Simulation of dynamic systems

Numerical solution of ordinary differential equations

Zdeněk Hurák

Advanced Algorithms for Control and Communications (AA4CC)
Department of Control Engineering, Faculty of Electrical Engineering
Czech Technical University in Prague, Prague, Czech Republic

November 26, 2019

Outline

Discretization and Taylor series approximation

Forward and Backward Euler algorithms

Numerical stability

Higher-order methods — single-step methods

Variable integration steps

Stiff systems

Multistep techniques

Problem statement

Find $x(t)$ satisfying

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

where $x(t)$ is a state vector, $u(t)$ is the input vector and the state is known at some (initial) time $t_0 < t$ and

$$x(t_0) = x_0.$$

Discretization in time — Taylor series

Assuming $x(t)$ sufficiently smooth,

$$x(t_0 + h) = x(t_0) + \left.\frac{\mathrm{d}x(t)}{\mathrm{d}t}\right|_{t_0} h + \left.\frac{\mathrm{d}^2 x(t)}{\mathrm{d}t^2}\right|_{t_0} \frac{h^2}{2!} + \dots.$$

Plugging in the nonlinear state space model yields

$$x(t_0 + h) = x(t_0) + f(x, u, t)|_{t_0} h + \left.\frac{\mathrm{d}f(x, u, t)}{\mathrm{d}t}\right|_{t_0} \frac{h^2}{2!} + \dots.$$

In the vector case when $\mathbf{x}(t) \in \mathbb{R}^n$, the expression is rewritten componentwise.

Alternative view of Taylor series description — numerical (approximate) integration

$$x(t_0+h) = x(t_0) + \underbrace{\left.\frac{\mathrm{d}x(t)}{\mathrm{d}t}\right|_{t_0} h + \left.\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}\right|_{t_0} \frac{h^2}{2!} + \ldots.}_{\int_{t_0}^{t_0+h} f(x,u,t)\mathrm{d}t}$$

An idea for the (class of) algorithms — truncate the Taylor series

For example, truncating the Taylor series

$$x(t_0 + h) = x(t_0) + f(x, u, t)|_{t_0} h + \left.\frac{\mathrm{d}f(x, u, t)}{\mathrm{d}t}\right|_{t_0} \frac{h^2}{2!} + \dots$$

after the second term yields

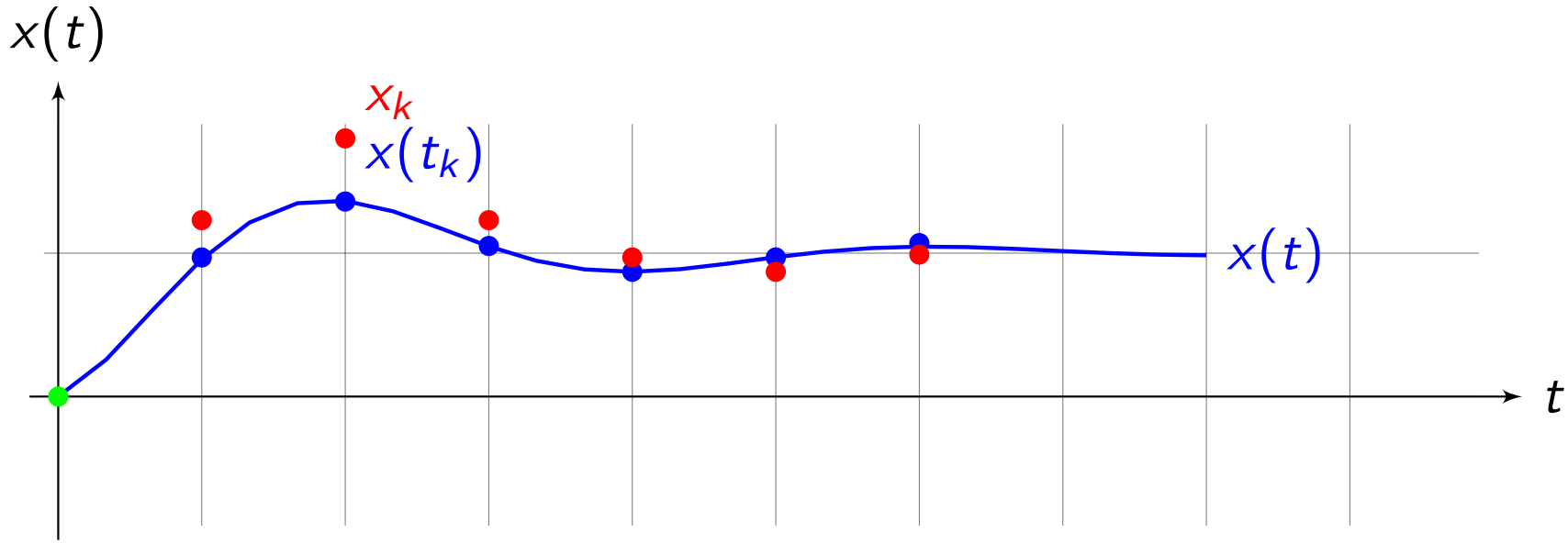
$$x(t_0 + h) \approx x(t_0) + f(x, u, t)|_{t_0} h$$

which suggests our first algorithm

$$x_{k+1} = x_k + f_k h.$$

Notation

$x(t_k)$ is a value of the solution at the (integration) time step t_k.
x_k is an approximation to the true value at the time step t_k.
f_k is an approximation to the true value of $f(x, u, t)$ at t_k.



Very often $t_k = t_0 + kh$, where h is an integration (time) step. But variable steps are actually more common in practice.

Truncation introduces errors, how to characterize them? Big-Oh concept

A function $e(x)$ is said to be $\mathcal{O}(g(x))$ if and only if

$$\lim_{x \to 0} \frac{e(x)}{g(x)} \leq K > 0.$$

Often $g(x)$ is a polynomial and the lowest powers matter most for assymptotic considerations ($x \to 0$). We then say that a function $e(x)$ is $\mathcal{O}(x^n)$ or simply n-th order.
The error introduced by truncating after three terms is $\mathcal{O}(x^3)$

$$x(t_0 + h) = x(t_0) + f(x, u, t)|_{t_0} h + \left. \frac{\mathrm{d}f(x, u, t)}{\mathrm{d}t} \right|_{t_0} \frac{h^2}{2!} + \mathcal{O}(h^3).$$

Beware of a different use in computer science in algorithmic complexity ($n \in \mathrm{Z}, n \to \infty$).

Our first algorithm — Forward Euler

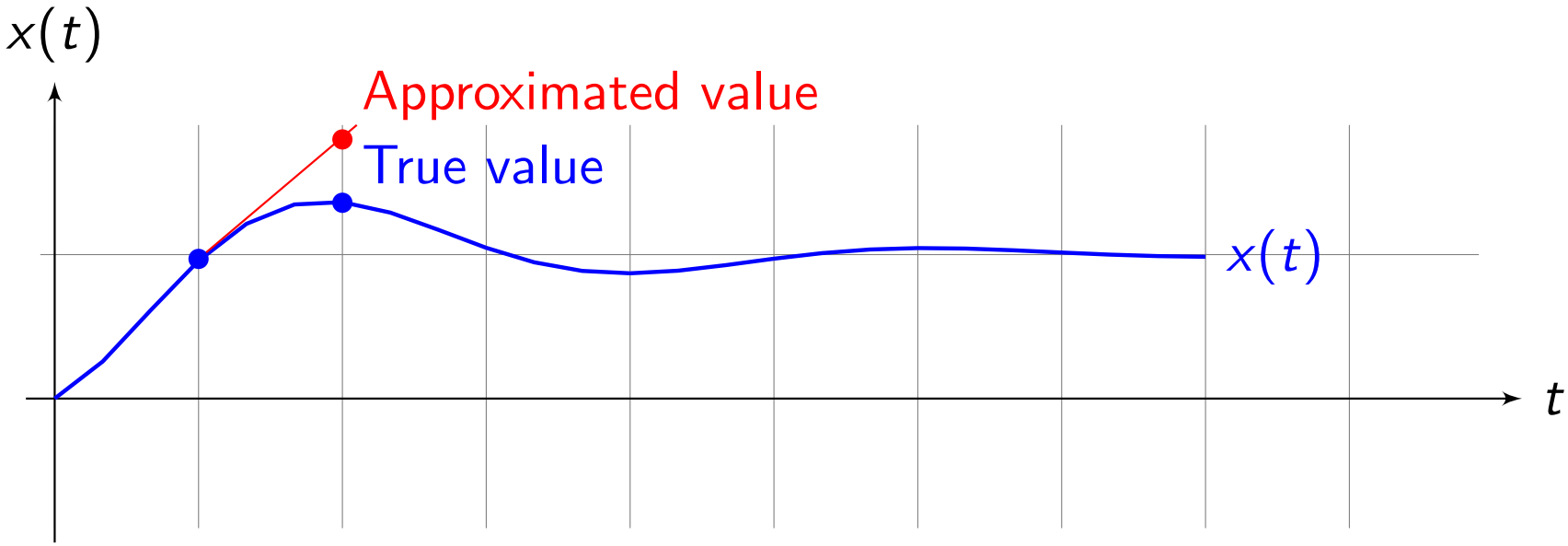
$$\boxed{x_{k+1} = x_k + f_k h}$$

In the linear case $\dot{x}(t) = ax(t)$ the truncated Taylor series turns into

$$\begin{aligned} x(t_0 + h) &\approx x(t_0) + ax(t_0)h \\ &\approx (1 + ah)x(t_0). \end{aligned}$$

(Do not let the linearity-in-x fool you. The derivatives in the Taylor approximation are done with respect to time.) The algorithm is

$$\boxed{x_{k+1} = (1 + ah)x_k}$$


$x(t)$
Approximated value
True value
$x(t)$
t

Pseudocode

Algorithm 1 Forward Euler integration (no input $u(t)$)

Require: $f(x(t), t), \quad t_0, \quad t_f, \quad x(t_0), \quad h$

Ensure: $\dot{x}(t) = f(x(t), t)$

$t_k \leftarrow t_0$

$x_k \leftarrow x(t_0)$

$\dot{x}_k \leftarrow f(x_0, t_0)$

while $t_k < t_f$ **do**

$x_k \leftarrow x_k + h\dot{x}_k$

$t_k \leftarrow t_k + h$

$\dot{x}_k \leftarrow f(x_k, t_k)$

end while

Numerical example

$$\dot{x}(t) = \frac{x(t) - 2tx^2(t)}{1+t}, \quad x(0) = 5$$

We are lucky that the analytical solution(s) can be found:

```
>> dsolve('Dx=(x-2*t*x^2)/(1+t)')
ans =
                    0
 (t + 1)/(t^2 + C2)
```

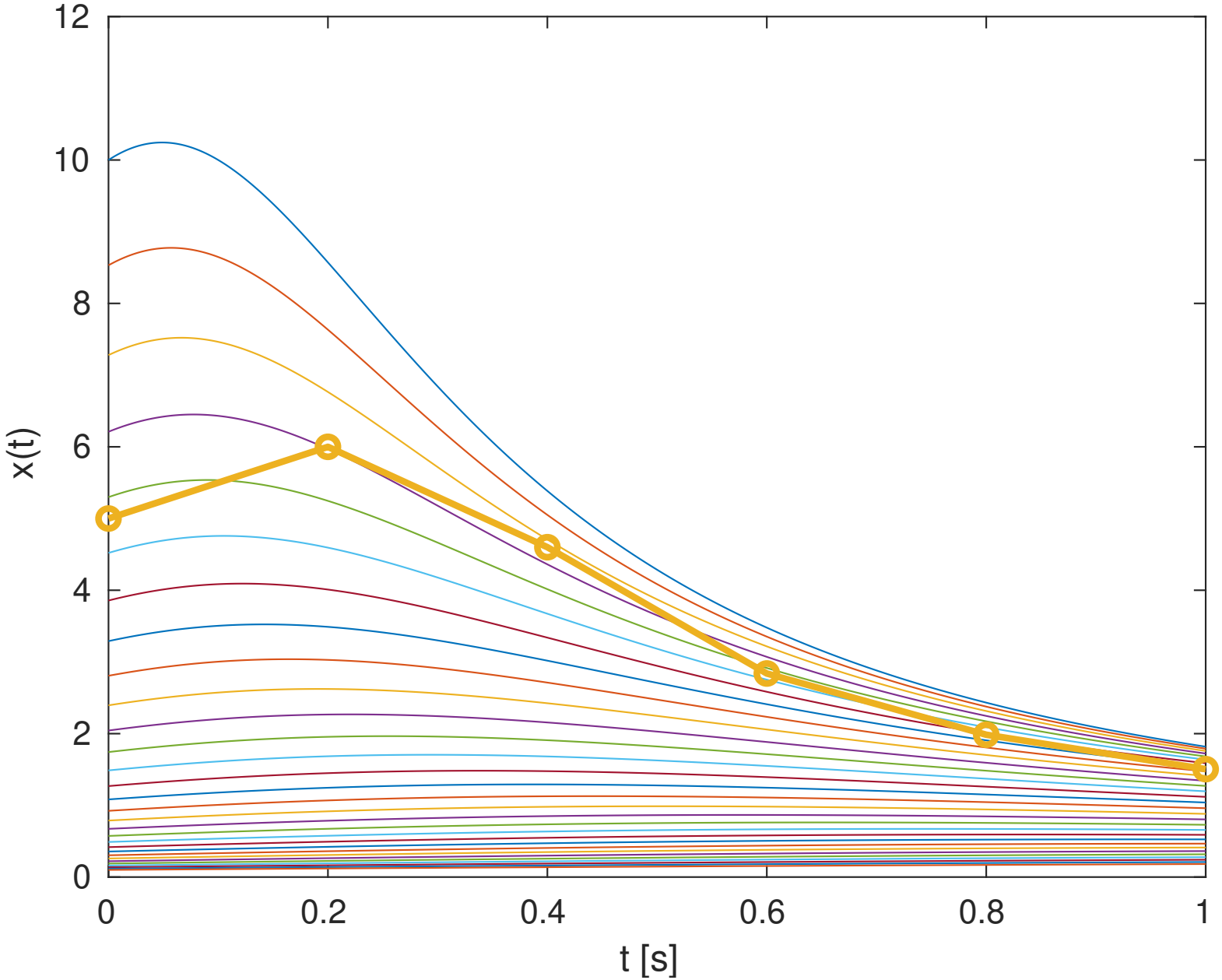
Plot the solutions for $h = 0.2, 0.1, 0.05, 0.025, 0.0125$

Snippet of a Matlab code

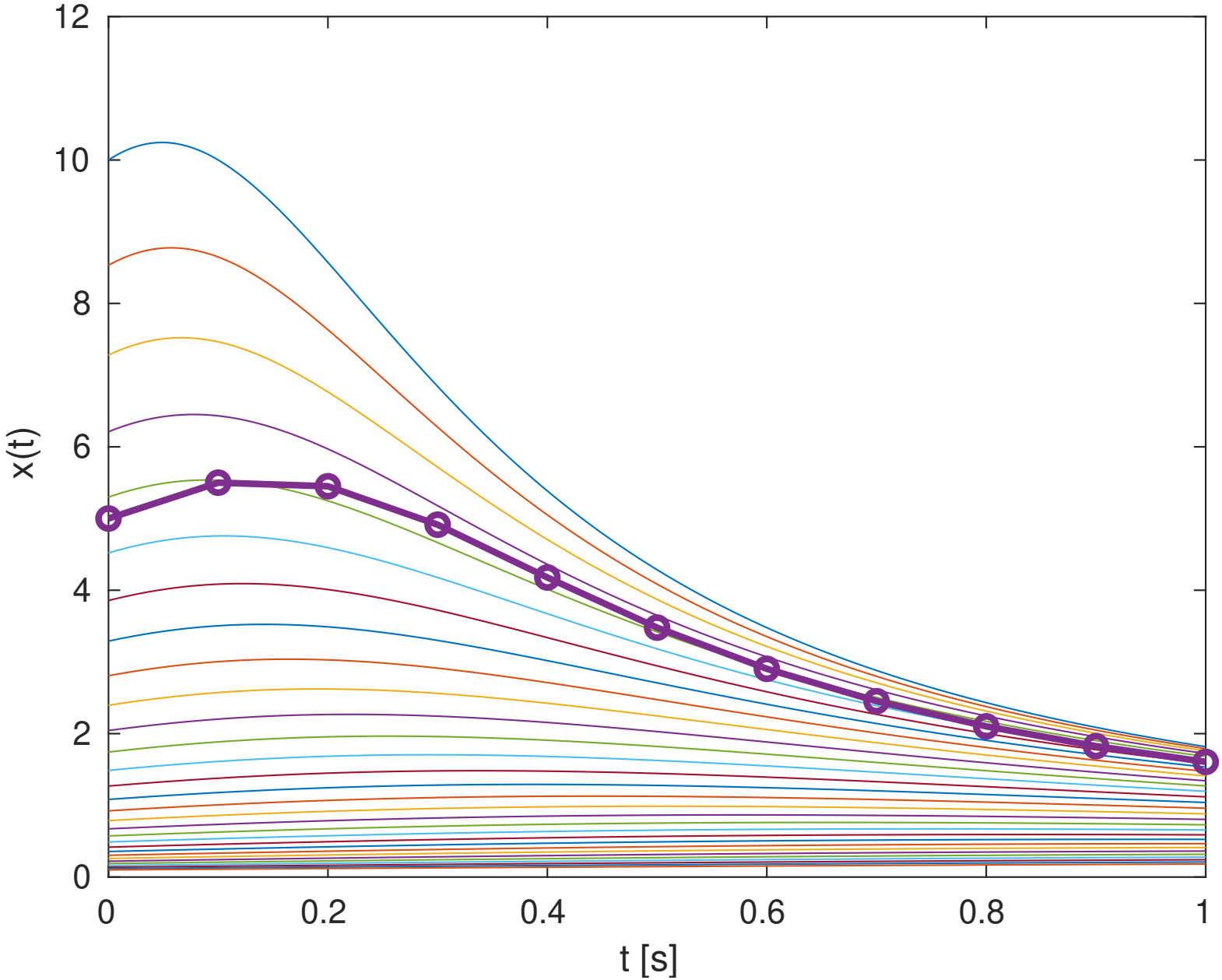
```
h = 0.1;

t = 0;
x = 5;
k = 1;

xtrue = @(t) (1+t)/(t^2+1/5);  % Exact solution

while t <= (tf-h)
    dxdt = (x(k)-2*t*x(k)^2)/(1+t);
    x(k+1) = x(k)+h*dxdt;
    t = t+h;
    error(k) = x(k)-xtrue(t);
    k = k+1;
end

global_error1 = error(end)
```

Note that k starts at 1. Matlab cannot start an integer index at 0.

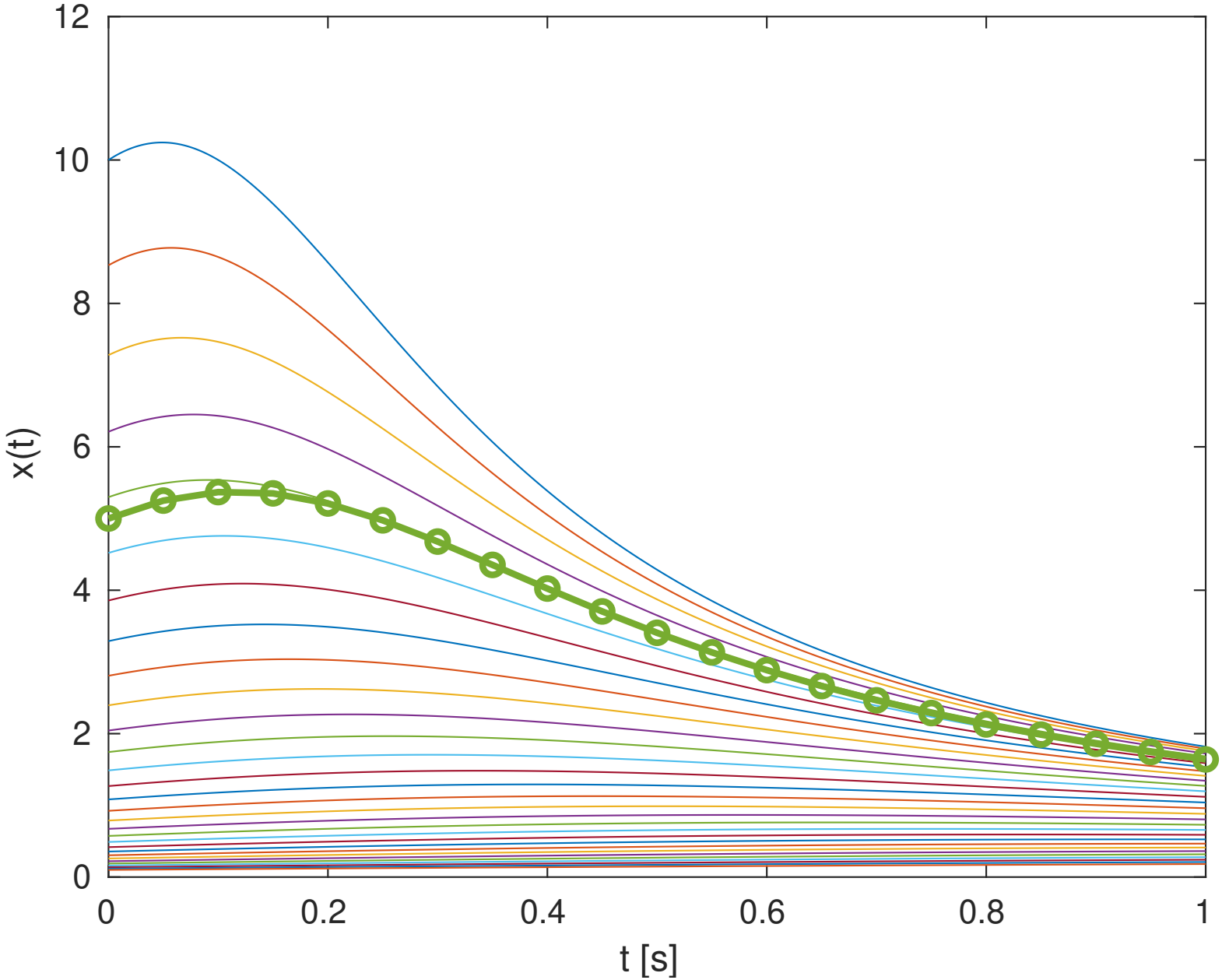
$h = 0.2$



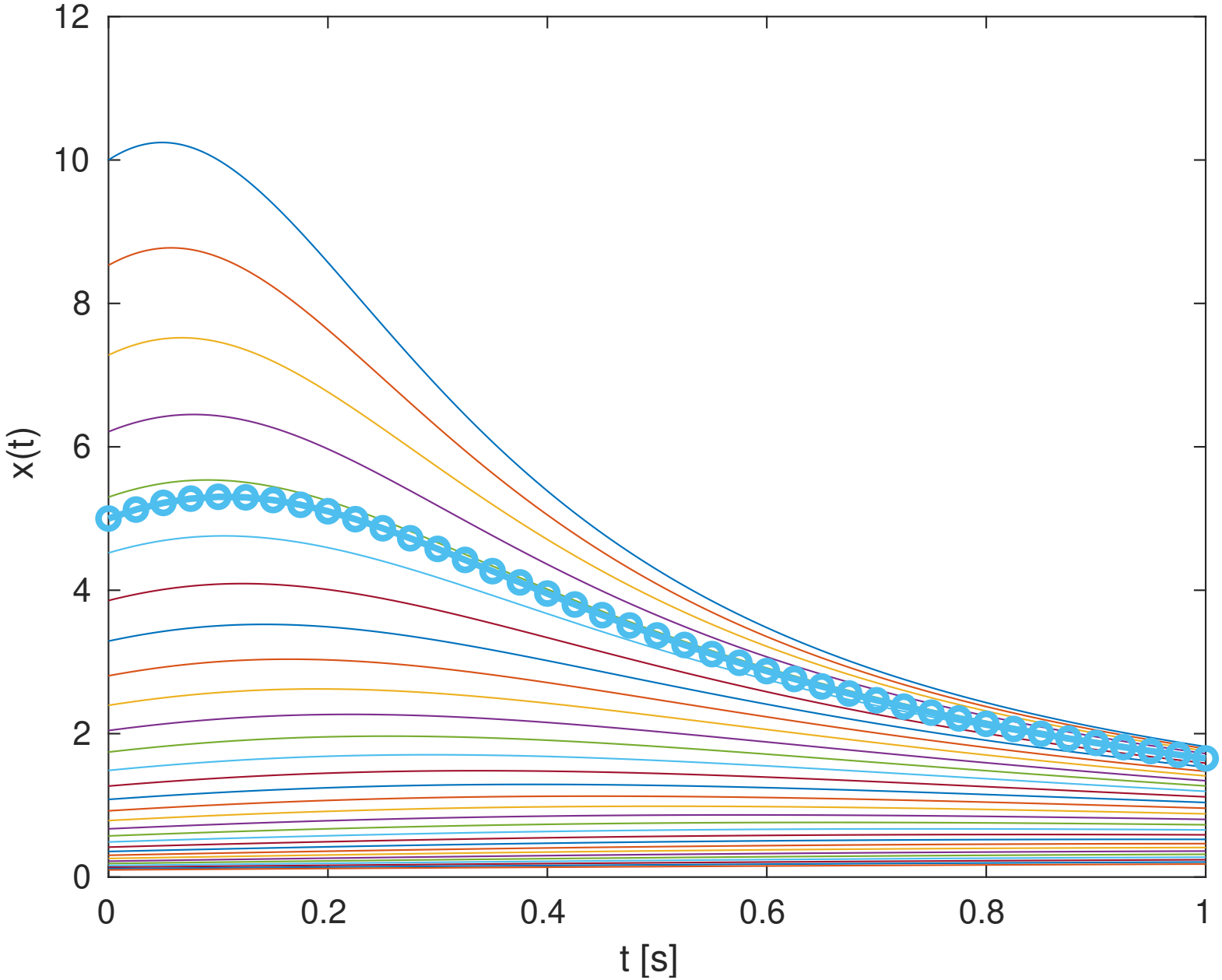
$h = 0.1$



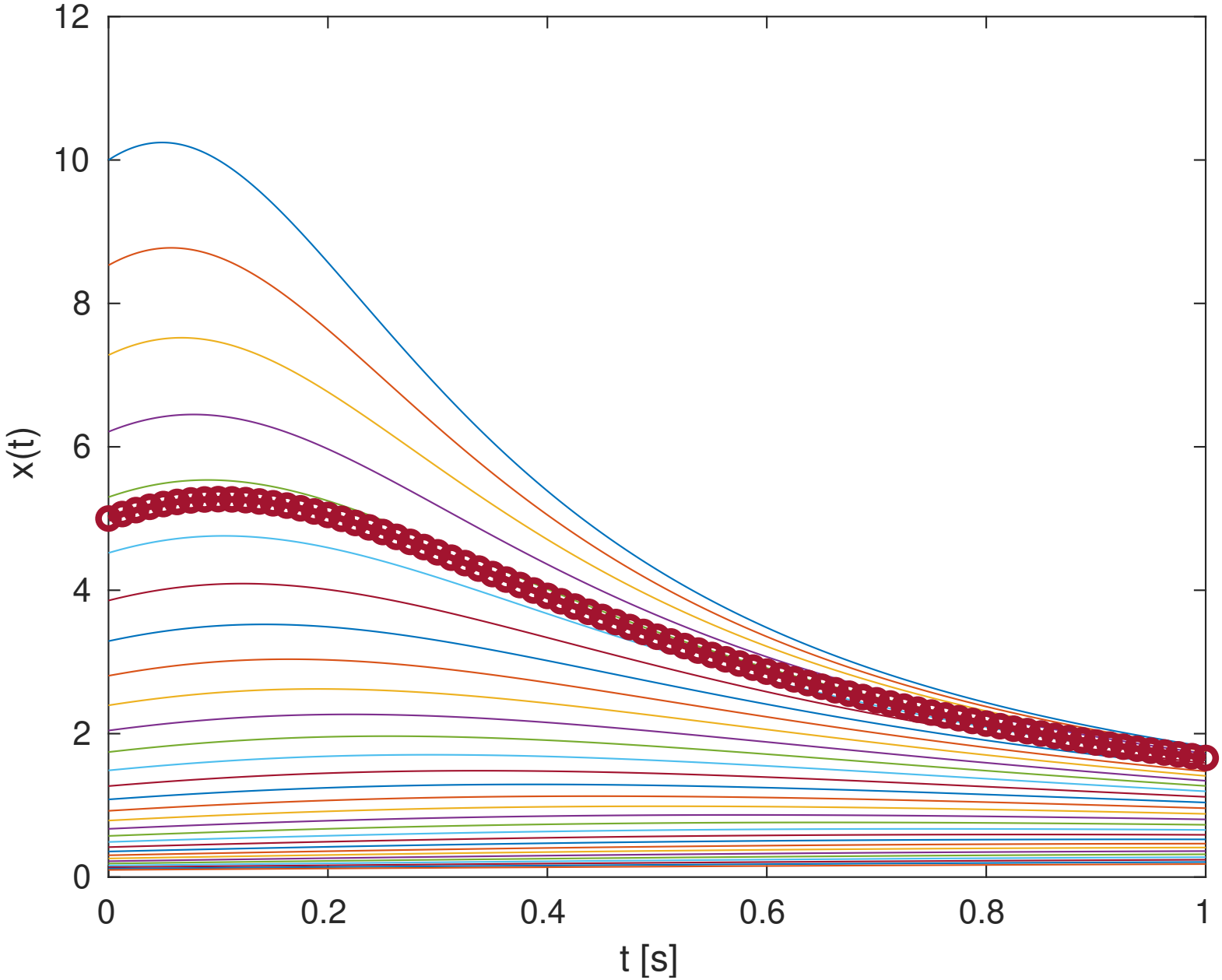
$h = 0.05$



$h = 0.025$



$h = 0.0125$



Analysis of errors

Global error (at the end of simulation interval) vs. Local error (at each step)

h	n	$\lvert x_n - x(t_f)\rvert$
0.2	5	3.1155
0.1	10	0.3956
0.05	20	0.1191
0.025	40	0.0542
0.0125	80	0.0261

As $h \to 0$, the global error $|x_n - x(t_f)|$ is (approximately) halved when h is halved $\Longrightarrow |x_n - x(t_f)| = \mathcal{O}(h) \Longrightarrow$ the FE method is not "cheap".
The findings correspond to the fact that the local error is $\mathcal{O}(h^2)$.

What next? Truncating after higher order?

Say, after the third term (the truncation error is then $\mathcal{O}(h^3)$)

$$x(t_0 + h) \approx x(t_0) + f(x, u, t)|_{t_0} h + \left.\frac{\mathrm{d}f(x, u, t)}{\mathrm{d}t}\right|_{t_0} \frac{h^2}{2!}.$$

But in order to create a functional algorithm, we need to find derivative(s) of $f()$ with respect to time. This can be accomplished by

1. symbolic differentiation
2. numerical approximations (see later in the lecture)

Higher order truncation of Taylor series by symbolic differentiation

Manually only feasible for small problems. Intensive research on automatic differentiation applicable to medium and larger problems.

Our example:

```
syms t x(t)
f(t) = (x(t)-2*t*(x(t))^2)/(1+t)
dfdt = diff(f(t),t)
```

Substituting f_k for $\dot{x}(t)$ and (as before) x_k for $x(t)$

$$\dot{f}_k = \frac{(4x_k + 1)f_k}{t_k + 1} - 4x_k f_k - \frac{x_k(2x_k + 1)}{(t_k + 1)^2}$$

Higher order truncations by numerical approximations

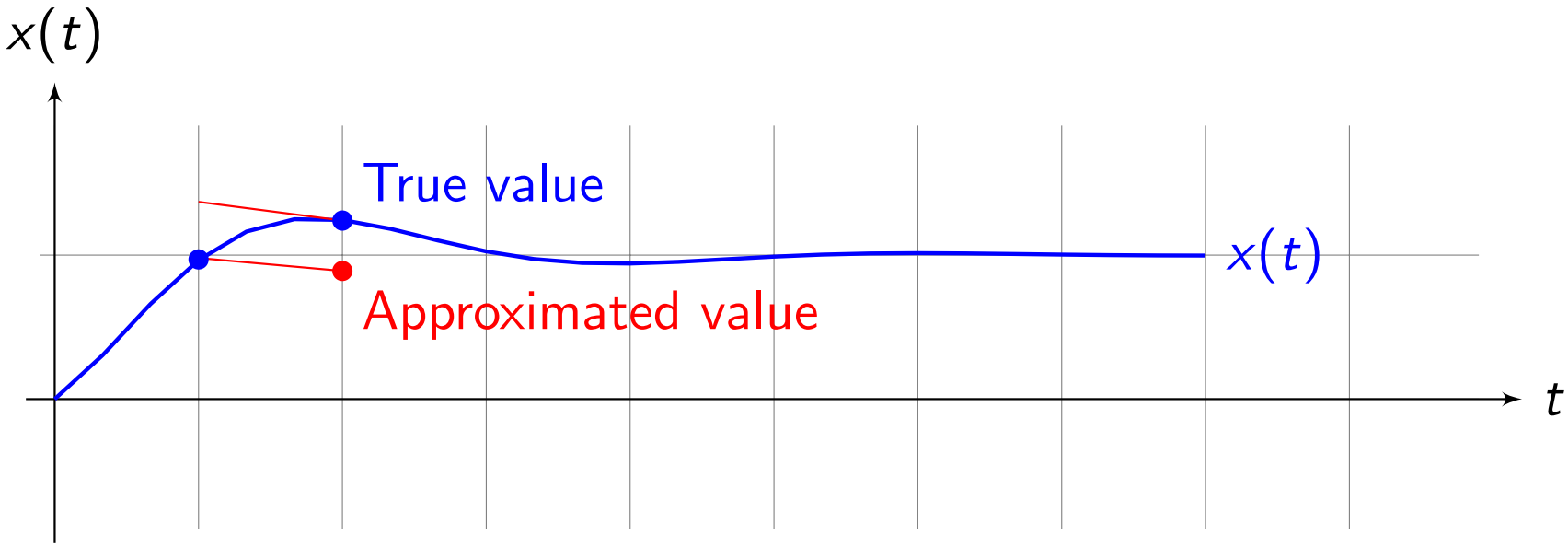
- higher-order single-step methods (Runge-Kutta)
- multistep methods

But now back to the second-order methods for a while.

Backward Euler method

Why using the "slope" at the beginning of the interval? Why not the one at the end?

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



In order to find x_{k+1} we need to know x_{k+1} ...

Backward Euler for a linear system and regular sampling

For a linear system and a regular sampling

$$x_{k+1} = x_k + hAx_{k+1} \quad \Longrightarrow \quad (I - Ah)x_{k+1} = x_k$$

Could be solved by a matrix inverse

$$x_{k+1} = \underbrace{(I - Ah)^{-1}}_{F} x_k.$$

but numerically not reliable and efficient. Use the backslash operator in Matlab instead to solve a set of equations.

Backward Euler method is implicit — calls for solving sets of equations

The neeed to solve (a set of) equations, in general nonlinear (algebraic).

Pseudocode

Algorithm 2 Backward Euler integration for a linear system (no input $u(t)$)

Require: $A, \quad t_0, \quad t_f, \quad x(t_0), \quad h$
Ensure: $\dot{x}(t) = Ax(t)$

$t_k \leftarrow t_0$
$x_k \leftarrow x(t_0)$
while $t_k < t_f$ **do**
  solve $(I - Ah)x_{k+1} = x_k$ for the unknown x_{k+1}
  $t_k \leftarrow t_k + h$
  $x_k \leftarrow x_{x+1}$
end while

Other options for second-order methods

- evaluate/approximate the slope (derivative) in the middle of the interval
- average the FE and BE derivatives

Write down the two algorithms on your own and experiment with them.

One general method for solving nonlinear equations — Fixed-point iterations

Solve

$$\boxed{x = g(x)}$$

Existence and uniqueness given by *contractivity* of $g(x)$. See *Fixed point theorem* or *Fixed point principle* or *Contractive mapping theorem*.
Algorithm: Substitute an initial guess x_0 into $g()$

$$x_1 = g(x_0)$$

and repeat. . .

Solving equations in BE methods — predictor-corrector scheme

Application to BE:

- the initial guess provided by FE — predictor stage
- the iterative improvement provided by BE — corrector stages

The right side is contractive provided $f()$ is *Lipschitz* (say, well-behaved...).

Algorithm 3 Predictor-corrector Backward Euler integration

Require: $f(x(t)), \quad t_0, \quad t_f, \quad x_0, \quad h, \quad \varepsilon$

Ensure: $\dot{x}(t) = f(x(t)), \quad x(t_0) = x_0$

$t_k \leftarrow t_0$

$x_k \leftarrow x(t_0)$

while $t < t_f$ **do**

$\dot{x}_k \leftarrow f(x_k, t_k)$

$x_{k+1}^P \leftarrow x_k + h\dot{x}_k$ {Predictor}

$\dot{x}_{k+1}^P \leftarrow f(x_{k+1}^P, t_k + h)$

$x_{k+1}^{C_1} \leftarrow x_k + h\dot{x}_{k+1}^P$ {1^{st} corrector}

$i \leftarrow 1$

while $e > \varepsilon$ **do**

$\dot{x}_{k+1}^{C_i} \leftarrow f(x_{k+1}^{C_i}, t_k + h)$

$x_{k+1}^{C_i} \leftarrow x_k + h\dot{x}_{k+1}^{C_i}$ {i^{th} corrector}

$e \leftarrow x^{C_i} - x^{C_{i-1}}$

$i \leftarrow i + 1$

end while

$x_k \leftarrow x_k^{C_i}$

$t_k \leftarrow t_k + h$

end while

PC scheme for a linear system

The sequence of matrices F in the discrete-time system $x(k+1) = Ax(k)$ is

$$
\begin{aligned}
F^P &= I + Ah \\
F^{C_1} &= I + Ah + (Ah)^2 \\
F^{C_2} &= I + Ah + (Ah)^2 + (Ah)^3 \\
F^{C_3} &= I + Ah + (Ah)^2 + (Ah)^3 + (Ah)^4
\end{aligned}
$$

For an infinite number of iterations this yields

$$F = I + Ah + (Ah)^2 + (Ah)^3 + (Ah)^4 + \ldots,$$

which can be shown to be equal to

$$F = (I - Ah)^{-1}.$$

Newton iterations for BE

Classical Newton method solves

$$g(x) = 0,$$

The i-th iteration is

$$\boxed{x^{i+1} = x^i - \frac{g(x^i)}{\dot{g}(x^i)}}.$$

The nonlinear equation to be solved in every iteration of BE is

$$x_{k+1} = x_k + hf(x_{k+1}, t_{k+1})$$

or, in the format ready for application of Newton method

$$x_k + hf(x_{k+1}, t_{k+1}) - x_{k+1} = 0.$$

The unknown variable is x_{k+1} and the Newton iteration is

$$\boxed{x_{k+1}^{i+1} = x_{k+1}^{i} - \frac{x_k + hf(x_{k+1}^{i}, t_{k+1}) - x_{k+1}^{i}}{h \left.\frac{\partial f(x,t)}{\partial x}\right|_{x_{k+1}^{i}, t_{k+1}} - 1.0}.}$$

Vector version of Newton iterations for BE

Replace the scalar $\frac{\mathrm{d}f(x)}{\mathrm{d}x}$ by its matrix counterpart — Jacobian matrix

$$J = \frac{\mathrm{d}f(x)}{\mathrm{d}x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}.$$

$$x_{k+1}^{i+1} = x_{k+1}^{i} - \left[hJ_{k+1}^{i} - I\right]^{-1} \cdot \left[x_k + hf(x_{k+1}^{i}) - x_{k+1}^{i}\right]$$

Jacobian for a linear model is just the matrix A; the vector version of the algorithms is then

$$\begin{aligned} x_{k+1}^{i+1} &= x_{k+1}^{i} - [Ah - I]^{-1} \cdot [x_k + (hA - I)x_{k+1}^{i}] \\ &= [I - Ah]^{-1} x_k \end{aligned}$$

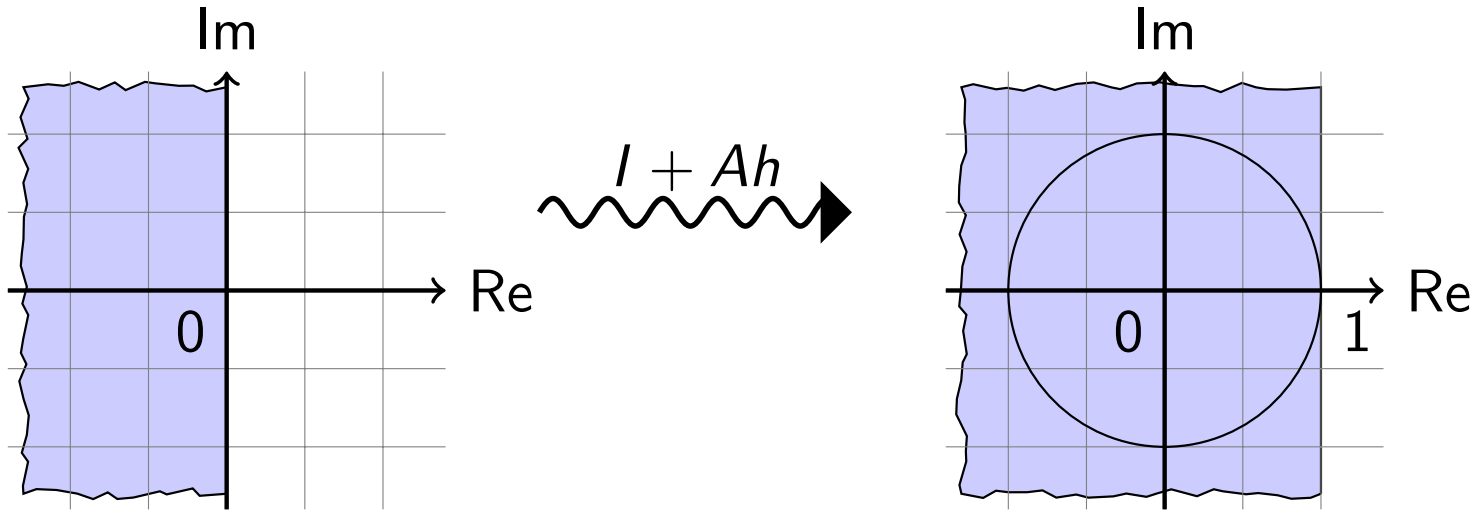
And we see that in the linear case the behavior of BE is revoked.

Numerical stability

Numerical stability = for slightly perturbed data only slightly perturbed simulation outcomes.
In control systems language: Is the discretized system stable when the original continuous-time system is stable?
Analytical results available only for linear systems.

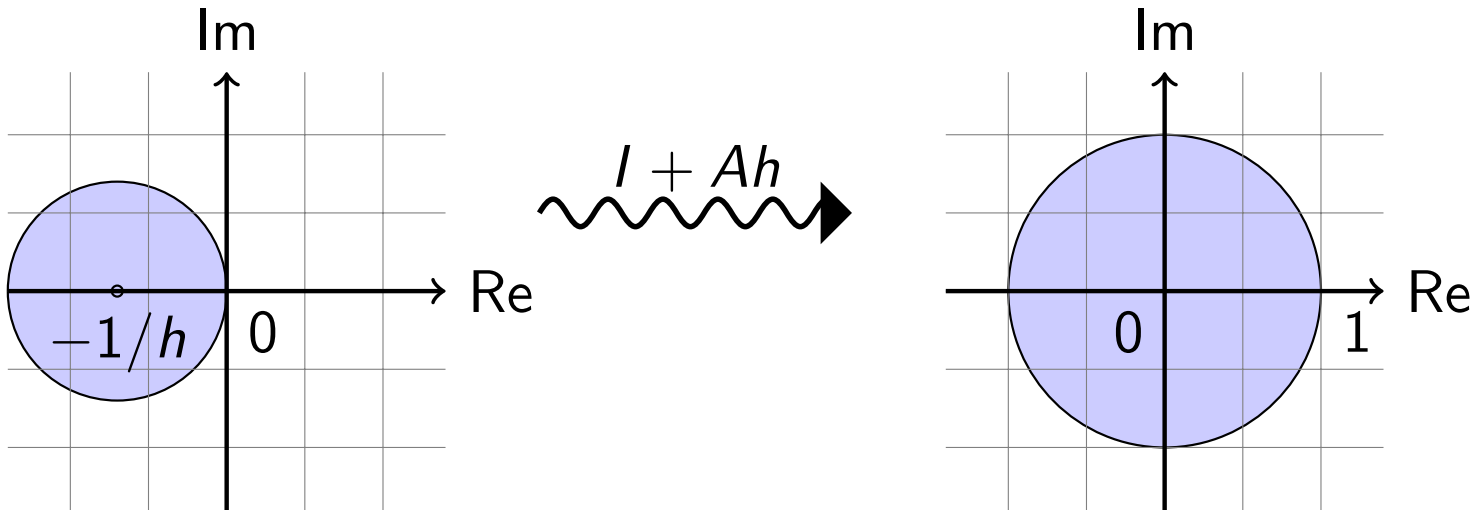
Stability for FE

The original continuous-time system $\dot{x}(t) = Ax(t)$. The discrete-time model $x(k+1) = Fx(k)$, where $F = I + Ah$.



Obviously some continous-time systems are unstable after discretization.

Domain of stability for FE — preimages of stable solutions



More often visualized not in the λ-plane but in the "normalized" $h\lambda$-plane. Does not matter.

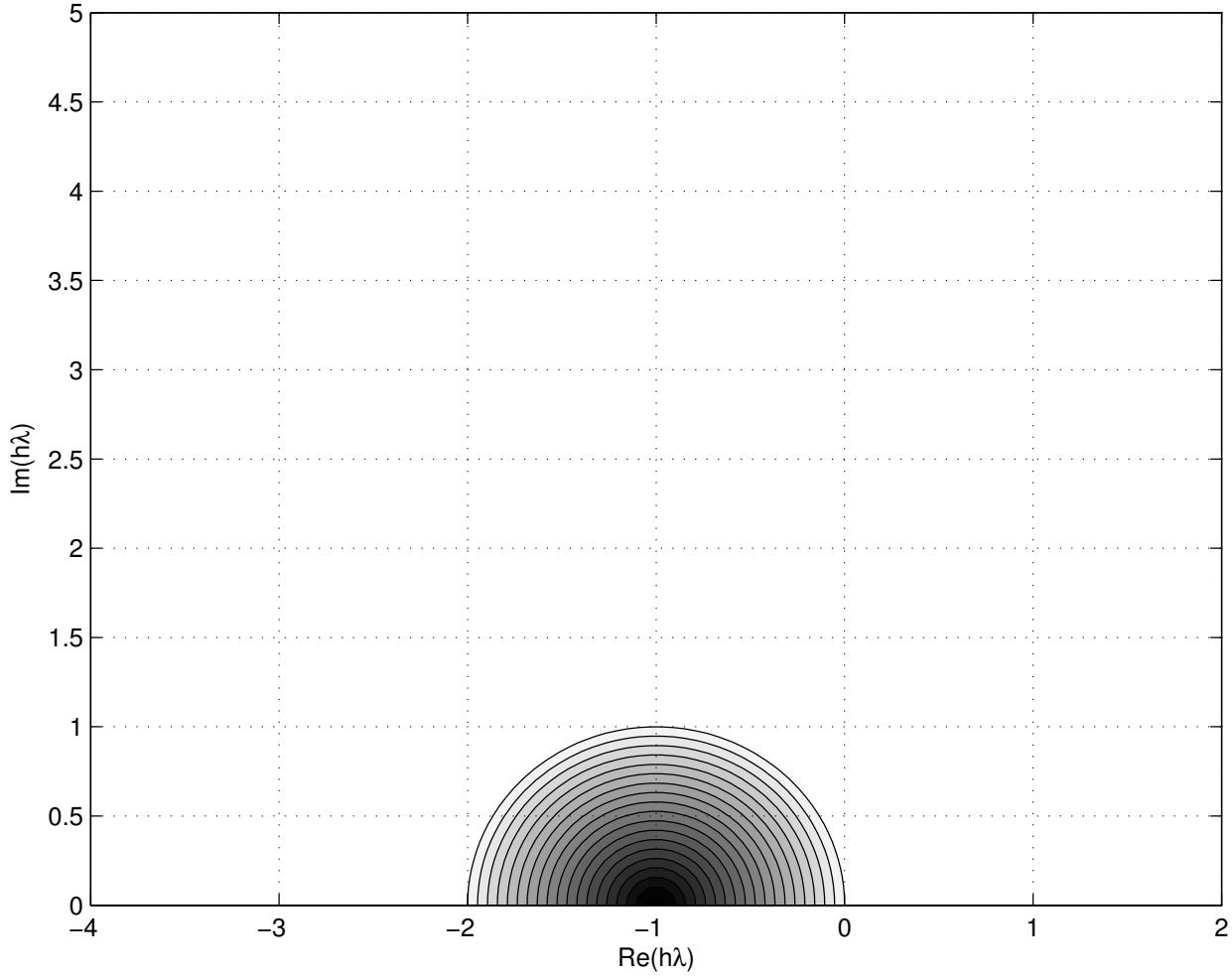
Finding the borders of stability domain numerically

Byl plotting the contours for spectral radius $\rho(F)$

```
h = 1; I = eye(2,2);

a = linspace(-4,2,100); % range of real values
b = linspace(0,5,100);     % range of imaginary values

rmax = zeros(length(a),length(b));

for ia = 1:length(a)
    for ib = 1:length(b)
        A = [a(ia) b(ib); -b(ib) a(ia)];
        F = I + A*h;
        rmax(ia,ib) = max(max(abs(eig(F))));
    end
end

[A,B] = meshgrid(a,b);

colormap(gray)
contourf(A,B,rmax',linspace(0,1,20))
xlabel('Re(h\lambda)'), ylabel('Im(h\lambda)')
axis([a(1) a(end) b(1) b(end)]), grid on
```

Could reuse by substituting a different F.

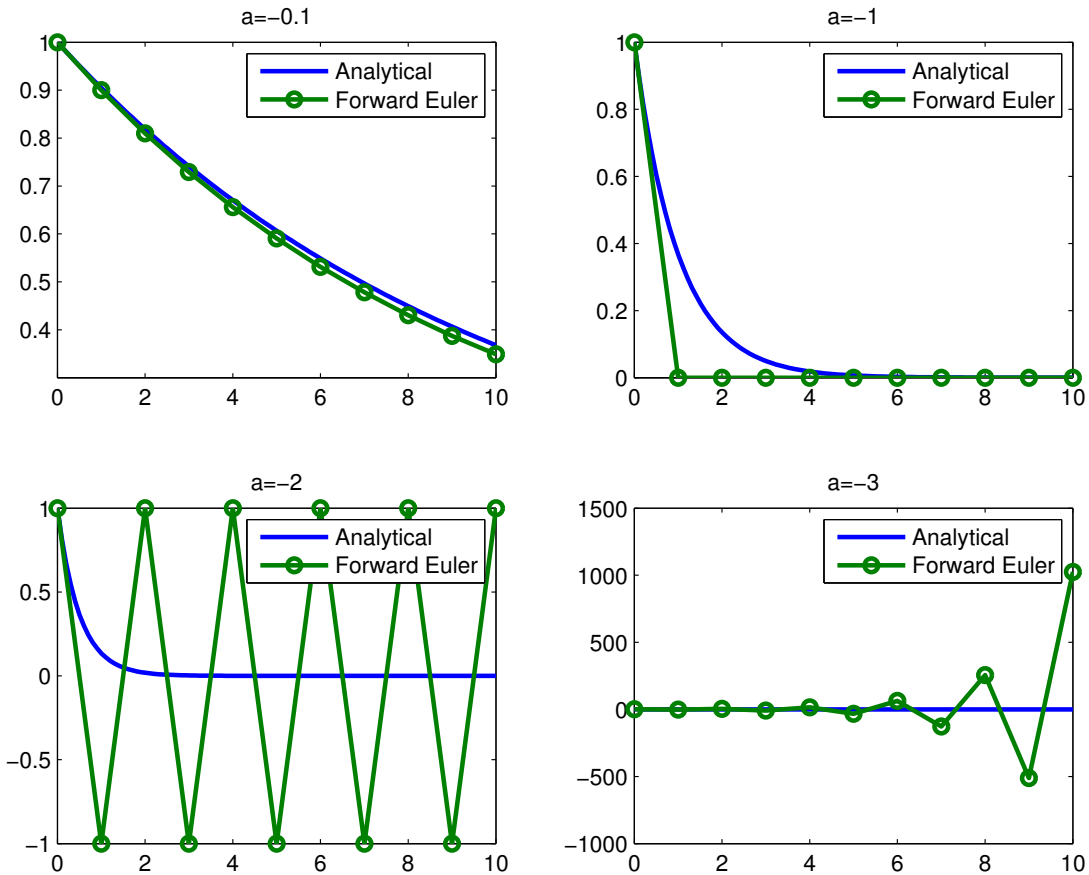
Plotting only the upper half-plane.



Conforms to the analytical findings.

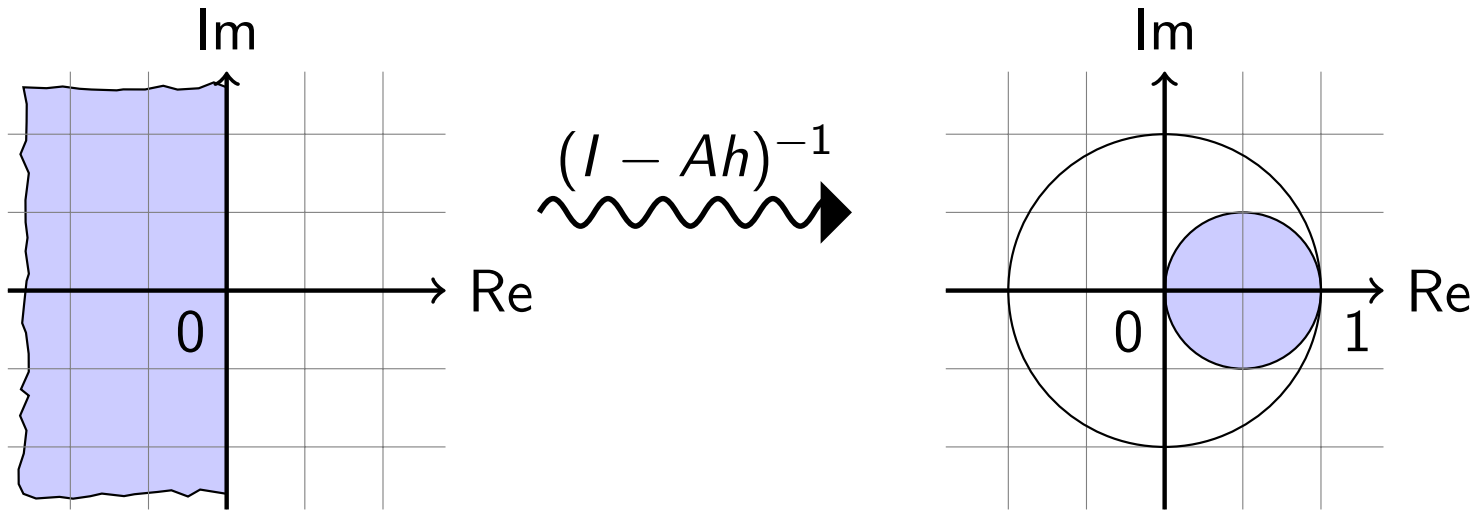
Numerical experiment for FE

$$\dot{x}(t) = ax(t), \quad x(0) = 1, \quad a = -0.1, -1.0, -2.0, -3.0, \quad h = 1.$$

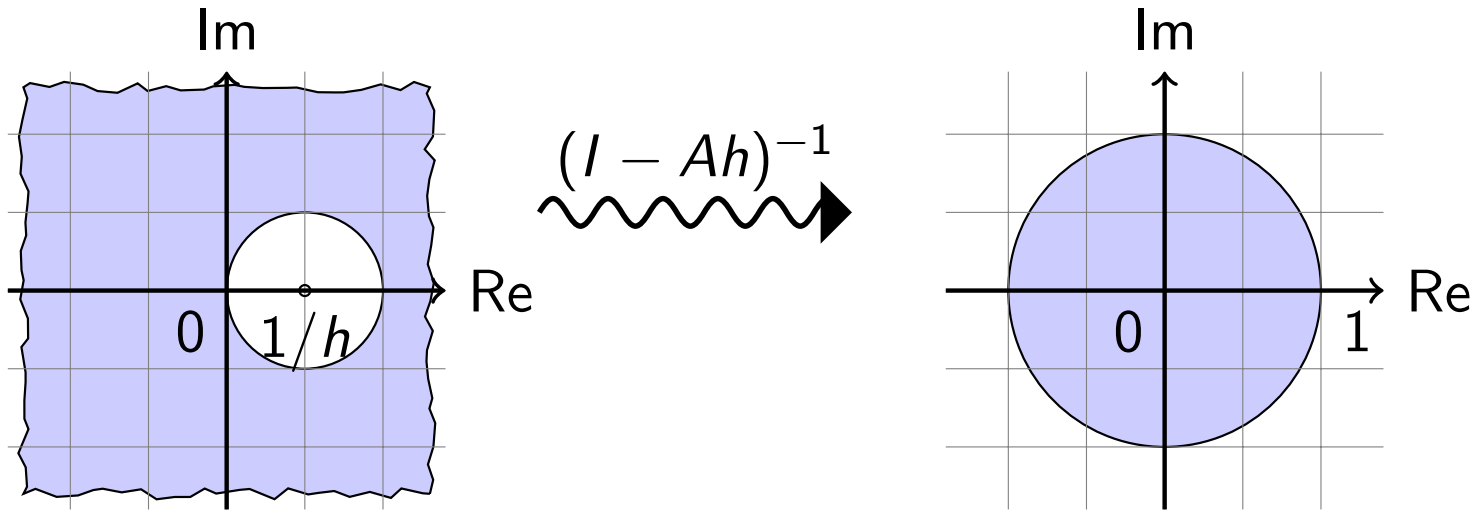


FE inappropriate for "fast but stable" or little damped systems.

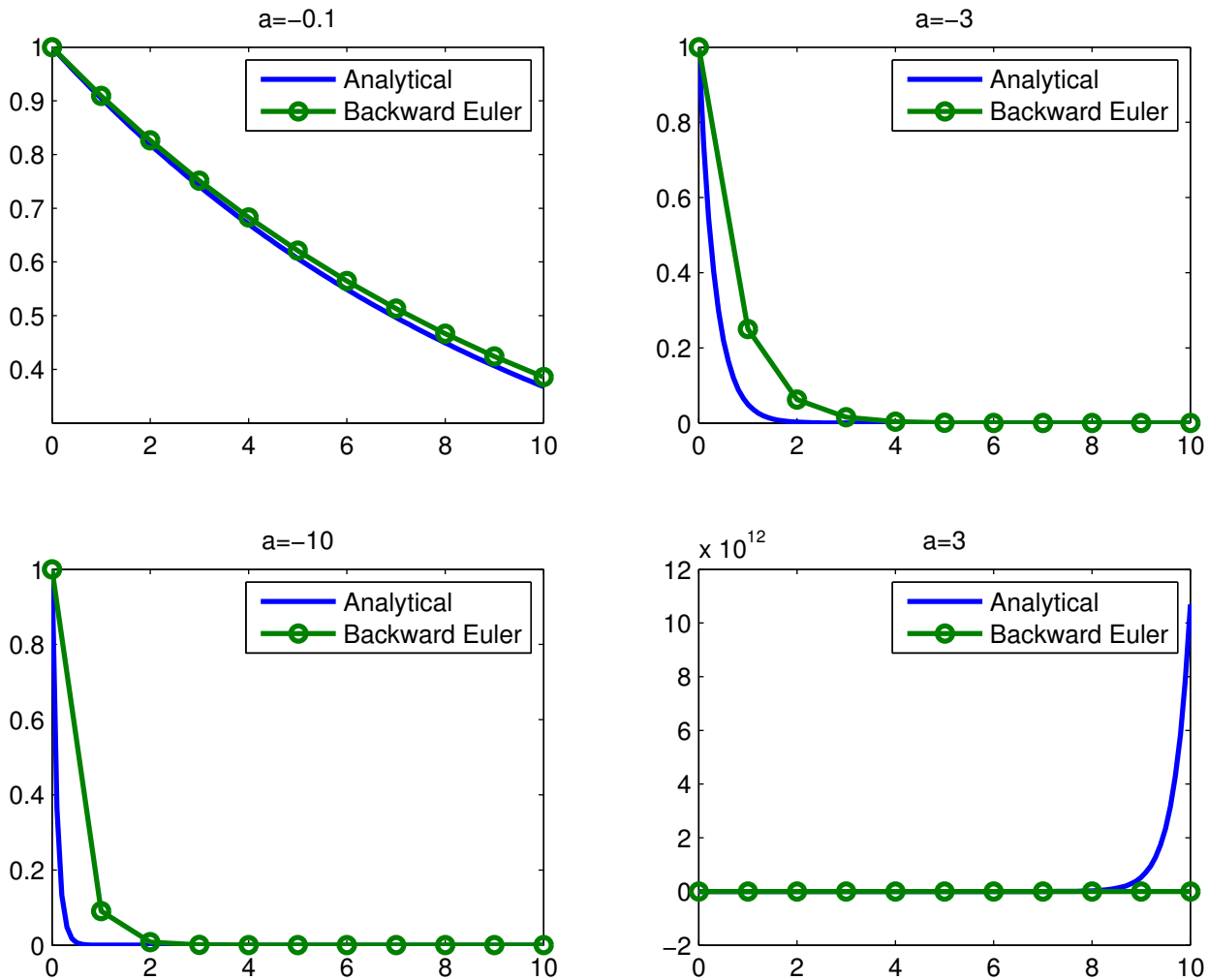
Analysis of stability for BE



Domain of stability for BE



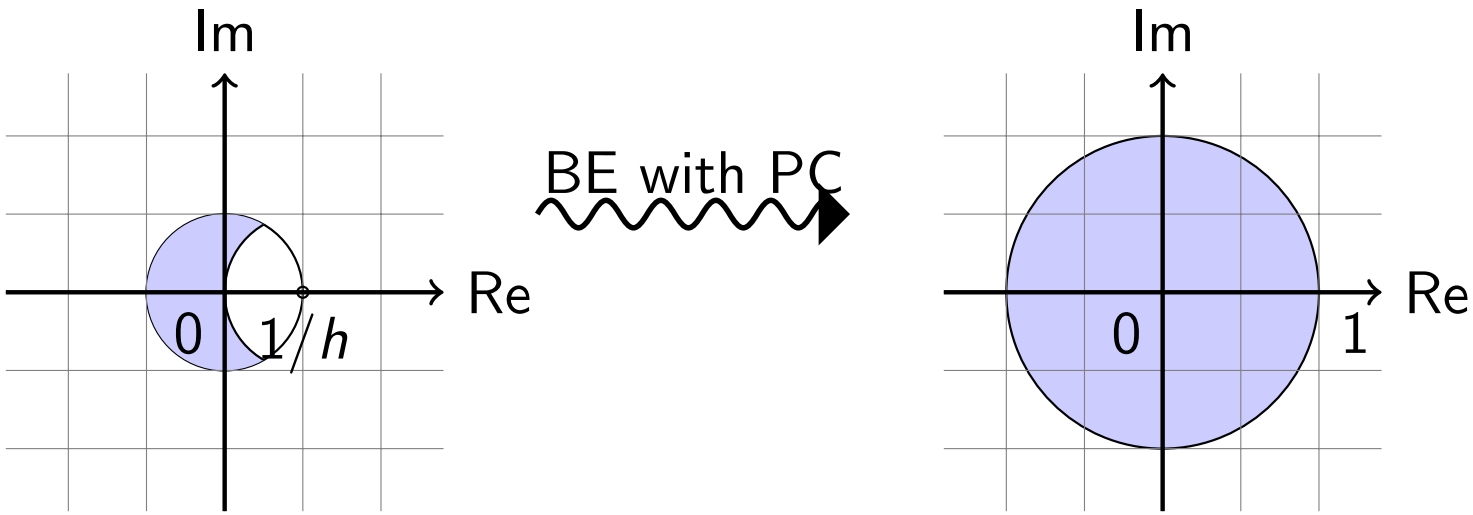
Numerical stability for BE — an experiment



BE reliable for fast stable and/or little damped, but unreliable for unstable (general feature of implicit methods, be careful).

Domain of stability for Predictor-Corrector BE (PC-BE)

When subtracting two ininite series, their regions of convergence must be considered (F converges for $\rho(Ah) < 1$)



Domain of stability for BE with Newton iterations

Newton iterations do not change the region of stability (see the linear case).

Heun's method — preview of single-steps methods

$$\begin{aligned}
\dot{x}_k &= f(x_k, t_k) \\
x^P_{k+1} &= x_k + h\dot{x}_k \\
\dot{x}^P_{k+1} &= f(x^P_{k+1}, t_k + h) \\
x^C_{k+1} &= x_k + h\dot{x}^P_{k+1}.
\end{aligned}$$

Substituting all the terms into just one we obtain

$$x_{k+1} = x_k + hf(x_k + hf_k, t_k + h).$$

Expand $f()$ in Taylor series and truncate after the linear term

$$f(x_k + hf_k, t_k + h) \approx f(x_k, t_k) + \left.\frac{\partial f(x,t)}{\partial x}\right|_{x_k,t_k} hf_k + \left.\frac{\partial f(x,t)}{\partial t}\right|_{x_k,t_k} h.$$

Plugging this into the equation above we obtain $\longrightarrow$

$$x_{k+1} = x_k + hf_k + h^2 \underbrace{\left(\left.\frac{\partial f(x,t)}{\partial x}\right|_{x_k,t_k} f_k + \left.\frac{\partial f(x,t)}{\partial t}\right|_{x_k,t_k} \right)}_{\dot{f}(x_k,t_k)}.$$

Now compare this with Taylor expansion of $x(t_k + h)$ truncated after the quadratic term

$$x(t_k + h) \approx x(t_k) + hf(x_k, t_k) + \frac{1}{2}h^2 \dot{f}(x(t_k), t_k).$$

Modify the PC algorithm to agree with the first three terms of Taylor series

$$x_{k+1}^{PC} = x_k + hf_k + h^2 \dot{f}(x_k, t_k)$$
$$x_{k+1} = \frac{1}{2}\left(x_{k+1}^{PC} + x_{k+1}^{FE}\right).$$

or

$$\dot{x}_k = f(x_k, t_k)$$
$$x_{k+1}^{P} = x_k + h\dot{x}_k$$
$$\dot{x}_{k+1}^{P} = f(x_{k+1}^{P}, t_{k+1})$$
$$x_{k+1}^{C} = x_k + \frac{1}{2}h\left(\dot{x}_{k+1}^{P} + \dot{x}_k\right).$$

LE of Heun's method is of third order, hence GE is second order.

Runge-Kutta methods

Generalize Heun's method

$$
\begin{aligned}
\dot{x}_k &= f(x_k, t_k) \\
x_{k+1}^P &= x_k + h\beta_{11}\dot{x}_k \\
\dot{x}_{k+1}^P &= f(x_{k+1}^P, t_k + \alpha h) \\
x_{k+1}^C &= x_k + h\left(\beta_{22}\dot{x}_{k+1}^P + \beta_{21}\dot{x}_k\right).
\end{aligned}
$$

Plugging the equations into each other and expanding into Taylor series yields

$$
\begin{aligned}
x_{k+1}^C = x_k &+ h(\beta_{21} + \beta_{22})f_k \\
&+ \frac{h^2}{2}\left(2\beta_{11}\beta_{22}\left.\frac{\partial f(x,t)}{\partial x}\right|_{x_t,t_k} f_k + 2\alpha_1\beta_{22}\left.\frac{\partial f(x,t)}{\partial t}\right|_{x_k,t_k}\right).
\end{aligned}
$$

Comparing with the Taylor expansion for $x(t_k + h)$ we learn that the following three equations need to be satisfied

$$\boxed{\begin{aligned} \beta_{21} + \beta_{22} &= 1 \\ 2\alpha_1\beta_{22} &= 1 \\ 2\beta_{11}\beta_{22} &= 1 \end{aligned}}$$

Heun's method

$$\alpha_1 = 1, \quad \beta_{11} = 1, \quad \beta_{21} = 0.5, \quad \beta_{22} = 0.5.$$

But another format popular — Butcher tableau

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

The Butcher tableau for the Explicit midpoint scheme

0	0	0
1/2	1/2	0
	0	1

The corresponding algorithm

$$
\begin{aligned}
\dot{x}_k &= f(x_k, t_k) \\
x^P_{k+\frac{1}{2}} &= x_k + \frac{h}{2}\dot{x}_k \\
\dot{x}^P_{k+\frac{1}{2}} &= f(x^P_{k+\frac{1}{2}}, t_{k+\frac{1}{2}}) \\
x^C_{k+1} &= x_k + h\dot{x}^P_{k+\frac{1}{2}}
\end{aligned}
$$

Fourth-order RK method

Matrix-vector format for the coefficients

$$\alpha = \begin{bmatrix} 1/2 \\ 1/2 \\ 1 \\ 1 \end{bmatrix}, \quad \beta = \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/6 & 1/3 & 1/3 & 1/6 \end{bmatrix}$$

Equivalently, the Butcher tableau is

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

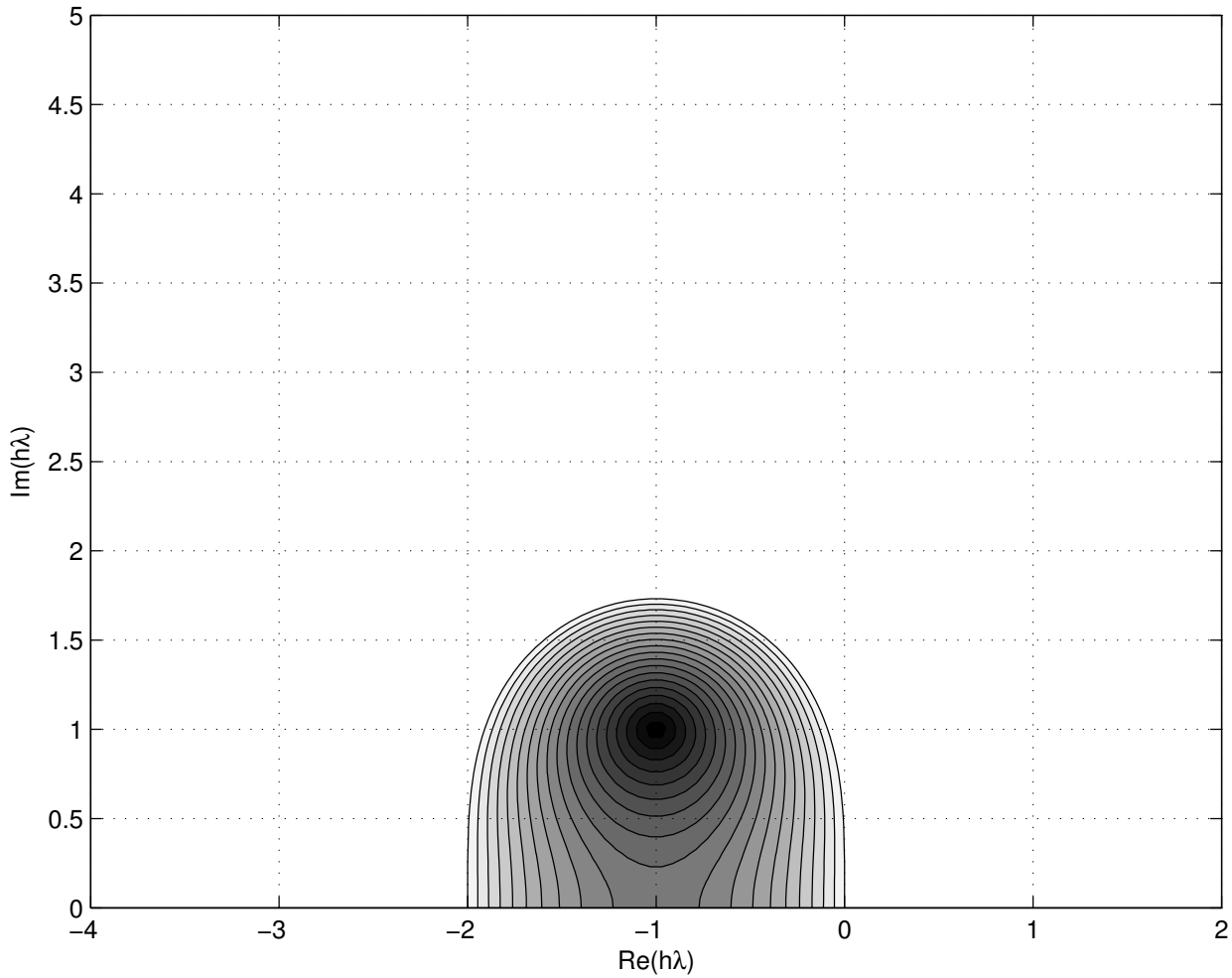
Fourt-order RK4 algorithm contains four "stages"

$$
\begin{aligned}
\dot{x}_k &= f(x_k, t_k) \\
x^{P_1} &= x_k + \frac{h}{2}\dot{x}_k \\
\dot{x}^{P_1} &= f(x^{P_1}, t_{k+\frac{1}{2}}) \\
x^{P_2} &= x_k + \frac{h}{2}\dot{x}^{P_1} \\
\dot{x}^{P_2} &= f(x^{P_2}, t_{k+\frac{1}{2}}) \\
x^{P_3} &= x_k + h\dot{x}^{P_2} \\
\dot{x}^{P_3} &= f(x^{P_3}, t_{k+1}) \\
x_{k+1} &= x_k + \frac{h}{6}\left(\dot{x}_k + 2\dot{x}^{P_1} + 2\dot{x}^{P_2} + \dot{x}^{P_3}\right).
\end{aligned}
$$

Stability regions for RK methods — Heun's method

$$F = I + Ah + A^2h^2$$

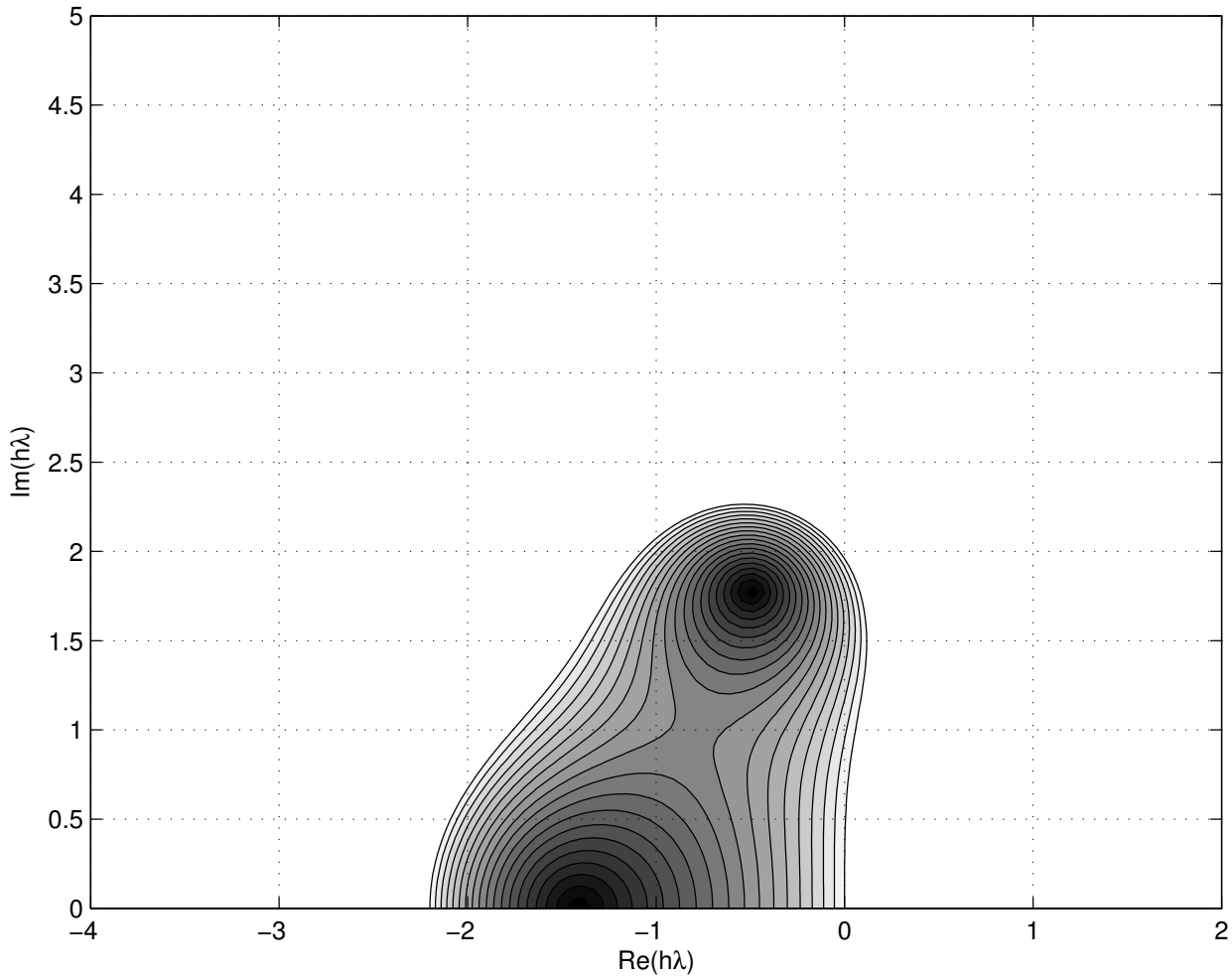
and its stability domain



Stability region for RK4 method

Finding F tedious but straightforward

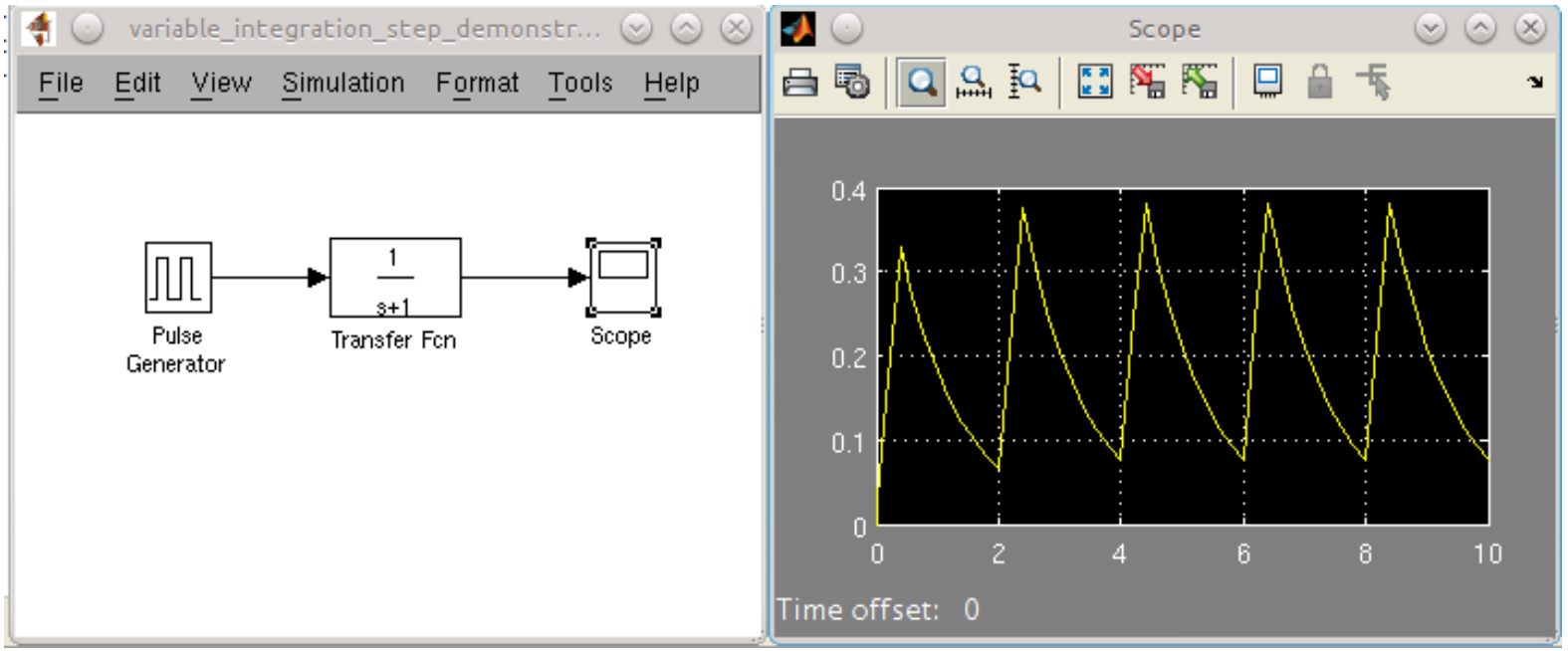
$$F = I + Ah + \frac{1}{2}A^2h^2 + \frac{1}{6}A^3h^3 + \frac{1}{24}A^4h^4.$$



Integration step size control

- The local error depends on the integration step h. If the current error is known and is not satisfactory, the step can be decreased.
- Where to get an error? Run two algorithms simultaneously and subtract their outcomes. Generally works.
- Why wasting numerical effort for two independent tasks? How about RK4 and RK5? Possibility of reuse of major part of effort.
- Voila! **ode45()** solver

Some experiment in Simulink




```
semilogy(tout(1:end-1),diff(tout))
```

Stiff systems — example (Cleve Moler))

$$\frac{dx}{dt} = x^2 - x^3, \quad x(0) = d, \quad 0 < t < 2/d, \quad d = 1/100.$$

Analytical solution possible:

```
x = dsolve('Dx=x^2-x^3','x(0) = 1/100');
x = simplify(x);
pretty(x)
ezplot(x,0,200)
```

```
d = 0.01;
F = @(t,x) x^2-x^3;
opts = odeset('RelTol',1.e-4);
ode45(F,[0 2/d],d,opts);
```

and then try with

```
ode23s(F,[0 2/d],d,opts);
```

One more note on errors — rounding errors

Errors come not only through truncating but also through finite precision arithmetic. Guess at the outcome

```
>> (0.1+0.2)==0.3
```

IEEE 754 single precision (type single) is valid only down to 7 decimal places ($2^{-23} \approx 1 \times 10^{-7}$). Let us see the impact for the truncation process. Think of some example where $h = 0.001$, $|x| \approx |f| \approx |\dot{f}| \approx |\ddot{f}| = 1$.

$$|x(t_0 + h)| \approx |x(t_0)| + |f(x(t_0))h| + \left|\frac{\mathrm{d}f(x(t_0))}{\mathrm{d}t}\frac{h^2}{2!}\right| + \left|\frac{\mathrm{d}^2 f(x(t_0))}{\mathrm{d}t^2}\frac{h^3}{3!}\right|$$
$$\approx 1.0 + 0.001 + 10^{-6} + 10^{-9}.$$

Multistep techniques — Why discarding the values from previous steps

Approximate the higher order derivatives of $x(t)$ numerically

$$x(t_0 + h) = x(t_0) + \left.\frac{\mathrm{d}x(t)}{\mathrm{d}t}\right|_{t_0} h + \left.\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}\right|_{t_0} \frac{h^2}{2!} + \mathcal{O}(h^3).$$

Alternatively, approximate derivatives of $f()$

$$x(t_0 + h) = x(t_0) + f(x, u, t)|_{t_0} h + \left.\frac{\mathrm{d}f(x, u, t)}{\mathrm{d}t}\right|_{t_0} \frac{h^2}{2!} + \mathcal{O}(h^3).$$

Approximate the derivative of $\dot{f}$ by a forward difference

Taylor series for the derivative

$$\dot{x}(t_0 + h) = \dot{x}(t_0) + \ddot{x}(t_0)h + \frac{1}{2}\dddot{x}(t_0)h^2 + \mathcal{O}(h^3)$$

From which

$$\ddot{x}(t_0)h = \dot{x}(t_0 + h) - \dot{x}(t_0) - \frac{1}{2}\dddot{x}(t_0)h^2 + \mathcal{O}(h^3)$$

Substitute to Taylor series for x

$$\begin{aligned} x(t_0 + h) &= x(t_0) + \dot{x}(t_0)h \\ &\quad + \left[\dot{x}(t_0 + h) - \dot{x}(t_0) - \frac{1}{2}\dddot{x}(t_0)h^2 + \mathcal{O}(h^3)\right]\frac{h}{2!} + \mathcal{O}(h^3) \\ &= x(t_0) + \frac{h}{2}\left(\dot{x}(t_0 + h) + \dot{x}(t_0)\right) + \mathcal{O}(h^3). \end{aligned}$$

Trapezoidal method — one step, order 2, implicit

$$\boxed{x_{k+1} = x_k + \frac{h}{2}\left(f_k + f_{k+1}\right)}$$

Adams-Bashforth method AB(2)

Another Taylor series for the derivative

$$\dot{x}(t_0 - h) = \dot{x}(t_0) - \ddot{x}(t_0)h + \frac{1}{2}\dddot{x}(t_0)h^2 + \mathcal{O}(h^3)$$

From which

$$\ddot{x}(t_0)h = -\dot{x}(t_0 + h) + \dot{x}(t_0) + \frac{1}{2}\dddot{x}(t_0)h^2 + \mathcal{O}(h^3)$$

Substitute to Taylor series for x

$$\begin{aligned} x(t_0 + h) &= x(t_0) + \dot{x}(t_0)h \\ &\quad + \left[-\dot{x}(t_0 + h) + \dot{x}(t_0) + \frac{1}{2}\dddot{x}(t_0)h^2 + \mathcal{O}(h^3)\right]\frac{h}{2!} + \mathcal{O}(h^3) \\ &= x(t_0) + \frac{3}{2}h\dot{x}(t_0) - \frac{1}{2}h\dot{x}(t_0 - h) + \mathcal{O}(h^3). \end{aligned}$$

AB(2) is explicit, two-step technique, order 2

$$\boxed{x_{k+2} = x_{k+1} + \frac{3}{2} h f_{k+1} - \frac{1}{2} h f_k}$$

Other two-step techniques

General structure

$$\boxed{x_{k+2} + \alpha_1 x_{k+1} + \alpha_0 x_k = h\left(\beta_2 f_{k+2} + \beta_1 f_{k+1} + \beta_0 f_k\right)}$$

Adams-Moulton (implicit, order 2)

$$x_{k+2} - x_{k+1} = \frac{1}{12} h\left(5 f_{k+2} + 8 f_{k+1} - f_k\right).$$

Simpson (order 4, implicit)

$$x_{k+2} - x_k = \frac{1}{3} h\left(f_{k+2} + 4 f_{k+1} + 1 f_k\right).$$

General multistep methods

$$x_{k+n}+\alpha_{n-1}x_{k+n-1}+\ldots+\alpha_0 x_k = h\left(\beta_n f_{k+n} + \beta_{n-1} f_{k+n-1} + \ldots + \beta_0 f_k\right).$$

$$\rho(z) = z^n + \alpha_{n-1}z^{n-1} + \ldots + \alpha_0,$$
$$\sigma(z) = \beta_n z^n + \beta_{n-1}z^{n-1} + \ldots + \beta_0.$$

Classic multistep methods

Adams-Bashforth — explicit, order n

$$\rho(z) = z^n - z^{n-1}$$

$$x_{k+n} - x_{k+n-1} = h\left(\beta_{n-1} f_{k+n-1} + \ldots + \beta_0 f_k\right).$$

Adams-Moulton — implicit, order $n+1$

$$\rho(z) = z^n - z^{n-1}$$

$$x_{k+n} - x_{k+n-1} = h\left(\beta_n f_{k+n} + \beta_{n-1} f_{k+n-1} + \ldots + \beta_0 f_k\right).$$

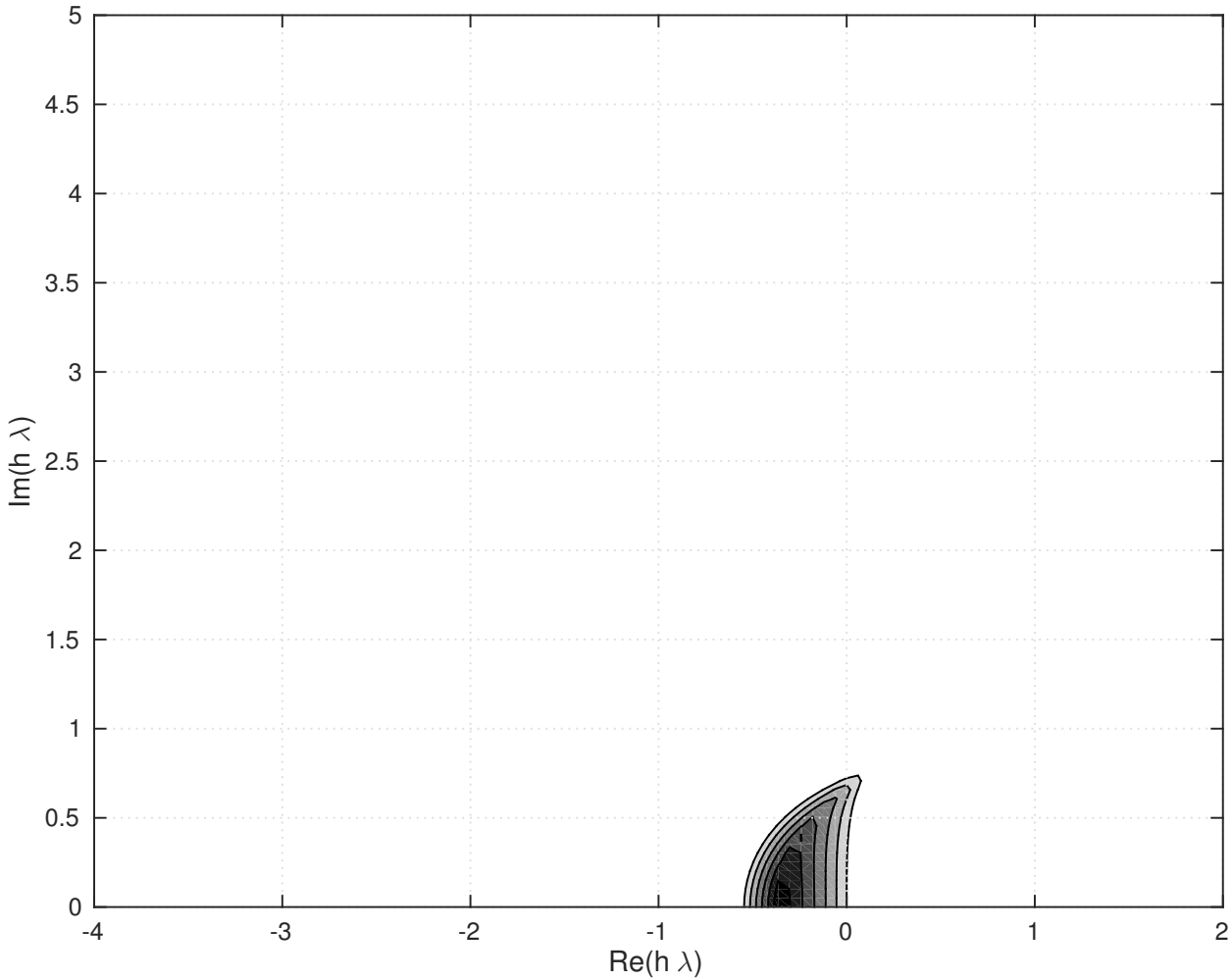
Backward differentiation formulas (BDFs) — implicit, generalization of backward Euler, order n

$$\sigma(z) = \beta_n z^n$$

$$x_{k+n} + \alpha_{n-1} x_{k+n-1} + \ldots + \alpha_0 x_k = h\beta_n f_{k+n}.$$

A-stability for multistep techniques

AB(3)

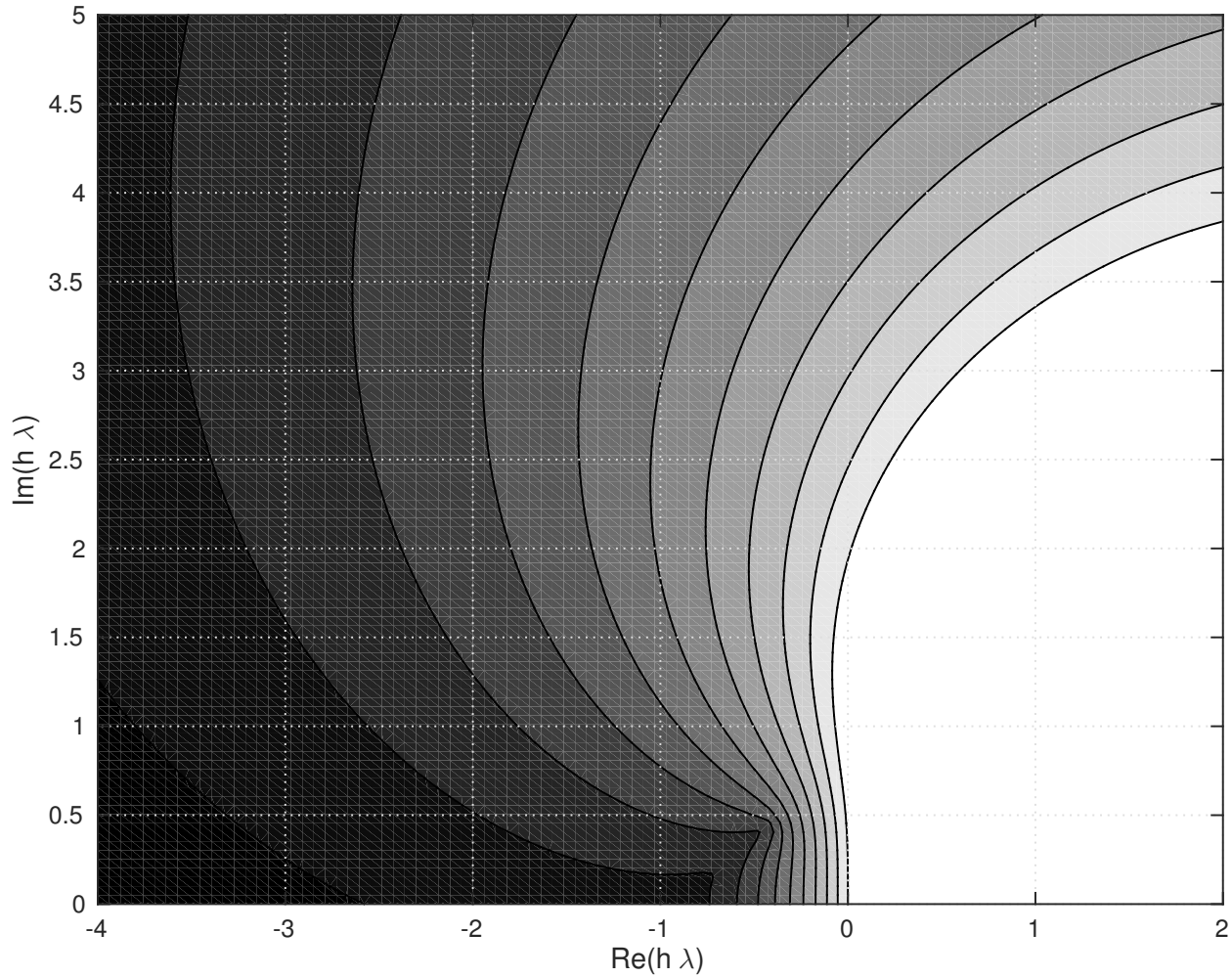


Compare with FE (=AB(1)), AB(2), ...

Not all multistep have small stability region — BDF rules

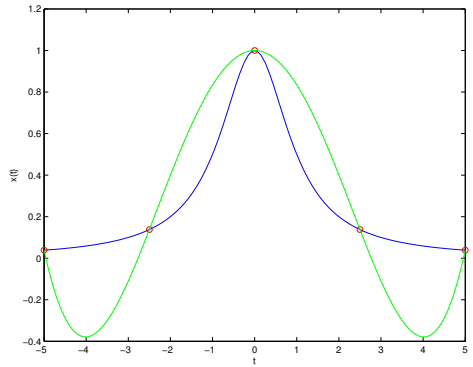
BDF(3)

$$x_{k+1} = \frac{18}{11}x_k - \frac{9}{11}x_{k-1} + \frac{2}{11}x_{k-2} + \frac{6}{11}hf_{k+1}$$

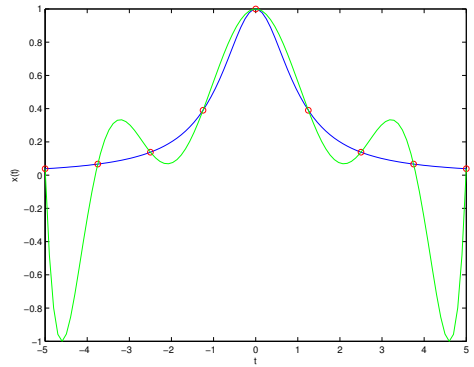


Why does the stability region shrinks for n growing?

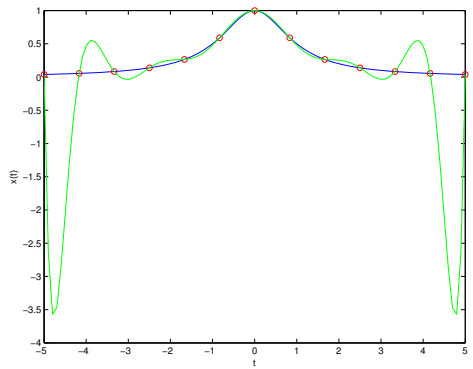
Interpolation of $x(t)$ at t_k, t_{k-1},..., t_{k-n}, by a polynomial.
Interpolation OK but extrapolation poor for growing n!



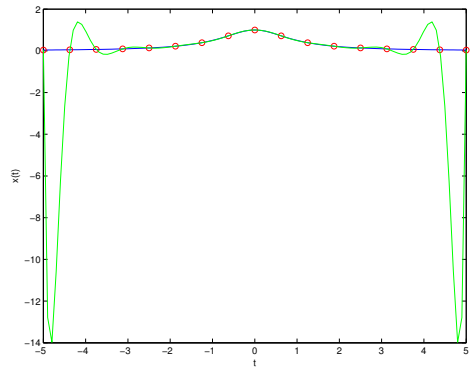
(a) $n = 4$



(b) $n = 8$



(c) $n = 12$



(d) $n = 16$

Polynomial interpolation — Newton method

Find a polynomial of nth order that passes through $n+1$ function values $f_0,\ f_1,\ldots,\ f_n$ at time instants $t_0,\ t_0+h,\ldots,\ t_0+nh$.

Forward differences

$$
\begin{aligned}
\Delta f_0 &= f_1 - f_0, \\
\Delta^2 f_0 &= \Delta f_1 - \Delta f_0 = (f_2 - f_1) - (f_1 - f_0) = f_2 - 2f_1 + f_0, \\
\Delta^3 f_0 &= \Delta^2 f_1 - \Delta^2 f_0 = f_3 - 3f_2 + 3f_1 - f_0, \\
&\vdots
\end{aligned}
$$

Newton-Gregory polynomial

$$s = \frac{t - t_0}{h}$$

$$\boxed{f(t) \approx f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \ldots + \binom{s}{n}\Delta^n f_0}$$

Backward differences and backward Newton-Gregory polynomial

$$
\begin{aligned}
\nabla f_i &= f_i - f_{i-1}, \\
\nabla^2 f_i &= \nabla f_i - \nabla f_{i-1} = f_i - 2f_{i-1} + f_{i-2}, \\
\nabla^3 f_i &= \nabla^2 f_i - \nabla^2 f_{i-1} = f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}, \\
&\vdots
\end{aligned}
$$

$$
f(t) \approx f_0 + s\nabla f_0 + \binom{s+1}{2}\nabla^2 f_0 + \binom{s+2}{3}\nabla^3 f_0 + \ldots + \binom{s+n-1}{n}\nabla^n f_0
$$

Using backward Newton polynomial to approximate integral of f

$$\dot{x}(t) = f_0 + s\nabla f_0 + \binom{s+1}{2}\nabla^2 f_0 + \binom{s+2}{3}\nabla^3 f_0 + \dots$$

$$\int_{t_0}^{t_1} \dot{x}(t)\mathrm{d}t = x(t_1) - x(t_0),$$

$$= \int_{t_0}^{t_1} \left[f_0 + s\nabla f_0 + \binom{s+1}{2}\nabla^2 f_0 + \binom{s+2}{3}\nabla^3 f_0 + \dots \right] \mathrm{d}t,$$

$$= \int_0^1 \left[f_0 + s\nabla f_0 + \binom{s+1}{2}\nabla^2 f_0 + \binom{s+2}{3}\nabla^3 f_0 + \dots \right] \frac{\mathrm{d}t}{\mathrm{d}s}\mathrm{d}s$$

$$\boxed{x(t_1) = x(t_0) + h\int_0^1 \left[f_0 + s\nabla f_0 + \binom{s+1}{2}\nabla^2 f_0 + \binom{s+2}{3}\nabla^3 f_0 + \dots \right] \mathrm{d}s}$$

Adams-Bashforth

$$\boxed{x(t_1) = x(t_0) + h\left[f_0 + \frac{1}{2}\nabla f_0 + \frac{5}{12}\nabla^2 f_0 + \frac{3}{8}\nabla^3 f_0 + \ldots\right] \mathrm{d}s}$$

AB(3) — cutting after the quadratic term and expanding

$$x_{k+1} = x_k + \frac{h}{12}(23f_k - 16f_{k-1} + 5f_{k-2}).$$

AB(4) — cutting after the cubic term and expanding

$$x_{k+1} = x_k + \frac{h}{24}(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}).$$