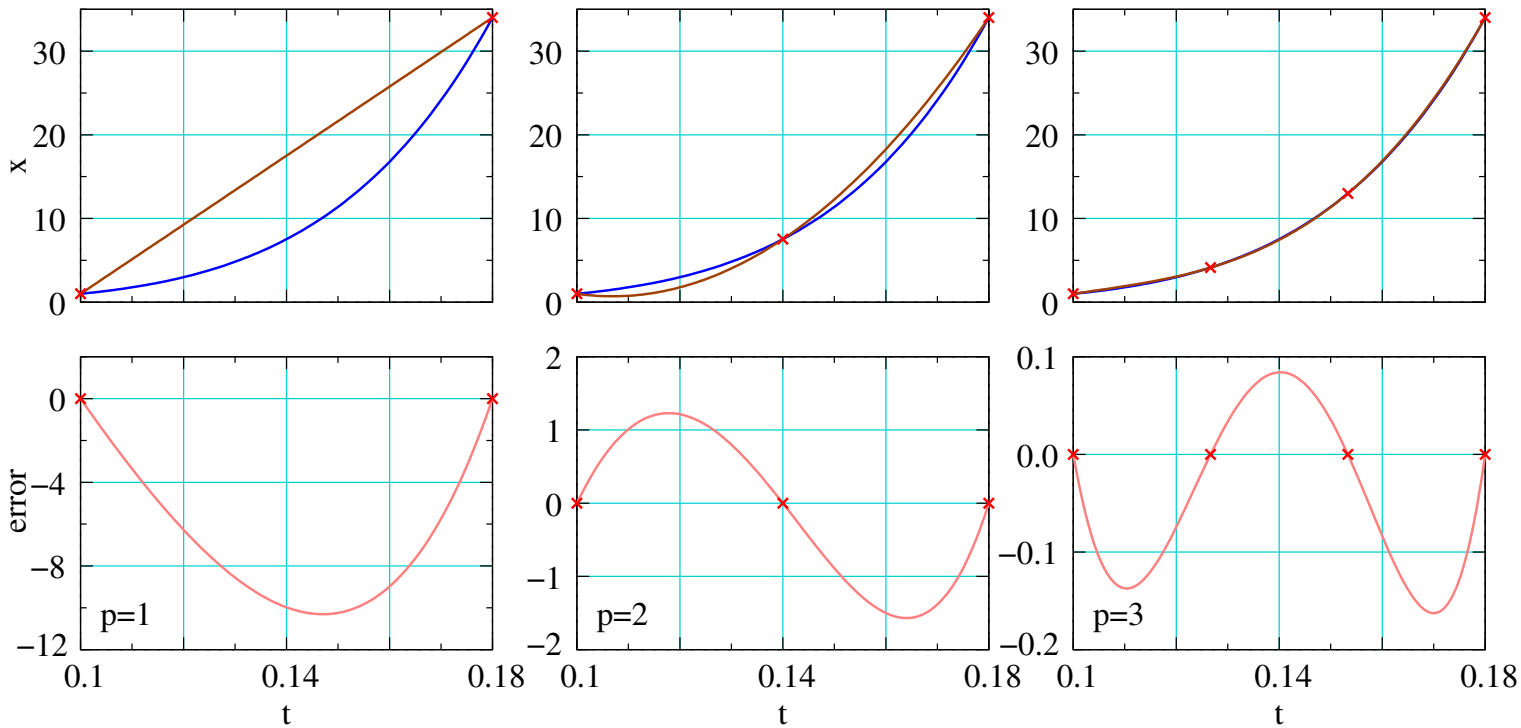
Use of a higher-order polynomial:



* The approximation is better when the order is increased.

Multi-step methods for solving $\dot{x} = f(t, x)$

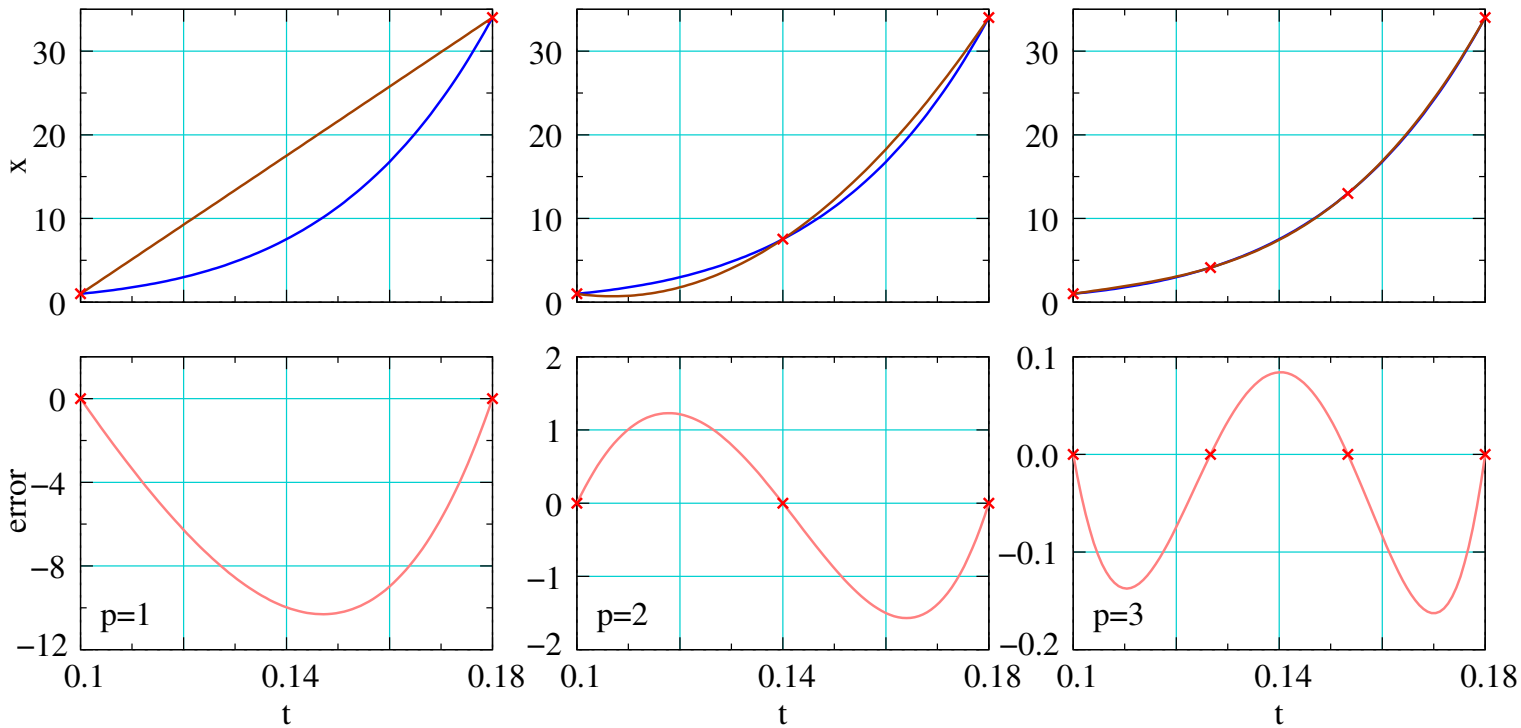
Use of a higher-order polynomial:



* The approximation is better when the order is increased.
* For fitting with a polynomial of order p, we need $(p + 1)$ points.

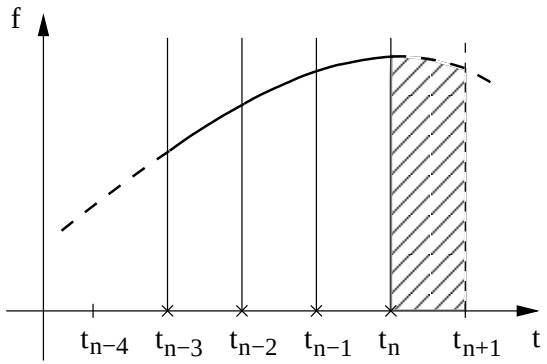
Multi-step methods for solving $\dot{x} = f(t, x)$

Use of a higher-order polynomial:

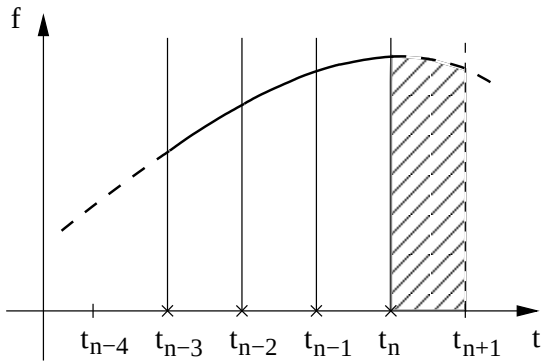


* The approximation is better when the order is increased.
* For fitting with a polynomial of order p, we need $(p + 1)$ points.
* The Adams-Bashforth and Adams-Moulton methods are based on approximating $x(t)$ with a polynomial.

Adams-Bashforth methods for $\dot{x} = f(t, x)$



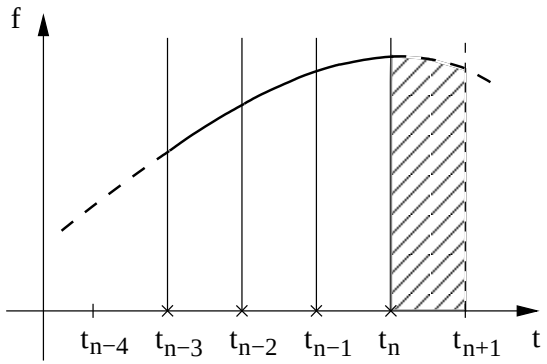
Adams-Bashforth methods for $\dot{x} = f(t, x)$



* Motivation:

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + \int_{t_n}^{t_{n+1}} x'(t)dt \\ &= x(t_n) + \int_{t_n}^{t_{n+1}} f\, dt\,. \end{aligned} \tag{3}$$

Adams-Bashforth methods for $\dot{x} = f(t, x)$

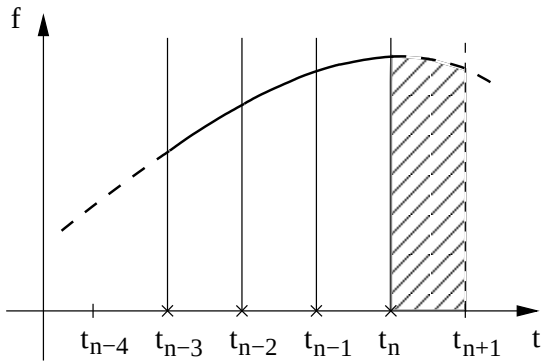


* Motivation:

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + \int_{t_n}^{t_{n+1}} x'(t)dt \\ &= x(t_n) + \int_{t_n}^{t_{n+1}} f\, dt\,. \end{aligned} \tag{3}$$

* Obtain a polynomial (in t) which passes through (t_n, f_n), (t_{n-1}, f_{n-1}), etc.

Adams-Bashforth methods for $\dot{x} = f(t, x)$



* Motivation:

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + \int_{t_n}^{t_{n+1}} x'(t)dt \\ &= x(t_n) + \int_{t_n}^{t_{n+1}} f\, dt\,. \end{aligned} \tag{3}$$

* Obtain a polynomial (in t) which passes through (t_n, f_n), (t_{n-1}, f_{n-1}), etc.
* Compute $\int_{t_n}^{t_{n+1}} f\, dt$ in Eq. 3 using the approximation for f $\Rightarrow$ Adams-Bashforth formula.

Adams-Bashforth methods for $\dot{x} = f(t, x)$

The AB formula of order p is given by,

$$x(t_{n+1}) = x(t_n) + h \sum_{i=0}^{p-1} \beta_i \, f_{n-i} \ . \tag{4}$$

Adams-Bashforth methods for $\dot{x} = f(t, x)$

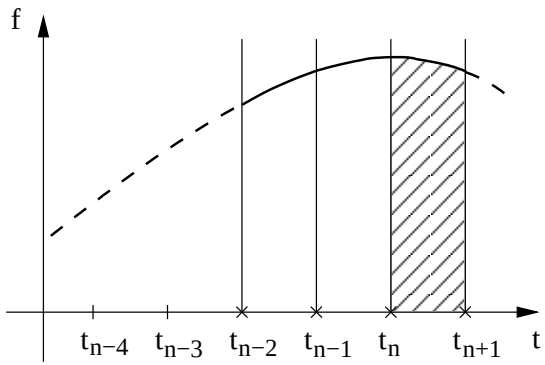
The AB formula of order p is given by,

$$x(t_{n+1}) = x(t_n) + h \sum_{i=0}^{p-1} \beta_i \, f_{n-i} \ . \tag{4}$$

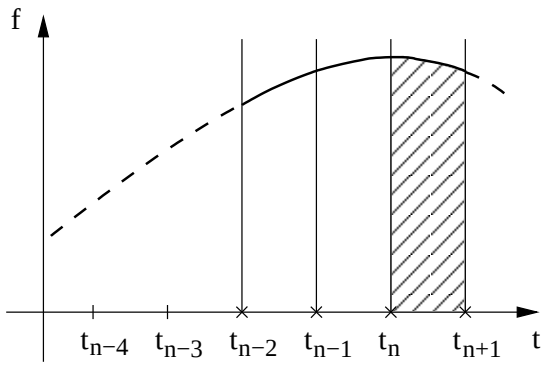
Order	β_0	β_1	β_2	β_3	β_4	β_5	LTE
1	1						$\frac{1}{2} h^2 x''(\xi_0)$
2	$\frac{3}{2}$	$-\frac{1}{2}$					$\frac{5}{12} h^3 x^{(3)}(\xi_0)$
3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$				$\frac{9}{24} h^4 x^{(4)}(\xi_0)$
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$			$\frac{251}{720} h^5 x^{(5)}(\xi_0)$
5	$\frac{1901}{720}$	$-\frac{2774}{720}$	$\frac{2616}{720}$	$-\frac{1274}{720}$	$\frac{251}{720}$		$\frac{475}{1440} h^6 x^{(6)}(\xi_0)$
6	$\frac{4277}{1440}$	$-\frac{7923}{1440}$	$\frac{9982}{1440}$	$-\frac{7298}{1440}$	$\frac{2877}{1440}$	$-\frac{475}{1440}$	$\frac{19,087}{60,480} h^7 x^{(7)}(\xi_0)$

(Note that the AB1 formula is the same as Forward Euler.)

Adams-Moulton methods for $\dot{x} = f(t, x)$



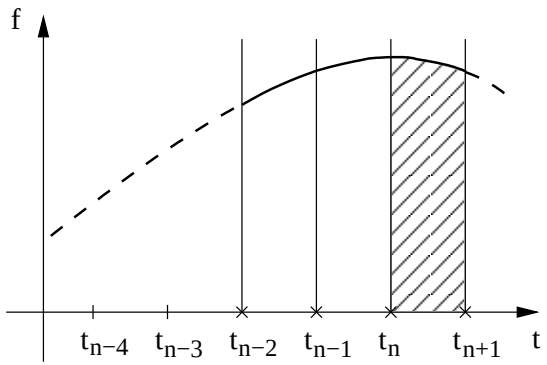
Adams-Moulton methods for $\dot{x} = f(t, x)$



* Motivation:

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + \int_{t_n}^{t_{n+1}} x'(t)dt \\ &= x(t_n) + \int_{t_n}^{t_{n+1}} f\, dt\,. \end{aligned} \tag{5}$$

Adams-Moulton methods for $\dot{x} = f(t, x)$

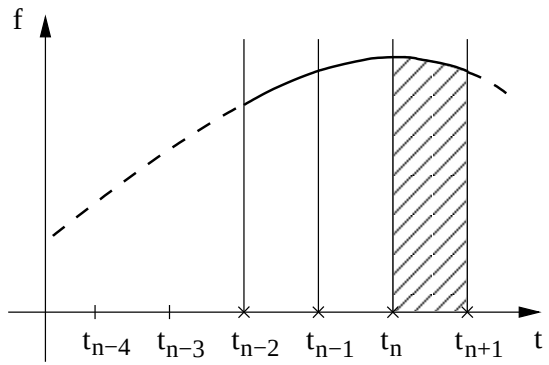


* Motivation:

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + \int_{t_n}^{t_{n+1}} x'(t)dt \\ &= x(t_n) + \int_{t_n}^{t_{n+1}} f\, dt\,. \end{aligned} \tag{5}$$

* Obtain a polynomial (in t) which passes through (t_{n+1}, f_{n+1}), (t_n, f_n), (t_{n-1}, f_{n-1}), etc. Note the involvement of f_{n+1} here, which makes the AM methods implicit in nature.

Adams-Moulton methods for $\dot{x} = f(t, x)$



* Motivation:

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + \int_{t_n}^{t_{n+1}} x'(t)dt \\ &= x(t_n) + \int_{t_n}^{t_{n+1}} f\, dt\,. \end{aligned} \tag{5}$$

* Obtain a polynomial (in t) which passes through (t_{n+1}, f_{n+1}), (t_n, f_n), (t_{n-1}, f_{n-1}), etc. Note the involvement of f_{n+1} here, which makes the AM methods implicit in nature.
* Compute $\int_{t_n}^{t_{n+1}} f\, dt$ in Eq. 5 using the approximation for f $\Rightarrow$ Adams-Moulton formula.

Adams-Moulton methods for $\dot{x} = f(t, x)$

The AM formula of order p is given by,

$$x(t_{n+1}) = x(t_n) + h \sum_{i=-1}^{p-2} \beta_i \, f_{n-i} \ . \tag{6}$$

Adams-Moulton methods for $\dot{x} = f(t, x)$

The AM formula of order p is given by,

$$x(t_{n+1}) = x(t_n) + h \sum_{i=-1}^{p-2} \beta_i \, f_{n-i} \ . \tag{6}$$

Order	β_{-1}	β_0	β_1	β_2	β_3	β_4	LTE
1	1						$-\frac{1}{2} h^2 x''(\xi_0)$
2	$\frac{1}{2}$	$\frac{1}{2}$					$-\frac{1}{12} h^3 x^{(3)}(\xi_0)$
3	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$				$-\frac{1}{24} h^4 x^{(4)}(\xi_0)$
4	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$			$-\frac{19}{720} h^5 x^{(5)}(\xi_0)$
5	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$		$-\frac{27}{1440} h^6 x^{(6)}(\xi_0)$
6	$\frac{475}{1440}$	$\frac{1427}{1440}$	$-\frac{798}{1440}$	$\frac{482}{1440}$	$-\frac{173}{1440}$	$\frac{27}{1440}$	$-\frac{863}{60,480} h^7 x^{(7)}(\xi_0)$

Adams-Moulton methods for $\dot{x} = f(t, x)$

The AM formula of order p is given by,

$$x(t_{n+1}) = x(t_n) + h \sum_{i=-1}^{p-2} \beta_i \, f_{n-i} \ . \tag{6}$$

Order	β_{-1}	β_0	β_1	β_2	β_3	β_4	LTE
1	1						$-\frac{1}{2} h^2 x''(\xi_0)$
2	$\frac{1}{2}$	$\frac{1}{2}$					$-\frac{1}{12} h^3 x^{(3)}(\xi_0)$
3	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$				$-\frac{1}{24} h^4 x^{(4)}(\xi_0)$
4	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$			$-\frac{19}{720} h^5 x^{(5)}(\xi_0)$
5	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$		$-\frac{27}{1440} h^6 x^{(6)}(\xi_0)$
6	$\frac{475}{1440}$	$\frac{1427}{1440}$	$-\frac{798}{1440}$	$\frac{482}{1440}$	$-\frac{173}{1440}$	$\frac{27}{1440}$	$-\frac{863}{60{,}480} h^7 x^{(7)}(\xi_0)$

* The AM1 and AM2 formulas are the same as the Backward Euler and trapezoidal methods, respectively.

Adams-Moulton methods for $\dot{x} = f(t, x)$

The AM formula of order p is given by,

$$x(t_{n+1}) = x(t_n) + h \sum_{i=-1}^{p-2} \beta_i \, f_{n-i} \ . \tag{6}$$

Order	β_{-1}	β_0	β_1	β_2	β_3	β_4	LTE
1	1						$-\frac{1}{2} h^2 x''(\xi_0)$
2	$\frac{1}{2}$	$\frac{1}{2}$					$-\frac{1}{12} h^3 x^{(3)}(\xi_0)$
3	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$				$-\frac{1}{24} h^4 x^{(4)}(\xi_0)$
4	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$			$-\frac{19}{720} h^5 x^{(5)}(\xi_0)$
5	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$		$-\frac{27}{1440} h^6 x^{(6)}(\xi_0)$
6	$\frac{475}{1440}$	$\frac{1427}{1440}$	$-\frac{798}{1440}$	$\frac{482}{1440}$	$-\frac{173}{1440}$	$\frac{27}{1440}$	$-\frac{863}{60,480} h^7 x^{(7)}(\xi_0)$

* The AM1 and AM2 formulas are the same as the Backward Euler and trapezoidal methods, respectively.
* By comparing the LTE columns in the AB and AM tables, we see that, for the same order, the AM formula is more accurate.

Backward Differentiation Formulas (BDF): Gear's formulas

We are looking for $x(t)$ which will satisfy the ODE at $t = t_{n+1}$, i.e.,

$$\dot{x}(t_{n+1}) = f(t_{n+1}, x_{n+1})\,. \tag{7}$$

Backward Differentiation Formulas (BDF): Gear's formulas

We are looking for $x(t)$ which will satisfy the ODE at $t = t_{n+1}$, i.e.,

$$\dot{x}(t_{n+1}) = f(t_{n+1}, x_{n+1})\,. \tag{7}$$

* First, obtain $\tilde{x}(t)$, a polynomial approximation for $x(t)$, passing through (t_{n+1},x_{n+1}), (t_n,x_n), (t_{n-1},x_{n-1}), $\cdots$.

Backward Differentiation Formulas (BDF): Gear's formulas

We are looking for $x(t)$ which will satisfy the ODE at $t = t_{n+1}$, i.e.,

$$\dot{x}(t_{n+1}) = f(t_{n+1}, x_{n+1})\,. \tag{7}$$

* First, obtain $\tilde{x}(t)$, a polynomial approximation for $x(t)$, passing through (t_{n+1},x_{n+1}), (t_n,x_n), (t_{n-1},x_{n-1}), $\cdots$.
* Differentiate to get an expression for $\dot{\tilde{x}}(t)$.

Backward Differentiation Formulas (BDF): Gear's formulas

We are looking for $x(t)$ which will satisfy the ODE at $t = t_{n+1}$, i.e.,

$$\dot{x}(t_{n+1}) = f(t_{n+1}, x_{n+1})\,. \tag{7}$$

* First, obtain $\tilde{x}(t)$, a polynomial approximation for $x(t)$, passing through (t_{n+1}, x_{n+1}), (t_n, x_n), (t_{n-1}, x_{n-1}), $\cdots$.
* Differentiate to get an expression for $\dot{\tilde{x}}(t)$.
* Replace the LHS of Eq. 7 with $\dot{\tilde{x}}(t)$ at $t = t_{n+1}$. $\Rightarrow$ BDF formula

Backward Differentiation Formulas (BDF): Gear's formulas

We are looking for $x(t)$ which will satisfy the ODE at $t = t_{n+1}$, i.e.,

$$\dot{x}(t_{n+1}) = f(t_{n+1}, x_{n+1})\,. \tag{7}$$

* First, obtain $\tilde{x}(t)$, a polynomial approximation for $x(t)$, passing through (t_{n+1},x_{n+1}), (t_n,x_n), (t_{n-1},x_{n-1}), $\cdots$.
* Differentiate to get an expression for $\dot{\tilde{x}}(t)$.
* Replace the LHS of Eq. 7 with $\dot{\tilde{x}}(t)$ at $t = t_{n+1}$. $\Rightarrow$ BDF formula
* BDFs are implicit in nature since $f(t_{n+1}, x_{n+1})$ appears in the formula.

BDFs for $\dot{x} = f(t, x)$

The general form of the BDF of order p is,

$$\sum_{i=-1}^{p-1} \alpha_i \, x_{n-i} = h \, f(t_{n+1}, x_{n+1}) \, . \tag{8}$$

BDFs for $\dot{x} = f(t, x)$

The general form of the BDF of order p is,

$$\sum_{i=-1}^{p-1} \alpha_i \, x_{n-i} = h \, f(t_{n+1}, x_{n+1}) \, . \tag{8}$$

Order	α_{-1}	α_0	α_1	α_2	α_3	α_4	α_5	LTE
1	1	-1						$-\frac{1}{2} h^2 x''(\xi)$
2	$\frac{3}{2}$	-2	$\frac{1}{2}$					$-\frac{2}{9} h^3 x'''(\xi)$
3	$\frac{11}{6}$	-3	$-\frac{3}{2}$	$-\frac{1}{3}$				$-\frac{3}{22} h^4 x^{(4)}(\xi)$
4	$\frac{25}{12}$	-4	3	$-\frac{4}{3}$	$\frac{1}{4}$			$-\frac{12}{125} h^5 x^{(5)}(\xi)$
5	$\frac{137}{60}$	-5	5	$-\frac{10}{3}$	$\frac{5}{4}$	$-\frac{1}{5}$		$-\frac{10}{137} h^6 x^{(6)}(\xi)$
6	$\frac{147}{60}$	-6	$\frac{15}{2}$	$-\frac{20}{3}$	$\frac{15}{4}$	$-\frac{6}{5}$	$\frac{1}{6}$	$-\frac{60}{1029} h^7 x^{(7)}(\xi)$

(Note that the BDF1 formula is the same as the Backward Euler method.)

Outline

* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

The AB, AM, and BDF methods are special cases of "linear multi-step methods" (LMM) given by,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

Method	k	α_i	β_i
AB	$p-1$	$\alpha_i = 0, \quad i = 1$ to k	$\beta_{-1} = 0$
AM	$p-2$	$\alpha_i = 0, \quad i = 1$ to k	–
BDF	$p-1$	–	$\beta_i = 0, \quad i = 0$ to k

The AB, AM, and BDF methods are special cases of "linear multi-step methods" (LMM) given by,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

Method	k	α_i	β_i
AB	$p-1$	$\alpha_i = 0, \quad i = 1$ to k	$\beta_{-1} = 0$
AM	$p-2$	$\alpha_i = 0, \quad i = 1$ to k	–
BDF	$p-1$	–	$\beta_i = 0, \quad i = 0$ to k

* Many other LMMs can be derived; all we need to do is to pick a polynomial passing through a suitable set of points.

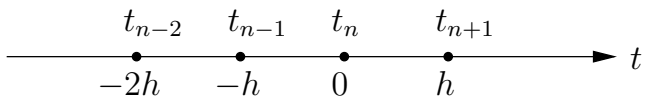
The AB, AM, and BDF methods are special cases of "linear multi-step methods" (LMM) given by,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

Method	k	α_i	β_i
AB	$p-1$	$\alpha_i = 0, \quad i = 1$ to k	$\beta_{-1} = 0$
AM	$p-2$	$\alpha_i = 0, \quad i = 1$ to k	–
BDF	$p-1$	–	$\beta_i = 0, \quad i = 0$ to k

* Many other LMMs can be derived; all we need to do is to pick a polynomial passing through a suitable set of points.
* What is so special about the AM, AB, and BDF methods? (to be discussed)

Exactness constraints for LMMs [6] for solving $\dot{x} = f(t, x)$

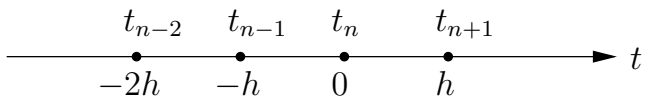


As an example, consider the LMM given by,

$$\alpha_{-1}\, x_{n+1} + \alpha_0\, x_n + \alpha_1\, x_{n-1} = h\, \beta_{-1}\, f_{n+1}\,. \tag{9}$$

There are three independent coefficients here $\Rightarrow$ the LMM formula is expected to accurately predict x_{n+1} if $x(t)$ is a second-order polynomial.

Exactness constraints for LMMs [6] for solving $\dot{x} = f(t, x)$



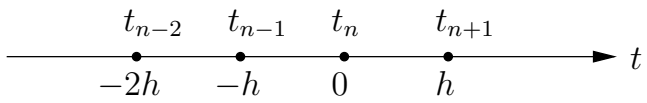
As an example, consider the LMM given by,

$$\alpha_{-1}\, x_{n+1} + \alpha_0\, x_n + \alpha_1\, x_{n-1} = h\, \beta_{-1}\, f_{n+1}\,. \tag{9}$$

There are three independent coefficients here $\Rightarrow$ the LMM formula is expected to accurately predict x_{n+1} if $x(t)$ is a second-order polynomial.

In particular, consider the special cases: (a) $x(t) = 1$, (b) $x(t) = t$, and (c) $x(t) = t^2$.

Exactness constraints for LMMs [6] for solving $\dot{x} = f(t, x)$



As an example, consider the LMM given by,

$$\alpha_{-1}\, x_{n+1} + \alpha_0\, x_n + \alpha_1\, x_{n-1} = h\, \beta_{-1}\, f_{n+1}\,. \tag{9}$$

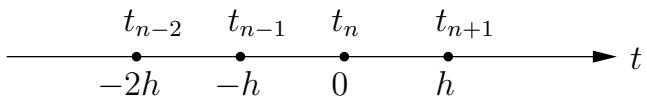
There are three independent coefficients here $\Rightarrow$ the LMM formula is expected to accurately predict x_{n+1} if $x(t)$ is a second-order polynomial.

In particular, consider the special cases: (a) $x(t) = 1$, (b) $x(t) = t$, and (c) $x(t) = t^2$.

For $x(t) = 1$, $f(t, x) = 0$, and $x_{n-1} = x_n = x_{n+1} = 1$. Substituting in (9), we get,

$$\alpha_{-1} + \alpha_0 + \alpha_1 = 0\,.$$

Exactness constraints for LMMs [6] for solving $\dot{x} = f(t, x)$



As an example, consider the LMM given by,

$$\alpha_{-1}\, x_{n+1} + \alpha_0\, x_n + \alpha_1\, x_{n-1} = h\, \beta_{-1}\, f_{n+1}\,. \tag{9}$$

There are three independent coefficients here $\Rightarrow$ the LMM formula is expected to accurately predict x_{n+1} if $x(t)$ is a second-order polynomial.

In particular, consider the special cases: (a) $x(t) = 1$, (b) $x(t) = t$, and (c) $x(t) = t^2$.

For $x(t) = 1$, $f(t, x) = 0$, and $x_{n-1} = x_n = x_{n+1} = 1$. Substituting in (9), we get,

$$\alpha_{-1} + \alpha_0 + \alpha_1 = 0\,.$$

Similarly, the other two exactness constraints can be derived.

Exactness constraints for LMMs for solving $\dot{x} = f(t, x)$

x	f	x_{n+1}	Constraint
1	0	1	$\alpha_{-1} + \alpha_0 + \alpha_1 = 0$
t	1	h	$\alpha_{-1} - \alpha_1 = \beta_{-1}$
t^2	$2\,t$	h^2	$\alpha_{-1} + \alpha_1 = 2\,\beta_{-1}$

Exactness constraints for LMMs for solving $\dot{x} = f(t, x)$

x	f	x_{n+1}	Constraint
1	0	1	$\alpha_{-1} + \alpha_0 + \alpha_1 = 0$
t	1	h	$\alpha_{-1} - \alpha_1 = \beta_{-1}$
t^2	$2\,t$	h^2	$\alpha_{-1} + \alpha_1 = 2\,\beta_{-1}$

* With $\beta_{-1} = 1$, we get $\alpha_{-1} = 3/2$, $\alpha_0 = -2$, and $\alpha_1 = 1/2$.

Exactness constraints for LMMs for solving $\dot{x} = f(t, x)$

x	f	x_{n+1}	Constraint
1	0	1	$\alpha_{-1} + \alpha_0 + \alpha_1 = 0$
t	1	h	$\alpha_{-1} - \alpha_1 = \beta_{-1}$
t^2	$2\,t$	h^2	$\alpha_{-1} + \alpha_1 = 2\,\beta_{-1}$

* With $\beta_{-1} = 1$, we get $\alpha_{-1} = 3/2$, $\alpha_0 = -2$, and $\alpha_1 = 1/2$.
* The LMM formula is therefore,

$$\frac{3}{2}\,x_{n+1} - 2\,x_n + \frac{1}{2}\,x_{n-1} = h\,f_{n+1}\,.$$

Exactness constraints for LMMs for solving $\dot{x} = f(t, x)$

x	f	x_{n+1}	Constraint
1	0	1	$\alpha_{-1} + \alpha_0 + \alpha_1 = 0$
t	1	h	$\alpha_{-1} - \alpha_1 = \beta_{-1}$
t^2	$2\,t$	h^2	$\alpha_{-1} + \alpha_1 = 2\,\beta_{-1}$

* With $\beta_{-1} = 1$, we get $\alpha_{-1} = 3/2$, $\alpha_0 = -2$, and $\alpha_1 = 1/2$.
* The LMM formula is therefore,

$$\frac{3}{2}\,x_{n+1} - 2\,x_n + \frac{1}{2}\,x_{n-1} = h\,f_{n+1}\,.$$

* This is the same as the BDF2 formula.

Exactness constraints for generalized LMM

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

Exactness constraints for generalized LMM

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

By following the procedure described earlier, we get the following constraints:

$$x(t) = 1 : \qquad \sum_{i=-1}^{k} \alpha_i = 0 \, ,$$

Exactness constraints for generalized LMM

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

By following the procedure described earlier, we get the following constraints:

$$x(t) = 1 : \qquad \sum_{i=-1}^{k} \alpha_i = 0 \, ,$$

$$x(t) = t : \qquad \sum_{i=-1}^{k} \alpha_i \, (-ih) = h \sum_{i=-1}^{k} \beta_i \, ,$$

Exactness constraints for generalized LMM

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

By following the procedure described earlier, we get the following constraints:

$$x(t) = 1 : \qquad \sum_{i=-1}^{k} \alpha_i = 0 \, ,$$

$$x(t) = t : \qquad \sum_{i=-1}^{k} \alpha_i \, (-ih) = h \sum_{i=-1}^{k} \beta_i \, ,$$

$$x(t) = t^2 : \qquad \sum_{i=-1}^{k} \alpha_i \, (-ih)^2 = h \sum_{i=-1}^{k} 2 \, \beta_i \, (-ih) \, ,$$

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

By following the procedure described earlier, we get the following constraints:

$$\begin{aligned}
x(t) = 1 : \quad & \sum_{i=-1}^{k} \alpha_i = 0 \, , \\
x(t) = t : \quad & \sum_{i=-1}^{k} \alpha_i \, (-ih) = h \sum_{i=-1}^{k} \beta_i \, , \\
x(t) = t^2 : \quad & \sum_{i=-1}^{k} \alpha_i \, (-ih)^2 = h \sum_{i=-1}^{k} 2 \, \beta_i \, (-ih) \, , \\
& \vdots \qquad\qquad\qquad \vdots \\
x(t) = t^p : \quad & \sum_{i=-1}^{k} \alpha_i \, (-ih)^p = h \sum_{i=-1}^{k} p \, \beta_i \, (-ih)^{p-1} \, .
\end{aligned} \tag{10}$$

Outline

* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.
* AB methods are explicit whereas AM methods are implicit.

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.
* AB methods are explicit whereas AM methods are implicit.
  $\Rightarrow$ For the same order, the AB method is faster.

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.
* AB methods are explicit whereas AM methods are implicit.
  $\Rightarrow$ For the same order, the AB method is faster.
* Can we combine the best of the AB and AM methods?

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.
* AB methods are explicit whereas AM methods are implicit.
  $\Rightarrow$ For the same order, the AB method is faster.
* Can we combine the best of the AB and AM methods?
  $\Rightarrow$ Predictor-Corrector (PC) methods

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.
* AB methods are explicit whereas AM methods are implicit.
  $\Rightarrow$ For the same order, the AB method is faster.
* Can we combine the best of the AB and AM methods?
  $\Rightarrow$ Predictor-Corrector (PC) methods
* In the PC method, in going from t_n to t_{n+1},

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.
* AB methods are explicit whereas AM methods are implicit.
  $\Rightarrow$ For the same order, the AB method is faster.
* Can we combine the best of the AB and AM methods?
  $\Rightarrow$ Predictor-Corrector (PC) methods
* In the PC method, in going from t_n to t_{n+1},
  - "Predict" x_{n+1} using an explicit method (such as AB).

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.
* AB methods are explicit whereas AM methods are implicit.
  $\Rightarrow$ For the same order, the AB method is faster.
* Can we combine the best of the AB and AM methods?
  $\Rightarrow$ Predictor-Corrector (PC) methods
* In the PC method, in going from t_n to t_{n+1},
  - "Predict" x_{n+1} using an explicit method (such as AB).
  - "Correct" x_{n+1} using an implicit method (such as AM). However, in this step, use the implicit formula as an "evaluation" formula, i.e., treat x_{n+1} in the RHS as a *known* value (given by the predicted x_{n+1}).

Predictor-corrector methods

* For the same order, AM methods are more accurate than AB methods.
* AB methods are explicit whereas AM methods are implicit.
  $\Rightarrow$ For the same order, the AB method is faster.
* Can we combine the best of the AB and AM methods?
  $\Rightarrow$ Predictor-Corrector (PC) methods
* In the PC method, in going from t_n to t_{n+1},
  - "Predict" x_{n+1} using an explicit method (such as AB).
  - "Correct" x_{n+1} using an implicit method (such as AM). However, in this step, use the implicit formula as an "evaluation" formula, i.e., treat x_{n+1} in the RHS as a *known* value (given by the predicted x_{n+1}).
  - Repeat to the desired tolerance.

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector.

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector.

$$\text{Prediction} \quad \text{(P)} \qquad x_{n+1}^{(0)} = x_n + h\, f_n \qquad \text{(AB1)}$$

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector.

$$\begin{array}{llll} \text{Prediction} \quad (\text{P}) & x_{n+1}^{(0)} & = & x_n + h\, f_n \qquad\qquad (\text{AB1}) \\ \text{Evaluation} \quad (\text{E}) & f_{n+1}^{(0)} & = & f(t_{n+1}, x_{n+1}^{(0)}) \end{array}$$

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector.

$$\begin{array}{llcll} \text{Prediction} & \text{(P)} & x_{n+1}^{(0)} & = & x_n + h\, f_n & \text{(AB1)} \\ \text{Evaluation} & \text{(E)} & f_{n+1}^{(0)} & = & f(t_{n+1}, x_{n+1}^{(0)}) & \\ \text{Correction} & \text{(C)} & x_{n+1}^{(1)} & = & x_n + \dfrac{h}{2}(f_n + f_{n+1}^{(0)}) & \text{(AM2)} \end{array}$$

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector.

$$
\begin{array}{llll}
\text{Prediction} & \text{(P)} & x_{n+1}^{(0)} = x_n + h\, f_n & \text{(AB1)} \\
\text{Evaluation} & \text{(E)} & f_{n+1}^{(0)} = f(t_{n+1}, x_{n+1}^{(0)}) & \\
\text{Correction} & \text{(C)} & x_{n+1}^{(1)} = x_n + \frac{h}{2}(f_n + f_{n+1}^{(0)}) & \text{(AM2)} \\
\text{Evaluation} & \text{(E)} & f_{n+1}^{(1)} = f(t_{n+1}, x_{n+1}^{(1)}) &
\end{array}
$$

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector.

$$
\begin{array}{llrcll}
\text{Prediction} & \text{(P)} & x_{n+1}^{(0)} & = & x_n + h\, f_n & \text{(AB1)} \\
\text{Evaluation} & \text{(E)} & f_{n+1}^{(0)} & = & f(t_{n+1}, x_{n+1}^{(0)}) & \\
\text{Correction} & \text{(C)} & x_{n+1}^{(1)} & = & x_n + \dfrac{h}{2}(f_n + f_{n+1}^{(0)}) & \text{(AM2)} \\
\text{Evaluation} & \text{(E)} & f_{n+1}^{(1)} & = & f(t_{n+1}, x_{n+1}^{(1)}) &
\end{array}
$$

* We can repeat this process, e.g., PECECE, or PE(CE)^k.

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector.

$$
\begin{array}{llrcll}
\text{Prediction} & \text{(P)} & x_{n+1}^{(0)} & = & x_n + h\, f_n & \text{(AB1)} \\
\text{Evaluation} & \text{(E)} & f_{n+1}^{(0)} & = & f(t_{n+1}, x_{n+1}^{(0)}) & \\
\text{Correction} & \text{(C)} & x_{n+1}^{(1)} & = & x_n + \dfrac{h}{2}(f_n + f_{n+1}^{(0)}) & \text{(AM2)} \\
\text{Evaluation} & \text{(E)} & f_{n+1}^{(1)} & = & f(t_{n+1}, x_{n+1}^{(1)}) &
\end{array}
$$

* We can repeat this process, e.g., PECECE, or PE(CE)^k.
* If the process is taken to covergence, we could obtain the same result as using the implicit (corrector) formula alone, i.e., solving the implicit equation of the corrector *exactly*.

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector.

Prediction	(P)	$x_{n+1}^{(0)}$	$=$	$x_n + h\, f_n$	(AB1)
Evaluation	(E)	$f_{n+1}^{(0)}$	$=$	$f(t_{n+1}, x_{n+1}^{(0)})$	
Correction	(C)	$x_{n+1}^{(1)}$	$=$	$x_n + \frac{h}{2}(f_n + f_{n+1}^{(0)})$	(AM2)
Evaluation	(E)	$f_{n+1}^{(1)}$	$=$	$f(t_{n+1}, x_{n+1}^{(1)})$	

* We can repeat this process, e.g., PECECE, or PE(CE)^k.
* If the process is taken to covergence, we could obtain the same result as using the implicit (corrector) formula alone, i.e., solving the implicit equation of the corrector *exactly*.
* Generally, one or two CE steps give a substantially better accuracy (over the predicted x_{n+1}).

Predictor-corrector method for solving $\dot{x} = f(t, x)$

Example: Use AB1 as predictor and AM2 as corrector for the ODE, $\dot{x} = 2x - x^2$, with $x(0) = 1$ (analytic solution: $x(t) = 2/(1 + \exp(-2t))$)

Predictor-corrector method for solving $\dot{x} = f(t, x)$

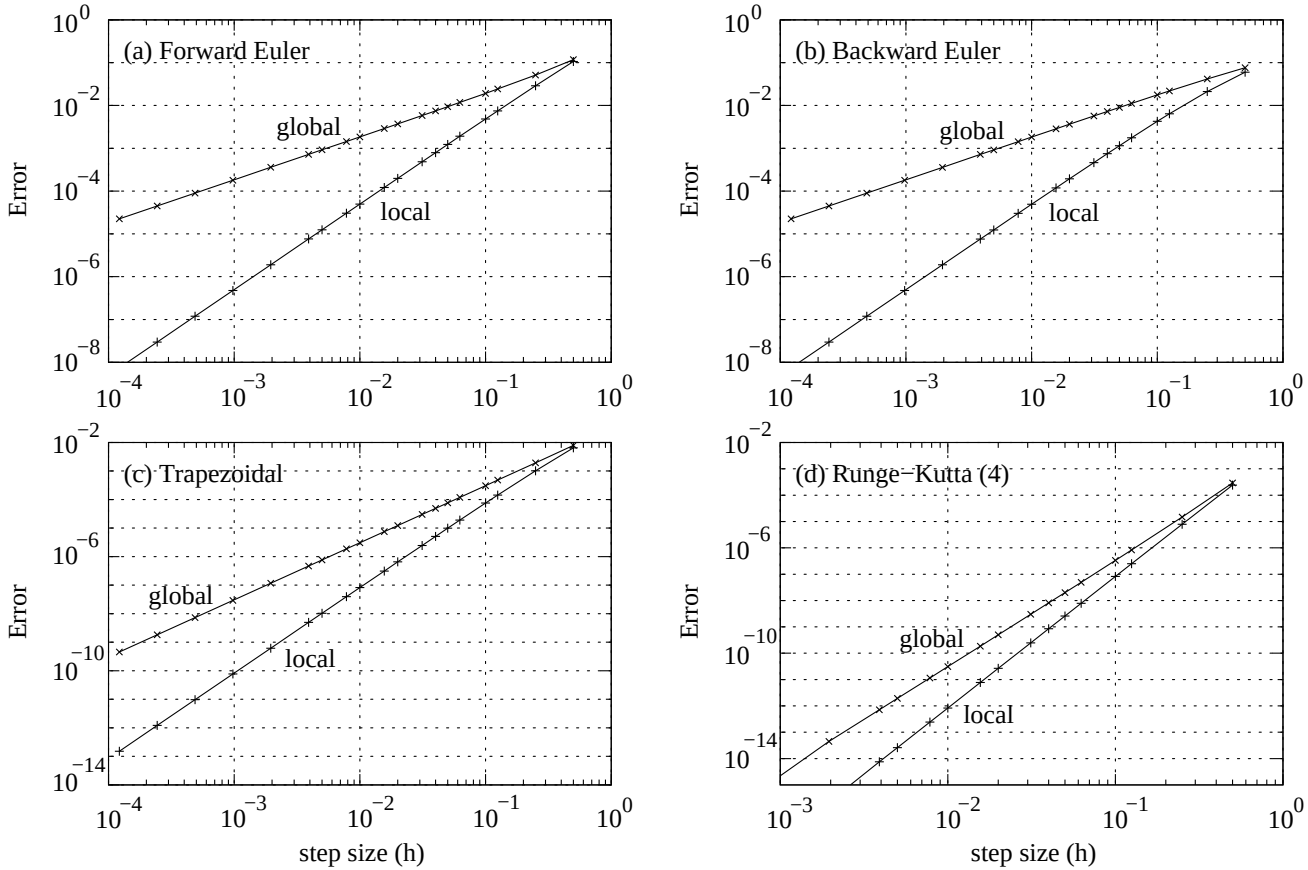
Example: Use AB1 as predictor and AM2 as corrector for the ODE, $\dot{x} = 2x - x^2$, with $x(0) = 1$ (analytic solution: $x(t) = 2/(1 + \exp(-2t))$)

step size (h)		0.05	0.1	0.2
predicted $x(h)$		1.05	1.1	1.2
corrected $x(h)$	(1)	1.04993750	1.09950000	1.19600000
	(2)	1.04993766	1.09950499	1.19615840
	(3)	1.04993766	1.09950494	1.19615219
	(4)	1.04993766	1.09950494	1.19615243
	(5)	1.04993766	1.09950494	1.19615242
$x(h)$ (TRZ)		1.04993766	1.09950494	1.19615242
$x(h)$ (exact)		1.04995837	1.09966799	1.19737532
LTE		2.0719×10^{-5}	1.6306×10^{-4}	1.2229×10^{-3}

Outline

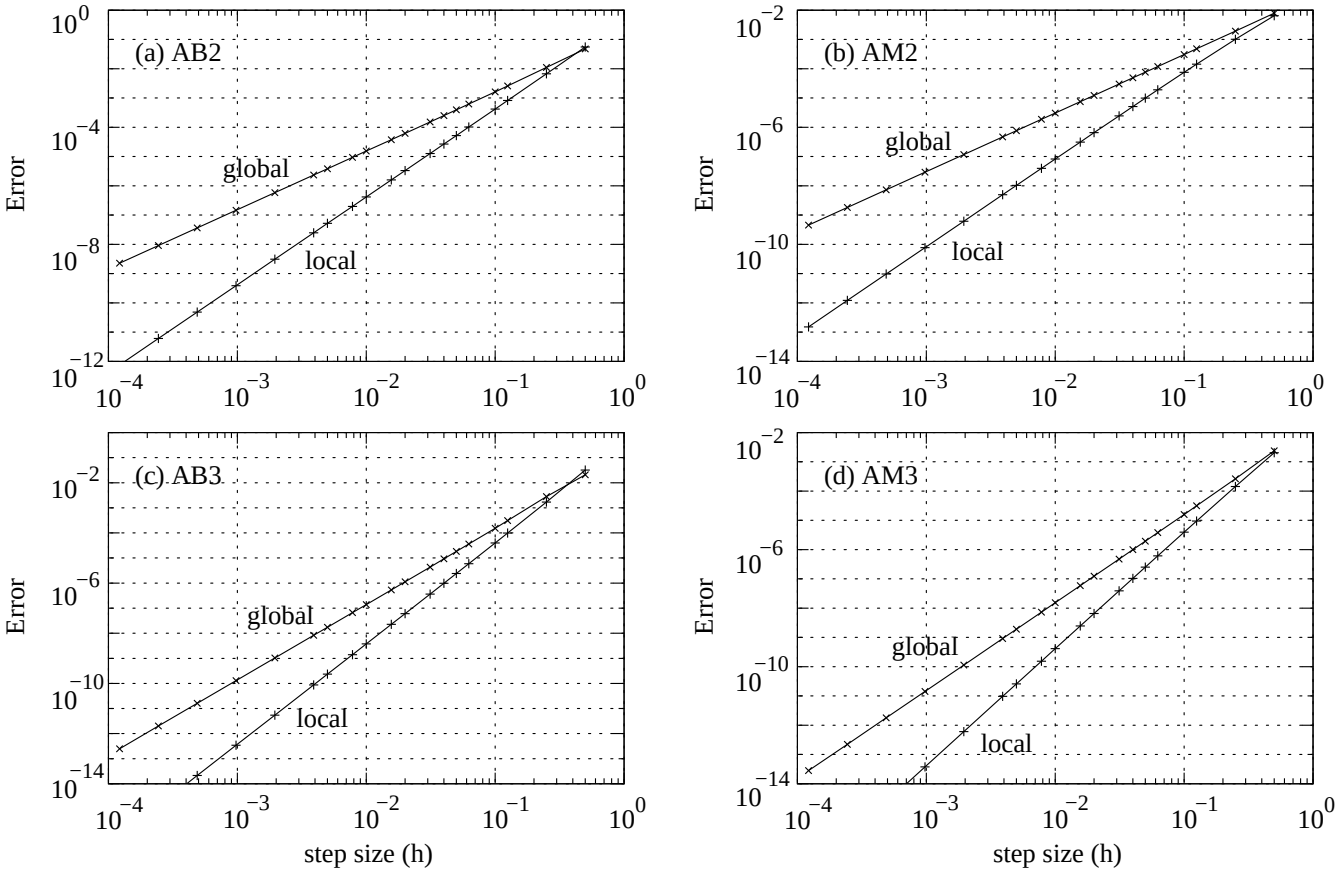
* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Numerical examples: local/global error versus h



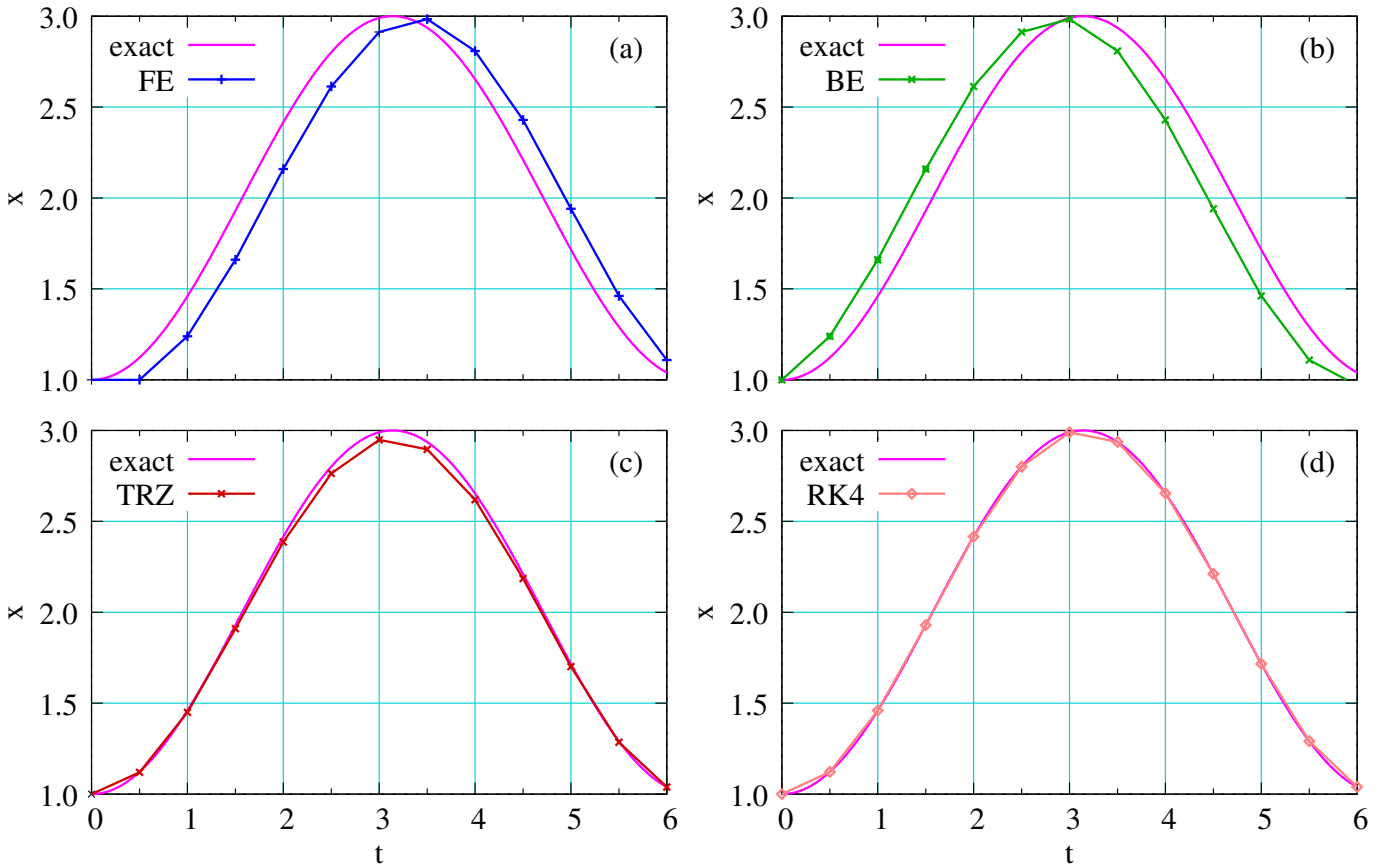
Local and global errors versus step size h for $\dot{x} = -x$, with $x(0) = 1$, for Forward Euler, Backward Euler, Trapezoidal, and Runge-Kutta (4^{th} order) methods. The local error has been computed for the first step, i.e., from $t = 0$ to $t = h$. The global error has been computed at $t = 1$.

Numerical examples: local/global error versus h



Local and global errors versus step size h for $\dot{x} = -x$, with $x(0) = 1$, for AB2, AM2, AB3, and AM3 methods. The local error has been computed for the first step, i.e., from $t = 0$ to $t = h$. The global error has been computed at $t = 1$.

Numerical examples: comparison of methods



Exact and numerical solutions for $\dot{x} = \sin(t)$, with $x(0) = 1$. (a) Forward Euler, (b) Backward Euler, (c) Trapezoidal, and (d) Runge-Kutta (4^{th} order).

Remarks on numerical examples

* For a method of order p,

Remarks on numerical examples

* For a method of order p,
  - the local error is $O(h^{p+1})$, i.e., if h is reduced by 10, the local error goes down by 10^{p+1}.

Remarks on numerical examples

* For a method of order p,
  - the local error is $O(h^{p+1})$, i.e., if h is reduced by 10, the local error goes down by 10^{p+1}.
  - the global error is $O(h^p)$, i.e., if h is reduced by 10, the global error goes down by 10^p.

Remarks on numerical examples

* For a method of order p,
  - the local error is $O(h^{p+1})$, i.e., if h is reduced by 10, the local error goes down by 10^{p+1}.
  - the global error is $O(h^p)$, i.e., if h is reduced by 10, the global error goes down by 10^p.
* A higher-order method is more accurate and therefore allows larger time steps to be taken.

Remarks on numerical examples

* For a method of order p,
  - the local error is $O(h^{p+1})$, i.e., if h is reduced by 10, the local error goes down by 10^{p+1}.
  - the global error is $O(h^p)$, i.e., if h is reduced by 10, the global error goes down by 10^p.
* A higher-order method is more accurate and therefore allows larger time steps to be taken.
* Should we *always* prefer a high-order method?

Remarks on numerical examples

* For a method of order p,
  - the local error is $O(h^{p+1})$, i.e., if h is reduced by 10, the local error goes down by 10^{p+1}.
  - the global error is $O(h^p)$, i.e., if h is reduced by 10, the global error goes down by 10^p.
* A higher-order method is more accurate and therefore allows larger time steps to be taken.
* Should we *always* prefer a high-order method?
  NO. Need to worry about *stability*.

Outline

* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Stability of numerical methods for $\dot{x} = f(t, x)$

* A numerical method generates a sequence of numbers, x_1, $x_2, \cdots$ to approximate the actual values of the solution $x(t_1)$, $x(t_2), \cdots$

Stability of numerical methods for $\dot{x} = f(t, x)$

* A numerical method generates a sequence of numbers, x_1, $x_2, \cdots$ to approximate the actual values of the solution $x(t_1)$, $x(t_2), \cdots$
* At a given time point t_k, there is an error ϵ_k associated with the numerical solution x_k due to algorithmic and round-off errors.

Stability of numerical methods for $\dot{x} = f(t, x)$

* A numerical method generates a sequence of numbers, $x_1, x_2, \cdots$ to approximate the actual values of the solution $x(t_1), x(t_2), \cdots$
* At a given time point t_k, there is an error ϵ_k associated with the numerical solution x_k due to algorithmic and round-off errors.
* If the numerical method causes this error to get amplified in the subsequent time intervals, $|x_n - x(t_n)|$ can become indefinitely large as $n \to \infty$, and the method is said to be *unstable*.

Stability of numerical methods for $\dot{x} = f(t, x)$

* A numerical method generates a sequence of numbers, x_1, $x_2, \cdots$ to approximate the actual values of the solution $x(t_1)$, $x(t_2), \cdots$
* At a given time point t_k, there is an error ϵ_k associated with the numerical solution x_k due to algorithmic and round-off errors.
* If the numerical method causes this error to get amplified in the subsequent time intervals, $|x_n - x(t_n)|$ can become indefinitely large as $n \to \infty$, and the method is said to be *unstable*.
* Need to consider two cases:

Stability of numerical methods for $\dot{x} = f(t, x)$

* A numerical method generates a sequence of numbers, x_1, $x_2, \cdots$ to approximate the actual values of the solution $x(t_1)$, $x(t_2), \cdots$
* At a given time point t_k, there is an error ϵ_k associated with the numerical solution x_k due to algorithmic and round-off errors.
* If the numerical method causes this error to get amplified in the subsequent time intervals, $|x_n - x(t_n)|$ can become indefinitely large as $n \to \infty$, and the method is said to be *unstable*.
* Need to consider two cases:
  - stability for small h ($h \to 0$)

Stability of numerical methods for $\dot{x} = f(t, x)$

* A numerical method generates a sequence of numbers, $x_1, x_2, \cdots$ to approximate the actual values of the solution $x(t_1), x(t_2), \cdots$
* At a given time point t_k, there is an error ϵ_k associated with the numerical solution x_k due to algorithmic and round-off errors.
* If the numerical method causes this error to get amplified in the subsequent time intervals, $|x_n - x(t_n)|$ can become indefinitely large as $n \to \infty$, and the method is said to be *unstable*.
* Need to consider two cases:
  - stability for small h ($h \to 0$)
  - stability for large h

Stability for small h

* If small discrepancies due to a slightly different initial condition, algorithmic errors, or round-off errors lead to correspondingly small changes in the computed solution, then the method is said to be stable (for small h).

Stability for small h

* If small discrepancies due to a slightly different initial condition, algorithmic errors, or round-off errors lead to correspondingly small changes in the computed solution, then the method is said to be stable (for small h).
* A method that is not stable in the above sense is of no practical use because errors are always introduced in solving an ODE numerically, which will cause an unstable method to "blow up" at some point of time.

Stability for small h

* If small discrepancies due to a slightly different initial condition, algorithmic errors, or round-off errors lead to correspondingly small changes in the computed solution, then the method is said to be stable (for small h).
* A method that is not stable in the above sense is of no practical use because errors are always introduced in solving an ODE numerically, which will cause an unstable method to "blow up" at some point of time.
* Runge-Kutta methods are stable.

Stability for small h

* If small discrepancies due to a slightly different initial condition, algorithmic errors, or round-off errors lead to correspondingly small changes in the computed solution, then the method is said to be stable (for small h).
* A method that is not stable in the above sense is of no practical use because errors are always introduced in solving an ODE numerically, which will cause an unstable method to "blow up" at some point of time.
* Runge-Kutta methods are stable.
* Linear multi-step methods (LMMs) can be unstable.

Stability for small h: LMMs

Consider a linear multi-step method of order p,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

Stability for small h: LMMs

Consider a linear multi-step method of order p,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

* Perturb the starting values; let Δ be the largest perturbation.

Stability for small h: LMMs

Consider a linear multi-step method of order p,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

* Perturb the starting values; let Δ be the largest perturbation.
* Perturb the recipe for evaluating x_{n+1} as,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) + \delta_n \, .$$

Stability for small h: LMMs

Consider a linear multi-step method of order p,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \, .$$

* Perturb the starting values; let Δ be the largest perturbation.
* Perturb the recipe for evaluating x_{n+1} as,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) + \delta_n \, .$$

* If the difference between the original numerical solution (x_n^{orig}) and the numerical solution of the perturbed problem (x_n^{new}) is such that,

$$\max |x_n^{\text{orig}} - x_n^{\text{new}}| \leq S \max(|\Delta|, \max |\delta_n|) \, ,$$

then the method is called *zero-stable* or *D-stable* (after Dahlquist) or simply *stable*.

Stability for small h: LMMs

Consider two explicit second-order LMMs [2]:

Stability for small h: LMMs

Consider two explicit second-order LMMs [2]:

(a) AB2: $x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$

Stability for small h: LMMs

Consider two explicit second-order LMMs [2]:

(a) AB2: $x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$

(b) New2: $x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$

Stability for small h: LMMs

Consider two explicit second-order LMMs [2]:

(a) AB2: $x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$

(b) New2: $x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$

* Both of these methods satisfy the exactness constraints (discussed earlier).

Stability for small h: LMMs

Consider two explicit second-order LMMs [2]:

(a) AB2: $x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$

(b) New2: $x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$

* Both of these methods satisfy the exactness constraints (discussed earlier).
* Apply the two methods to $\dot{x} = -x$, $x(0) = 1$.

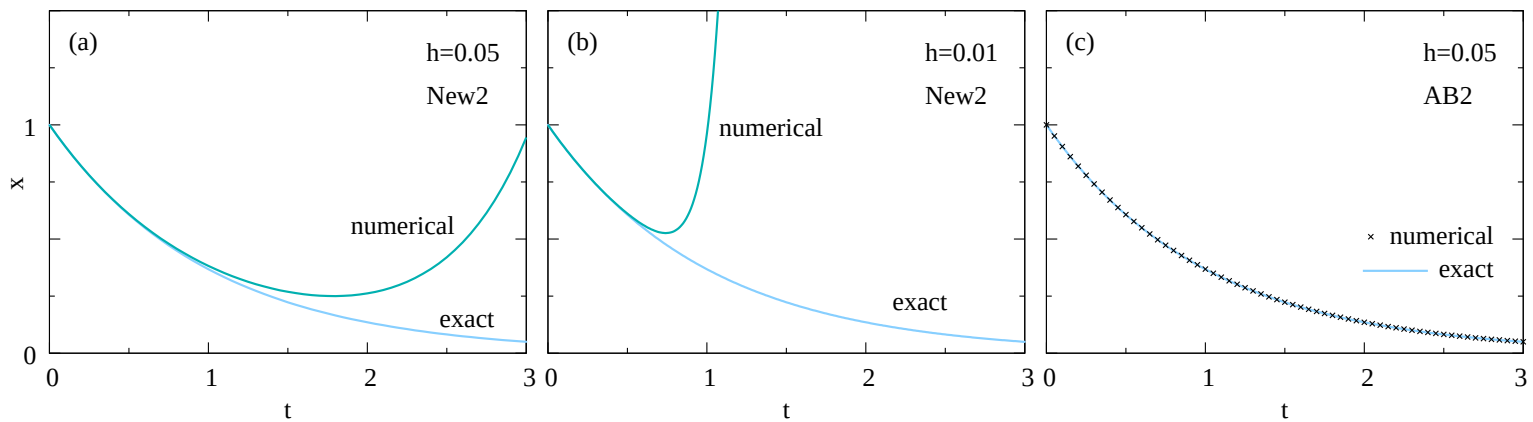
Stability for small h: LMMs

Consider two explicit second-order LMMs [2]:

(a) AB2: $x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$

(b) New2: $x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$

* Both of these methods satisfy the exactness constraints (discussed earlier).
* Apply the two methods to $\dot{x} = -x$, $x(0) = 1$.



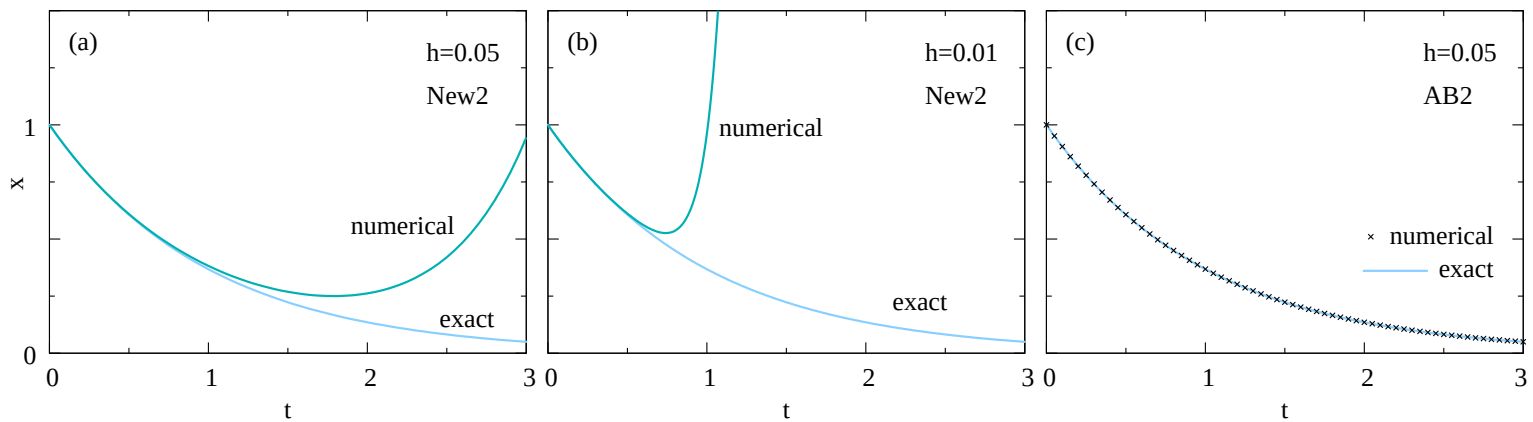
Stability for small h: LMMs

Consider two explicit second-order LMMs [2]:

(a) AB2: $x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$

(b) New2: $x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$

* Both of these methods satisfy the exactness constraints (discussed earlier).
* Apply the two methods to $\dot{x} = -x$, $x(0) = 1$.



* New2 is unstable while AB2 is stable.

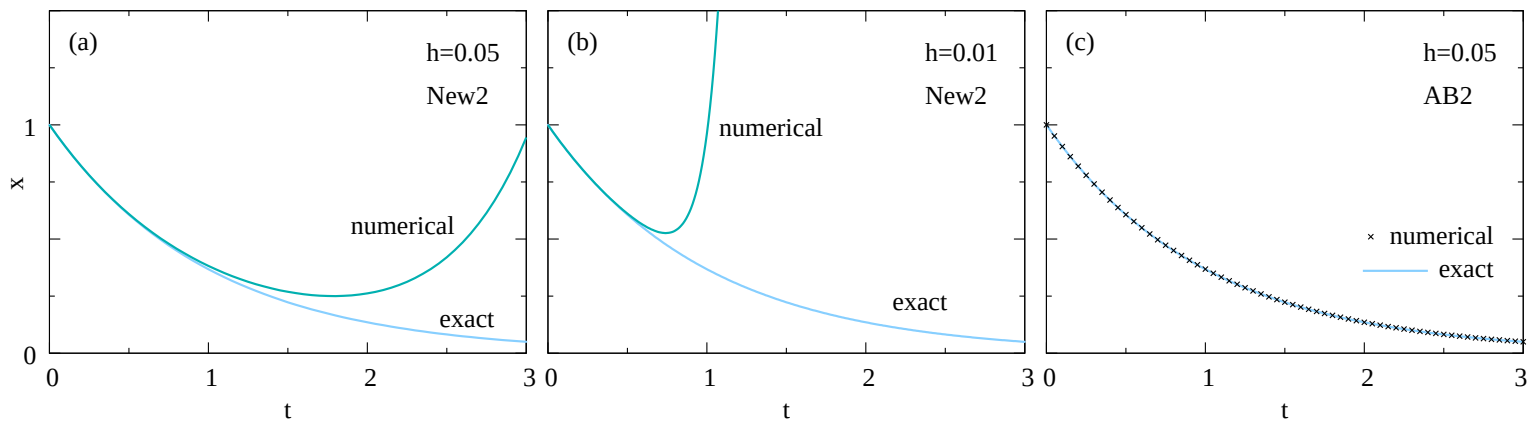
Stability for small h: LMMs

Consider two explicit second-order LMMs [2]:

(a) AB2: $x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$

(b) New2: $x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$

* Both of these methods satisfy the exactness constraints (discussed earlier).
* Apply the two methods to $\dot{x} = -x$, $x(0) = 1$.



* New2 is unstable while AB2 is stable.
* Why are these two methods so different?

Difference equations: general solution

Consider a linear difference equation with constant (real) coefficients,

$$a_k x_{m+k} + a_{k-1} x_{m+k-1} + \cdots + a_1 x_{m+1} + a_0 x_m = b\,. \tag{11}$$

Difference equations: general solution

Consider a linear difference equation with constant (real) coefficients,

$$a_k x_{m+k} + a_{k-1} x_{m+k-1} + \cdots + a_1 x_{m+1} + a_0 x_m = b\,. \tag{11}$$

* The solution has two parts: (a) homogeneous, (b) particular.

Difference equations: general solution

Consider a linear difference equation with constant (real) coefficients,

$$a_k x_{m+k} + a_{k-1} x_{m+k-1} + \cdots + a_1 x_{m+1} + a_0 x_m = b\,. \tag{11}$$

* The solution has two parts: (a) homogeneous, (b) particular.
* For the homogeneous part (i.e., $b = 0$), we seek a solution of the form $x_i^{(h)} = z^i$, substitute it in Eq. 11, and obtain

$$z^m \left[a_k z^k + a_{k-1} z^{k-1} + \cdots + a_1 z + a_0 \right] = 0\,. \tag{12}$$

Difference equations: general solution

Consider a linear difference equation with constant (real) coefficients,

$$a_k x_{m+k} + a_{k-1} x_{m+k-1} + \cdots + a_1 x_{m+1} + a_0 x_m = b\,. \tag{11}$$

* The solution has two parts: (a) homogeneous, (b) particular.
* For the homogeneous part (i.e., $b = 0$), we seek a solution of the form $x_i^{(h)} = z^i$, substitute it in Eq. 11, and obtain

$$z^m \left[a_k z^k + a_{k-1} z^{k-1} + \cdots + a_1 z + a_0 \right] = 0\,. \tag{12}$$

* Eq. 12 has the trivial solution $z = 0$, or

$$a_k z^k + a_{k-1} z^{k-1} + \cdots + a_1 z + a_0 = 0\,. \tag{13}$$

Difference equations: general solution

Consider a linear difference equation with constant (real) coefficients,

$$a_k x_{m+k} + a_{k-1} x_{m+k-1} + \cdots + a_1 x_{m+1} + a_0 x_m = b\,. \tag{11}$$

* The solution has two parts: (a) homogeneous, (b) particular.
* For the homogeneous part (i.e., $b = 0$), we seek a solution of the form $x_i^{(h)} = z^i$, substitute it in Eq. 11, and obtain

$$z^m \left[a_k z^k + a_{k-1} z^{k-1} + \cdots + a_1 z + a_0 \right] = 0\,. \tag{12}$$

* Eq. 12 has the trivial solution $z = 0$, or

$$a_k z^k + a_{k-1} z^{k-1} + \cdots + a_1 z + a_0 = 0\,. \tag{13}$$

* Eq. 13 is called the *characteristic equation* of the difference equation (Eq. 11).

Difference equations: general solution

* If the roots of the characteristic equation, z_1, $z_2, \cdots$, z_k, are distinct, then the general solution of Eq. 12 is given by

$$x_i^{(h)} = c_1 z_1^i + c_2 z_2^i + \cdots + c_k z_k^i \,. \tag{14}$$

Difference equations: general solution

* If the roots of the characteristic equation, $z_1, z_2, \cdots, z_k$, are distinct, then the general solution of Eq. 12 is given by

$$x_i^{(h)} = c_1 z_1^i + c_2 z_2^i + \cdots + c_k z_k^i \,. \tag{14}$$

* If the roots are not distinct, then the general form for $x_i^{(h)}$ gets modified. As an example, if $z_1, z_2, \cdots, z_l$ are identical, and the other roots are distinct, then $x_i^{(h)}$ is given by,

$$x_i^{(h)} = (c_1 + c_2 n + \cdots + c_l n^{l-1}) z_1^i + c_{l+1} z_{l+1}^i + \cdots + c_k z_k^i \,. \tag{15}$$

Difference equations: general solution

* If the roots of the characteristic equation, z_1, $z_2, \cdots$, z_k, are distinct, then the general solution of Eq. 12 is given by

$$x_i^{(h)} = c_1 z_1^i + c_2 z_2^i + \cdots + c_k z_k^i \,. \tag{14}$$

* If the roots are not distinct, then the general form for $x_i^{(h)}$ gets modified. As an example, if z_1, $z_2, \cdots$, z_l are identical, and the other roots are distinct, then $x_i^{(h)}$ is given by,

$$x_i^{(h)} = (c_1 + c_2 n + \cdots + c_l n^{l-1}) z_1^i + c_{l+1} z_{l+1}^i + \cdots + c_k z_k^i \,. \tag{15}$$

* The complete solution of Eq. 11 is then given by,

$$x_i = x_i^{(h)} + x_i^{(p)} \,, \tag{16}$$

where $x_i^{(p)}$ is a particular solution. The constants c_1, c_2, etc. can be determined from the initial condition(s), i.e., the starting values in the sequence $\{x_i\}$.

* Consider the characteristic equation

$$\sum_{i=-1}^{k} \alpha_i \, z^{1-i} = 0 \,, \tag{17}$$

associated with the LMM,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \,. \tag{18}$$

Let the roots of Eq. 17 be z_1, $z_2, \cdots$, z_k. If $|z_i| \leq 1$ for all i, and all roots with magnitude 1 are simple, then the LMM is said to satisfy the *root condition*.

Stability of LMMs

* Consider the characteristic equation

$$\sum_{i=-1}^{k} \alpha_i \, z^{1-i} = 0 \,, \tag{17}$$

associated with the LMM,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \,. \tag{18}$$

Let the roots of Eq. 17 be z_1, $z_2, \cdots$, z_k. If $|z_i| \leq 1$ for all i, and all roots with magnitude 1 are simple, then the LMM is said to satisfy the *root condition*.

* If an LMM satisfies the root condition, and if the only root of the associated characteristic equation with magnitude one is equal to 1, then the LMM is said to be *strongly stable*.

* Consider the characteristic equation

$$\sum_{i=-1}^{k} \alpha_i \, z^{1-i} = 0 \,, \tag{17}$$

associated with the LMM,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \,. \tag{18}$$

Let the roots of Eq. 17 be z_1, $z_2, \cdots$, z_k. If $|z_i| \leq 1$ for all i, and all roots with magnitude 1 are simple, then the LMM is said to satisfy the *root condition*.

* If an LMM satisfies the root condition, and if the only root of the associated characteristic equation with magnitude one is equal to 1, then the LMM is said to be *strongly stable*.
* If an LMM satisfies the root condition, and if more than one (distinct) roots of the associated characteristic equation have magnitude one, then the LMM is said to be *weakly stable*.

* Consider the characteristic equation

$$\sum_{i=-1}^{k} \alpha_i \, z^{1-i} = 0 \,, \tag{17}$$

associated with the LMM,

$$\sum_{i=-1}^{k} \alpha_i \, x_{n-i} = h \sum_{i=-1}^{k} \beta_i \, f(t_{n-i}, x_{n-i}) \,. \tag{18}$$

Let the roots of Eq. 17 be z_1, $z_2, \cdots$, z_k. If $|z_i| \leq 1$ for all i, and all roots with magnitude 1 are simple, then the LMM is said to satisfy the *root condition*.

* If an LMM satisfies the root condition, and if the only root of the associated characteristic equation with magnitude one is equal to 1, then the LMM is said to be *strongly stable*.
* If an LMM satisfies the root condition, and if more than one (distinct) roots of the associated characteristic equation have magnitude one, then the LMM is said to be *weakly stable*.
* If an LMM does not satisfy the root condition, it is said to be *unstable*.

Coming back to AB2 and New2,

AB2	$x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$ char. eqn.: $z - 1 = 0$. roots: $z_1 = 1$.
New2	$x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$ char. eqn.: $z^2 - 2.1\,z + 1.1 = 0$. roots: $z_1 = 1,\ z_2 = 1.1$.

Coming back to AB2 and New2,

AB2	$x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$ char. eqn.: $z - 1 = 0$. roots: $z_1 = 1$.
New2	$x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$ char. eqn.: $z^2 - 2.1\,z + 1.1 = 0$. roots: $z_1 = 1,\ z_2 = 1.1$.

* AB2 satisfies the root condition; New2 does not.

Coming back to AB2 and New2,

AB2	$x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$ char. eqn.: $z - 1 = 0$. roots: $z_1 = 1$.
New2	$x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$ char. eqn.: $z^2 - 2.1\,z + 1.1 = 0$. roots: $z_1 = 1,\ z_2 = 1.1$.

* AB2 satisfies the root condition; New2 does not.
* For the New2 method, the general solution, $c_1 z_1^n + c_2 z_2^n$, can grow indefinitely since $|z_2| > 1$.

Coming back to AB2 and New2,

AB2	$x_{n+1} = x_n + h\,(1.5\,f_n - 0.5\,f_{n-1})$ char. eqn.: $z - 1 = 0$. roots: $z_1 = 1$.
New2	$x_{n+1} = 2.1\,x_n - 1.1\,x_{n-1} + h\,(0.95\,f_n - 1.05\,f_{n-1})$ char. eqn.: $z^2 - 2.1\,z + 1.1 = 0$. roots: $z_1 = 1,\ z_2 = 1.1$.

* AB2 satisfies the root condition; New2 does not.
* For the New2 method, the general solution, $c_1 z_1^n + c_2 z_2^n$, can grow indefinitely since $|z_2| > 1$.
* Even if c_2 is forced to be zero because of initial conditions, numerical errors can make it non-zero.

Stability for large h

* A numerical method that is unstable even for small values of h (e.g., the "New2" method seen earlier) is practically useless since it is unstable for *any* problem.

Stability for large h

* A numerical method that is unstable even for small values of h (e.g., the "New2" method seen earlier) is practically useless since it is unstable for *any* problem.
* A method that is stable for small h (e.g., the AB2 method) may still be unstable in a different sense, viz., unstable if h exceeds a certain value, say, $h_{\max}$.

Stability for large h

* A numerical method that is unstable even for small values of h (e.g., the "New2" method seen earlier) is practically useless since it is unstable for *any* problem.
* A method that is stable for small h (e.g., the AB2 method) may still be unstable in a different sense, viz., unstable if h exceeds a certain value, say, $h_{\max}$.
* $h_{\max}$ would depend on the ODE being solved. Generally, it is determined for the *test equation*,

$$\dot{x} = \lambda x, \qquad x(0) = 1\,, \tag{19}$$

where λ is a constant, a complex number in general. Eq. 19 is representative of several problems of practical importance, such as RC circuits.

Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

Let λ be real and negative. Consider the Forward Euler method,

$$\begin{aligned} x_{n+1} &= x_n + h\, f(t_n, x_n) \\ &= x_n + h\lambda\, x_n \\ &= x_n\,(1 + h\lambda) \end{aligned} \tag{20}$$

Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

Let λ be real and negative. Consider the Forward Euler method,

$$\begin{aligned} x_{n+1} &= x_n + h\, f(t_n, x_n) \\ &= x_n + h\lambda\, x_n \\ &= x_n\,(1 + h\lambda) \end{aligned} \tag{20}$$

The characteristic equation for this difference equation is,

$$z - (1 + h\lambda) = 0\,, \tag{21}$$

Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

Let λ be real and negative. Consider the Forward Euler method,

$$\begin{aligned} x_{n+1} &= x_n + h\, f(t_n, x_n) \\ &= x_n + h\lambda\, x_n \\ &= x_n\,(1 + h\lambda) \end{aligned} \tag{20}$$

The characteristic equation for this difference equation is,

$$z - (1 + h\lambda) = 0\,, \tag{21}$$

for which the general solution is given by,

$$\begin{aligned} x_i &= c_1\, z_1^i \\ &= (1 + h\lambda)^i\,. \end{aligned} \tag{22}$$

($c_1 = 1$ is required to satisfy the initial condition, $x_0 = 1$.)

Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

* The exact solution is $x(t) = \exp(\lambda t)$ which, for $t_k = kh$ and $|h\lambda| \ll 1$, is

$$\begin{aligned} x(t_k) &= e^{\lambda t_k} = e^{kh\lambda} \\ &\approx (1 + h\lambda)^k \, . \end{aligned}$$

Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

* The exact solution is $x(t) = \exp(\lambda t)$ which, for $t_k = kh$ and $|h\lambda| \ll 1$, is

$$\begin{aligned} x(t_k) &= e^{\lambda t_k} = e^{kh\lambda} \\ &\approx (1 + h\lambda)^k \, . \end{aligned}$$

* Comparing with the numerical solution,

$$x_k = (1 + h\lambda)^k \, ,$$

we see that the numerical solution will approximate the true solution if $h\lambda$ is small.

Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

* The exact solution is $x(t) = \exp(\lambda t)$ which, for $t_k = kh$ and $|h\lambda| \ll 1$, is

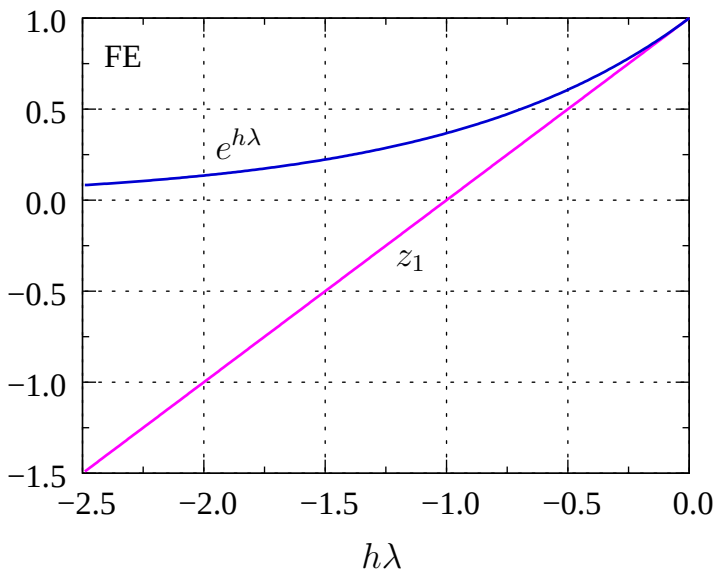
$$\begin{aligned} x(t_k) &= e^{\lambda t_k} = e^{kh\lambda} \\ &\approx (1 + h\lambda)^k \, . \end{aligned}$$

* Comparing with the numerical solution,

$$x_k = (1 + h\lambda)^k \, ,$$

  we see that the numerical solution will approximate the true solution if $h\lambda$ is small.

* As h is increased, $z_1 = 1 + h\lambda$ decreases (since $\lambda < 0$), and for $h\lambda = -2$, z_1 becomes equal to -1 (see figure). Beyond this point, $|z_1| > 1$, and the numerical solution ($x_k = c_1 \, z_1^k$) grows indefinitely with $k \Rightarrow$ instability.



Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

Consider the second-order Adams-Bashforth method. The difference equation is,

$$x_{n+1} = x_n + h\lambda \left[\frac{3}{2} x_n - \frac{1}{2} x_{n-1} \right] .$$

Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

Consider the second-order Adams-Bashforth method. The difference equation is,

$$x_{n+1} = x_n + h\lambda \left[\frac{3}{2} x_n - \frac{1}{2} x_{n-1} \right] .$$

The characteristic equation for this case is,

$$z^2 - \left(1 + \frac{3h\lambda}{2} \right) z + \left(\frac{h\lambda}{2} \right) = 0 ,$$

Stability for large h (for $\dot{x} = \lambda x$, $x(0) = 1$)

Consider the second-order Adams-Bashforth method. The difference equation is,

$$x_{n+1} = x_n + h\lambda \left[\frac{3}{2} x_n - \frac{1}{2} x_{n-1} \right] .$$

The characteristic equation for this case is,

$$z^2 - \left(1 + \frac{3h\lambda}{2} \right) z + \left(\frac{h\lambda}{2} \right) = 0 ,$$

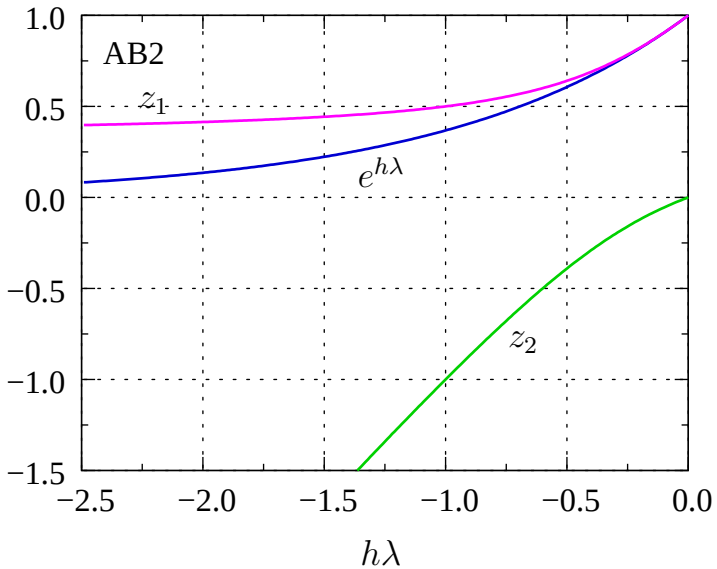
with the roots,

$$z_{1,2} = \frac{1}{2} \left\{ \left(1 + \frac{3h\lambda}{2} \right) \pm \sqrt{\left(1 + \frac{3h\lambda}{2} \right)^2 - 2h\lambda} \right\} ,$$

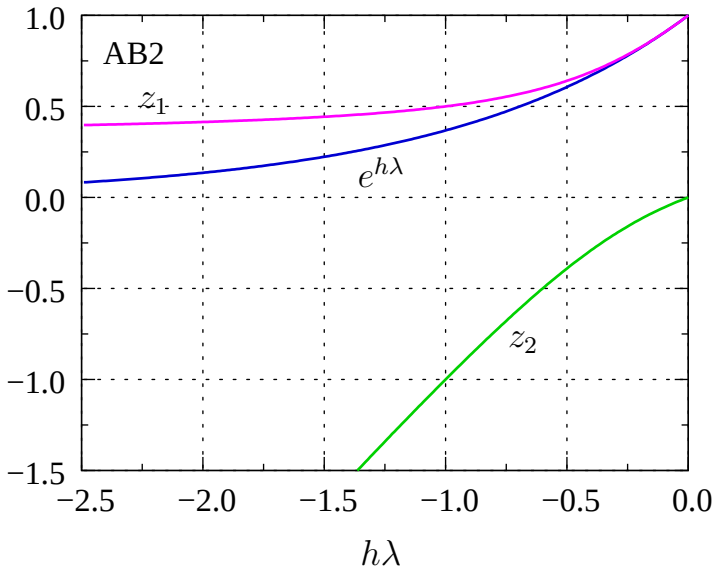
and the general solution,

$$x_i = c_1 z_1^i + c_2 z_2^i .$$

Stability for large h (AB2 method for $\dot{x} = \lambda x$, $x(0) = 1$)

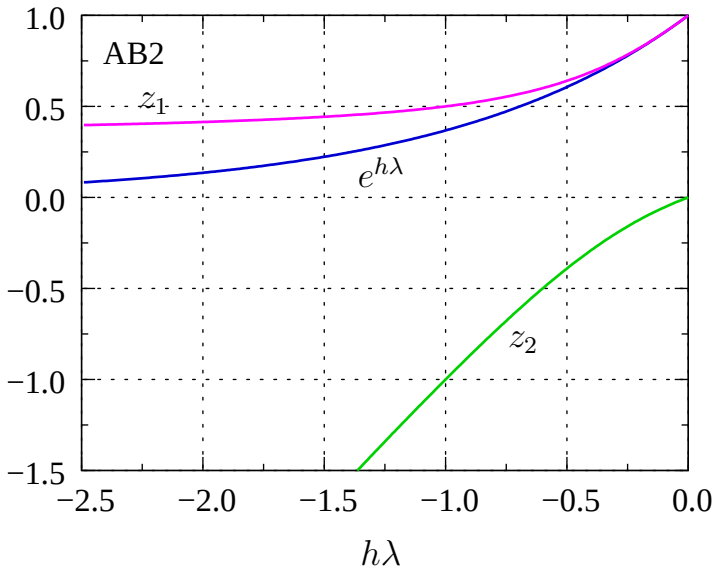


Stability for large h (AB2 method for $\dot{x} = \lambda x$, $x(0) = 1$)



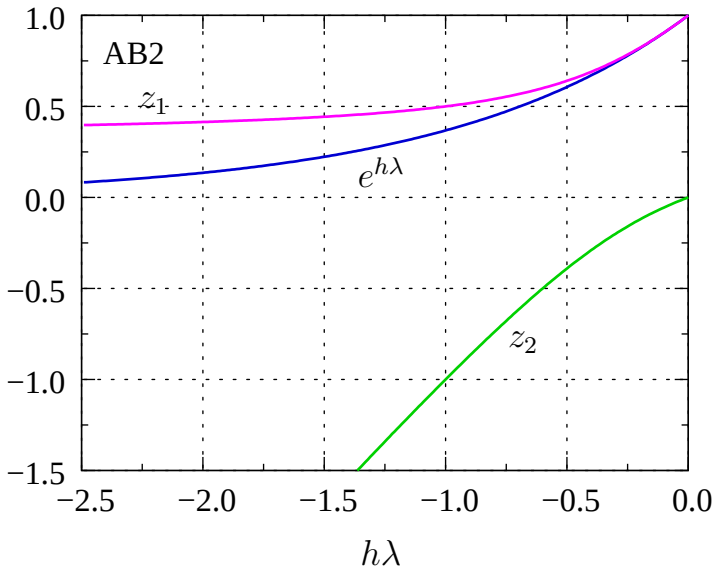
* For small values of $h\lambda$, z_1 represents $e^{h\lambda}$ more closely than in the FE method, as we would expect from a second-order method.

Stability for large h (AB2 method for $\dot{x} = \lambda x$, $x(0) = 1$)



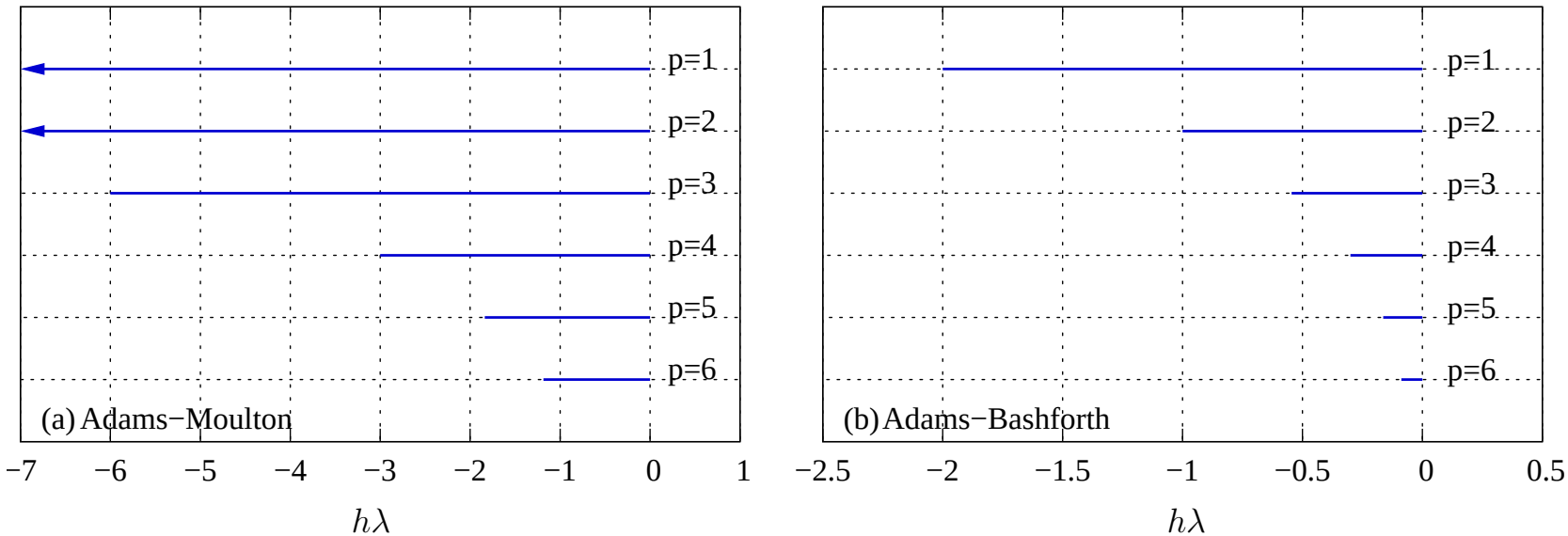
* For small values of $h\lambda$, z_1 represents $e^{h\lambda}$ more closely than in the FE method, as we would expect from a second-order method.
* What is of concern, from the stability angle, is the other root z_2 which starts off at zero, but becomes greater than one in magnitude at $h\lambda = -1$, thus leading to instability.

Stability for large h (AB2 method for $\dot{x} = \lambda x$, $x(0) = 1$)

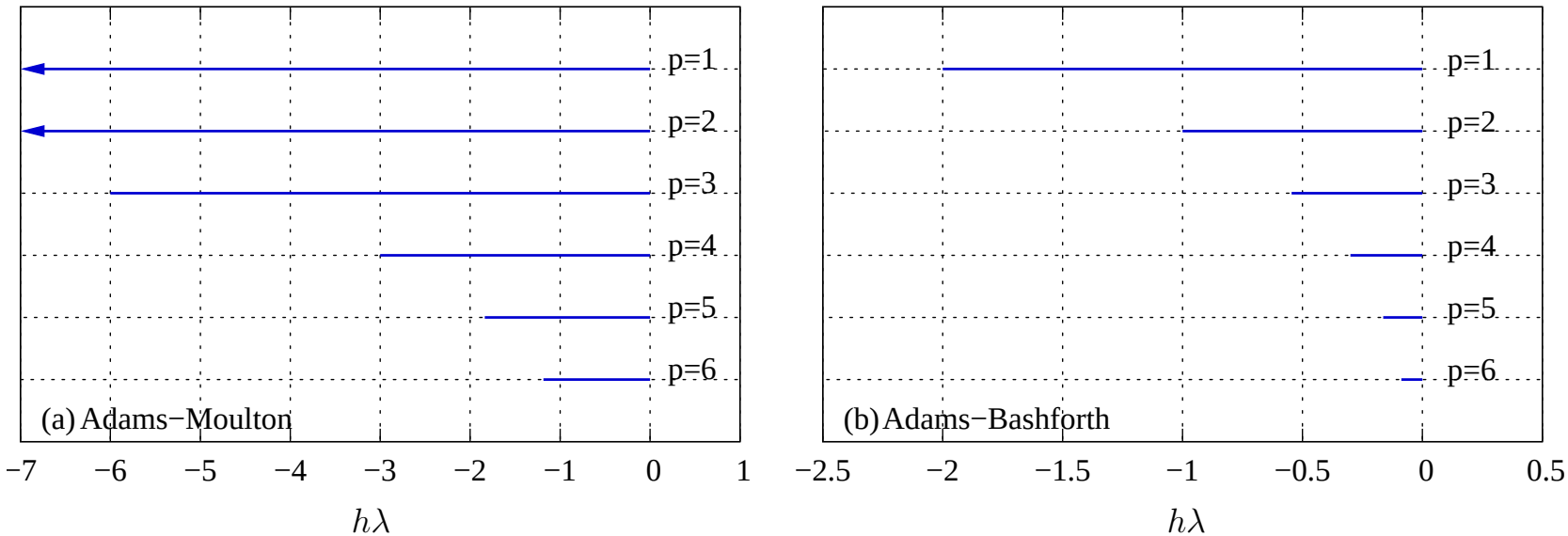


* For small values of $h\lambda$, z_1 represents $e^{h\lambda}$ more closely than in the FE method, as we would expect from a second-order method.
* What is of concern, from the stability angle, is the other root z_2 which starts off at zero, but becomes greater than one in magnitude at $h\lambda = -1$, thus leading to instability.
* This root is not *required* to represent $e^{h\lambda}$, and in that sense, it is a *parasitic* or *spurious* root. In contrast, the root z_1, which approximates $e^{h\lambda}$, is called the *principal* root.

Stability for large h: AB and AM methods for $\dot{x} = \lambda x$, $\lambda < 0$

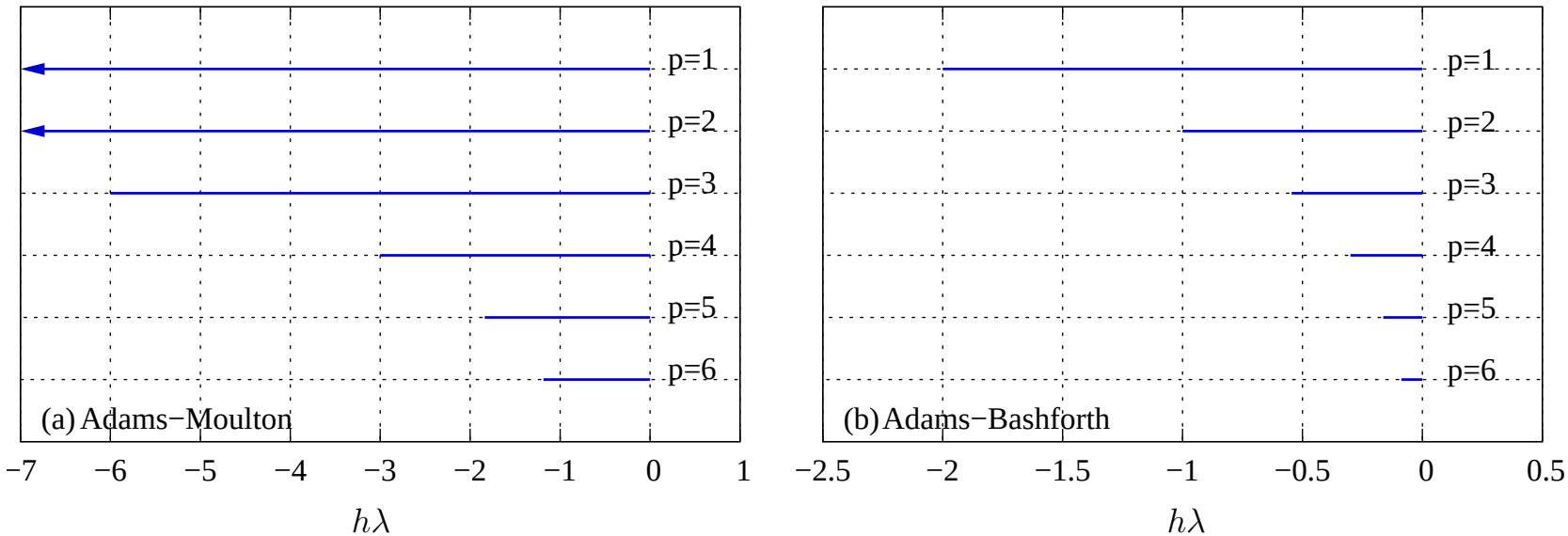


Stability for large h: AB and AM methods for $\dot{x} = \lambda x$, $\lambda < 0$



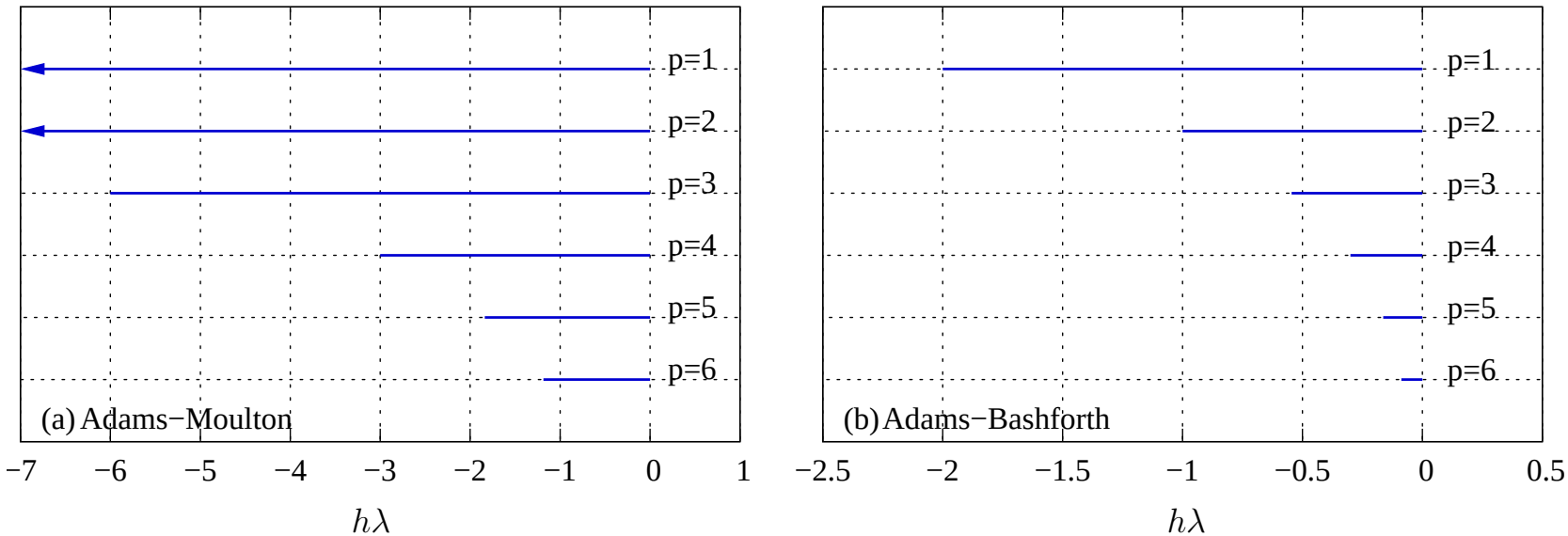
* The AM methods, for the same order, are more stable than the AB methods.

Stability for large h: AB and AM methods for $\dot{x} = \lambda x$, $\lambda < 0$



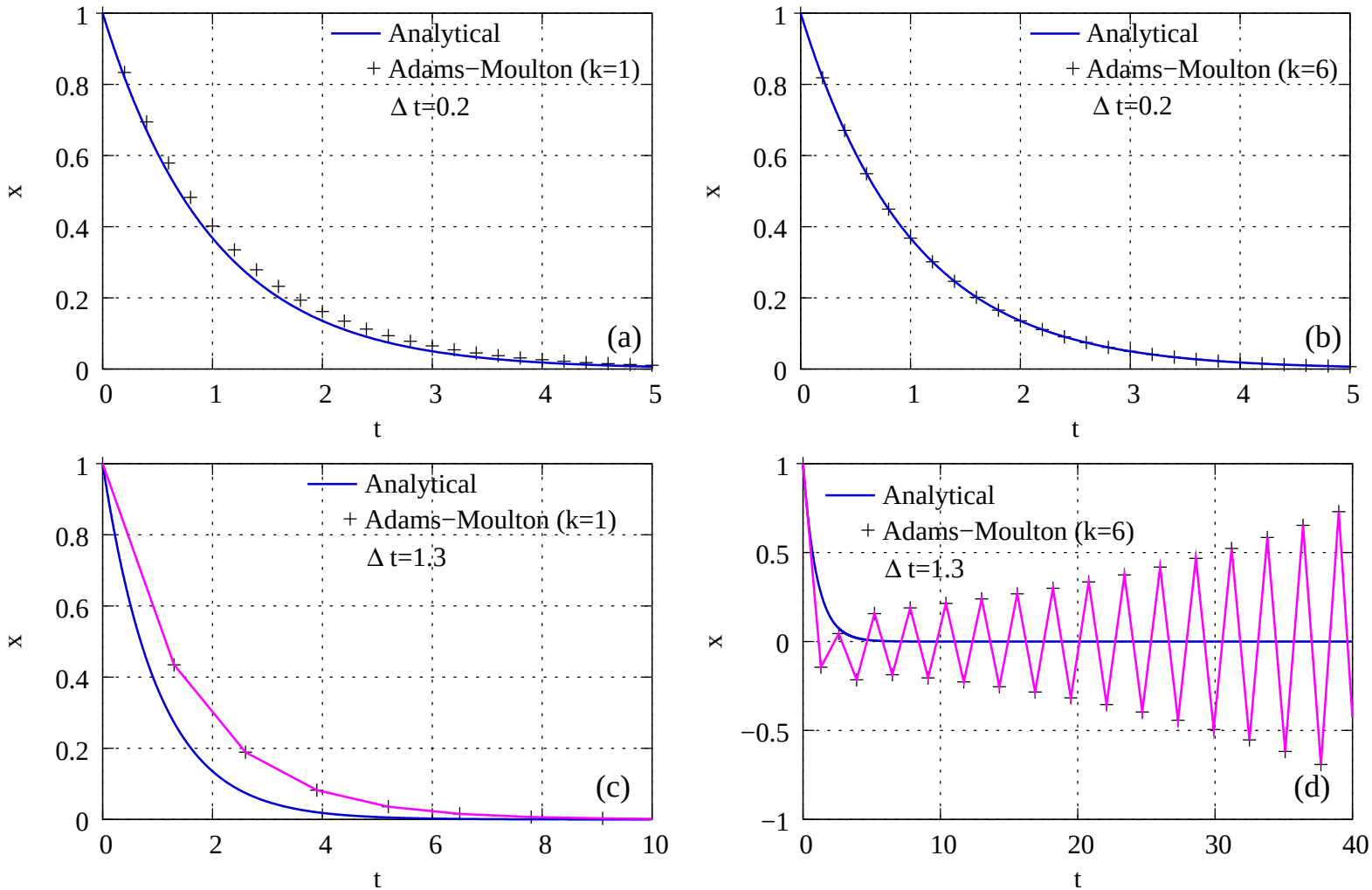
* The AM methods, for the same order, are more stable than the AB methods.
* The AM methods of order 1 and 2 (the BE and TRZ methods) are stable for all values of $h\lambda$.

Stability for large h: AB and AM methods for $\dot{x} = \lambda x$, $\lambda < 0$

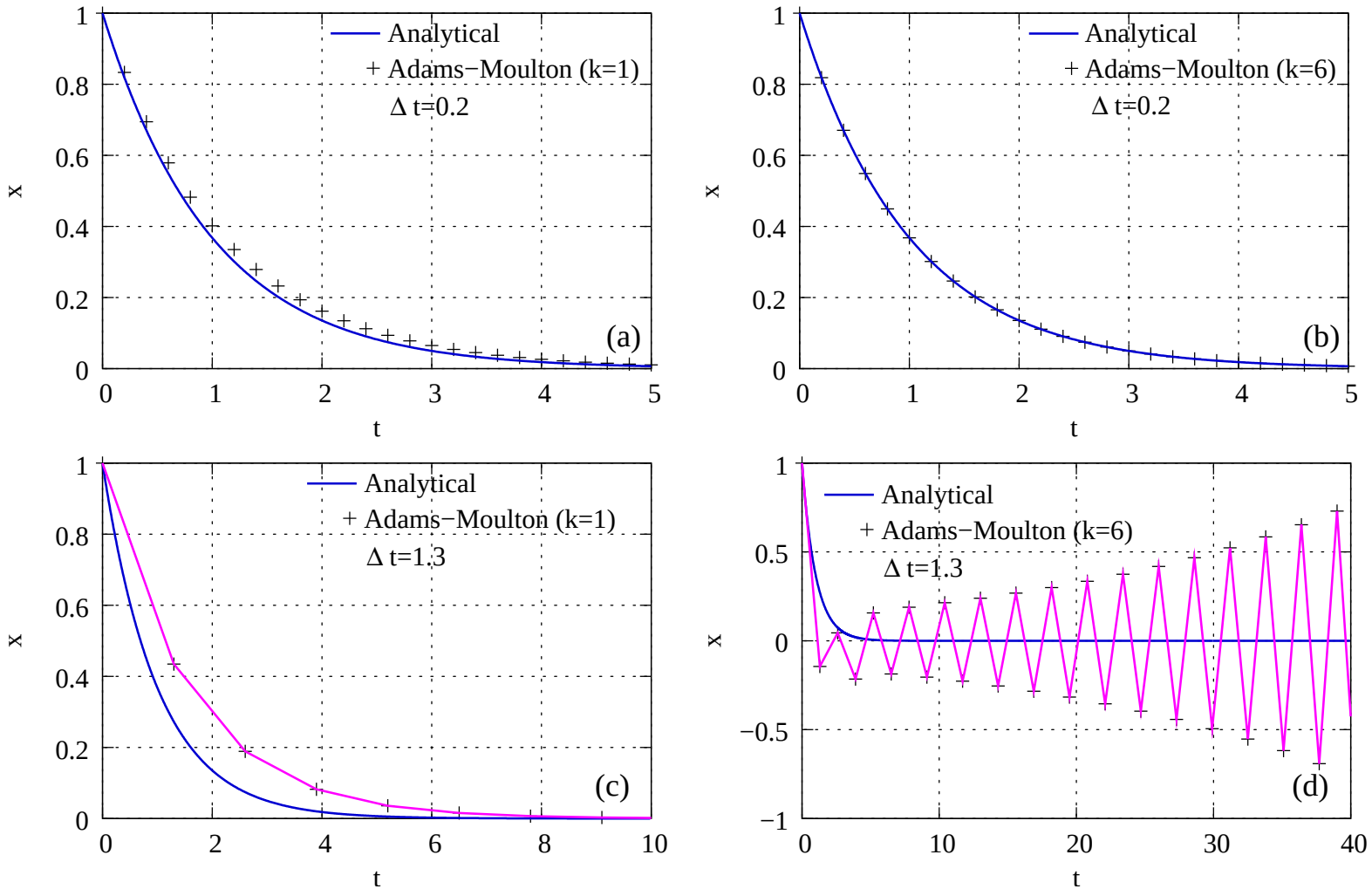


* The AM methods, for the same order, are more stable than the AB methods.
* The AM methods of order 1 and 2 (the BE and TRZ methods) are stable for all values of $h\lambda$.
* As the order increases, the range of stability becomes smaller for both AB and AM methods. This explains why higher-order methods are not used in circuit simulation.

Stability for large h: AM1 and AM6 methods for $\dot{x} = -x$, $x(0) = 1$

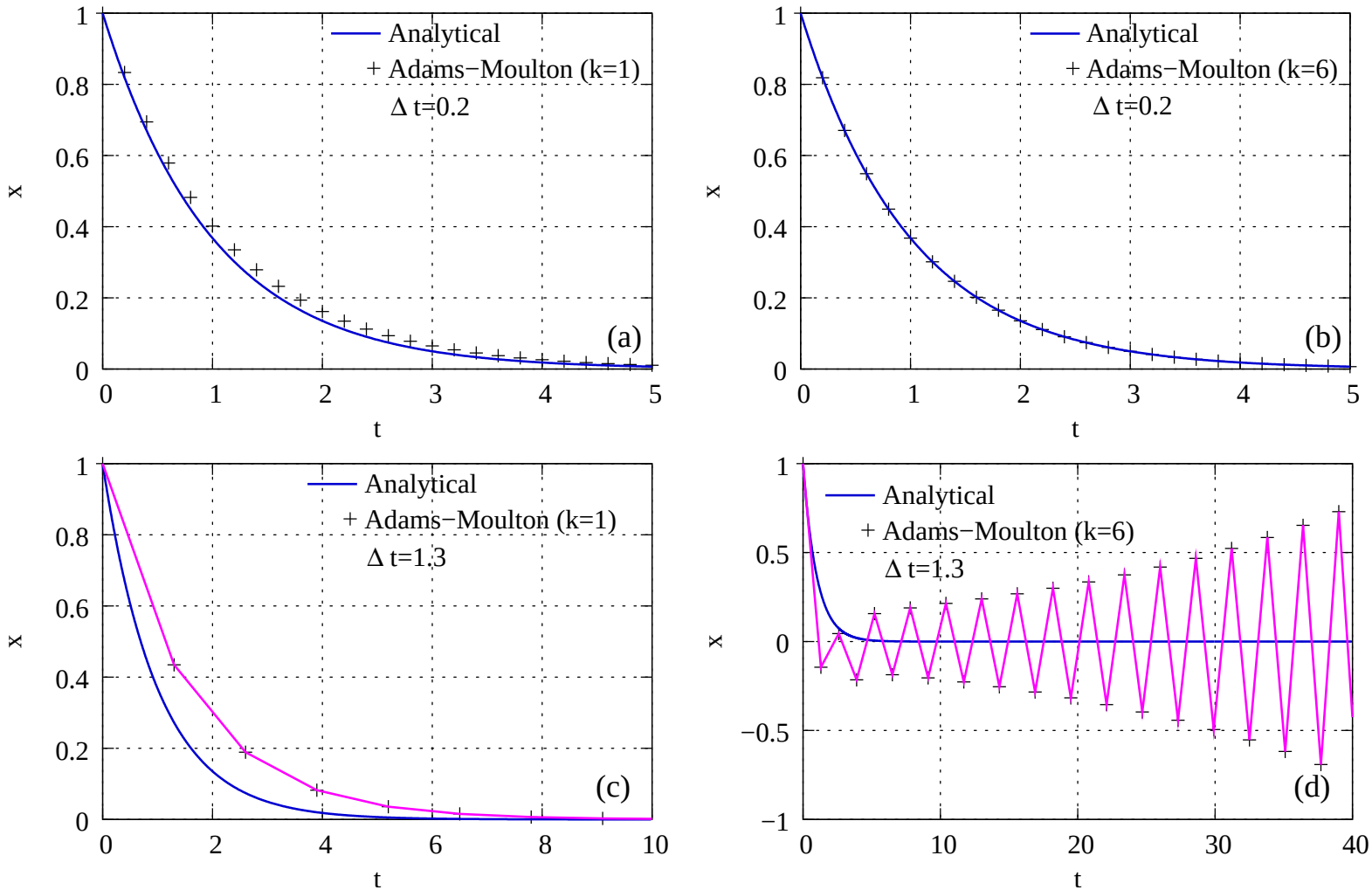


Stability for large h: AM1 and AM6 methods for $\dot{x} = -x$, $x(0) = 1$



* AM6 is more accurate than AM1 (upper figures).

Stability for large h: AM1 and AM6 methods for $\dot{x} = -x$, $x(0) = 1$



* AM6 is more accurate than AM1 (upper figures).
* For $\Delta t = 1.3$, AM1 is stable, but AM6 is not (lower figures).

Consider the 2×2 system of ODEs, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, which may be written in the expanded form,

$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,, \end{aligned} \tag{23}$$

with $x_1(0) = x_1^0$ and $x_2(0) = x_2^0$.

Consider the 2×2 system of ODEs, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, which may be written in the expanded form,

$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,, \end{aligned} \tag{23}$$

with $x_1(0) = x_1^0$ and $x_2(0) = x_2^0$.

Let λ_1, λ_2 be the eigenvalues (assumed to be distinct) of $\mathbf{A}$, and $\mathbf{S_1}$, $\mathbf{S_2}$ be the corresponding eigenvectors.

Consider the 2×2 system of ODEs, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, which may be written in the expanded form,

$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,, \end{aligned} \tag{23}$$

with $x_1(0) = x_1^0$ and $x_2(0) = x_2^0$.

Let λ_1, λ_2 be the eigenvalues (assumed to be distinct) of $\mathbf{A}$, and $\mathbf{S_1}$, $\mathbf{S_2}$ be the corresponding eigenvectors.

Eq. 23 can be re-written in a diagonalized form,

$$\begin{aligned} \dot{y_1} &= \lambda_1 y_1 \,, \\ \dot{y_2} &= \lambda_2 y_2 \,. \end{aligned} \tag{24}$$

Consider the 2×2 system of ODEs, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, which may be written in the expanded form,

$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,, \end{aligned} \tag{23}$$

with $x_1(0) = x_1^0$ and $x_2(0) = x_2^0$.

Let λ_1, λ_2 be the eigenvalues (assumed to be distinct) of $\mathbf{A}$, and $\mathbf{S_1}$, $\mathbf{S_2}$ be the corresponding eigenvectors.

Eq. 23 can be re-written in a diagonalized form,

$$\begin{aligned} \dot{y_1} &= \lambda_1 y_1 \,, \\ \dot{y_2} &= \lambda_2 y_2 \,. \end{aligned} \tag{24}$$

The new variables y_1 and y_2 are given by,

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \mathbf{S_1} & \mathbf{S_2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} . \tag{25}$$

Consider the 2×2 system of ODEs, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, which may be written in the expanded form,

$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,, \end{aligned} \tag{23}$$

with $x_1(0) = x_1^0$ and $x_2(0) = x_2^0$.

Let λ_1, λ_2 be the eigenvalues (assumed to be distinct) of $\mathbf{A}$, and $\mathbf{S_1}$, $\mathbf{S_2}$ be the corresponding eigenvectors.

Eq. 23 can be re-written in a diagonalized form,

$$\begin{aligned} \dot{y_1} &= \lambda_1 y_1 \,, \\ \dot{y_2} &= \lambda_2 y_2 \,. \end{aligned} \tag{24}$$

The new variables y_1 and y_2 are given by,

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \mathbf{S_1} & \mathbf{S_2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} . \tag{25}$$

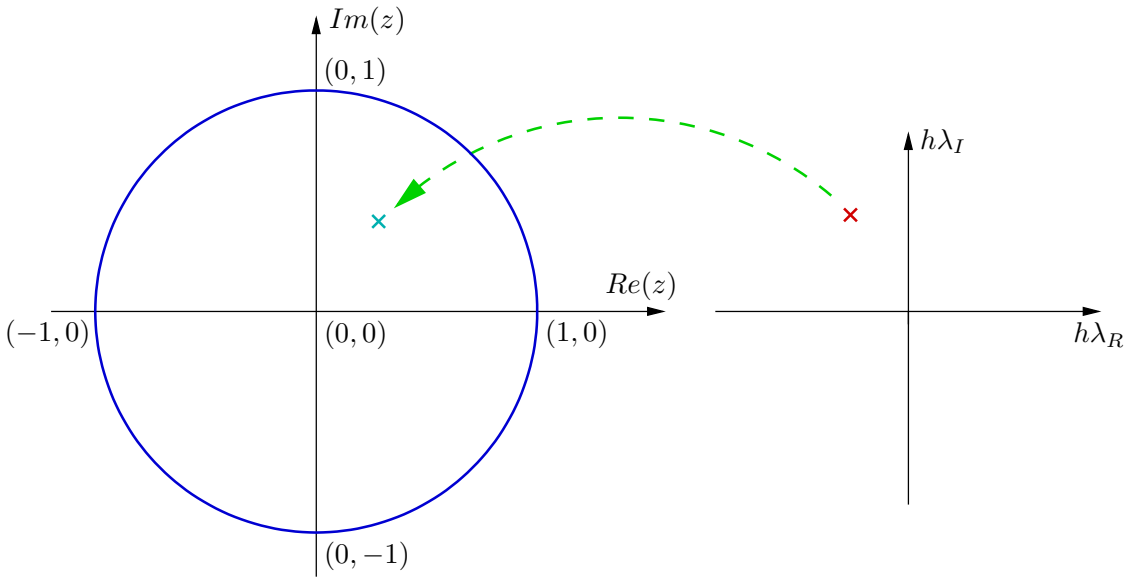
Solving the system of ODEs, Eq. 23, is thus equivalent to solving two separate ODEs. Since λ_1, λ_2 are generally complex, we are interested in solving $\dot{x} = \lambda x$ when λ is complex.

Outline

* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

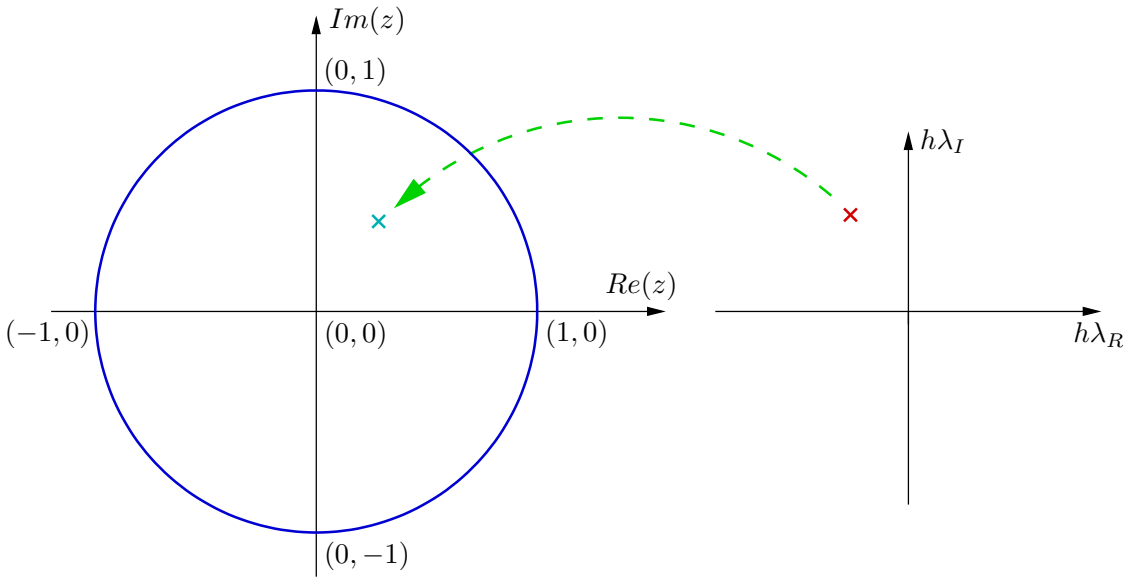
Stability for large h ($\dot{x} = \lambda x$, complex λ)

* A method is said to be *absolutely stable* (with respect to the test equation) for a given $h\lambda$ with $\mathrm{Re}(\lambda) < 0$ if all the roots of the characteristic equation lie inside the unit circle in the $h\lambda$ plane. The set of all such $h\lambda$ is called the *region of absolute stability* of the method [5].



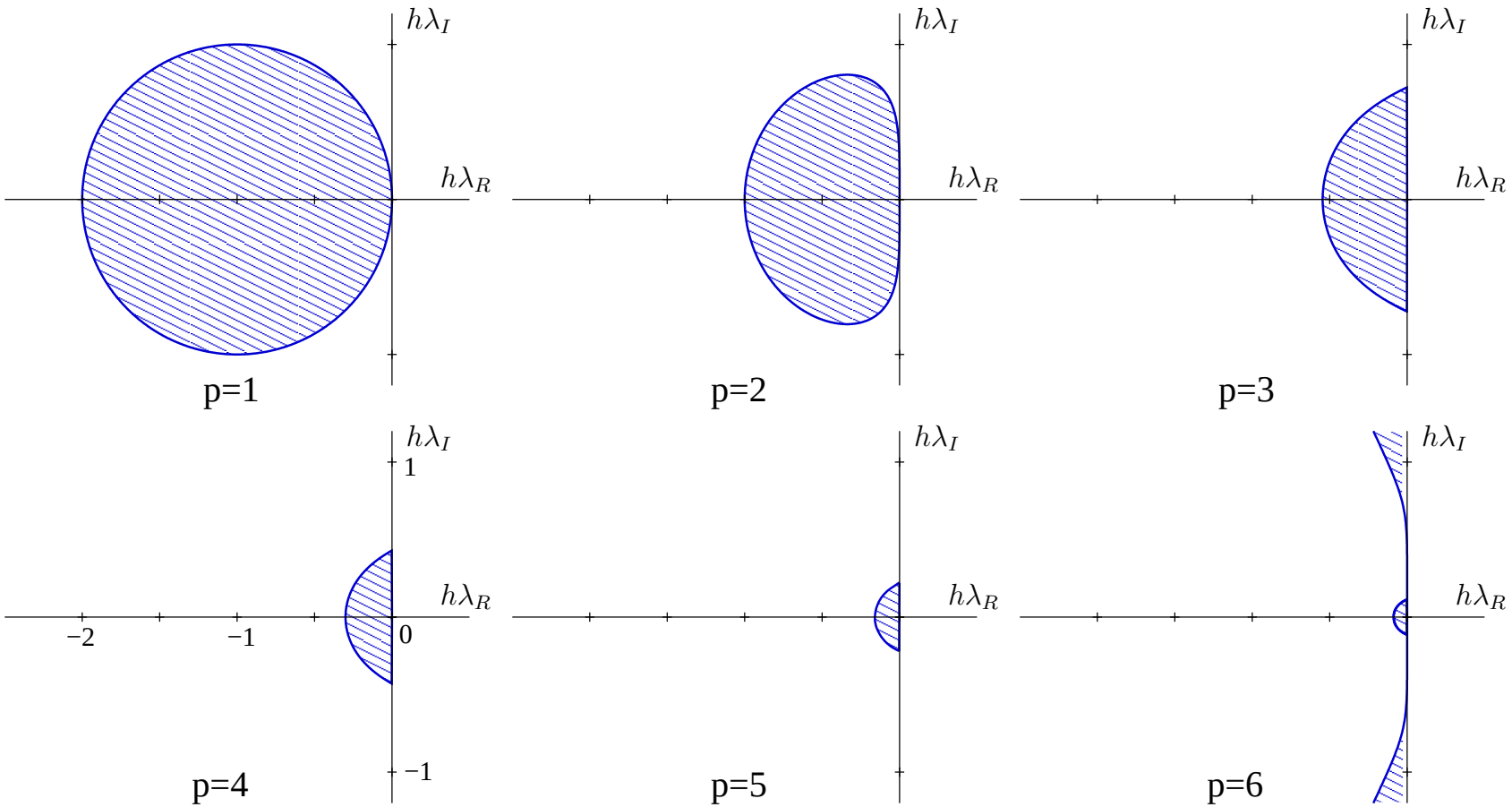
Stability for large h ($\dot{x} = \lambda x$, complex λ)

* A method is said to be *absolutely stable* (with respect to the test equation) for a given $h\lambda$ with $\mathrm{Re}(\lambda) < 0$ if all the roots of the characteristic equation lie inside the unit circle in the $h\lambda$ plane. The set of all such $h\lambda$ is called the *region of absolute stability* of the method [5].

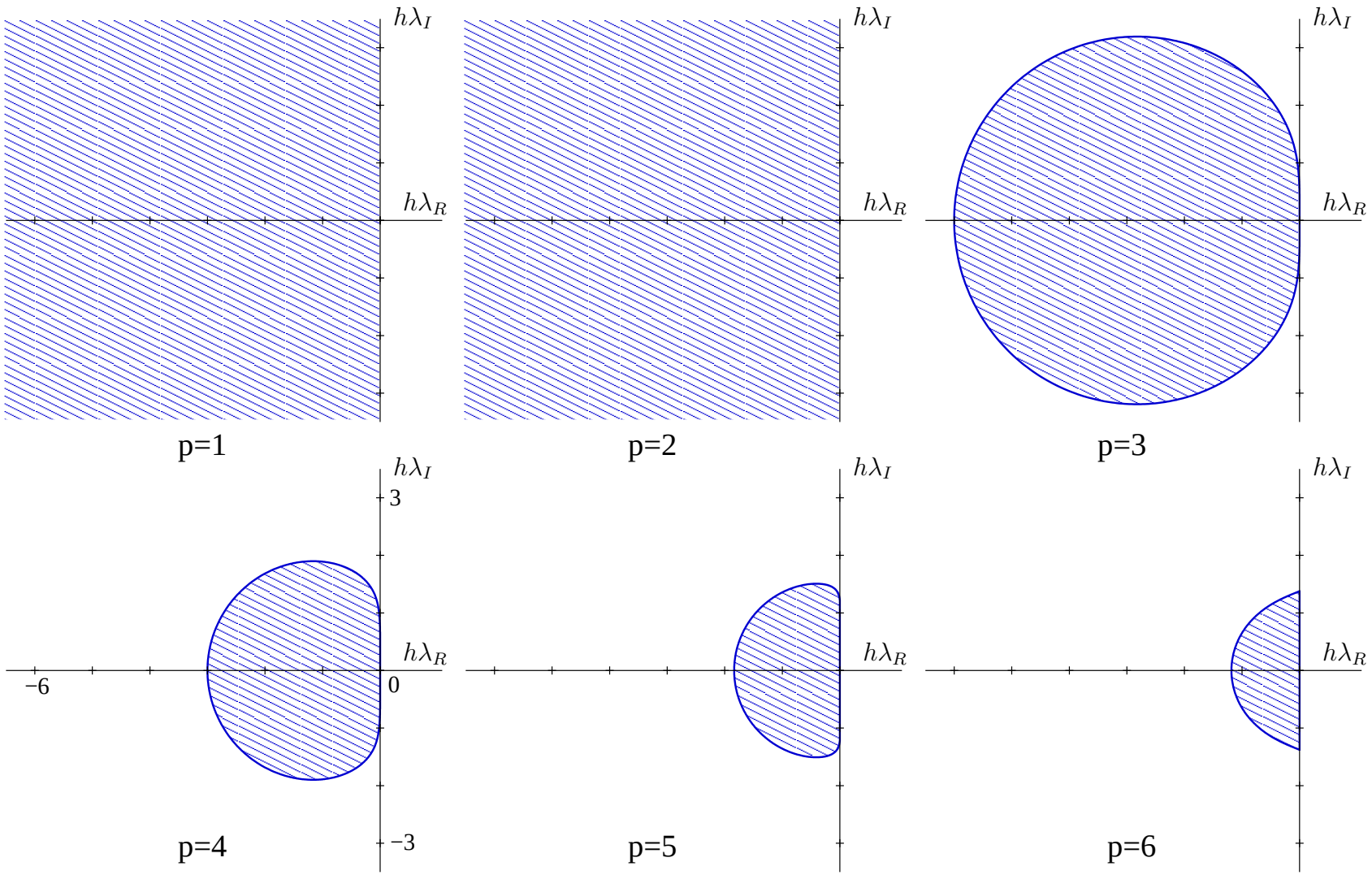


* Methods that are stable for all λ with $\mathrm{Re}(\lambda) < 0$ are called *A-stable*.

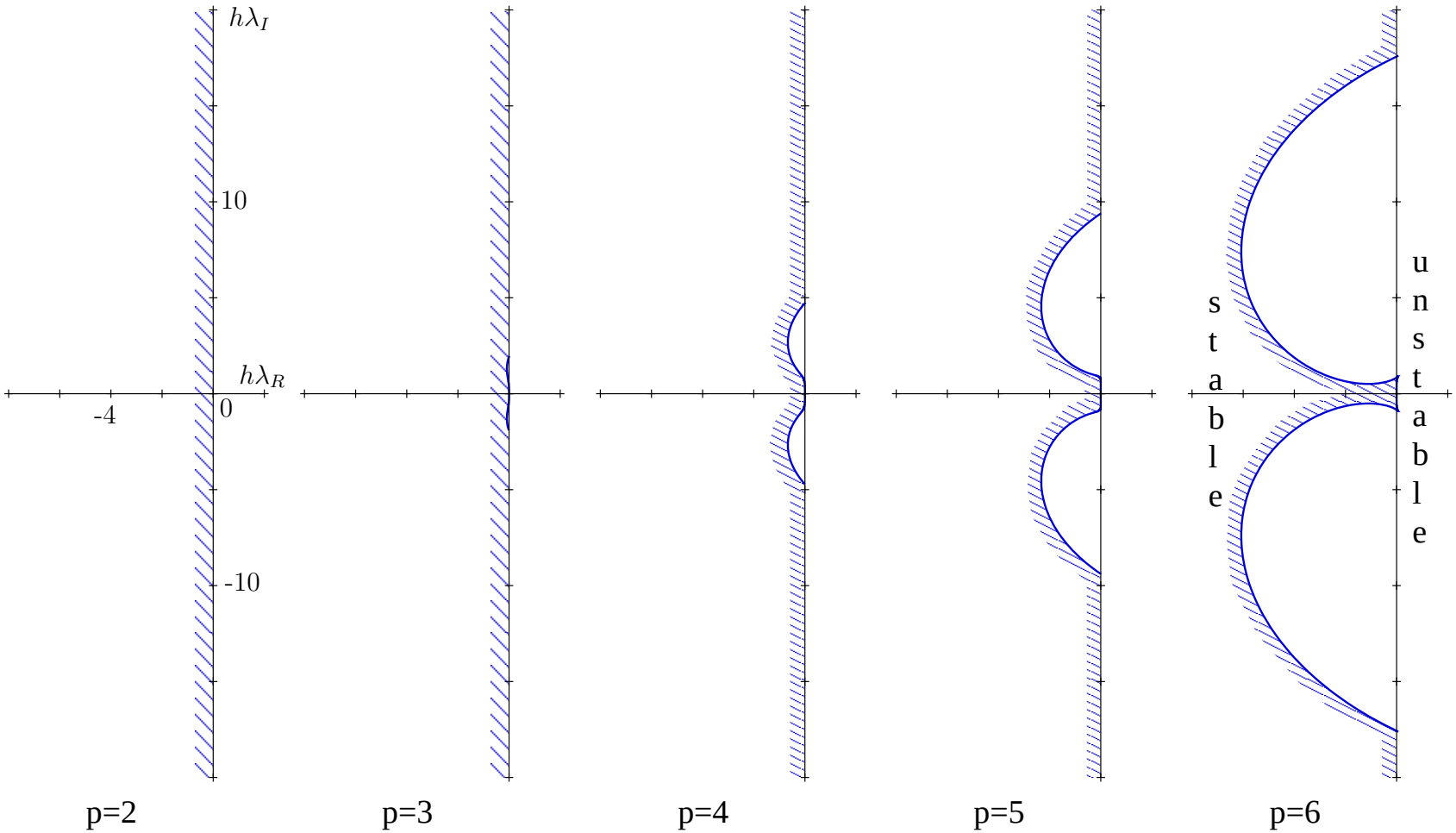
Region of stability for AB methods



Region of stability for AM methods



Region of stability for BDF methods



Stability for large h ($\dot{x} = \lambda x$, complex λ)

* The AM1 (Backward Euler), AM2 (Trapezoidal), and second-order BDF methods are A-stable; other methods are *conditionally* stable.

Stability for large h ($\dot{x} = \lambda x$, complex λ)

* The AM1 (Backward Euler), AM2 (Trapezoidal), and second-order BDF methods are A-stable; other methods are *conditionally* stable.
* The region of absolute stability for each method shrinks significantly as the order increases.

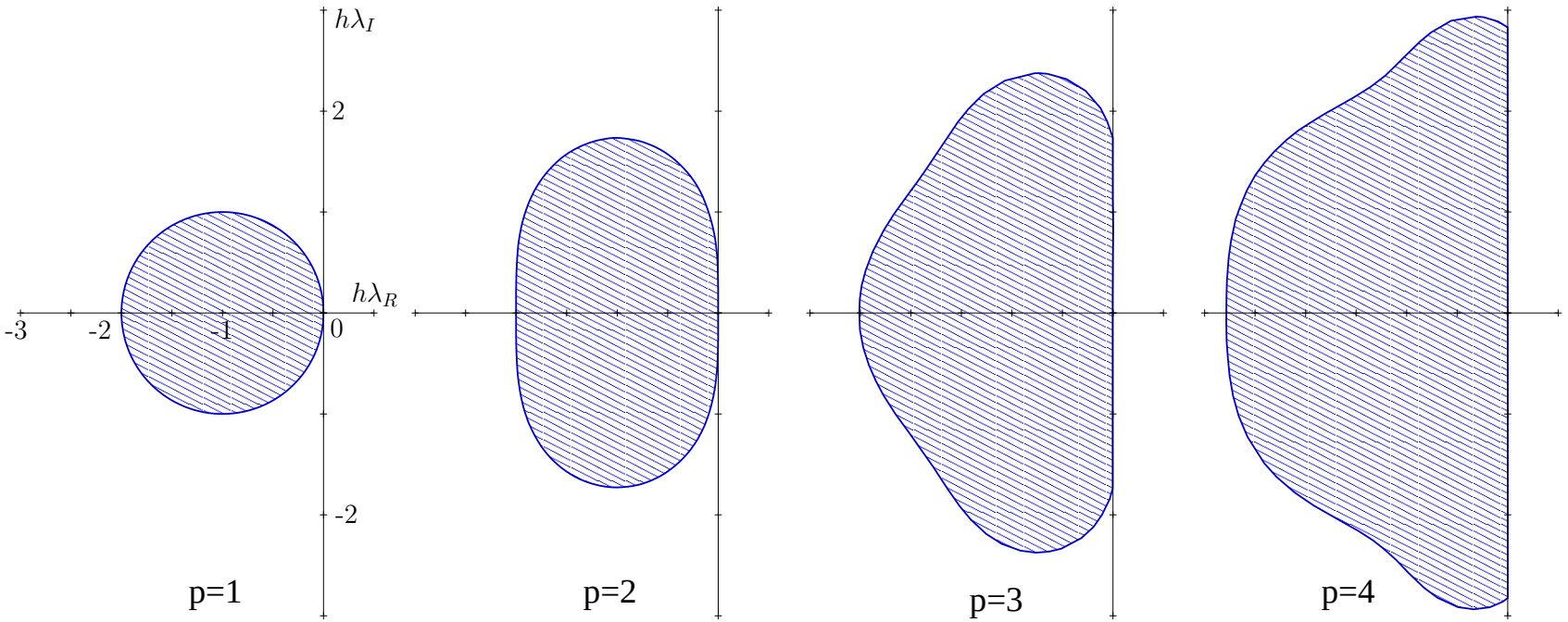
Stability for large h ($\dot{x} = \lambda x$, complex λ)

* The AM1 (Backward Euler), AM2 (Trapezoidal), and second-order BDF methods are A-stable; other methods are *conditionally* stable.
* The region of absolute stability for each method shrinks significantly as the order increases.
* For purely real values of λ, the BDF methods (up to order six) are unconditionally stable, while the AB and AM (except AM1 and AM2) methods are conditionally stable.

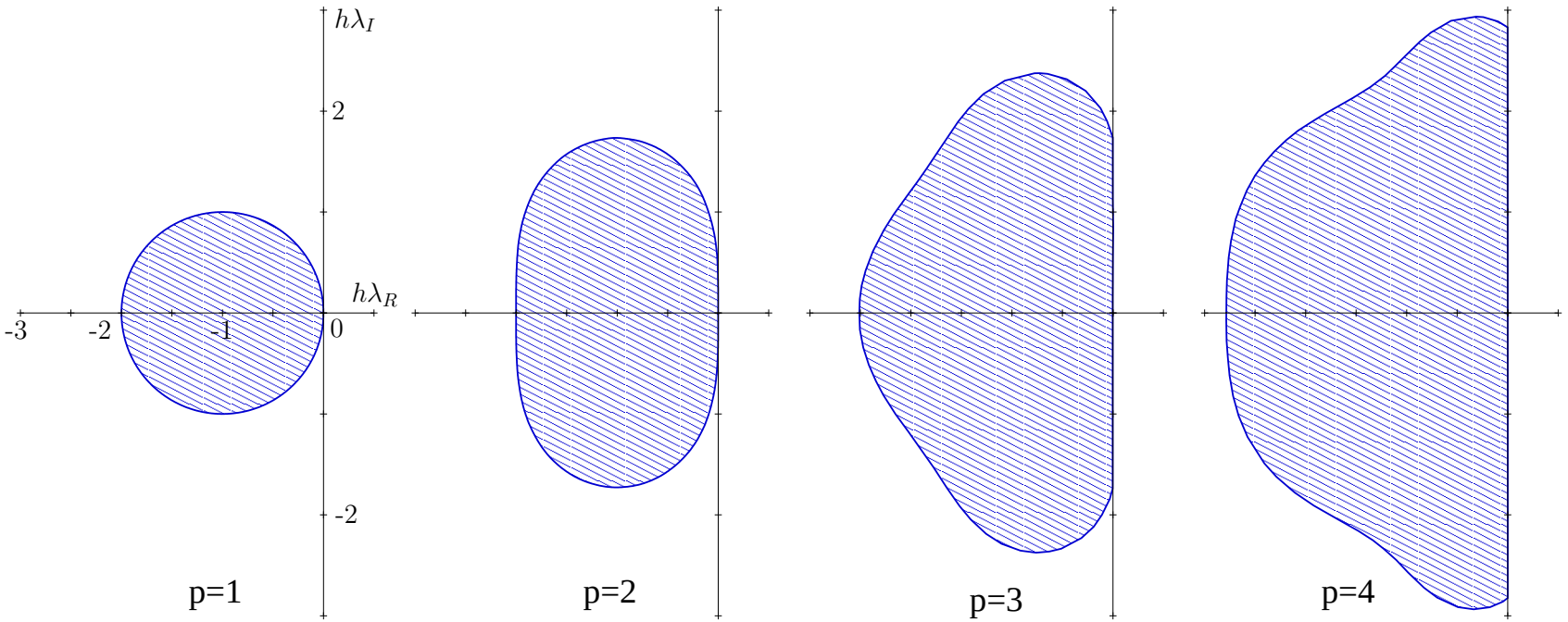
Stability for large h ($\dot{x} = \lambda x$, complex λ)

* The AM1 (Backward Euler), AM2 (Trapezoidal), and second-order BDF methods are A-stable; other methods are *conditionally* stable.
* The region of absolute stability for each method shrinks significantly as the order increases.
* For purely real values of λ, the BDF methods (up to order six) are unconditionally stable, while the AB and AM (except AM1 and AM2) methods are conditionally stable.
* Stability conditions impose restrictions on the choice of methods for circuit simulation.

Region of stability for explicit Runge-Kutta methods

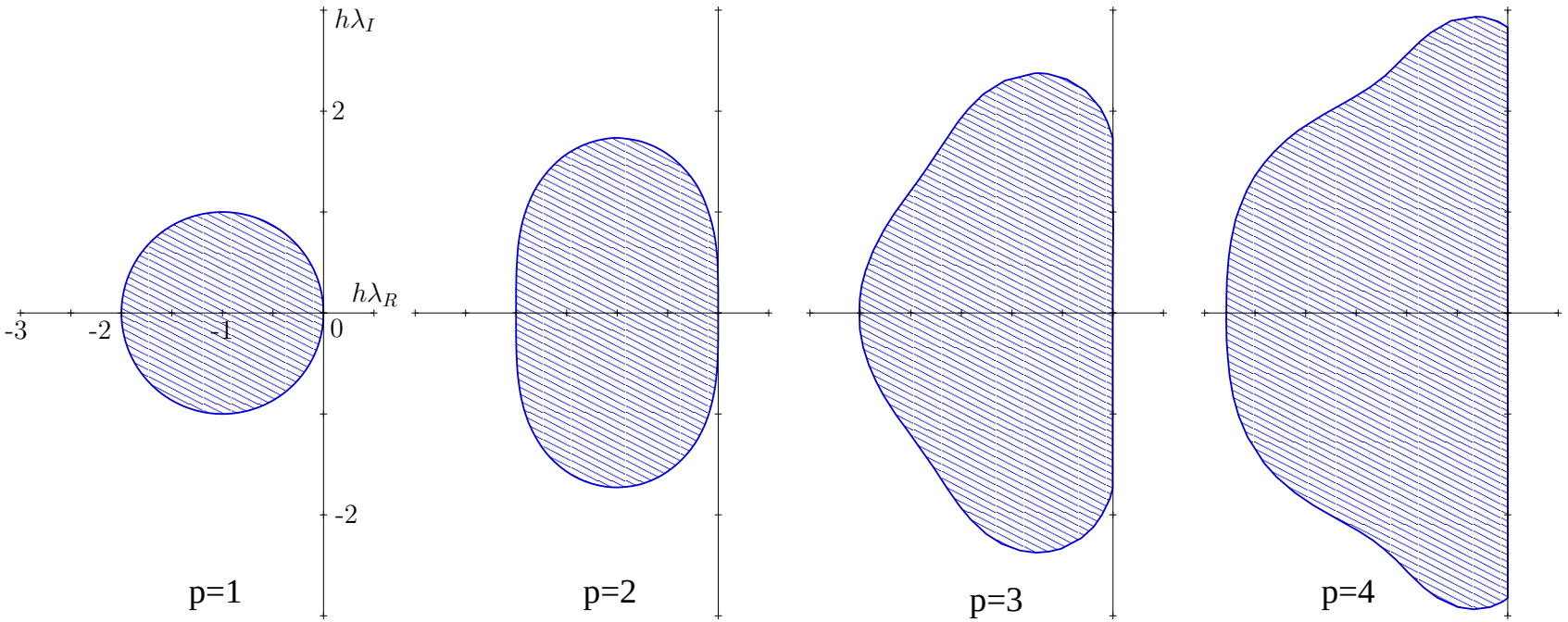


Region of stability for explicit Runge-Kutta methods



* Explicit Runge-Kutta methods are conditionally stable.

Region of stability for explicit Runge-Kutta methods



* Explicit Runge-Kutta methods are conditionally stable.
* On the other hand, implicit Runge-Kutta methods are A-stable [4].

Outline

* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Stiff equations

Consider the 2×2 system of ODEs,

$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,. \end{aligned} \tag{26}$$

Stiff equations

Consider the 2×2 system of ODEs,

$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,. \end{aligned} \tag{26}$$

* If the magnitudes of the eigenvalues λ_1 and λ_2 of the **A** matrix are significantly different, the system of ODEs is said to be *stiff*. (The same idea applies to larger systems as well.)

Stiff equations

Consider the 2×2 system of ODEs,

$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,. \end{aligned} \tag{26}$$

* If the magnitudes of the eigenvalues λ_1 and λ_2 of the **A** matrix are significantly different, the system of ODEs is said to be *stiff*. (The same idea applies to larger systems as well.)
* There are several physical examples of stiff systems, such as motion of masses connected by springs, chemical reactions involving several reactants, and electrical circuits.

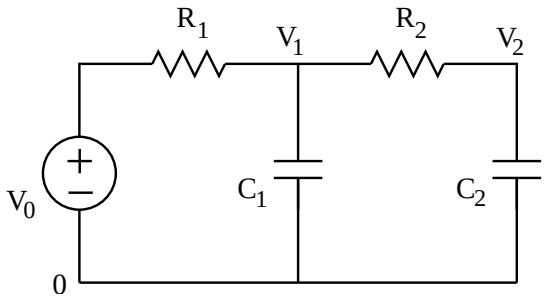
Stiff equations

Consider the 2×2 system of ODEs,

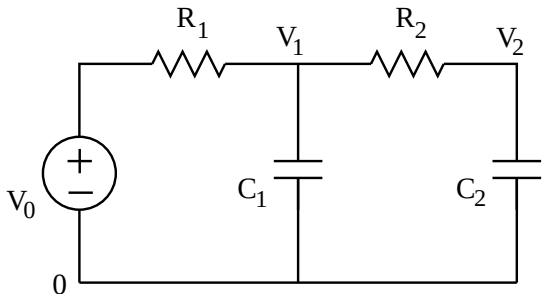
$$\begin{aligned} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 \,, \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 \,. \end{aligned} \tag{26}$$

* If the magnitudes of the eigenvalues λ_1 and λ_2 of the **A** matrix are significantly different, the system of ODEs is said to be *stiff*. (The same idea applies to larger systems as well.)
* There are several physical examples of stiff systems, such as motion of masses connected by springs, chemical reactions involving several reactants, and electrical circuits.
* Stiff equations present a challenge because they involve vastly different time constants. In some cases, it is important for the numerical method to be able to resolve transients on a time scale corresponding to the smallest time constant.

Stiff equations: *RC* circuit example

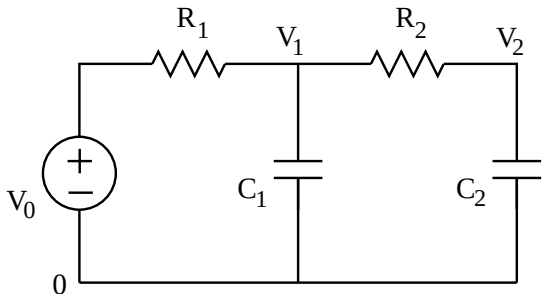


Stiff equations: *RC* circuit example



$$\begin{aligned} \frac{dV_1}{dt} &= \left(\frac{-1}{C_1}\right)\left(\frac{1}{R_1}+\frac{1}{R_2}\right)V_1 + \left(\frac{1}{C_1R_2}\right)V_2 + \left(\frac{V_0}{C_1R_1}\right), \\ \frac{dV_2}{dt} &= \left(\frac{1}{C_2R_2}\right)V_1 - \left(\frac{1}{C_2R_2}\right)V_2 . \end{aligned} \tag{27}$$

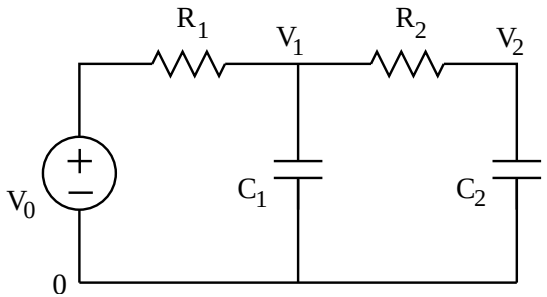
Stiff equations: *RC* circuit example



$$\begin{aligned} \frac{dV_1}{dt} &= \left(\frac{-1}{C_1}\right)\left(\frac{1}{R_1}+\frac{1}{R_2}\right)V_1 + \left(\frac{1}{C_1R_2}\right)V_2 + \left(\frac{V_0}{C_1R_1}\right), \\ \frac{dV_2}{dt} &= \left(\frac{1}{C_2R_2}\right)V_1 - \left(\frac{1}{C_2R_2}\right)V_2 . \end{aligned} \tag{27}$$

* This is a coupled system of ODEs (see Eq. 26).

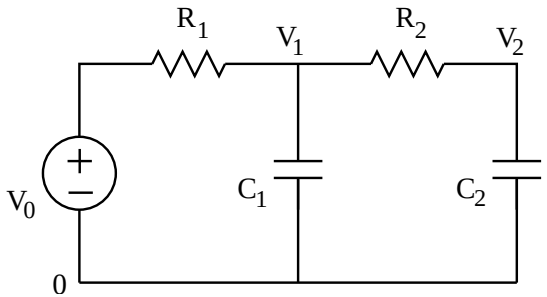
Stiff equations: *RC* circuit example



$$\begin{aligned} \frac{dV_1}{dt} &= \left(\frac{-1}{C_1}\right)\left(\frac{1}{R_1}+\frac{1}{R_2}\right)V_1 + \left(\frac{1}{C_1 R_2}\right)V_2 + \left(\frac{V_0}{C_1 R_1}\right), \\ \frac{dV_2}{dt} &= \left(\frac{1}{C_2 R_2}\right)V_1 - \left(\frac{1}{C_2 R_2}\right)V_2 . \end{aligned} \tag{27}$$

* This is a coupled system of ODEs (see Eq. 26).
* For $R_1 = 0.5\,\Omega$, $R_2 = 5\,\Omega$, $C_1 = 0.01\,F$, $C_2 = 1\,F$, the eigenvalues are, $\lambda_1 = -0.182\,s^{-1}$, $\lambda_2 = -220\,s^{-1}$, and the time constants are $\tau_1 = 5.5\,s$, $\tau_2 = 0.0045\,s$.

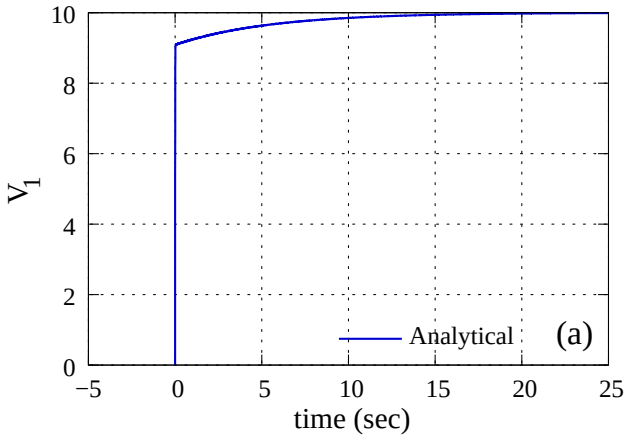
Stiff equations: *RC* circuit example

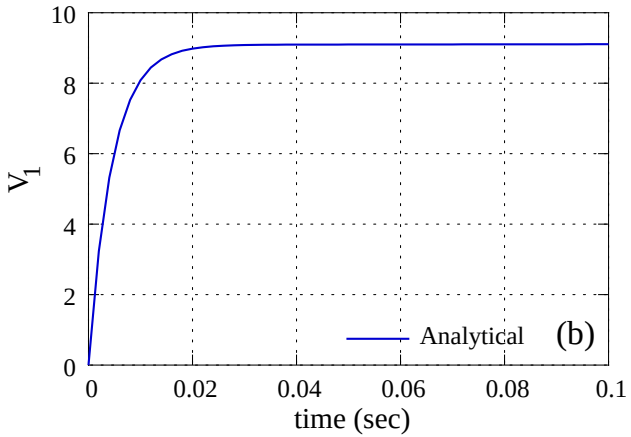


$$\begin{aligned} \frac{dV_1}{dt} &= \left(\frac{-1}{C_1}\right)\left(\frac{1}{R_1}+\frac{1}{R_2}\right)V_1 + \left(\frac{1}{C_1 R_2}\right)V_2 + \left(\frac{V_0}{C_1 R_1}\right), \\ \frac{dV_2}{dt} &= \left(\frac{1}{C_2 R_2}\right)V_1 - \left(\frac{1}{C_2 R_2}\right)V_2 . \end{aligned} \tag{27}$$

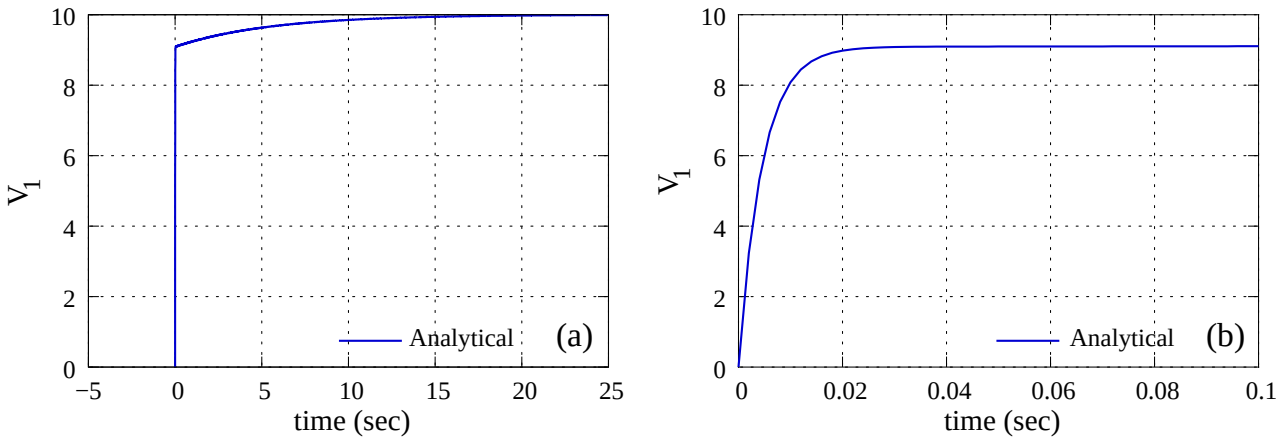
* This is a coupled system of ODEs (see Eq. 26).
* For $R_1 = 0.5\,\Omega$, $R_2 = 5\,\Omega$, $C_1 = 0.01\,F$, $C_2 = 1\,F$, the eigenvalues are, $\lambda_1 = -0.182\,s^{-1}$, $\lambda_2 = -220\,s^{-1}$, and the time constants are $\tau_1 = 5.5\,s$, $\tau_2 = 0.0045\,s$.
* Note that $\tau_1 \approx 1000 \times \tau_2$.

Stiff equations: *RC* circuit example





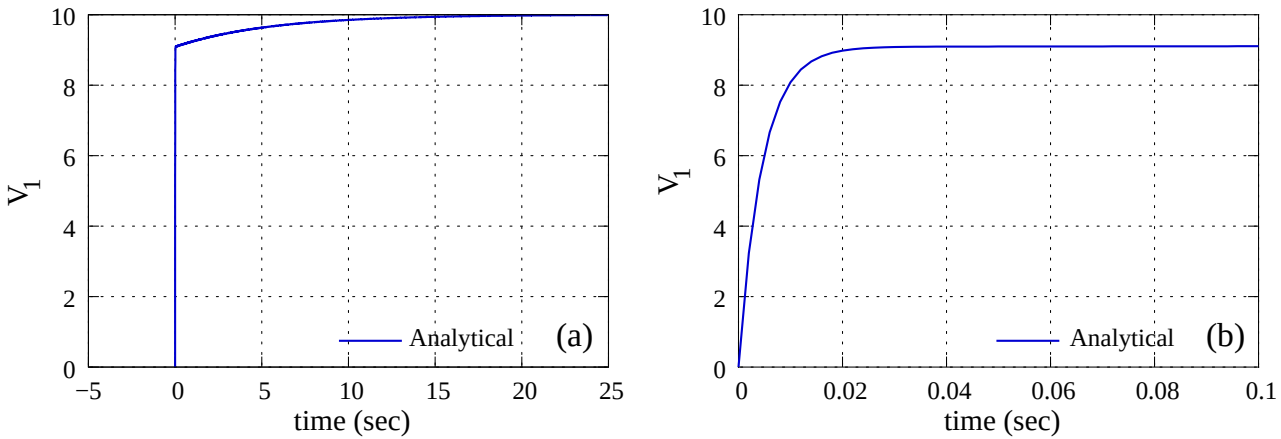
Stiff equations: *RC* circuit example



* The currents and voltages in the circuit are given by the general form,

$$x(t) = Ae^{-t/\tau_1} + Be^{-t/\tau_2} + C\,. \tag{28}$$

Stiff equations: *RC* circuit example

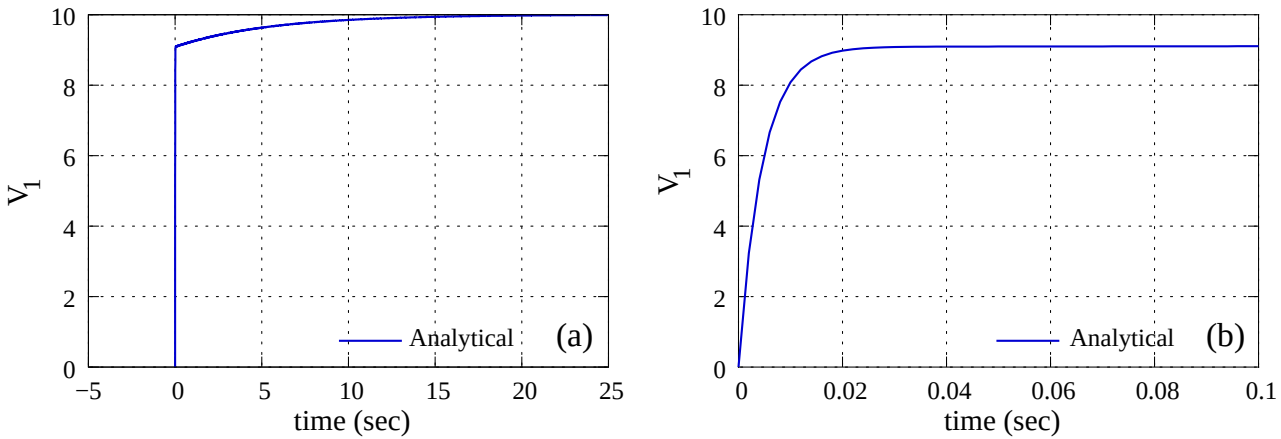


* The currents and voltages in the circuit are given by the general form,

$$x(t) = Ae^{-t/\tau_1} + Be^{-t/\tau_2} + C\,. \tag{28}$$

* Two transients can be seen in (a) – an initial fast transient due to τ_2, followed by a slow transient due to τ_1.

Stiff equations: *RC* circuit example

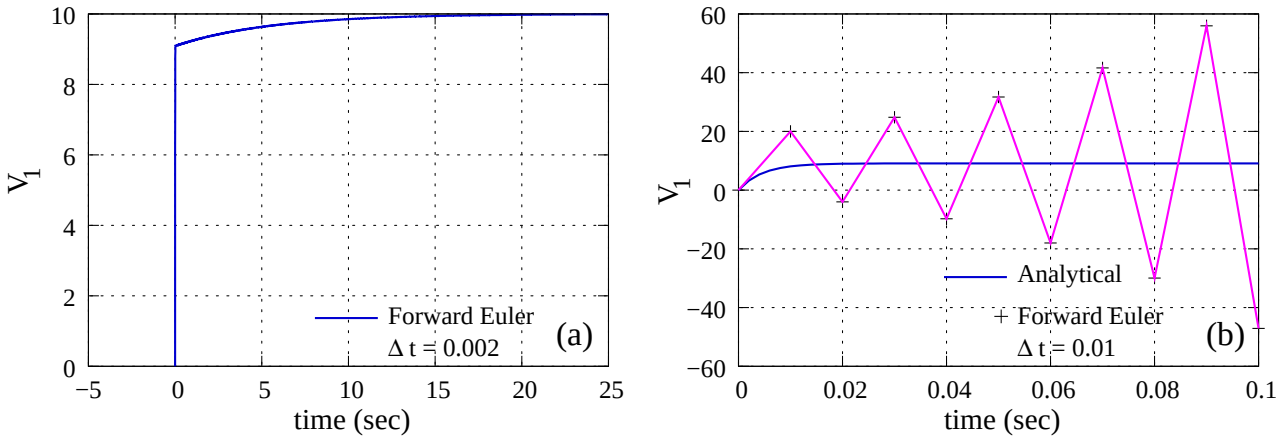


* The currents and voltages in the circuit are given by the general form,

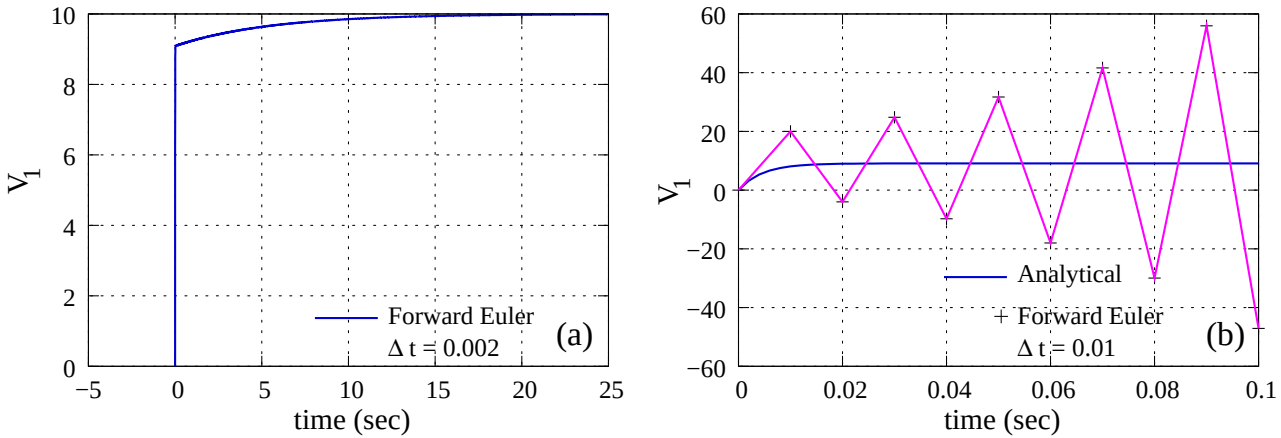
$$x(t) = Ae^{-t/\tau_1} + Be^{-t/\tau_2} + C\,. \tag{28}$$

* Two transients can be seen in (a) – an initial fast transient due to τ_2, followed by a slow transient due to τ_1.
* An expanded view of the fast transient is shown in (b).

Stiff equations: *RC* circuit example (FE results)

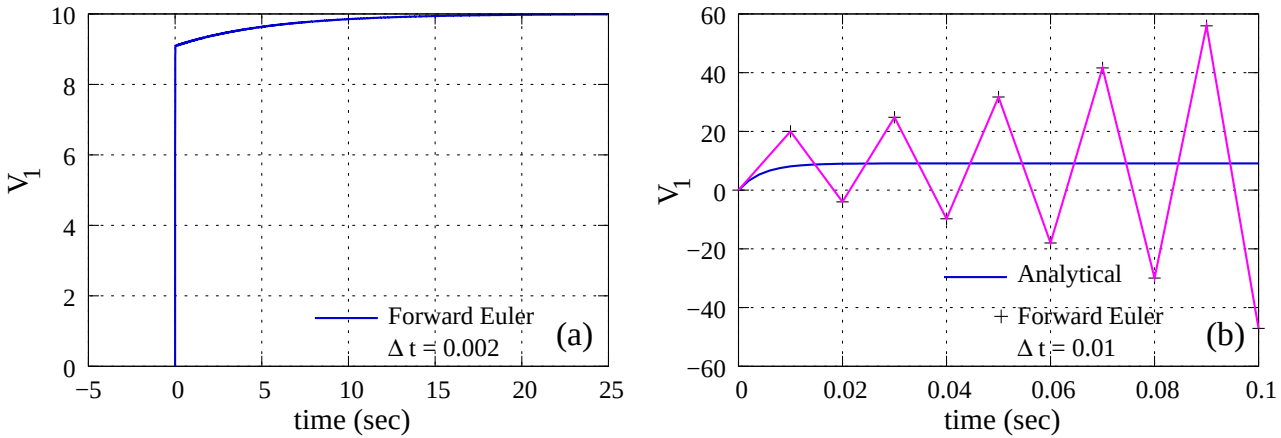


Stiff equations: *RC* circuit example (FE results)



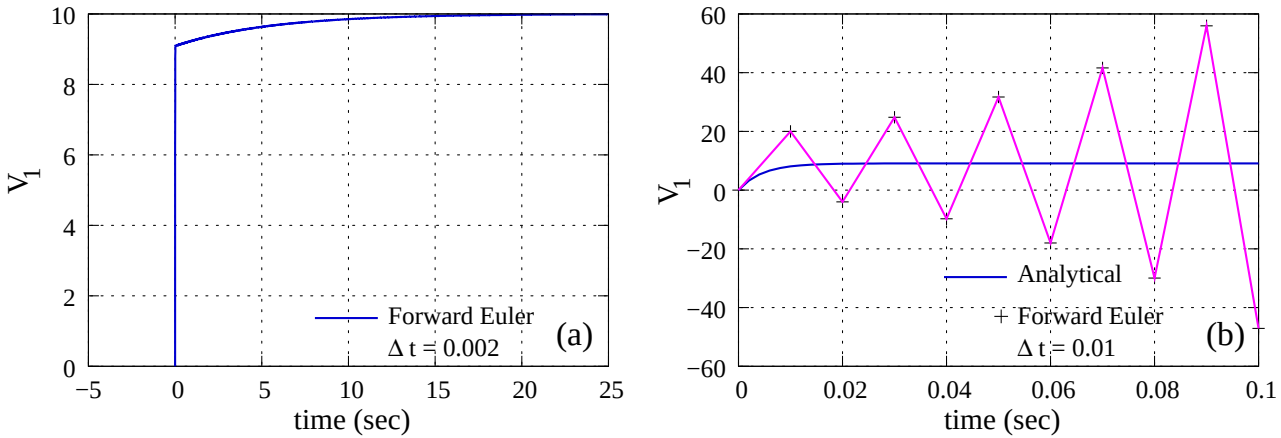
* For $\Delta t = 0.002$ s, Forward Euler results are acceptable.

Stiff equations: *RC* circuit example (FE results)



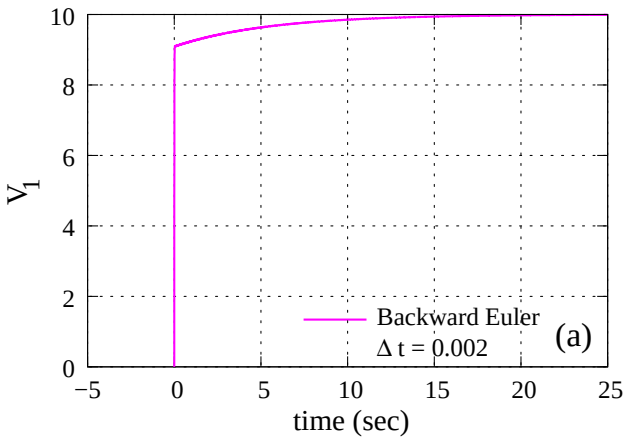
* For $\Delta t = 0.002$ s, Forward Euler results are acceptable.
* For $\Delta t = 0.01$ s, which is large than $2\,\tau_2$ but much smaller than τ_1, the Forward Euler method is unstable.

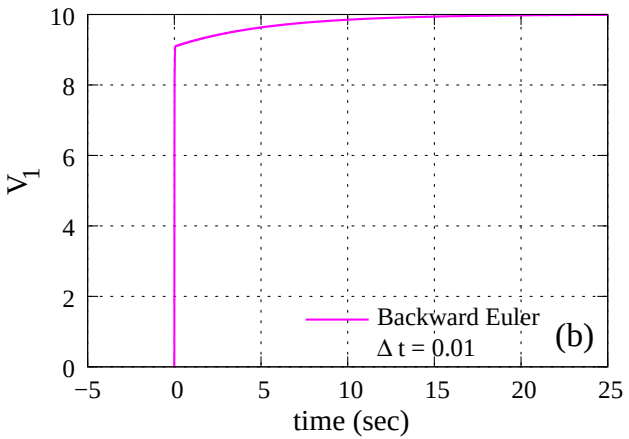
Stiff equations: *RC* circuit example (FE results)



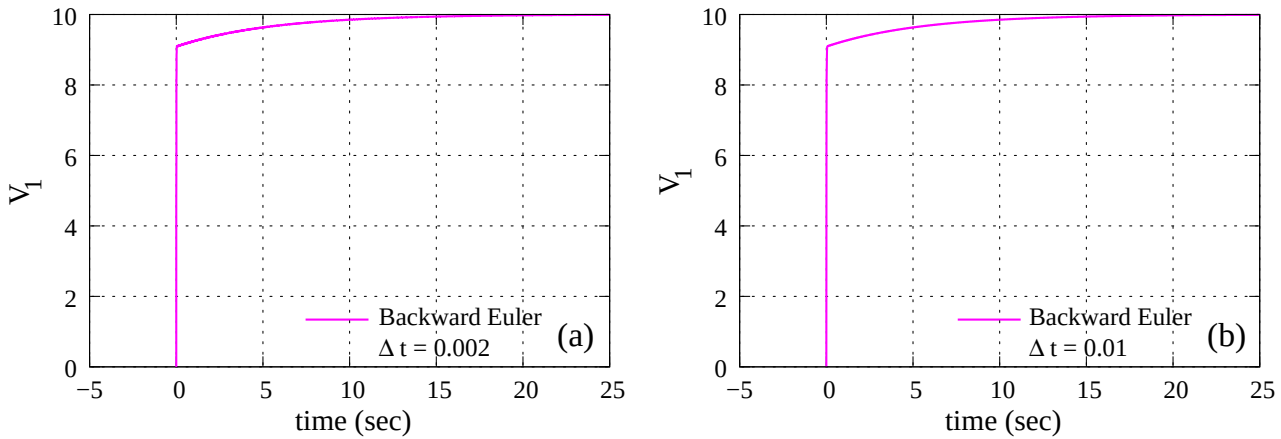
* For $\Delta t = 0.002$ s, Forward Euler results are acceptable.
* For $\Delta t = 0.01$ s, which is large than $2\,\tau_2$ but much smaller than τ_1, the Forward Euler method is unstable.
* To prevent the unstable behaviour, a small time step is required *throughout*, i.e., even *after* the fast transient has vanished $\Rightarrow$ extremely inefficient simulation.

Stiff equations: *RC* circuit example (BE results)



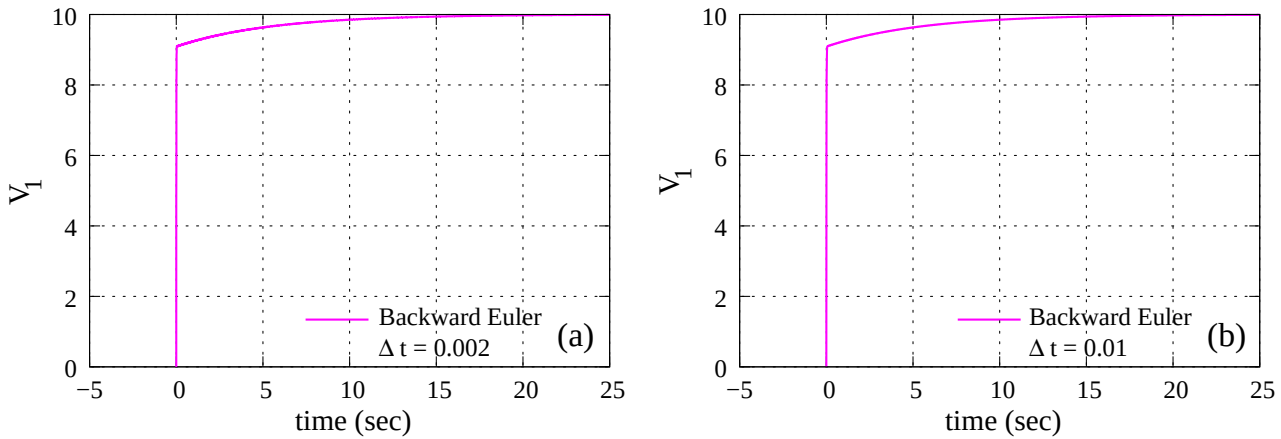


Stiff equations: *RC* circuit example (BE results)



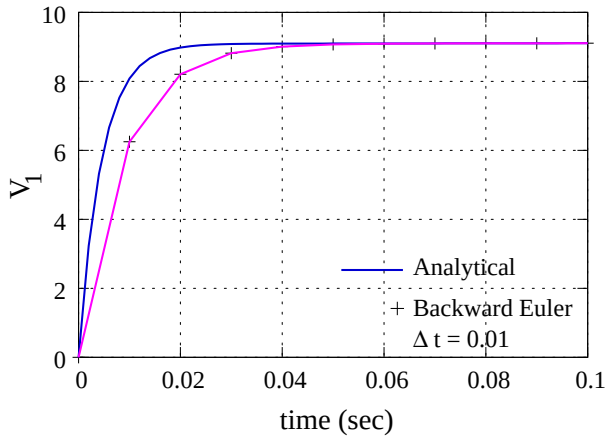
* The Backward Euler method is stable for For $\Delta t = 0.01$ s as well, which is expected from an A-stable method.

Stiff equations: *RC* circuit example (BE results)

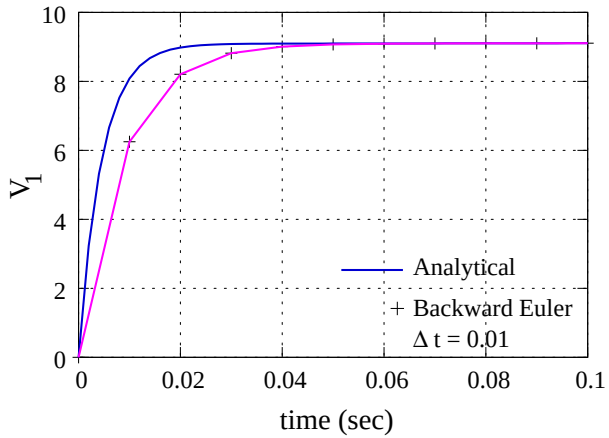


* The Backward Euler method is stable for For $\Delta t = 0.01$ s as well, which is expected from an A-stable method.
* The BE method allows much larger time steps than the FE method.

Stiff equations: *RC* circuit example (BE results)

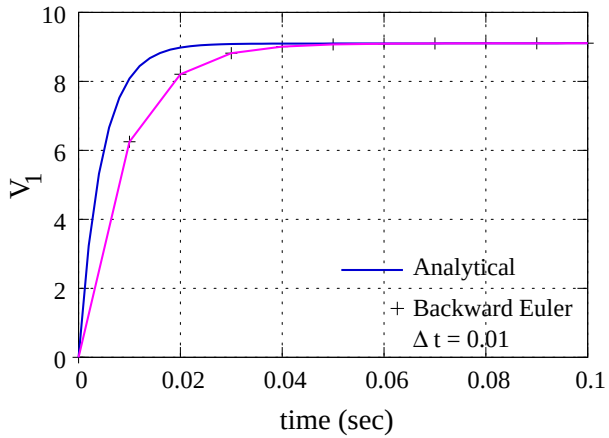


Stiff equations: *RC* circuit example (BE results)



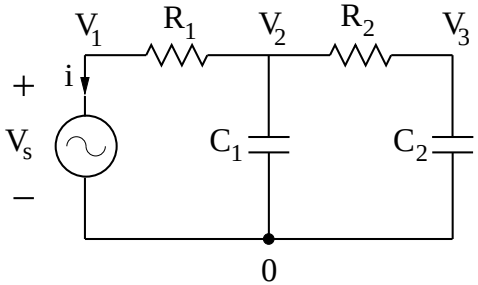
* Although the BE method with $\Delta t = 0.01$ s works well for the slow transient, it does not capture the fast transient accurately, as shown in this expanded view.

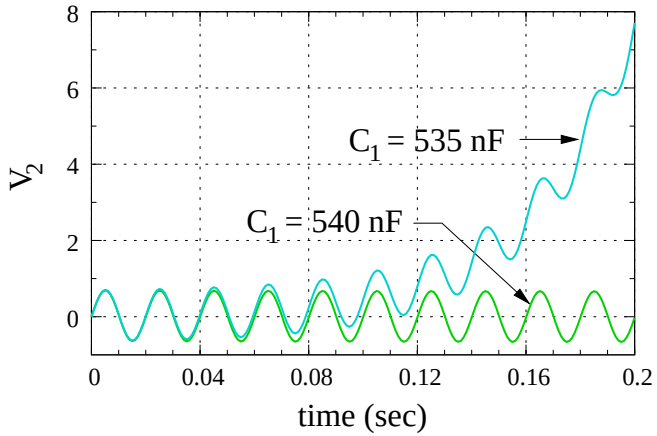
Stiff equations: *RC* circuit example (BE results)



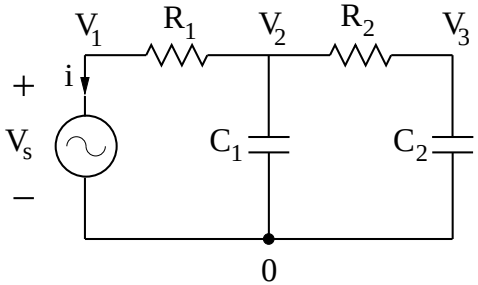
* Although the BE method with $\Delta t = 0.01$ s works well for the slow transient, it does not capture the fast transient accurately, as shown in this expanded view.
* In practice, the time step is made small when things are changing rapidly, and large otherwise. This strategy makes the simulation faster without compromising on accuracy.

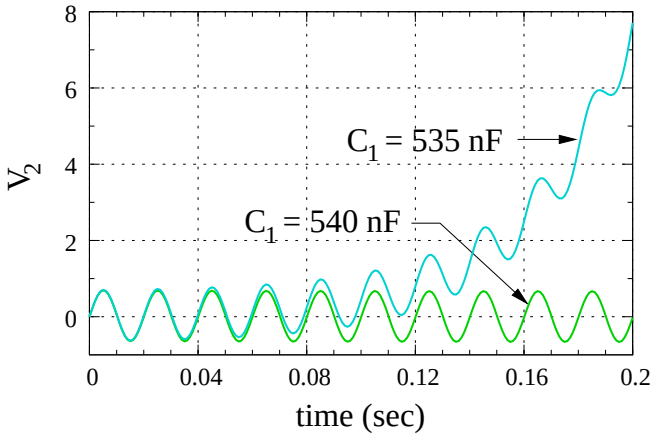
Stiff equations: *RC* circuit example (RK4 results)





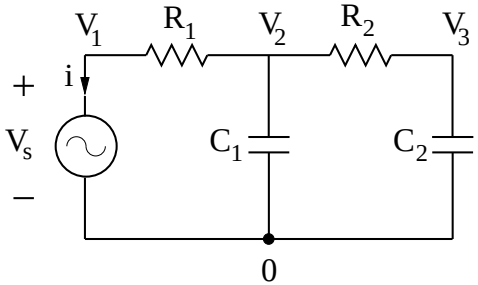
Stiff equations: *RC* circuit example (RK4 results)

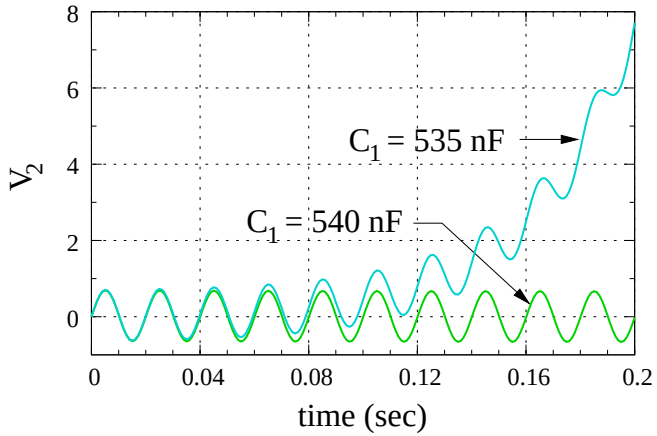




* Parameters: $R_1 = 1$ kΩ, $R_2 = 2$ kΩ, $C_2 = 1$ mF, $f = 50$ Hz, $\hat{V}$ (amplitude of V_s)=1 V, and h (step size)=1 ms.

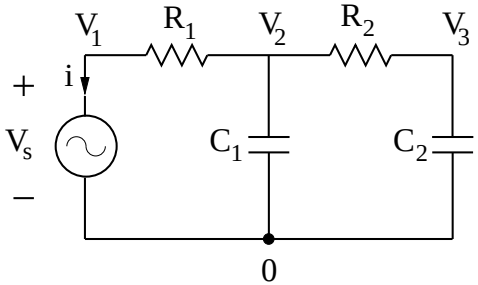
Stiff equations: *RC* circuit example (RK4 results)

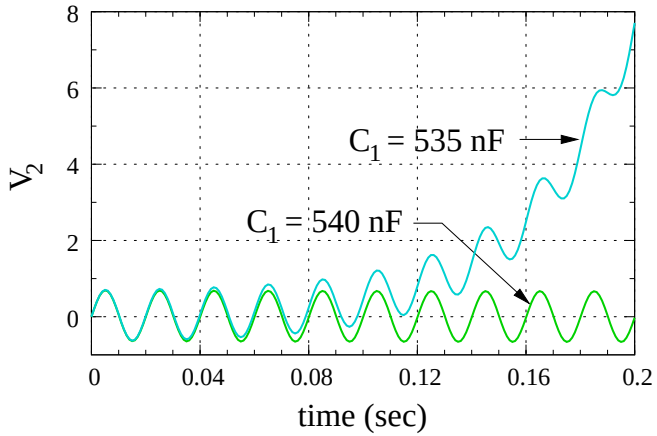




* Parameters: $R_1 = 1$ kΩ, $R_2 = 2$ kΩ, $C_2 = 1$ mF, $f = 50$ Hz, $\hat{V}$ (amplitude of V_s)=1 V, and h (step size)=1 ms.
* For $f = 50$ Hz, $X_{C1} = 295$ kΩ, and $X_{C2} = 160$ Ω. $\Rightarrow$ C_1 is effectively an open circuit, and its exact value should have no effect on the results.

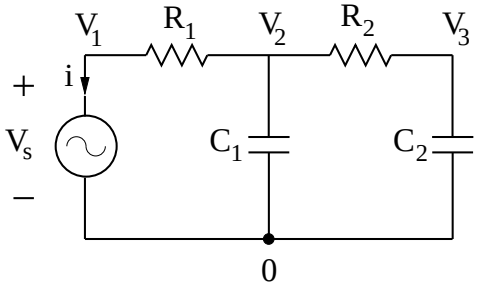
Stiff equations: *RC* circuit example (RK4 results)

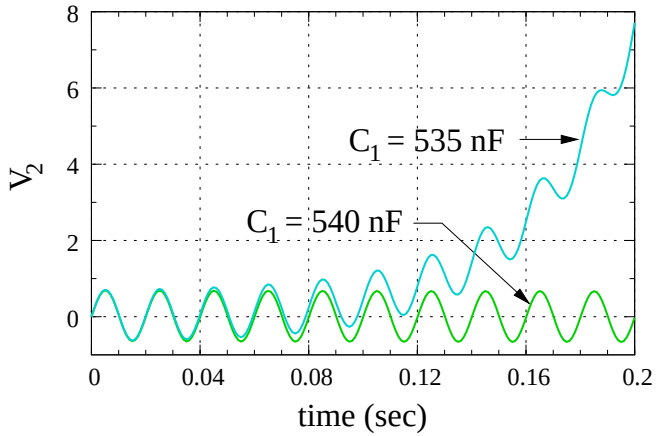




* Parameters: $R_1 = 1$ kΩ, $R_2 = 2$ kΩ, $C_2 = 1$ mF, $f = 50$ Hz, $\hat{V}$ (amplitude of V_s)=1 V, and h (step size)=1 ms.
* For $f = 50$ Hz, $X_{C1} = 295$ kΩ, and $X_{C2} = 160$ Ω. $\Rightarrow$ C_1 is effectively an open circuit, and its exact value should have no effect on the results.
* However, for $C_1 = 540$ nF and $C_1 = 535$ nF, the RK4 results are dramatically different. Why?

Stiff equations: *RC* circuit example (RK4 results)



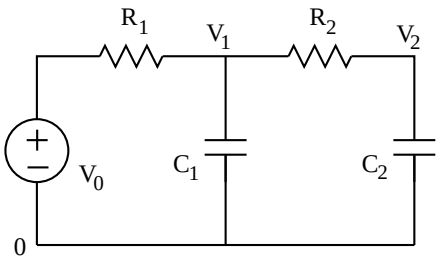


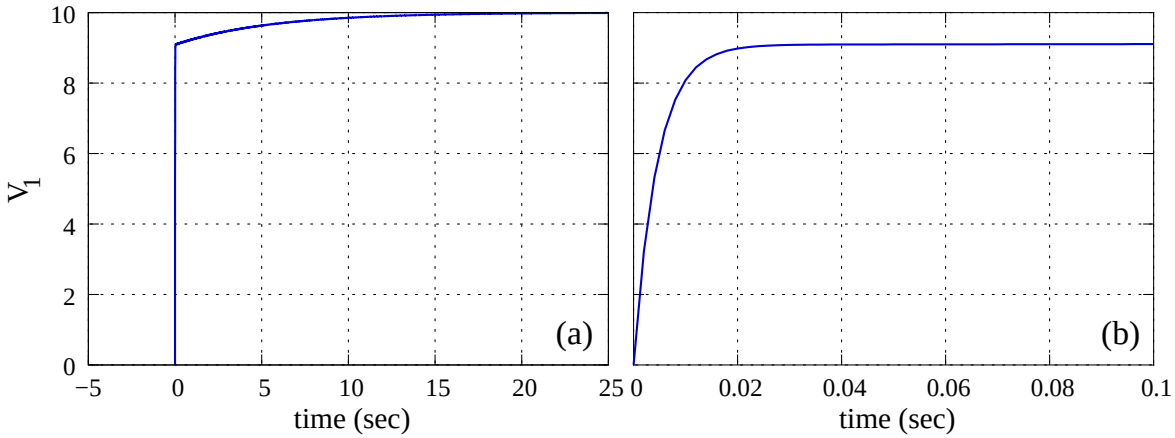
* Parameters: $R_1 = 1$ kΩ, $R_2 = 2$ kΩ, $C_2 = 1$ mF, $f = 50$ Hz, $\hat{V}$ (amplitude of V_s)=1 V, and h (step size)=1 ms.
* For $f = 50$ Hz, $X_{C1} = 295$ kΩ, and $X_{C2} = 160$ Ω. $\Rightarrow$ C_1 is effectively an open circuit, and its exact value should have no effect on the results.
* However, for $C_1 = 540$ nF and $C_1 = 535$ nF, the RK4 results are dramatically different. Why?
* $C_1 = 535$ nF makes one of the time constants in the circuit small enough (with respect to $h = 1$ ms) to make the RK4 method unstable.

Outline

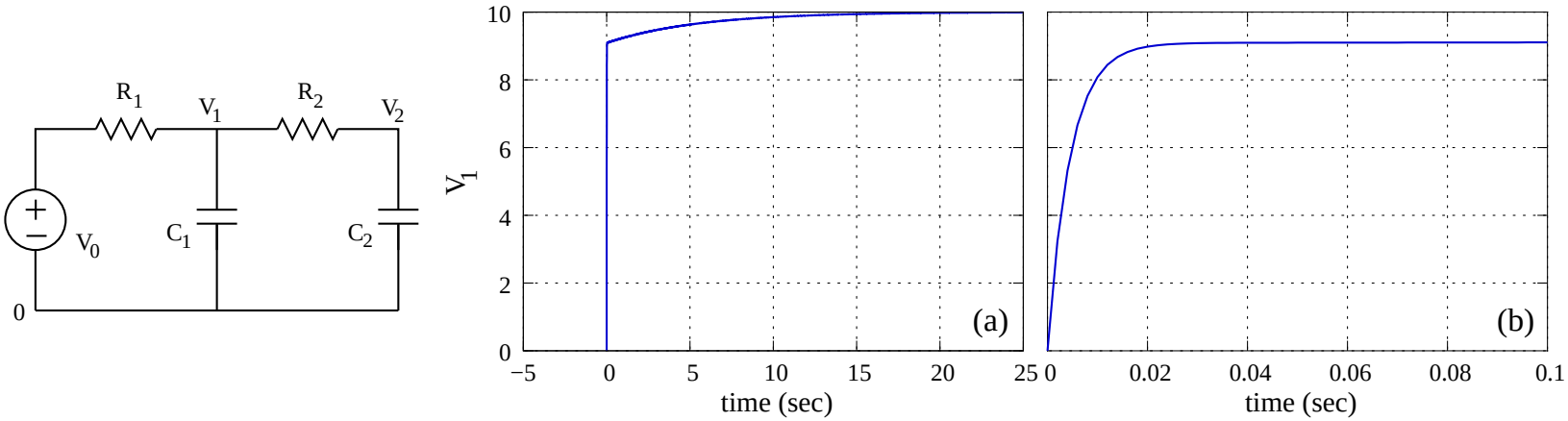
* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Adaptive step size



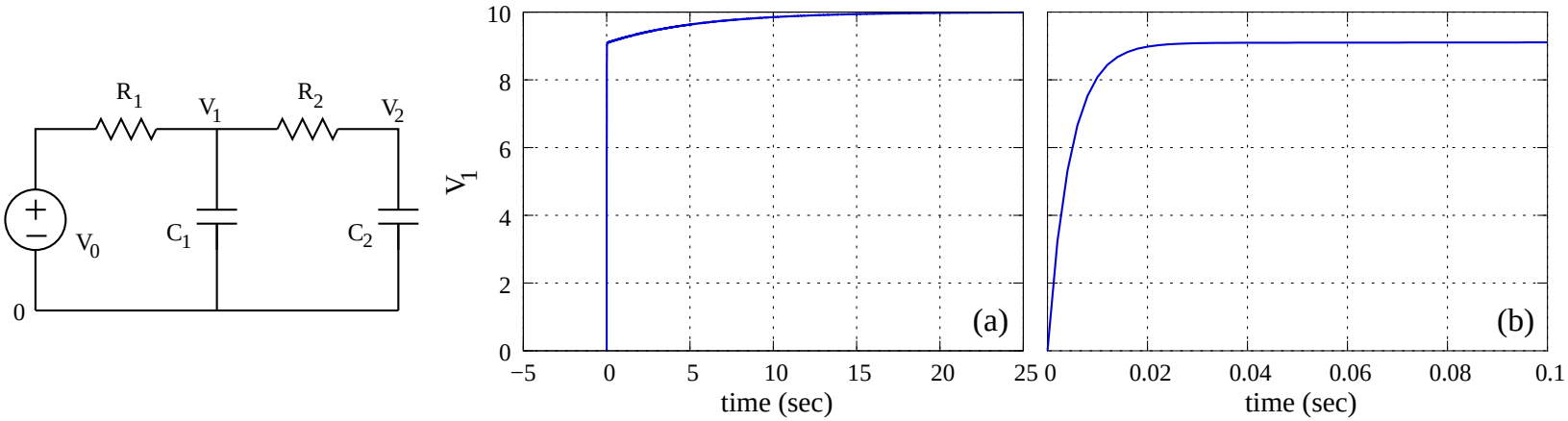


Adaptive step size



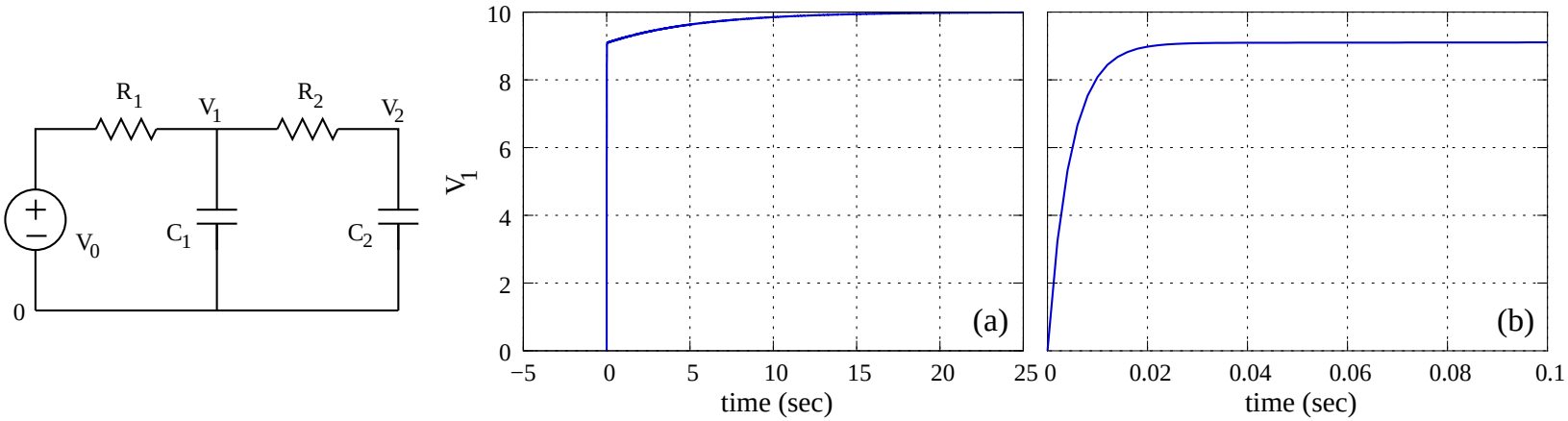
* It is desirable to use small time steps when the solution is changing rapidly, and large time steps otherwise.

Adaptive step size



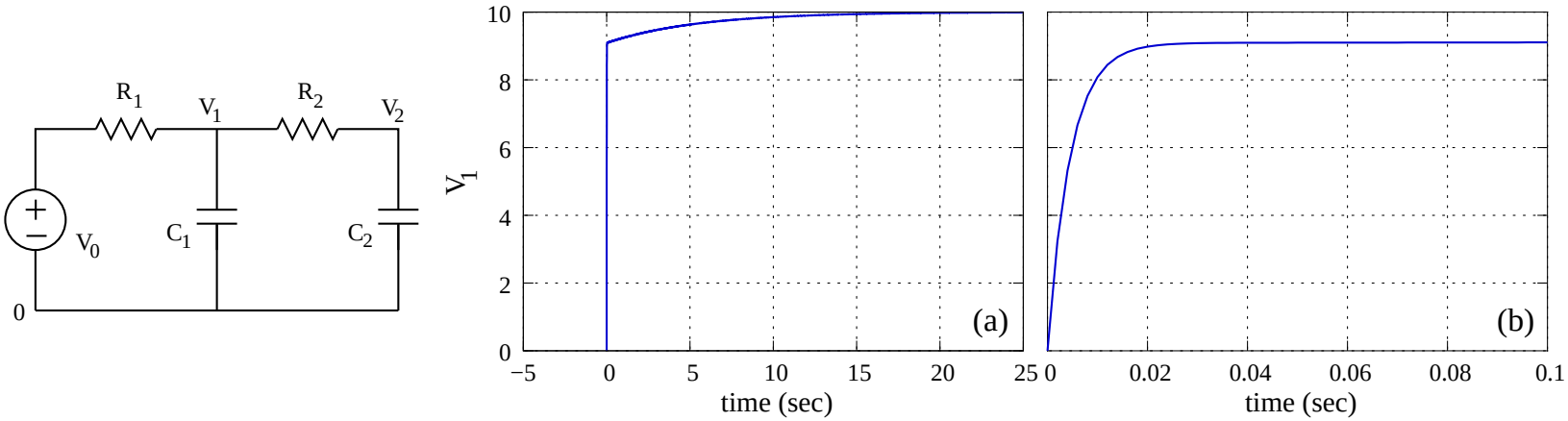
* It is desirable to use small time steps when the solution is changing rapidly, and large time steps otherwise.
* For automatic time step computation, the next time step can be computed on the basis of

Adaptive step size



* It is desirable to use small time steps when the solution is changing rapidly, and large time steps otherwise.
* For automatic time step computation, the next time step can be computed on the basis of
  - an estimate of the local truncation error (LTE)

Adaptive step size



* It is desirable to use small time steps when the solution is changing rapidly, and large time steps otherwise.
* For automatic time step computation, the next time step can be computed on the basis of
  - an estimate of the local truncation error (LTE)
  - convergence behaviour of Newton-Raphson algorithm (for nonlinear problems)

LTE estimation: use of two time steps

* The local error of a numerical method of order p is given by,

$$x(t_n + h) - x_{n+1} = h^{p+1}\psi(t_n, x_n) + O(h^{p+2}), \tag{29}$$

where ψ is called the principal error function of the method.

LTE estimation: use of two time steps

* The local error of a numerical method of order p is given by,

$$x(t_n + h) - x_{n+1} = h^{p+1}\psi(t_n, x_n) + O(h^{p+2}), \tag{29}$$

where ψ is called the principal error function of the method.

* If, instead of a single step of h, we take two steps of $h/2$ each, then the local error would be

$$x(t_n + h) - \tilde{x}_{n+1} = 2\left(\frac{h}{2}\right)^{p+1}\psi(t_n, x_n) + O(h^{p+2}), \tag{30}$$

where $\tilde{x}_{n+1}$ denotes the computed solution after the second step.

LTE estimation: use of two time steps

* The local error of a numerical method of order p is given by,

$$x(t_n + h) - x_{n+1} = h^{p+1}\psi(t_n, x_n) + O(h^{p+2}), \tag{29}$$

where ψ is called the principal error function of the method.

* If, instead of a single step of h, we take two steps of $h/2$ each, then the local error would be

$$x(t_n + h) - \tilde{x}_{n+1} = 2\left(\frac{h}{2}\right)^{p+1}\psi(t_n, x_n) + O(h^{p+2}), \tag{30}$$

where $\tilde{x}_{n+1}$ denotes the computed solution after the second step.

* By subtracting Eq. 29 from Eq. 30, we get an estimate for the LTE,

$$\text{LTE}^{\text{est}} = \left(\frac{2^p}{2^p - 1}\right)|\tilde{x}_{n+1} - x_{n+1}|. \tag{31}$$

LTE estimation: use of two time steps

* Suppose τ has been specified as the maximum allowed LTE.

LTE estimation: use of two time steps

* Suppose τ has been specified as the maximum allowed LTE.
* If $\text{LTE}^{\text{est}} < \tau$, the current step is accepted, and the next step is allowed to increase (since a larger step may continue to fulfill the the constraint on the LTE).

LTE estimation: use of two time steps

* Suppose τ has been specified as the maximum allowed LTE.
* If $\text{LTE}^{\text{est}} < \tau$, the current step is accepted, and the next step is allowed to increase (since a larger step may continue to fulfill the the constraint on the LTE).
* If $\text{LTE}^{\text{est}} > \tau$, the current step is rejected, and a new trial step h_n' is computed such that it would result in an LTE equal to τ.

$$h_n' = h_n \left(\frac{\tau}{\text{LTE}^{\text{est}}} \right)^{1/p+1} . \tag{32}$$

LTE estimation: use of two time steps

* Suppose τ has been specified as the maximum allowed LTE.
* If $\text{LTE}^{\text{est}} < \tau$, the current step is accepted, and the next step is allowed to increase (since a larger step may continue to fulfill the the constraint on the LTE).
* If $\text{LTE}^{\text{est}} > \tau$, the current step is rejected, and a new trial step h_n' is computed such that it would result in an LTE equal to τ.

$$h_n' = h_n \left(\frac{\tau}{\text{LTE}^{\text{est}}} \right)^{1/p+1} . \tag{32}$$

* LTE^{est} may also be used to improve the accuracy of the solution.

LTE estimation: use of methods of different orders

Consider two methods of orders p and $(p+1)$. If $x_n = x(t_n)$ is assumed, we get

$$\text{LTE}^{(p)} = x(t_{n+1}) - x_{n+1}\,, \tag{33}$$

$$\text{LTE}^{(p+1)} = x(t_{n+1}) - \tilde{x}_{n+1}\,, \tag{34}$$

where the superscript on LTE indicates the order of the method, and x_{n+1}, $\tilde{x}_{n+1}$ denote the numerical solutions corresponding to the two methods.

LTE estimation: use of methods of different orders

Consider two methods of orders p and $(p+1)$. If $x_n = x(t_n)$ is assumed, we get

$$\text{LTE}^{(p)} = x(t_{n+1}) - x_{n+1}\,, \tag{33}$$

$$\text{LTE}^{(p+1)} = x(t_{n+1}) - \tilde{x}_{n+1}\,, \tag{34}$$

where the superscript on LTE indicates the order of the method, and x_{n+1}, $\tilde{x}_{n+1}$ denote the numerical solutions corresponding to the two methods.

Subtracting Eq. 34 from Eq. 33 yields,

$$\text{LTE}^{(p)} - \text{LTE}^{(p+1)} = \tilde{x}_{n+1} - x_{n+1}\,. \tag{35}$$

LTE estimation: use of methods of different orders

Consider two methods of orders p and $(p+1)$. If $x_n = x(t_n)$ is assumed, we get

$$\text{LTE}^{(p)} = x(t_{n+1}) - x_{n+1}\,, \tag{33}$$

$$\text{LTE}^{(p+1)} = x(t_{n+1}) - \tilde{x}_{n+1}\,, \tag{34}$$

where the superscript on LTE indicates the order of the method, and x_{n+1}, $\tilde{x}_{n+1}$ denote the numerical solutions corresponding to the two methods.

Subtracting Eq. 34 from Eq. 33 yields,

$$\text{LTE}^{(p)} - \text{LTE}^{(p+1)} = \tilde{x}_{n+1} - x_{n+1}\,. \tag{35}$$

Assuming that $\text{LTE}^{(p+1)}$ can be neglected in comparison with $\text{LTE}^{(p)}$ (since $\text{LTE}^{(p+1)}$ is for a higher-order method), we get an estimate for $\text{LTE}^{(p)}$.

LTE estimation: use of methods of different orders

Consider two methods of orders p and $(p+1)$. If $x_n = x(t_n)$ is assumed, we get

$$\text{LTE}^{(p)} = x(t_{n+1}) - x_{n+1}\,, \tag{33}$$

$$\text{LTE}^{(p+1)} = x(t_{n+1}) - \tilde{x}_{n+1}\,, \tag{34}$$

where the superscript on LTE indicates the order of the method, and x_{n+1}, $\tilde{x}_{n+1}$ denote the numerical solutions corresponding to the two methods.

Subtracting Eq. 34 from Eq. 33 yields,

$$\text{LTE}^{(p)} - \text{LTE}^{(p+1)} = \tilde{x}_{n+1} - x_{n+1}\,. \tag{35}$$

Assuming that $\text{LTE}^{(p+1)}$ can be neglected in comparison with $\text{LTE}^{(p)}$ (since $\text{LTE}^{(p+1)}$ is for a higher-order method), we get an estimate for $\text{LTE}^{(p)}$.

Note that an additional cost of computing $\tilde{x}_{n+1}$ with a higher-order method is involved here. In practice, the low- and high-order methods are chosen so that some of the computation of the low-order method can be used for the high-order method.

Runge-Kutta-Fehlberg 4/5 method

0					0					
$\frac{1}{4}$	$\frac{1}{4}$				$\frac{1}{4}$	$\frac{1}{4}$				
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$			$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$			
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$		$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$		
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	
					$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$

4^{th}-order array

5^{th}-order array

Runge-Kutta-Fehlberg 4/5 method

0					0						
$\frac{1}{4}$	$\frac{1}{4}$				$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$			$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$		$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		
					$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$

4^{th}-order array 5^{th}-order array

* Since the function values, $f_0, f_1, \cdots, f_4$ are the same in the two methods, only six function evaluations are required in each time step.

Runge-Kutta-Fehlberg 4/5 method

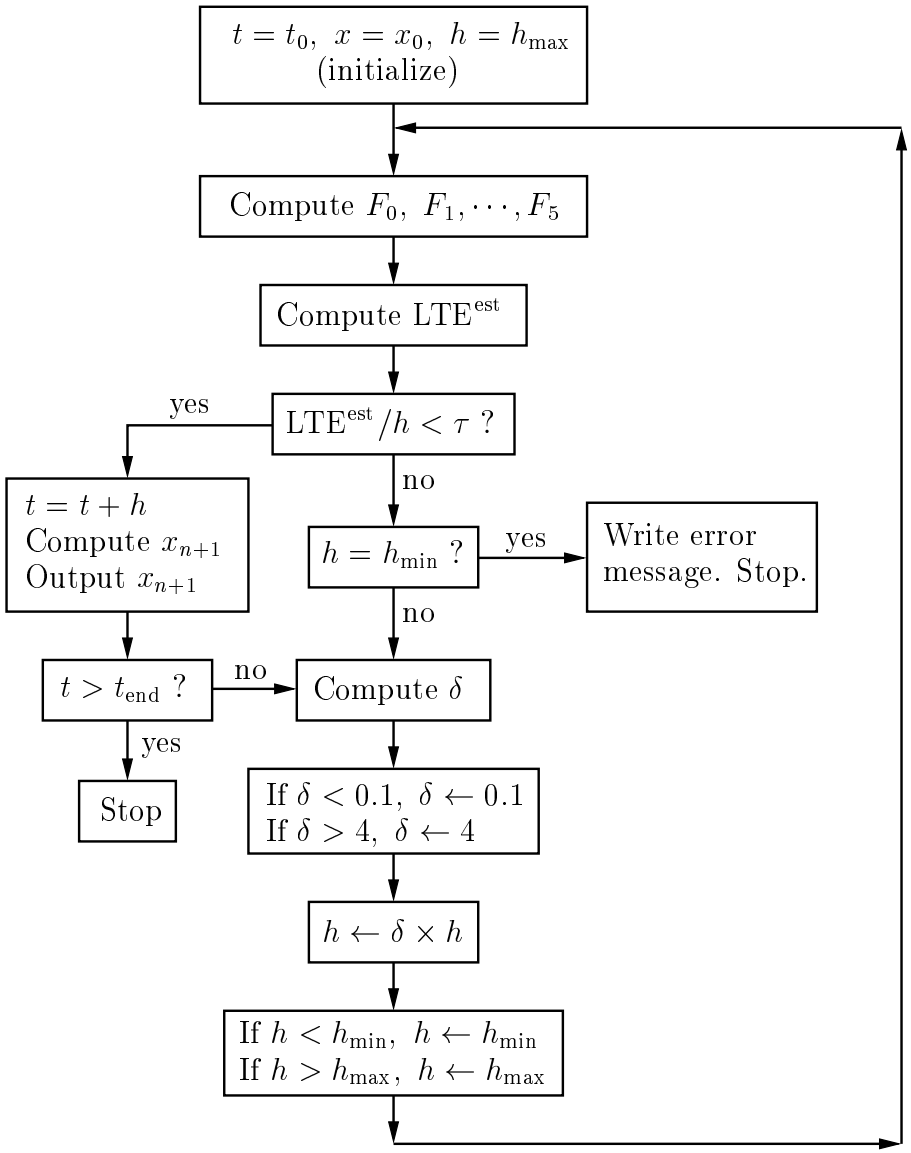
0					0						
$\frac{1}{4}$	$\frac{1}{4}$				$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$			$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$		$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		
					$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$

4^{th}-order array 5^{th}-order array

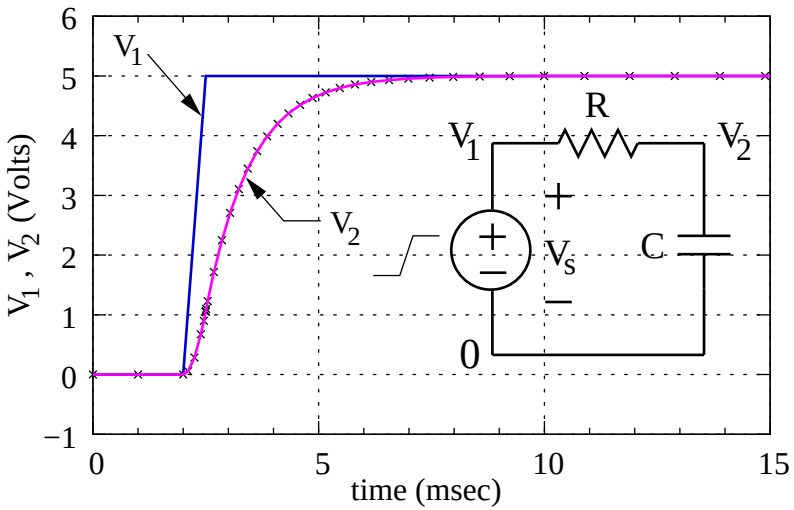
* Since the function values, $f_0, f_1, \cdots, f_4$ are the same in the two methods, only six function evaluations are required in each time step.
* The estimated LTE of the fourth-order formula is given by,

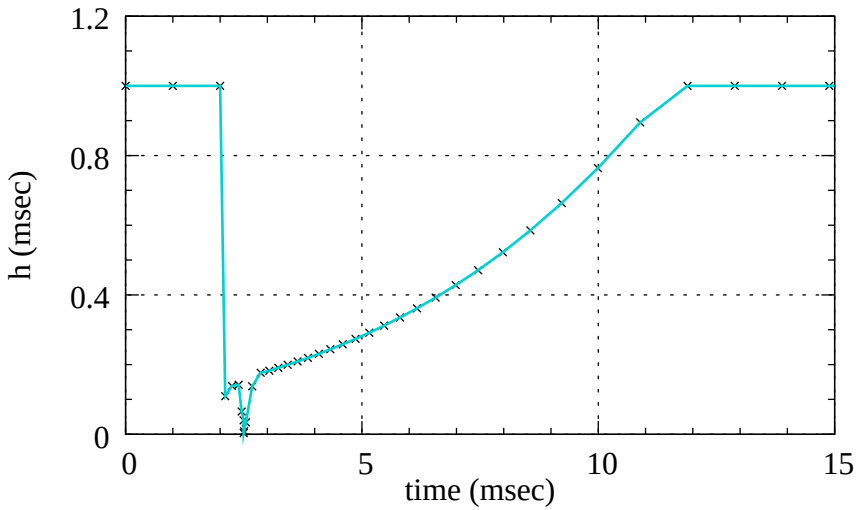
$$\text{LTE}^{\text{est}} = h\left[\frac{1}{360}f_0 - \frac{128}{4275}f_2 - \frac{2197}{25740}f_3 + \frac{1}{50}f_4 + \frac{2}{55}f_5\right]. \qquad (36)$$

Runge-Kutta-Fehlberg 4/5 method: flow chart [1]

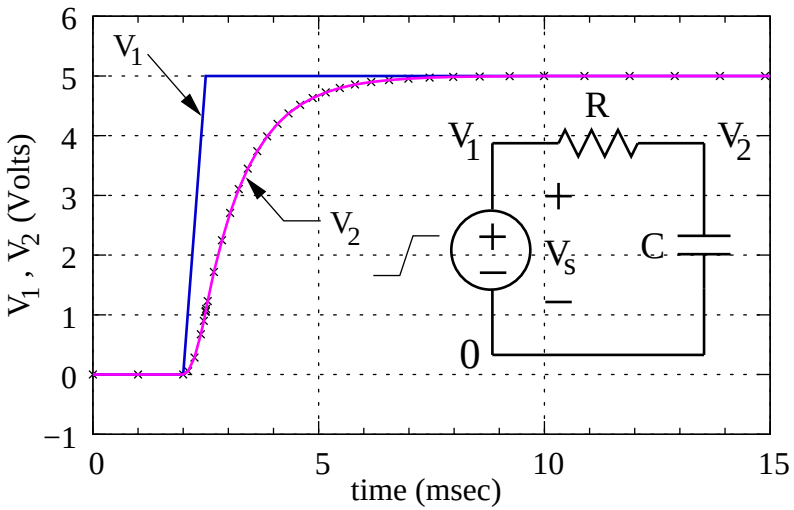


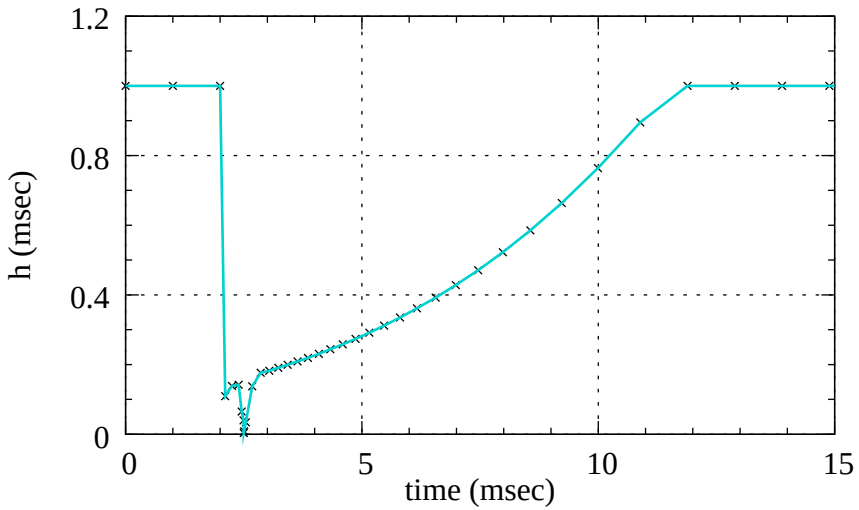
Runge-Kutta-Fehlberg 4/5 method: example





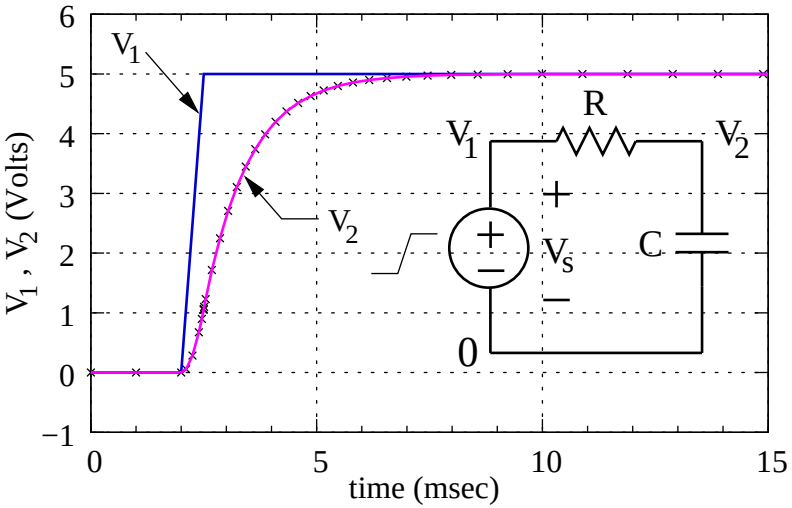
Runge-Kutta-Fehlberg 4/5 method: example

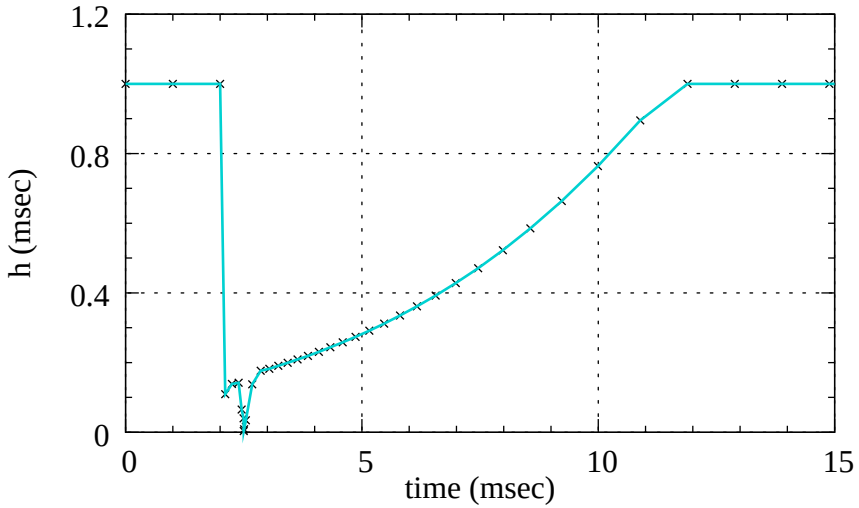




* Parameters: $R = 1$ kΩ, $C = 1$ μF, $h_{\text{min}} = 1$ ns, $h_{\text{max}} = 1$ ms, ϵ (tolerance)=10 mV.

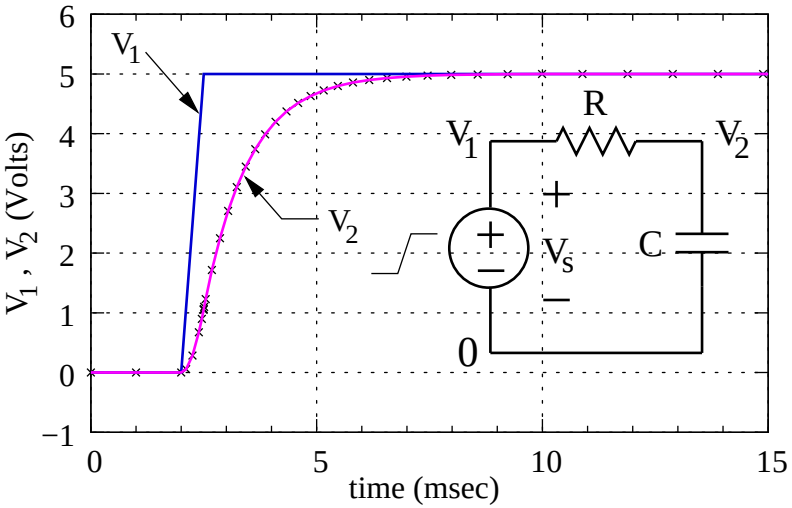
Runge-Kutta-Fehlberg 4/5 method: example

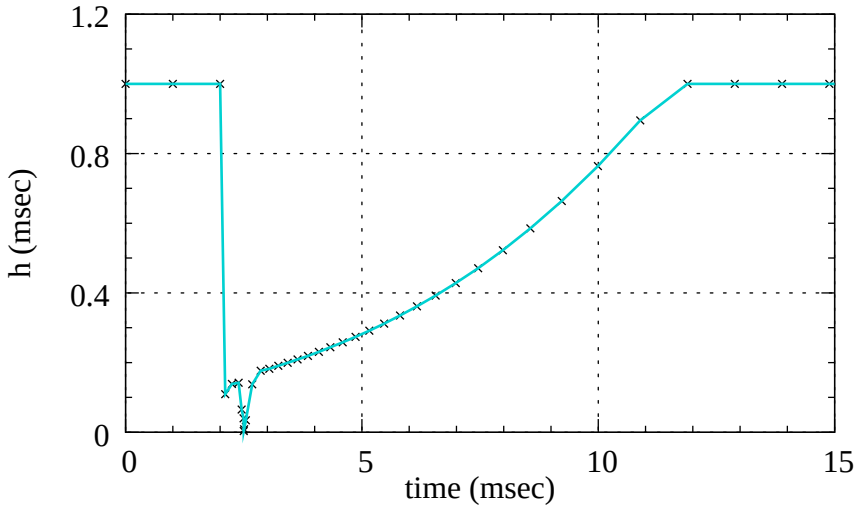




* Parameters: $R = 1$ kΩ, $C = 1$ μF, $h_{\text{min}} = 1$ ns, $h_{\text{max}} = 1$ ms, ϵ (tolerance)=10 mV.
* When the solution is changing rapidly, the time step is made small in order to meet the tolerance requirement.

Runge-Kutta-Fehlberg 4/5 method: example





* Parameters: $R = 1$ kΩ, $C = 1$ μF, $h_{\text{min}} = 1$ ns, $h_{\text{max}} = 1$ ms, ϵ (tolerance)=10 mV.
* When the solution is changing rapidly, the time step is made small in order to meet the tolerance requirement.
* When the solution is changing slowly, the time step is made large (capped by a user-specified h_{max}).

Adaptive step size: convergence of N-R iterations

* The Newton-Raphson process is more likely to converge (in a given number of iterations) if the starting point, i.e., the "initial guess", is close to the solution.

Adaptive step size: convergence of N-R iterations

* The Newton-Raphson process is more likely to converge (in a given number of iterations) if the starting point, i.e., the "initial guess", is close to the solution.
* In transient analysis, $\mathbf{x}_n$ serves as the starting point, and $\mathbf{x}_{n+1}$ is the solution being sought.

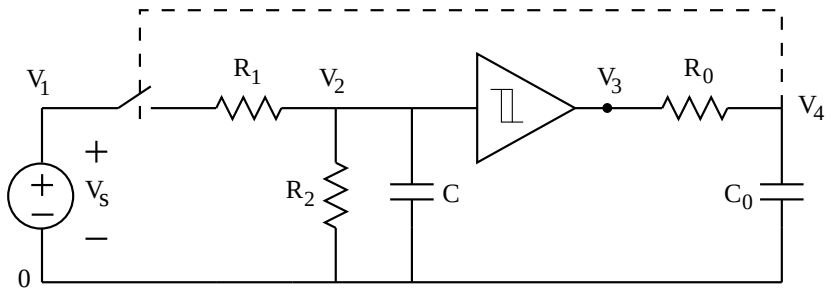
Adaptive step size: convergence of N-R iterations

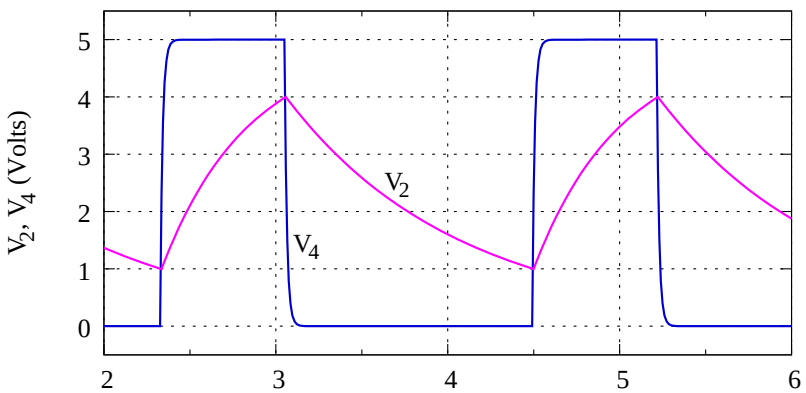
* The Newton-Raphson process is more likely to converge (in a given number of iterations) if the starting point, i.e., the "initial guess", is close to the solution.
* In transient analysis, $\mathbf{x}_n$ serves as the starting point, and $\mathbf{x}_{n+1}$ is the solution being sought.
* If the time step, $(t_{n+1} - t_n)$, is made smaller, the initial guess $\mathbf{x}_n$ is expected to be closer to the solution $\mathbf{x}_{n+1}$, and the N-R process is more likely to converge.

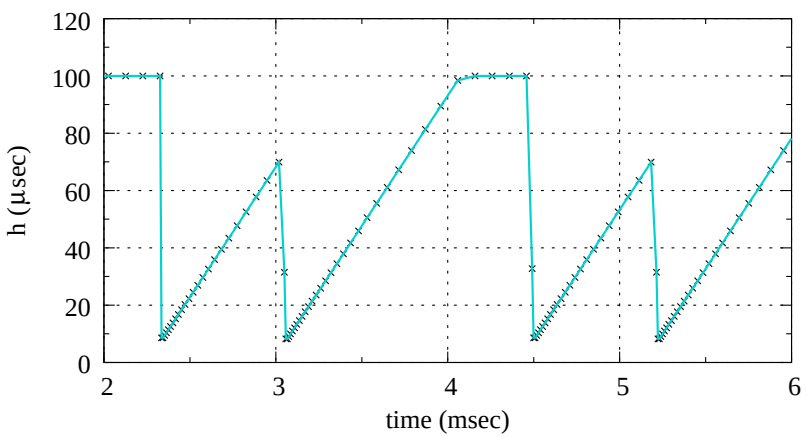
Adaptive step size: convergence of N-R iterations

* The Newton-Raphson process is more likely to converge (in a given number of iterations) if the starting point, i.e., the "initial guess", is close to the solution.
* In transient analysis, $\mathbf{x}_n$ serves as the starting point, and $\mathbf{x}_{n+1}$ is the solution being sought.
* If the time step, $(t_{n+1} - t_n)$, is made smaller, the initial guess $\mathbf{x}_n$ is expected to be closer to the solution $\mathbf{x}_{n+1}$, and the N-R process is more likely to converge.
* The above observation is commonly used in circuit simulation for controlling the time step. (It works only for non-linear problems.)

Adaptive step size: convergence of N-R iterations







Circuit parameters:

$V_s = 10\ V$,

$R_1 = R_2 = 1\ \text{k}\Omega$,

$C = 1\ \mu F$,

$R_0 = 100\ \Omega$,

$C_0 = 0.1\ \mu F$,

$V_{IL} = 1\ V$,

$V_{IH} = 4\ V$,

$V_{OL} = 0\ V$,

$V_{OH} = 5\ V$.

Algorithm parameters:

$\tau = 10^{-12}$,

$h_{\text{min}} = 10^{-9}$ s,

$h_{\text{max}} = 10^{-4}$ s,

$k_{\text{up}} = 1.1$,

$k_{\text{down}} = 0.8$,

$N_{\text{NR}}^{\text{max}} = 10$.

* The method must be zero-stable (i.e., stable for small h).

* The method must be zero-stable (i.e., stable for small h).
* There are small time constants involved in many circuits. Methods which are conditionally stable are not practical since they will require unacceptably small time steps. $\Rightarrow$ The method must be A-stable.

* The method must be zero-stable (i.e., stable for small h).
* There are small time constants involved in many circuits. Methods which are conditionally stable are not practical since they will require unacceptably small time steps. $\Rightarrow$ The method must be A-stable.
* The above considerations severely restrict the choice of methods available: Only AM1 (BE), AM2 (TRZ), BDF2, and implicit Runge-Kutta methods may be used.

* The method must be zero-stable (i.e., stable for small h).
* There are small time constants involved in many circuits. Methods which are conditionally stable are not practical since they will require unacceptably small time steps. $\Rightarrow$ The method must be A-stable.
* The above considerations severely restrict the choice of methods available: Only AM1 (BE), AM2 (TRZ), BDF2, and implicit Runge-Kutta methods may be used.
* Application of the MNA method to circuits would generally yield a set of equations of the following type [7]:

$$\mathbf{F}(\mathbf{x}', \mathbf{x}, \mathbf{y}, t) = 0\,, \tag{37}$$

$$\mathbf{G}(\mathbf{x}, \mathbf{y}, t) = 0\,. \tag{38}$$

Equations in the above form are called "Differential-algebraic equations (DAEs)."

* The method must be zero-stable (i.e., stable for small h).
* There are small time constants involved in many circuits. Methods which are conditionally stable are not practical since they will require unacceptably small time steps. $\Rightarrow$ The method must be A-stable.
* The above considerations severely restrict the choice of methods available: Only AM1 (BE), AM2 (TRZ), BDF2, and implicit Runge-Kutta methods may be used.
* Application of the MNA method to circuits would generally yield a set of equations of the following type [7]:

$$\mathbf{F}(\mathbf{x}', \mathbf{x}, \mathbf{y}, t) = 0\,, \tag{37}$$

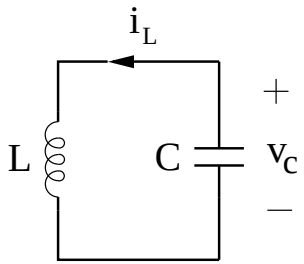
$$\mathbf{G}(\mathbf{x}, \mathbf{y}, t) = 0\,. \tag{38}$$

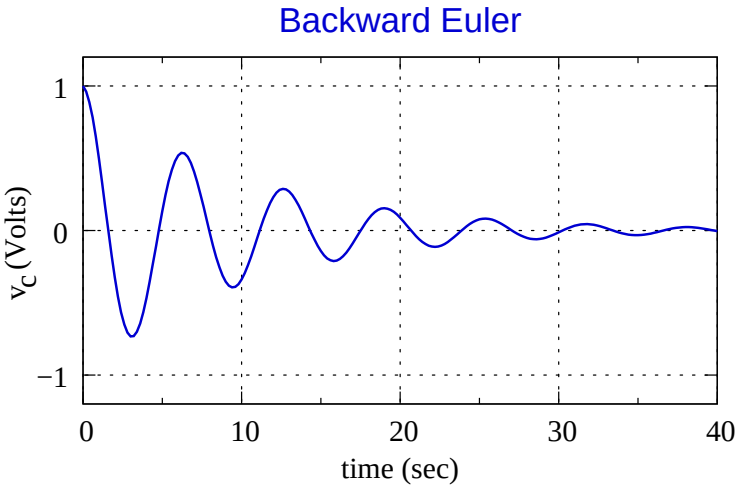
  Equations in the above form are called "Differential-algebraic equations (DAEs)."
* Runge-Kutta methods are not suitable for DAEs. $\Rightarrow$ The choice is further reduced to BE, TRZ, BDF2.

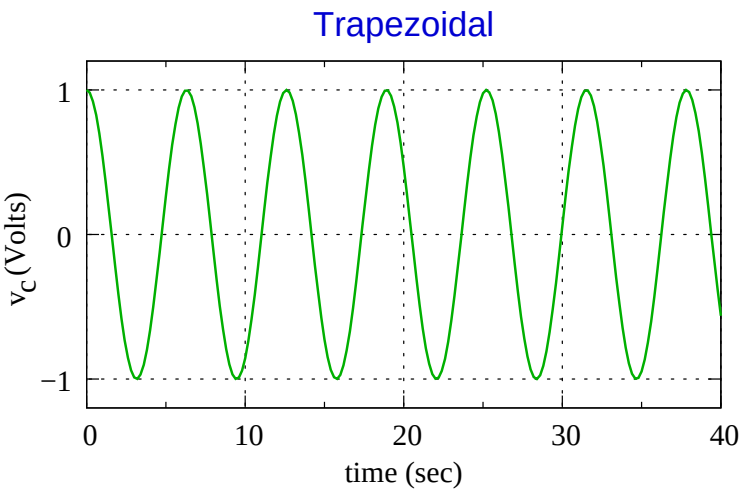
Outline

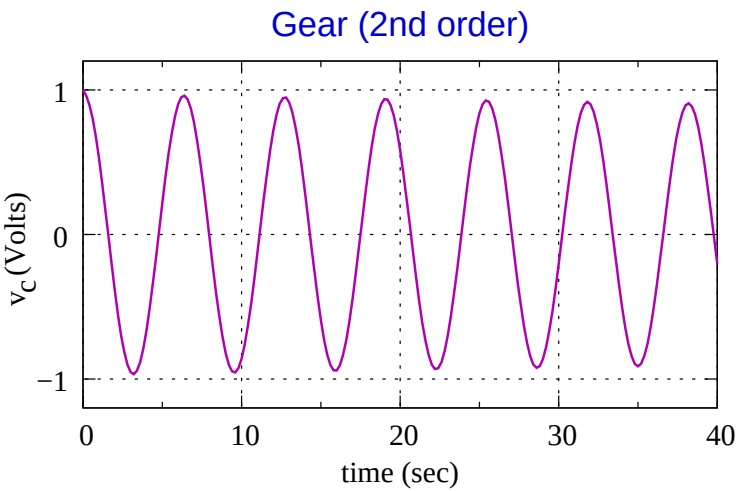
* Introduction and problem definition
* Taylor series methods
* Runge-Kutta methods
* Specific multi-step methods
* Generalized multi-step methods
* Predictor-corrector methods
* Numerical results
* Stability of numerical methods
* Regions of stability
* Stiff equations
* Adaptive step size
* Miscellaneous topics

Undamped oscillations









($L = 1$ H, $C = 1$ F, and h (time step)=0.2 sec in all cases.)

Undamped oscillations

* BE and BDF2 (Gear2) methods introduce artificial damping; they should not be used when there is little or no damping in the circuit.

Undamped oscillations

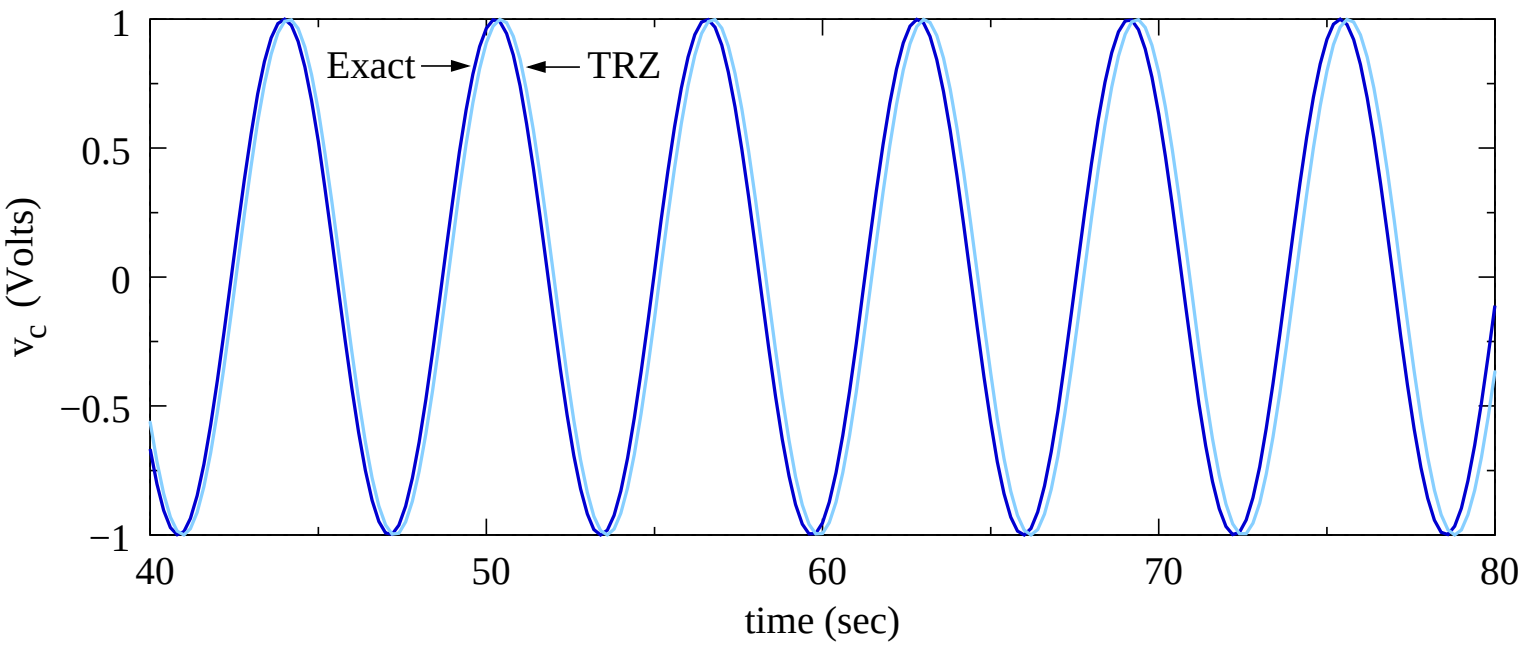
* BE and BDF2 (Gear2) methods introduce artificial damping; they should not be used when there is little or no damping in the circuit.
* TRZ method does not introduce artificial damping.

Undamped oscillations

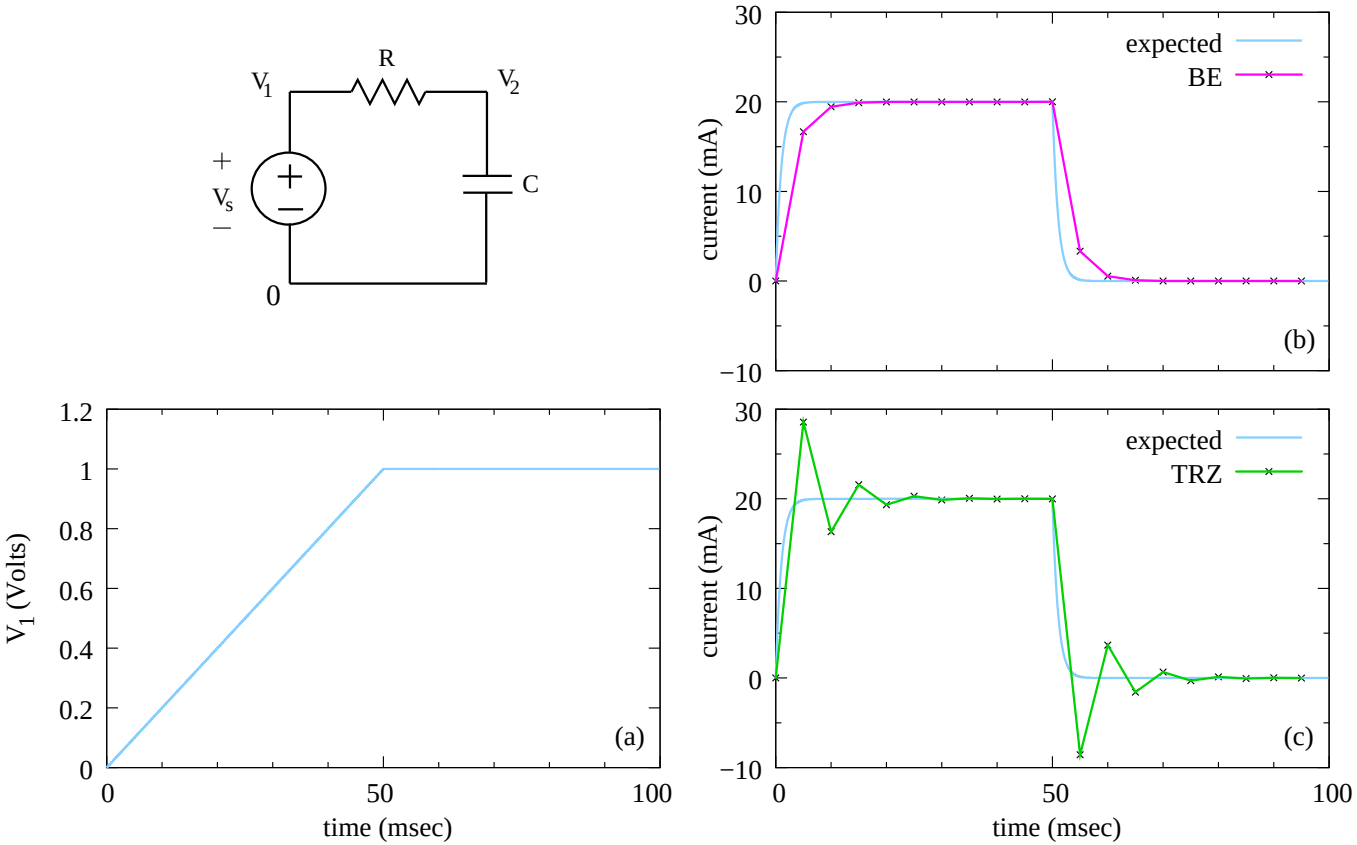
* BE and BDF2 (Gear2) methods introduce artificial damping; they should not be used when there is little or no damping in the circuit.
* TRZ method does not introduce artificial damping.
* However, even the TRZ method is not perfect for purely oscillator problems since it does introduce some phase error. $\Rightarrow$ need to select a small time step.

Undamped oscillations

* BE and BDF2 (Gear2) methods introduce artificial damping; they should not be used when there is little or no damping in the circuit.
* TRZ method does not introduce artificial damping.
* However, even the TRZ method is not perfect for purely oscillator problems since it does introduce some phase error. $\Rightarrow$ need to select a small time step.

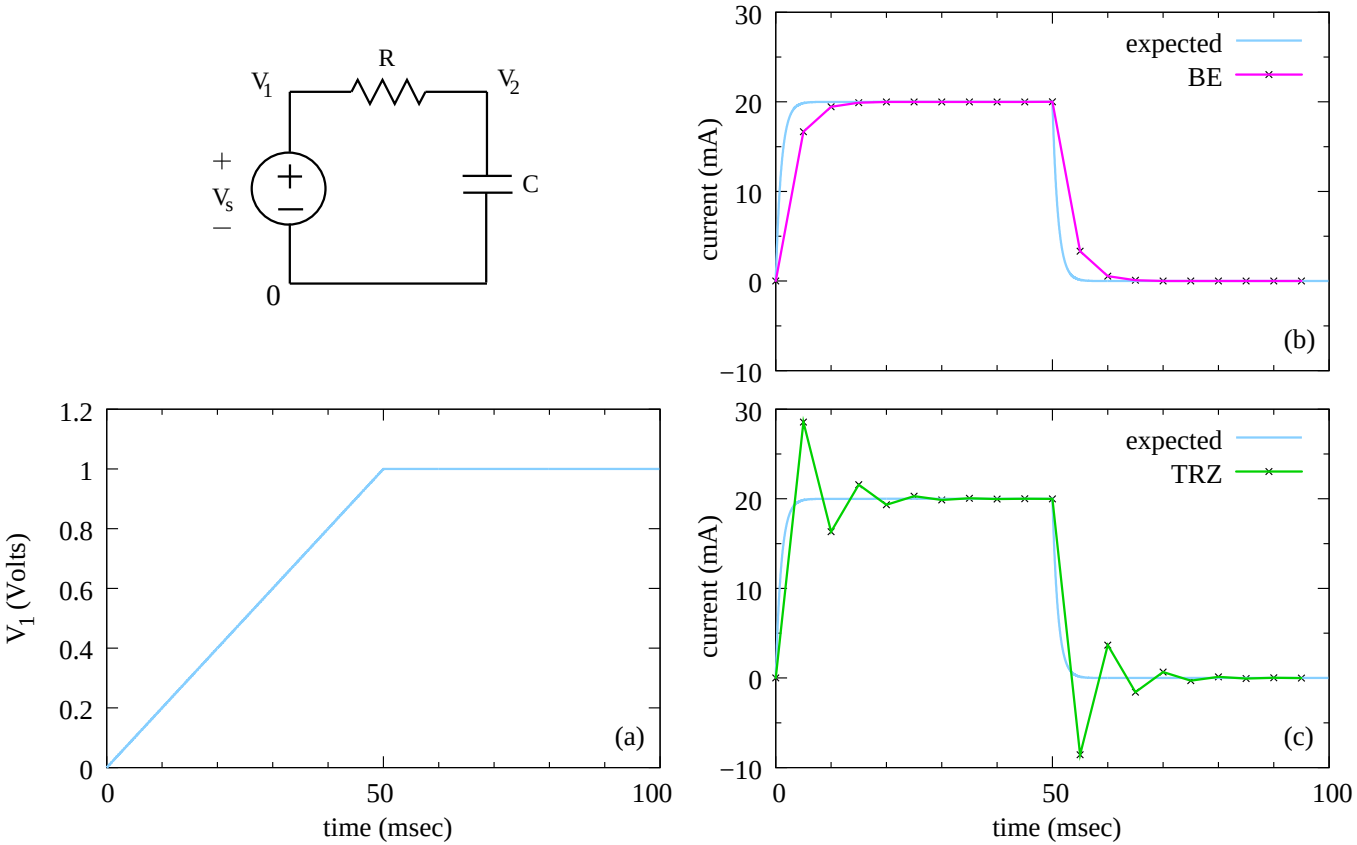


Ringing



(Parameters: $R = 1\ \Omega$, $C = 1$ mF, and h (time step) $= 5$ msec.)

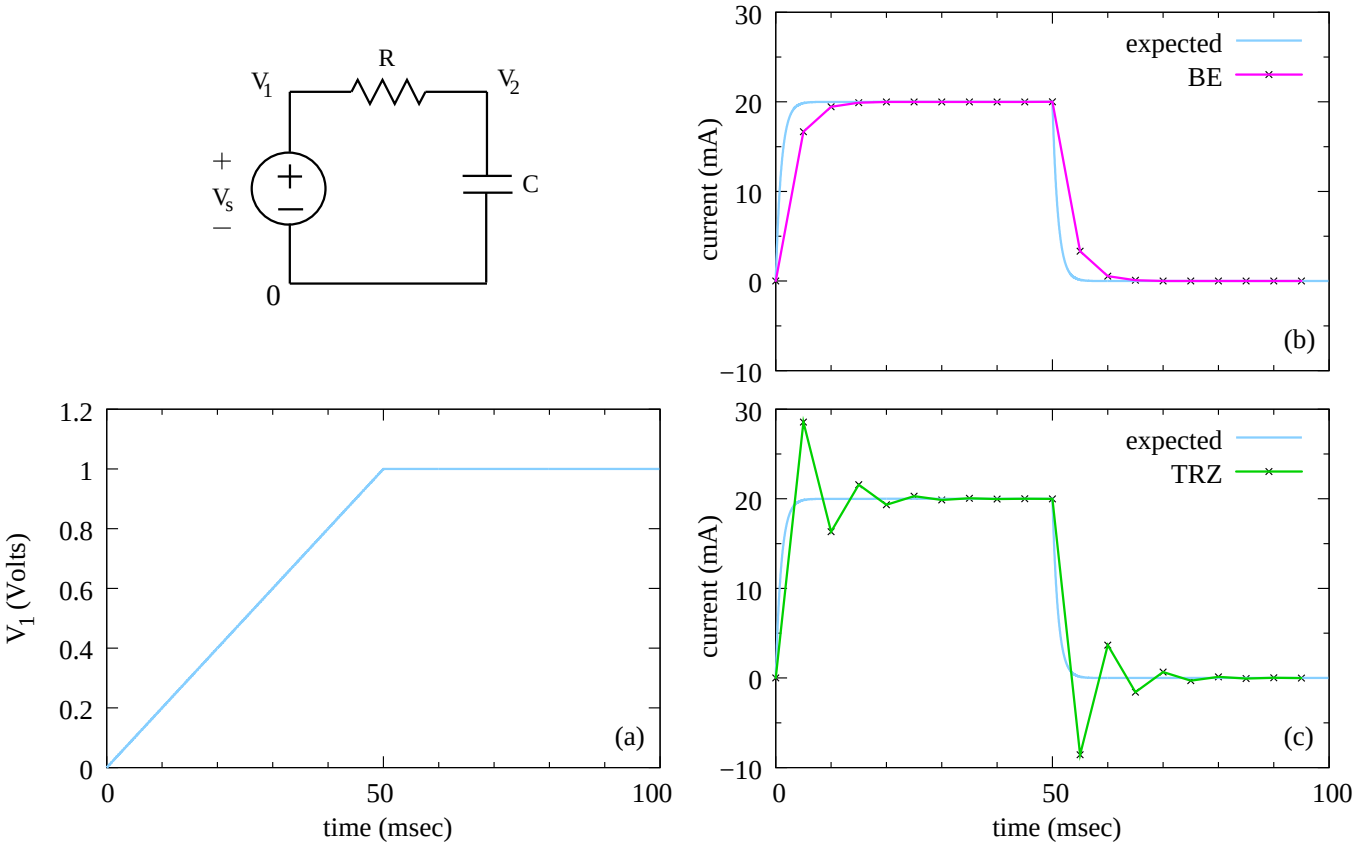
Ringing



(Parameters: $R = 1\ \Omega$, $C = 1$ mF, and h (time step) $= 5$ msec.)

* If h is large, TRZ results in ringing.

Ringing



(Parameters: $R = 1\ \Omega$, $C = 1$ mF, and h (time step) $= 5$ msec.)

* If h is large, TRZ results in ringing.
* Ringing can be reduced by using a smaller time step.

References

[1] R. L. Burden and J. D. Faires, *Numerical Analysis*, Singapore: Thomson, 2001.

[2] M. B. Patil, V. Ramanarayanan, and V. T. Ranganathan, *Simulation of Power Electronic Circuits,* to be published.

[3] C. F. Gerald and P. O. Whitley, *Applied Numerical Analysis*, Delhi: Pearson Education India, 1999.

[4] L. Lapidus and J. H. Seinfeld, *Numerical solution of ordinary differential*, New York: Academic Press, 1971.

[5] L. F. Shampine, *Numerical solution of ordinary differential equations*, New York: Chapman and Hall, 1994.

[6] L. O. Chua and P. M. Lin, *Computer-Aided Analysis of Electronic Circuits*, Englewood Cliffs: Prentice-Hall, 1976.

[7] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations,* New York: North-Holland, 1989.