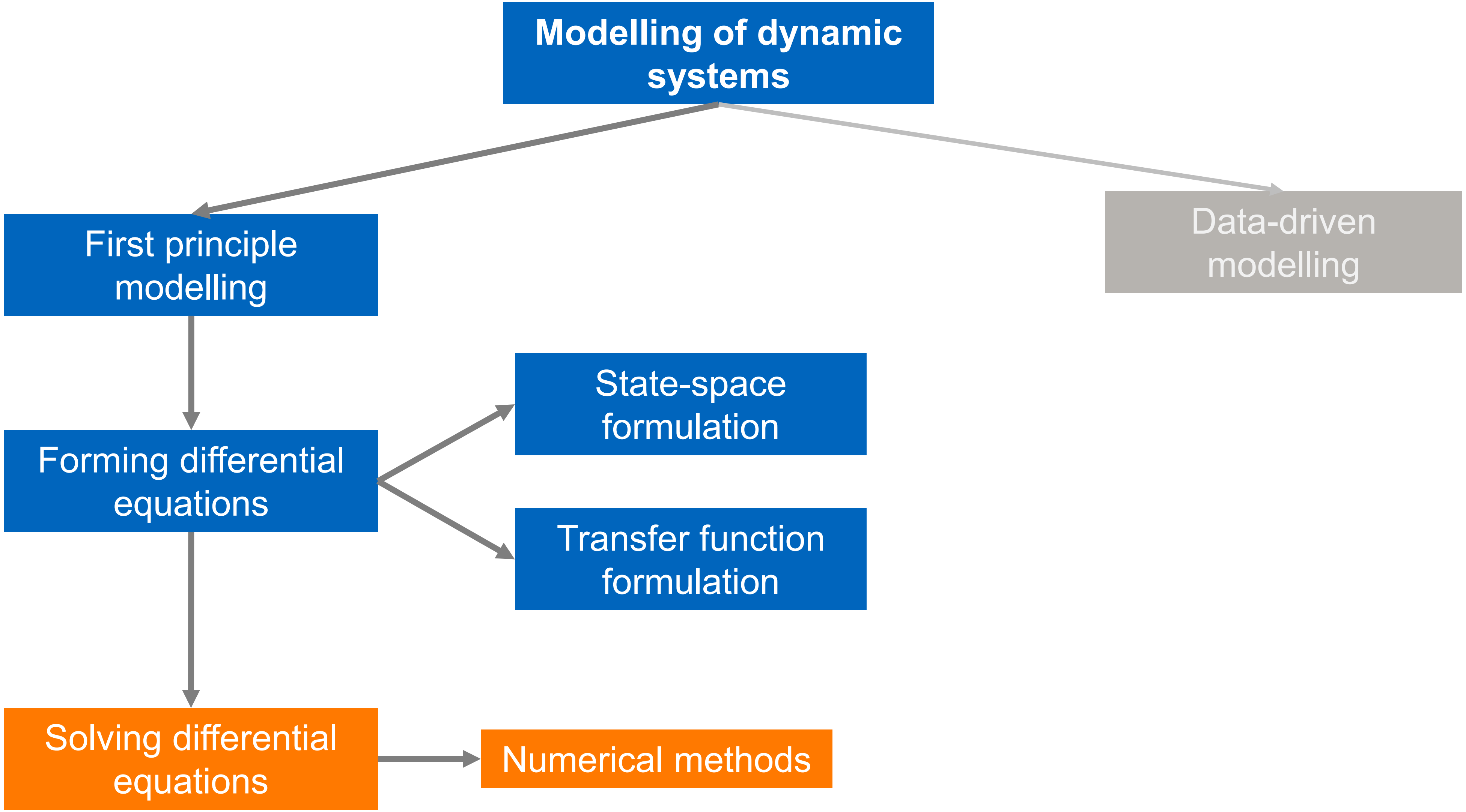
Aalto University
School of Electrical
Engineering

ELEC-E8103 Modelling, Estimation and Dynamic Systems

Simulation

Quan Zhou,
Department of Electrical Engineering and Automation
Aalto University, School of Electrical Engineering
Email: quan.zhou@aalto.fi

Overview


Modelling of dynamic systems
First principle modelling
Data-driven modelling
Forming differential equations
State-space formulation
Transfer function formulation
Solving differential equations
Numerical methods

Learning Goals

Course Learning Outcomes

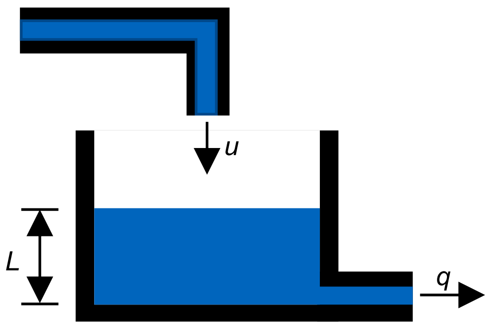
- Select proper modeling approach for specific practical problems,
- Formulate mathematical models of physical systems,
- Construct models of systems using modeling tools such as MATLAB and Simulink,
- Estimate the parameters of linear and nonlinear static systems from measurement data,
- Identify the models of linear dynamic systems from measurement data

将常微分方程(ODE)求解为初值问题

- Solving ordinary differential equations (ODEs) as initial value problems
- Solving ordinary differential equations as boundary value problems

常微分方程作为边值问题求解

Example: flow system…



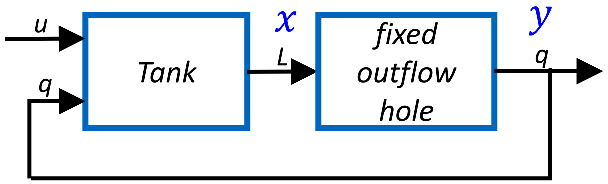
$$\frac{d}{dt}L(t) = -\frac{a\sqrt{2g}}{A}\cdot\sqrt{L(t)} + \frac{1}{A}u(t)$$
$$q(t) = a\sqrt{2g}\cdot\sqrt{L(t)}$$

Generalizing to

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

ODE with three variables: *u*, *y*, *x*

If we are interested in the outflow rate:



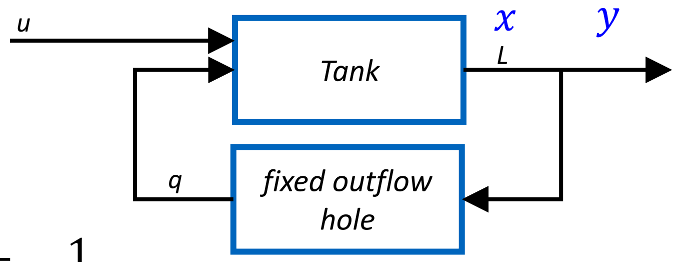
input → $u(t) = u(t),$

Internal state → $x(t) = L(t),$

output → $y(t) = q(t)$

$$f(x(t), u(t)) = -\frac{a\sqrt{2g}}{A}\cdot\sqrt{x(t)} + \frac{1}{A}u(t)$$
$$h\big(x(t), u(t)\big) = a\sqrt{2g}\cdot\sqrt{x(t)}$$

If we are only interested in the height:



$$x(t) = L(t)$$
$$y(t) = x(t)$$

$$f(x(t), u(t)) = -\frac{a\sqrt{2g}}{A}\cdot\sqrt{x(t)} + \frac{1}{A}u(t)$$
$$h\big(x(t), u(t)\big) = x(t)$$

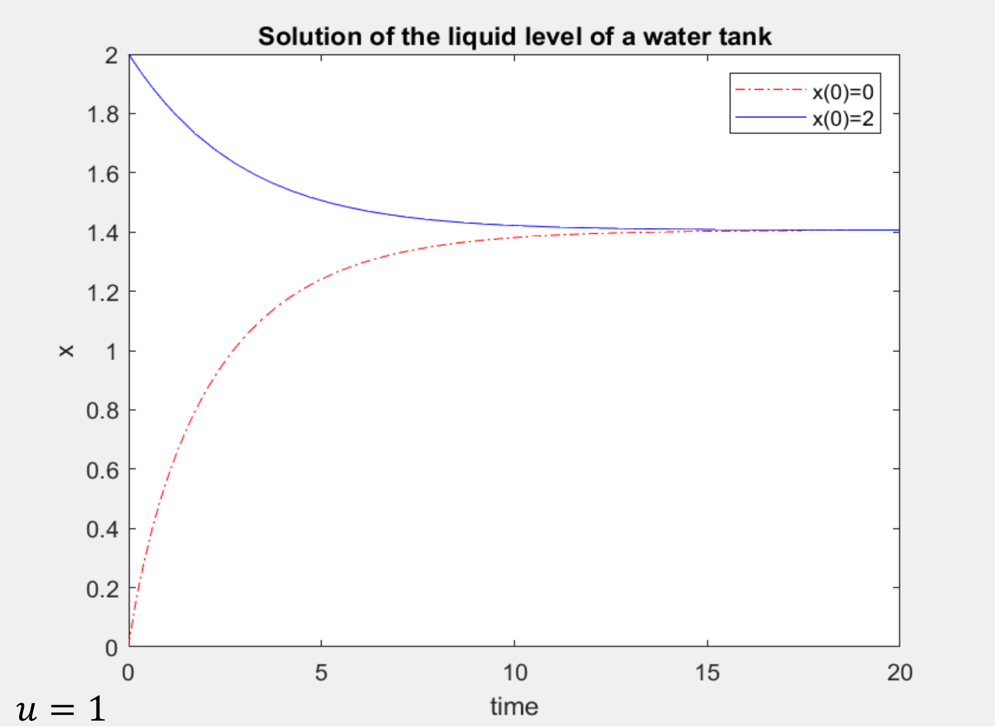
In both cases, the system is written in the same form.

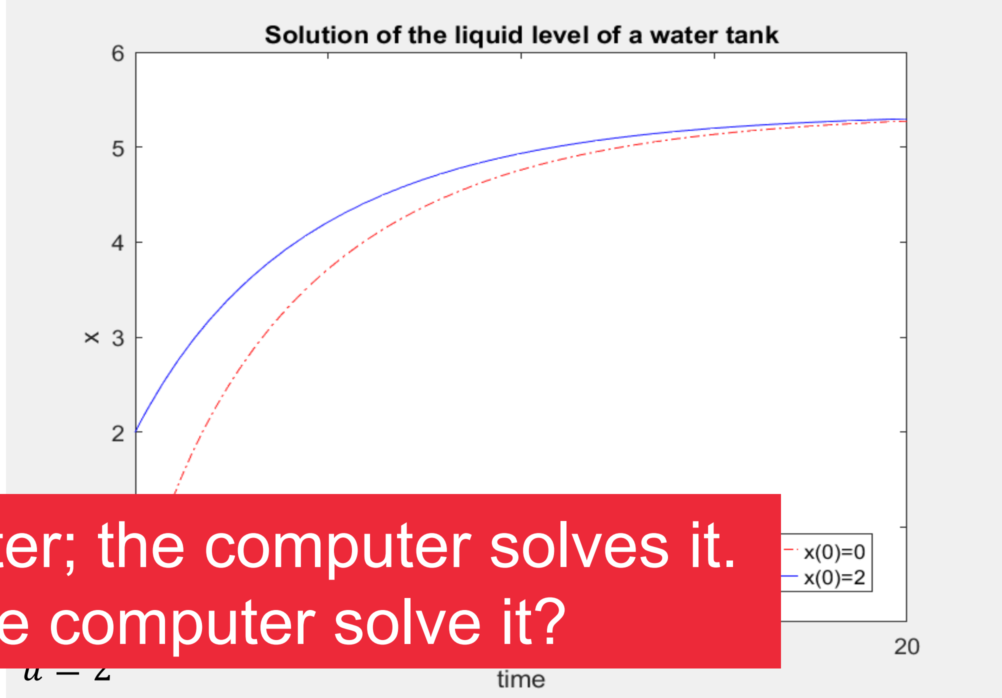
How to do it in MATLAB

$$f(x(t),u(t)) = -\frac{a\sqrt{2g}}{A} \cdot \sqrt{x(t)} + \frac{1}{A}u(t)$$

$$h\big(x(t),u(t)\big) = x(t)$$

```
%% Ljung 1994 case 3: flow system
u = 1;
A = 1;
g = 9.8;
a = 0.2;

model = @(t,x) -a*sqrt(2*g)/A*sqrt(x(1)) + 1/A*u;
hfunc = @(x) x;

options = odeset('RelTol',1e-4,'AbsTol',1e-6);
timeSpan = [0 20];
initCond = 0;
[T1,X1] = ode45(model,timeSpan, initCond,options);
initCond = 2;
[T2,X2] = ode45(model,timeSpan, initCond,options);

Y1 = hfunc(X1); % we are only interested in the height
Y2 = hfunc(X2);

clf
plot(T1,Y1,'r-.'); hold on; plot(T2,Y2,'b-');
title('Solu
xlabel('tim
ylabel('x')
legend('x(0
```





We give the equation to the computer; the computer solves it.
That’s great! But how does the computer solve it?

rsity
lectrical

Ordinary differential equation (ODE)

包含一个或多个函数及其导数且相对于一个自变量的微分方程

- A differential equation containing one or more functions and their derivatives, with respect to one independent variable
  - Ordinary because it has only derivatives of one independent variable, not a partial differential equation involving two ore more 只有一个自变量的导数
  - What is this independent variable in dynamic systems?
  - Do ODEs have analytical solutions?

$$\frac{d}{dt}L(t) = -\frac{a\sqrt{2g}}{A}\cdot\sqrt{L(t)} + \frac{1}{A}u(t)$$

$$\frac{d}{dt}N_1(t) = (\lambda_1 - \gamma_1)N_1(t) + \alpha_1 N_1(t)N_2(t)$$
$$\frac{d}{dt}N_2(t) = (\lambda_2 - \gamma_2)N_2(t) - \alpha_2 N_1(t)N_2(t)$$

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

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

In this lecture, we mainly deal with the form: $\dfrac{dx(t)}{dt} = f\big(t, x(t)\big)$

$$\begin{cases} \dfrac{dx(t)}{dt} = f(t, x(t)) \\ x(t_0) = x_0 \end{cases}$$

Initial value problem

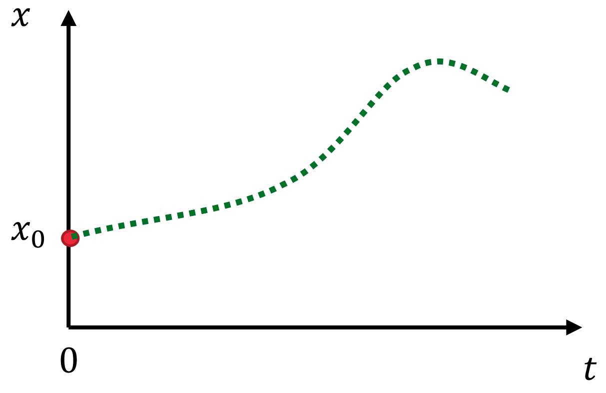
For an ordinary differential equation (ODE)

Solution

$$\frac{dx(t)}{dt} = f(t, x(t))$$

System dynamics

Boundry condition: $x(t_0) = x_0$



We want to solve this with a computer capable of:

- $+ \; - \; \times \; \div$
- Iterations
- Conditional branches

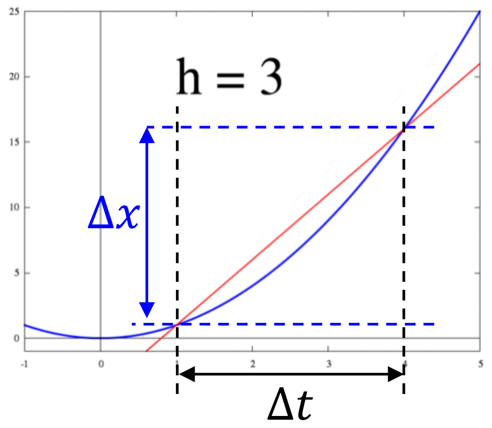
Euler's method

$$\begin{cases} \dfrac{dx(t)}{dt} = f\big(t, x(t)\big) \\ x(t_0) = x_0 \end{cases}$$

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t}$$

Let $h = \Delta t, \Delta x = x(t+h) - x(t)$

$$\frac{x(t+h) - x(t)}{h} \approx f\big(t, x(t)\big)$$



$$x(t+h) \approx x(t) + hf\big(t, x(t)\big)$$

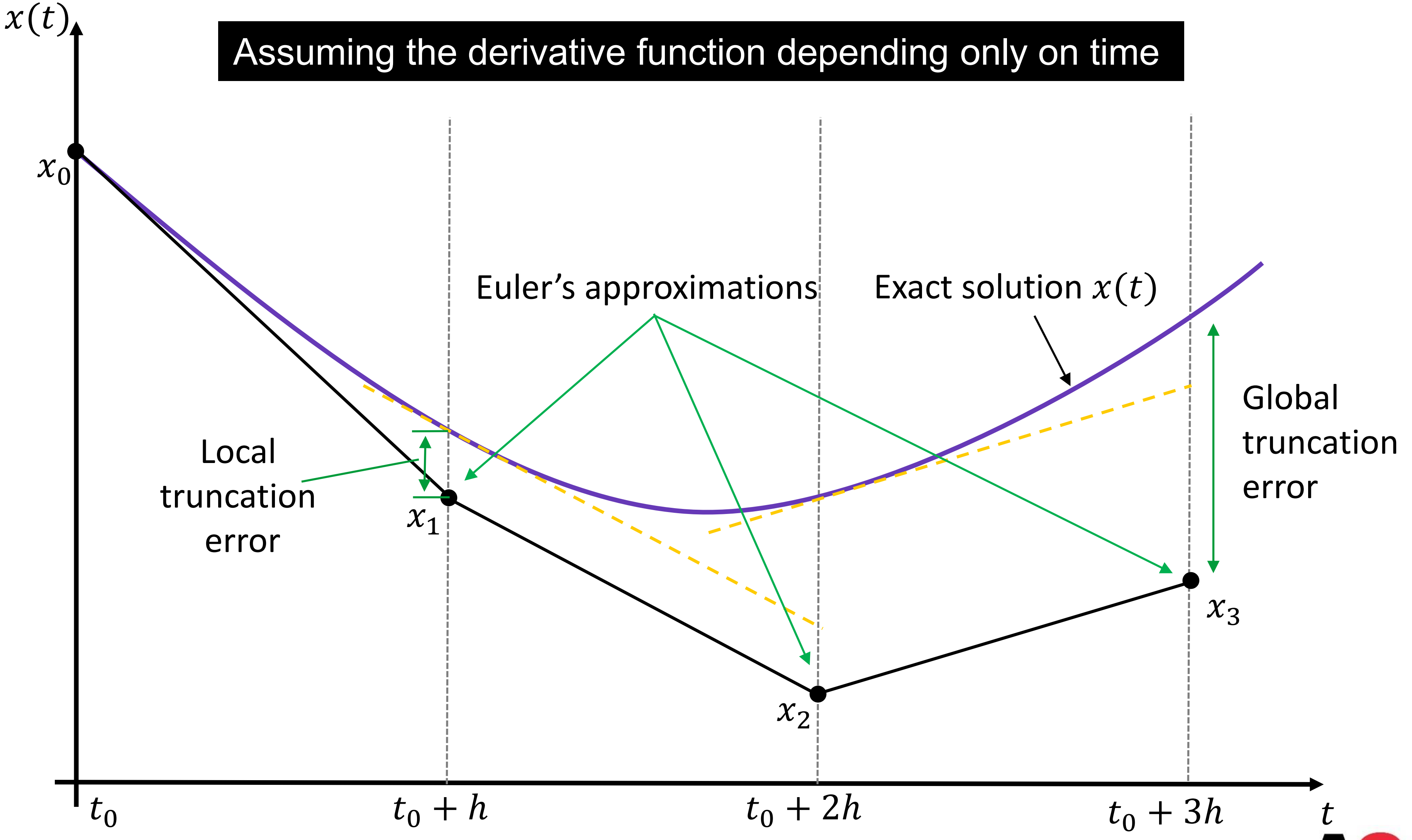
with known initial condition $t = t_0$ and $x(t_0) = x_0$, we can simulate the output of the ODE in n steps until $t = t_n$, where $t_n = t_0 + nh$

Solving Initial Value Problem using Euler's method

$x(t+h) \approx x(t) + hf\left(t, x(t)\right)$

$$\begin{cases} \dfrac{dx(t)}{dt} = f\left(t, x(t)\right) \\ x(t_0) = x_0 \end{cases}$$

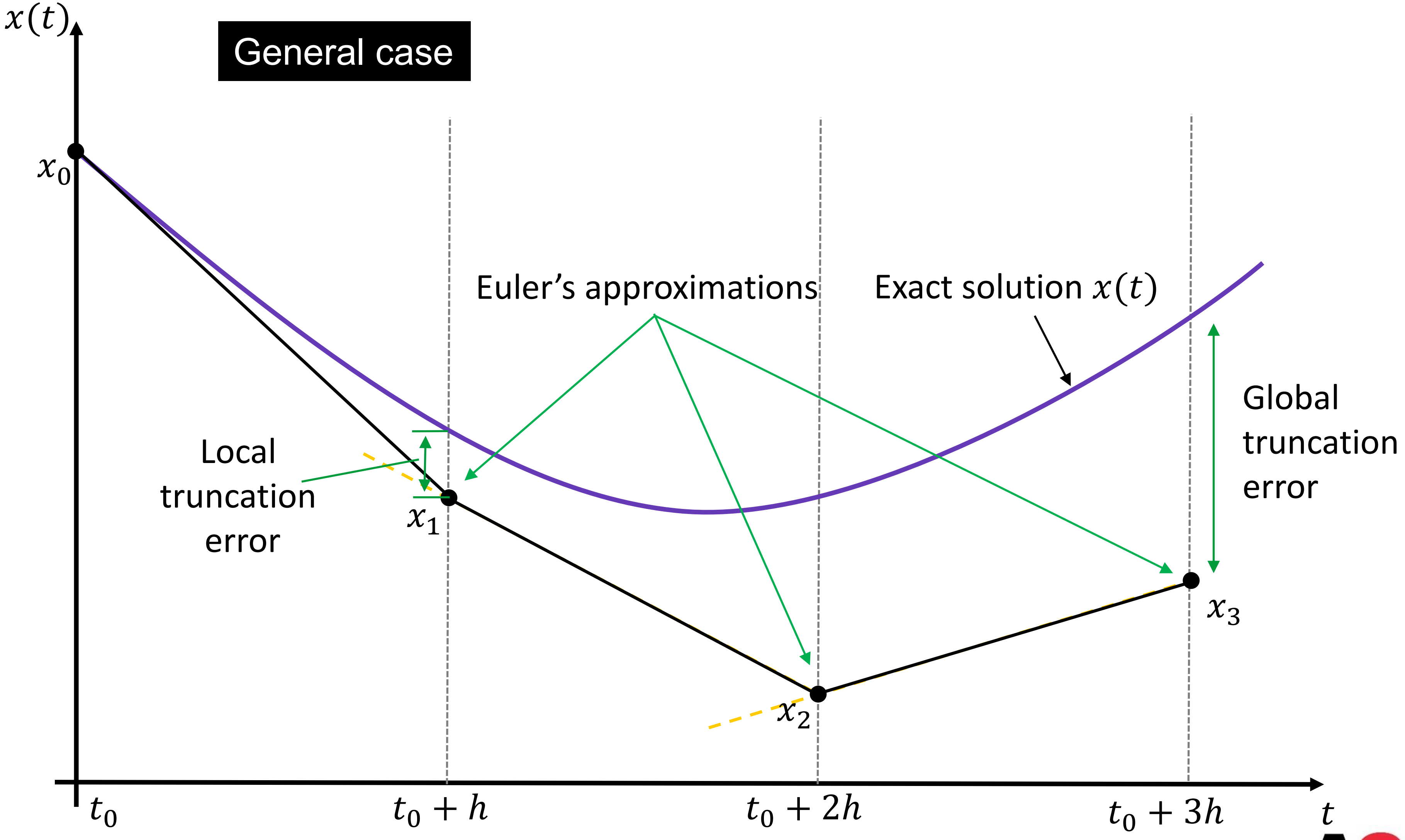
Assuming the derivative function depending only on time



Solving Initial Value Problem using Euler's method

$$x(t+h) \approx x(t) + hf\big(t, x(t)\big)$$

$$\begin{cases} \dfrac{dx(t)}{dt} = f\big(t, x(t)\big) \\ x(t_0) = x_0 \end{cases}$$



How to reduce the error?

Discussion:
how to reduce the error?

$$\begin{cases} \dfrac{dx(t)}{dt} = f\big(t, x(t)\big) \\ x(t_0) = x_0 \end{cases}$$

Euler's method

- Taylor's theorem 泰勒定理

$$x(t+h) = x(t) + h\frac{dx(t)}{dt} + \frac{1}{2!}h^2\frac{d^2x(t)}{dt^2} + \cdots \frac{1}{m!}h^m\frac{d^mx(t)}{dt^m} + \cdots$$

- For small h, we can use the 1st order Taylor polynomial

$$x(t+h) \approx x(t) + h\frac{dx(t)}{dt}$$

- Therefore

$$\boxed{x(t+h) \approx x(t) + hf\big(t, x(t)\big)}$$

Local truncation error: $\mathcal{O}(h^2)$

Global truncation error: $\mathcal{O}(h^1)$

What is $f\big(t, x(t)\big)$ in the figure on the previous slide?

It's the slope of $x(t)$ at time t.

Better methods

$$\begin{cases} \dfrac{dx(t)}{dt} = f\big(t, x(t)\big) \\ x(t_0) = x_0 \end{cases}$$

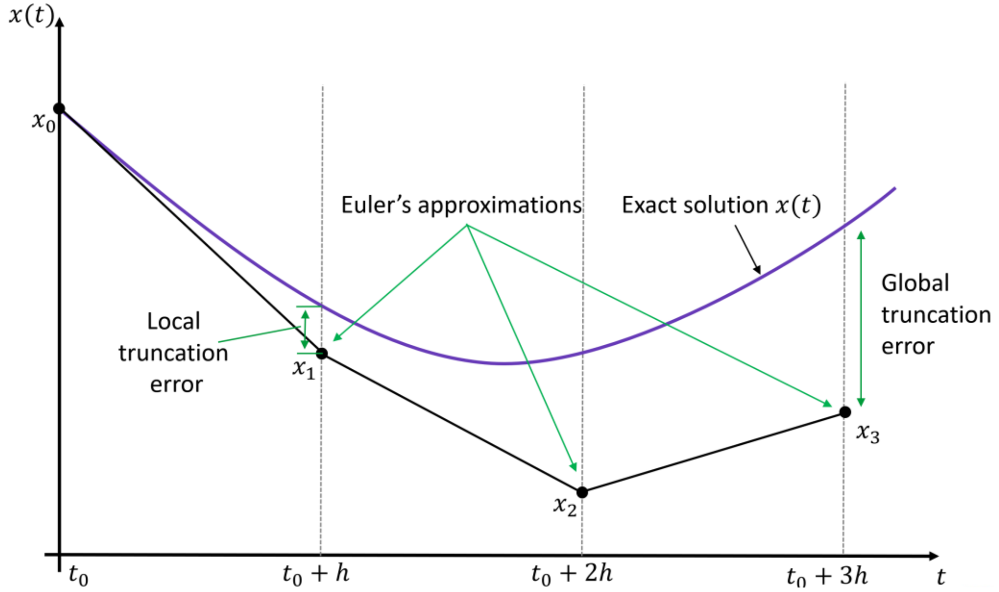
$x(t+h) = x(t) + h\,\phi\big(t, x(t)\big)$ or $x_{i+1} = x_i + h\,\phi\big(t, x(t)\big)$

New value = old value + step size x increment function

- increment function = slope estimate

In Euler's method $\phi\big(t, x(t)\big) = f\big(t, x(t)\big)$

$$x(t+h) \approx x(t) + hf\big(t, x(t)\big)$$



How can we get a better slope estimate?

Better methods

$$\begin{cases} \dfrac{dx(t)}{dt} = f\big(t, x(t)\big) \\ x(t_0) = x_0 \end{cases}$$

- From Euler's method: $\tilde{x}(t+h) = x(t) + hf\big(t, x(t)\big)$
- We can use the prediction to improve accuracy:

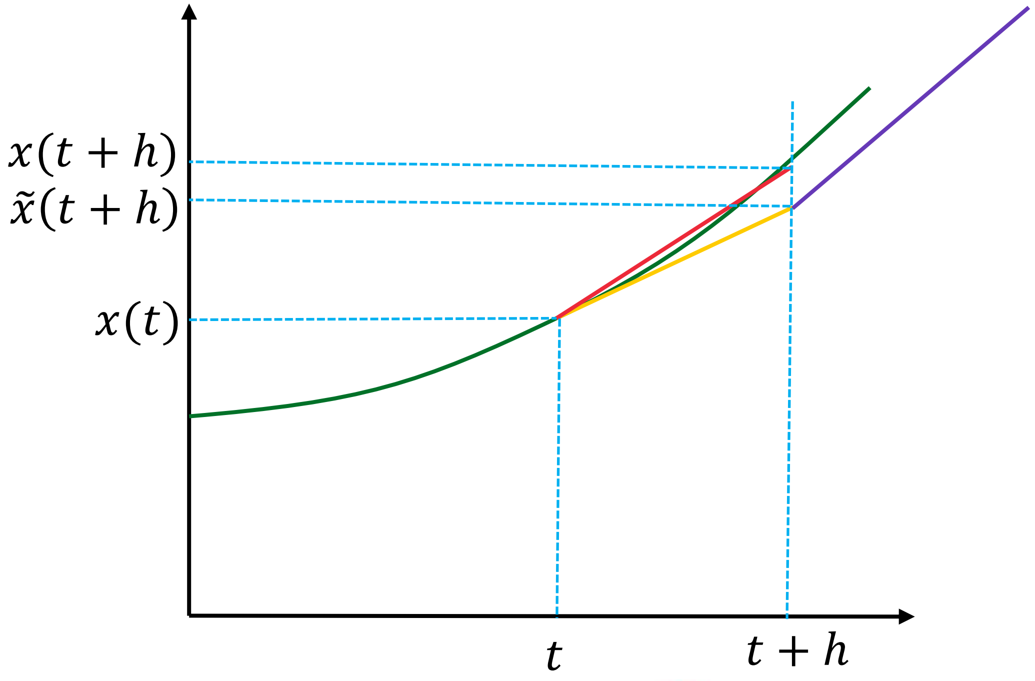
$$x(t+h) = x(t) + h\frac{1}{2}\Big(f\big(t, x(t)\big) + f\big(t+h, \tilde{x}(t+h)\big)\Big)$$

Heun's method

$$\phi = \frac{1}{2}\Big(f\big(t, x(t)\big) + f\big(t+h, \tilde{x}(t+h)\big)\Big)$$

Local truncation error: $\mathcal{O}(h^3)$

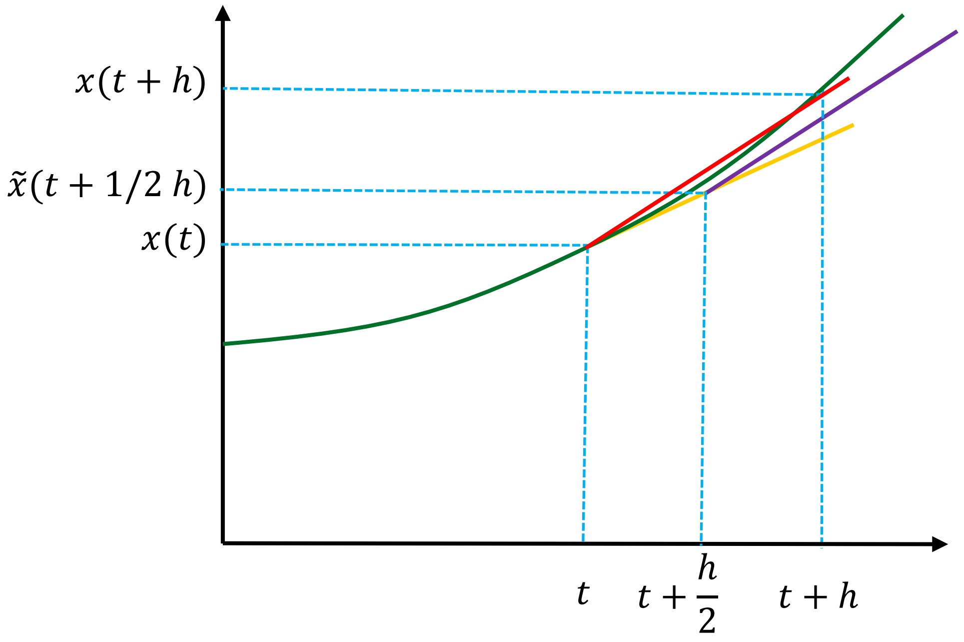
Global truncation error: $\mathcal{O}(h^2)$



Better methods…

$$\tilde{x}\left(t+\frac{h}{2}\right)=x(t)+\frac{h}{2}f\big(t,x(t)\big)$$

$$x(t+h)=x(t)+hf\left(t+\frac{1}{2}h,\tilde{x}\left(t+\frac{h}{2}\right)\right)$$



Midpoint method

$$\phi=f\left(t+\frac{1}{2}h,\tilde{x}\left(t+\frac{h}{2}\right)\right)$$

Local truncation error: $\mathcal{O}(h^3)$

Global truncation error: $\mathcal{O}(h^2)$

Runge-Kutta methods

$$\begin{cases} \frac{dx(t)}{dt} = f\big(t, x(t)\big) \\ x(t_0) = x_0 \end{cases}$$

$$x(t+h) = x(t) + h\,\phi\big(t, x(t)\big)$$

$$\phi\big(t, x(t)\big) = a_1 k_1 + a_2 k_2 + \cdots + a_n k_n$$

- For 2nd order methods

$$k_1 = f(t, x(t))$$

$$k_2 = f(t + \alpha h, x(t) + \beta k_1 h)$$

$$\phi\big(t, x(t)\big) = a_1 k_1 + a_2 k_2$$

Method	a_1	a_2	α	β
Explicit Euler	1	0	0	0
Heun's (Trapezoidal)	1/2	1/2	1	1
Midpoint	0	1	1/2	1/2
Ralston's	1/4	3/4	2/3	2/3

$$x(t+h) \approx x(t) + hf\big(t, x(t)\big) \quad \phi = 1 \cdot f\big(t, x(t)\big)$$

$$\phi = \frac{1}{2} f\big(t, x(t)\big) + \frac{1}{2} f\big(t + 1 \cdot h, \tilde{x}(t + 1 \cdot h)\big)$$

$$\phi = 0 \cdot f\big(t, x(t)\big) + 1 \cdot f\left(t + \frac{1}{2} h, \tilde{x}\left(t + \frac{h}{2}\right)\right)$$

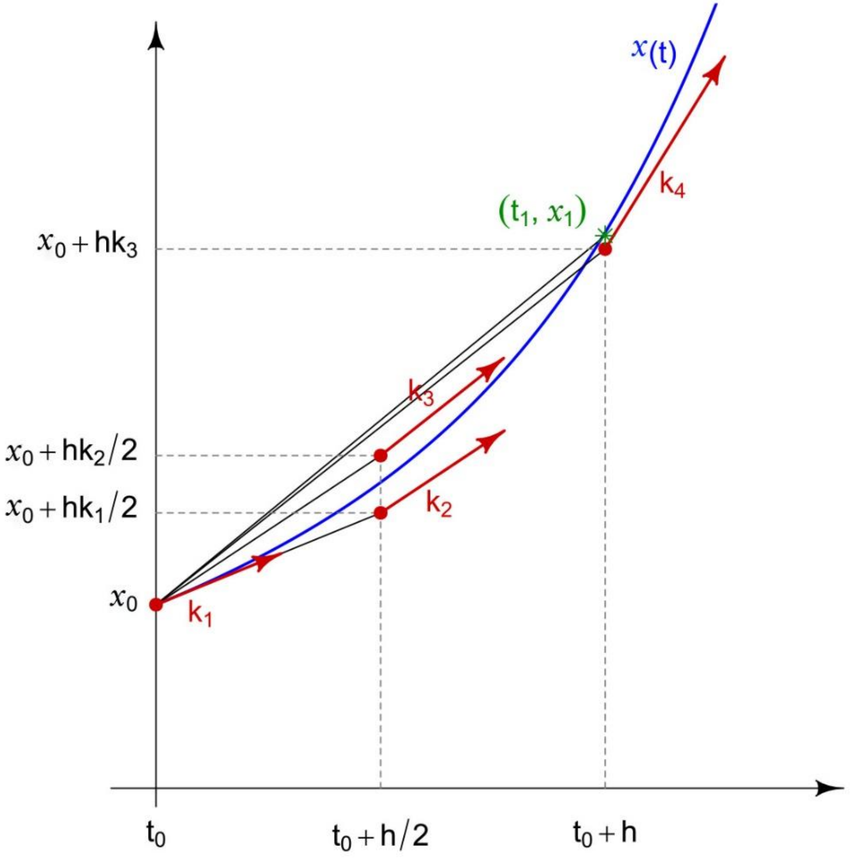
Aalto University
School of Electrical Engineering

4th order Runge Kutta

$$\begin{cases} \dfrac{dx(t)}{dt} = f\left(t, x(t)\right) \\ x(t_0) = x_0 \end{cases}$$

$$x(t+h) = x(t) + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

$$\begin{cases} k_1 = hf(t, x) \\ k_2 = hf\left(t + \frac{1}{2}h, x + \frac{1}{2}k_1\right) \\ k_3 = hf\left(t + \frac{1}{2}h, x + \frac{1}{2}k_2\right) \\ k_4 = hf(t + h, x + k_3) \end{cases}$$

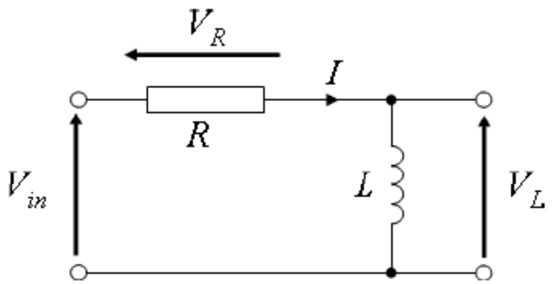


Local truncation error: $\mathcal{O}(h^5)$

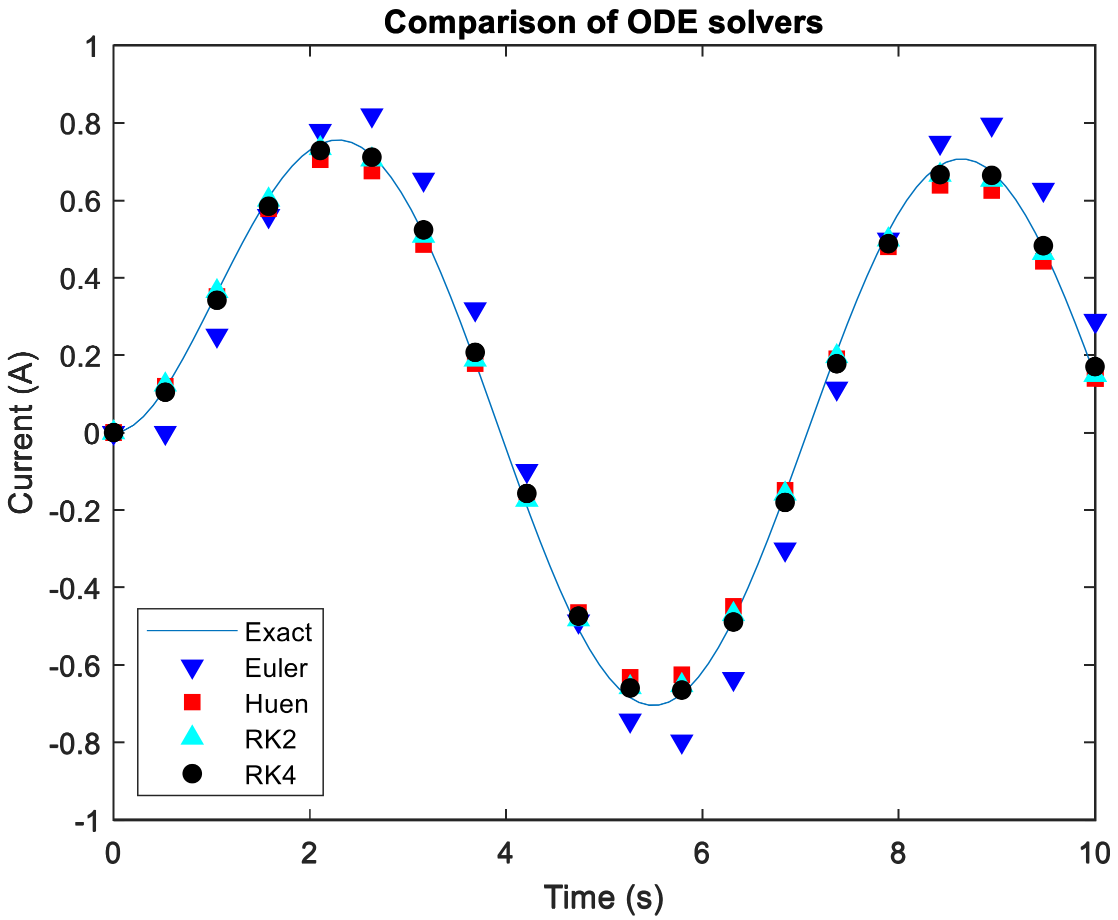
Global truncation error: $\mathcal{O}(h^4)$

Aalto University
School of Electrical Engineering

Initial value problems - Example



$$Ri + L\frac{di}{dt} = V_{in}$$
$$R = 1$$
$$L = 1$$
$$V_{in} = \sin(t)$$
$$\frac{di}{dt} = -i + \sin(t)$$



Code

```matlab
%% Comparism of different methods solving the RL problem
clear;
close all;
N = 20; t0 = 0; tN = 10;
t = linspace(t0,tN,N);
x = zeros(size(t));
f = @(t,x) -x+sin(t);
x(1) = 0;
h = (t(end)-t(1))/length(t);

%% Exact solution
t_exact = linspace(t0,tN,100);
x_exact = 0.5*exp(-t_exact) + sin(t_exact)/2 - cos(t_exact)/2;
plot(t_exact,x_exact);
axis([0 10 -1 1]); hold on;

%% Euler method
for i = 1:length(t)-1
    x(i+1) = x(i) + h * f(t(i),x(i));
end
plot(t,x,'bv','MarkerFaceColor','b');

%% Huen's method
for i = 1:length(t)-1
    x(i+1) = x(i) + (h/2) * (f(t(i),x(i)) + f(t(i)+h,x(i)+h*f(t(i),x(i))));
end
plot(t,x,'rs','MarkerFaceColor','r');

%% Second order Runge-Kutta
for i = 1:length(t)-1
    x(i+1) = x(i) + h * f(t(i)+h/2, x(i)+(h/2)*f(t(i),x(i)));
end
plot(t,x,'c^','MarkerFaceColor','c');

%% Fourth order Runge-Kutta
for i = 1:length(t)-1
    K1 = f(t(i),x(i));
    K2 = f(t(i) + h/2, x(i) + K1*h/2);
    K3 = f(t(i) + h/2, x(i) + K2*h/2);
    K4 = f(t(i) + h, x(i) + K3*h);
    x(i+1) = x(i) + (h/6)*(K1+2*K2+2*K3+K4);
end
plot(t,x,'ko','MarkerFaceColor','k');

legend('Exact','Euler','Huen','RK2','RK4','Location','southwest');
xlabel('Time (s)'); ylabel('Current (A)'); title('Comparison of ODE solvers');
hold off;
```

Aalto University
School of Electrical
Engineering

Solving higher-order ODEs

- So far, we are dealing with: $\begin{cases} \frac{dx(t)}{dt} = f\big(t, x(t)\big) \\ x(t_0) = x_0 \end{cases}$

- How to deal with higher-order ODE? e.g.:

$$\frac{d^3u}{dt^3} + u^2\frac{du}{dt} + \cos t \cdot u = g(t)$$

Step 1: Write the higher order ODE as first-order ODEs

Step 2: Solve the first-order ODE system with the time stepping methods.

Higher-order to system of first-order ODEs

$$\frac{d^3u}{dt^3} + u^2\frac{du}{dt} + \cos t \cdot u = g(t)$$

Let

$$x_1 = u$$
$$x_2 = \frac{dx_1}{dt}$$
$$x_3 = \frac{dx_2}{dt}$$

We have

$$x_3 = \frac{d^2u}{dt^2}$$
$$\Rightarrow \frac{dx_3}{dt} = -u^2\frac{du}{dt} - \cos t \cdot u + g(t)$$
$$= -{x_1}^2 x_2 - \cos t \cdot x_1 + g(t)$$

$$\Rightarrow \frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ -{x_1}^2 x_2 - \cos t \cdot x_1 + g(t) \end{bmatrix}$$

$$\begin{cases} \dfrac{dx(t)}{dt} = f\big(t, x(t)\big) \\ x(t_0) = x_0 \end{cases}$$

MATLAB ODE solvers: Example

$$\frac{d^2x(t)}{dt^2} - \mu(1 - x(t)^2)\frac{dx(t)}{dt} + x(t) = 0$$

Van der Pol oscillator

$x(t)$: position

t : time

$$x_1 = x(t)$$

$$x_2 = \frac{dx(t)}{dt}$$

$$\frac{dx_1}{dt} = x_2$$

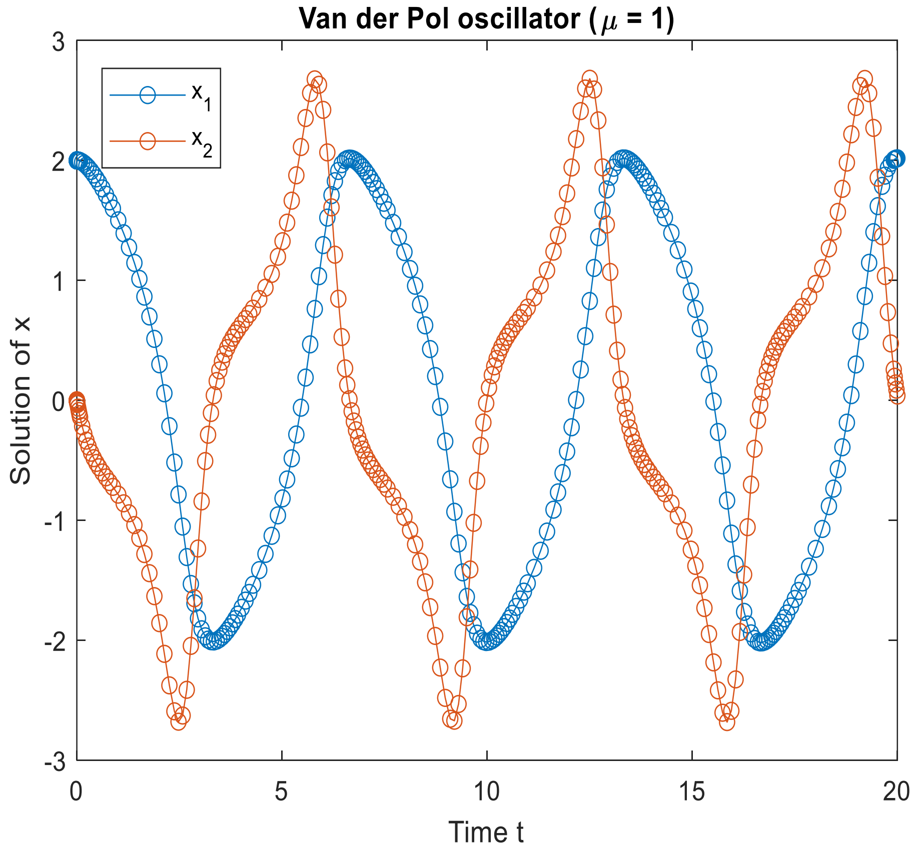
$$\frac{dx_2}{dt} = \mu(1 - {x_1}^2)x_2 - x_1$$

$$\Rightarrow \frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \mu(1 - {x_1}^2)x_2 - x_1 \end{bmatrix}$$

Code

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \mu(1 - {x_1}^2)x_2 - x_1 \end{bmatrix}$$

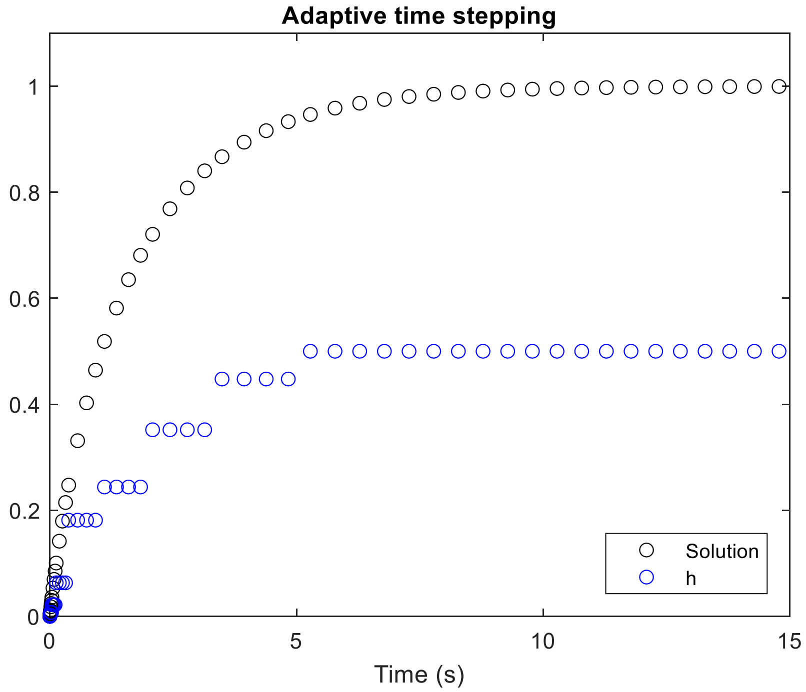
```
% Van der Pol oscillator
% Adpated from Matlab ODE45 example
clear all;
close all;

mu = 1;
f = @(t,x) [x(2); mu*(1-x(1)^2)*x(2)-x(1)];

timeSpan = [0 20];  % time span: [T0 TFinal]
initCond = [2; 0];  % initial consitions: [x1; x2]
options = odeset('RelTol',1e-4,'AbsTol',1e-6);

[t,x] = ode45(f,timeSpan,initCond);

plot(t,x(:,1),'-o',t,x(:,2),'-o');
title('Van der Pol oscillator (\mu = 1)');
xlabel('Time t');
ylabel('Solution of x');
legend('x_1','x_2','Location','NorthWest');
```



Time step

- What kind of time step should we use in simulation?
- Adaptive time stepping
  - When the solution changes slowly → Time steps are large
  - When the solution changes rapidly → Time steps are small

valid step: $|e| < AbsTol$ or $|e/x| < RelTol$




```matlab
%% Adaptive time stepping
clear all;
close all;

options = odeset('AbsTol',1e-6,'RelTol',1e-3,'Stats','on');
timeSpan = [0 20];
initCond = 0;
f = @(t,x) 1-sqrt(x);

[t,x] = ode45(f, timeSpan, initCond, options);

figure
for i = 1:length(t)-1
    h(i) = t(i+1)-t(i);
end
plot(t,x,'ko',t(1:end-1),h,'bo');
legend('Solution','h','Location','SouthEast');
axis([0 15 0 1.1]);
xlabel('Time (s)');
title('Adaptive time stepping');
```

What happens when time step *h* reduce to half?

MATLAB ODE solvers

Many solvers, e.g.

- ode45: 4th/5th order Runge-Kutta method.
  - Method of the first choice, very efficient in smooth solutions
- ode23: 2nd/3rd order Runge-Kutta
  - It may be more efficient than ode45 at crude tolerances and in the presence of moderate stiffness
- ode15s: variable order method
  - A good first choice for most stiff problems

MATLAB solvers have built-in adaptive time stepping

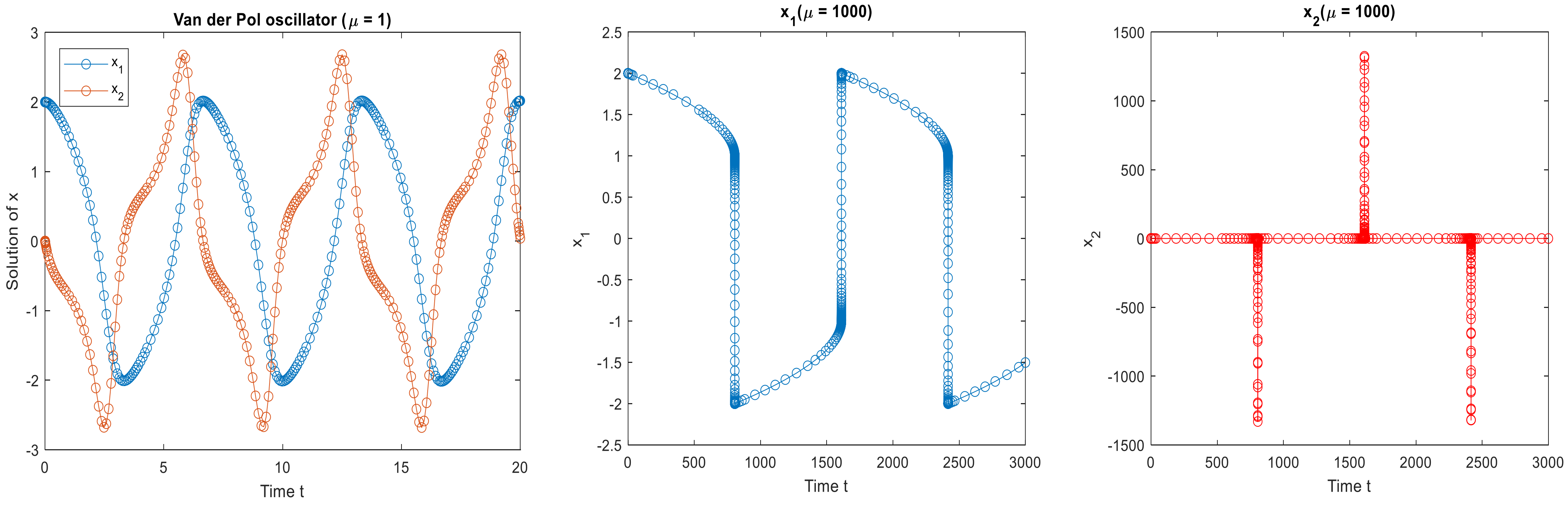
Stiff problem

- Differential equations make some solvers numerically unstable, unless the step size is extremely small
  - This happens even when the solution curve is smooth.
- It’s a phenomena, no precise mathematical definition
  - Often quick changes + slow changes is an indicator of a possible stiff problem
  - But the nature is more complicated, lies in the behavior of numerical methods
- Stiff solvers exists
  - The order is usually low

Stiff problem: example

- Van der Pol oscillator became a stiff system if μ is large

$$\frac{d^2x(t)}{dt^2} - \mu(1 - x(t)^2)\frac{dx(t)}{dt} + x(t) = 0$$



Use stiff solvers such as ode15s, ode23s → ode45 is not efficient for stiff equations.

Code

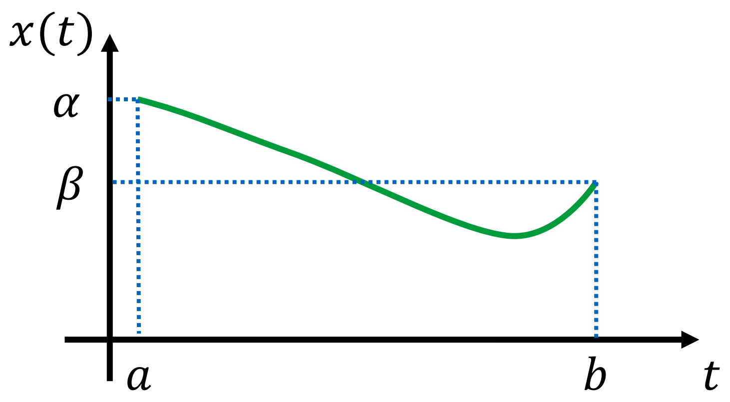
```
%% Van der Pol oscillator whenA mu is large
%
clear all;
close all;

timeSpan = [0 3000]; % time span : [Tleft Tright]
initCond = [2; 0];  % initial consitions : [y1 y2]
mu = 1000;


%% ODE45
%
t0=tic;
f = @(t,x) [x(2); mu*(1-x(1)^2)*x(2)-x(1)];
[t,x] = ode45(f,timeSpan,initCond);   %ode45 vs ode15s
T45 = toc(t0);

subplot(1,2,1);
plot(t,x(:,1),'-o');
xlabel('Time t');
ylabel('x_1');
title(['x_1'  '(\mu = ' num2str(mu) '), time used: ' num2str(T45) 's']);
axis([0 3000 -2.5 2.5]);

subplot(1,2,2);
plot(t,x(:,2),'r-o');
xlabel('Time t');
ylabel('x_2');
title(['x_2'  '(\mu = ' num2str(mu) ')']);
axis([0 3000 -1500 1500]);
pause;

%% ODE15s
%
t0=tic;
f = @(t,x) [x(2); mu*(1-x(1)^2)*x(2)-x(1)];
[t,x] = ode15s(f,timeSpan,initCond);   %ode45 vs ode15s
T15s = toc(t0);

subplot(1,2,1);
plot(t,x(:,1),'-o');
xlabel('Time t');
ylabel('x_1');
title(['x_1'  '(\mu = ' num2str(mu) '), time used: ' num2str(T15s) 's']);
axis([0 3000 -2.5 2.5]);

subplot(1,2,2);
plot(t,x(:,2),'r-o');
xlabel('Time t');
ylabel('x_2');
title(['x_2'  '(\mu = ' num2str(mu) ')']);
axis([0 3000 -1500 1500]);
```

More info can be found in MATLAB documents about ode15s, ode23s

Boundary-Value Problems

- Find solution of $x(t)$ with a given boundary values

$$\begin{cases} \dfrac{d^2x(t)}{dt^2} = f\left(t, x(t), \dfrac{dx(t)}{dt}\right) \\ x(a) = \alpha \quad x(b) = \beta \end{cases}$$



Can we solve the problem with the method we just learnt?

To solve the problem using methods discussed previously, we need two initial values, now we have only one

We need: $x(a) = \alpha \quad \dot{x}(a) = \gamma$

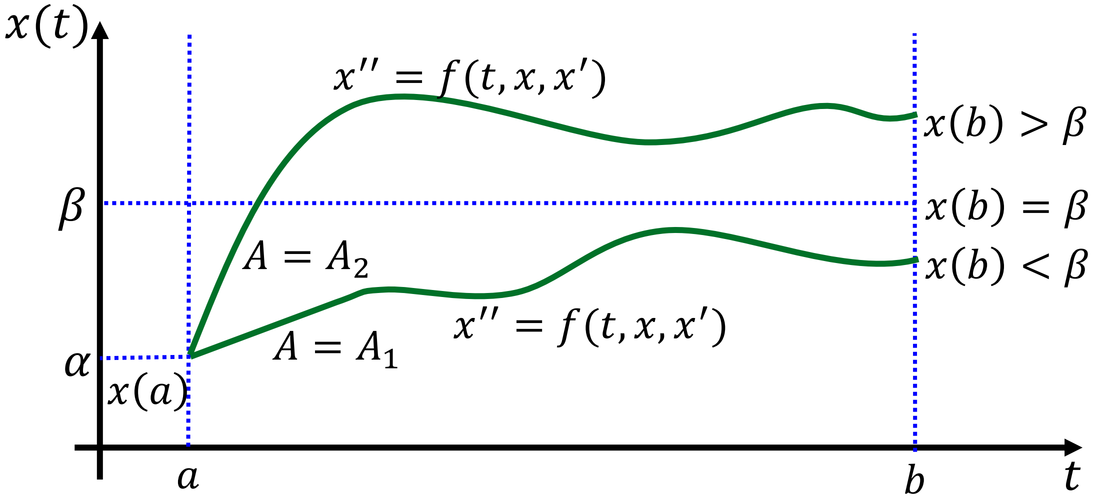
Shooting method

$$\begin{cases} \dfrac{d^2x(t)}{dt^2} = f\left(t, x(t), \dfrac{dx(t)}{dt}\right) \\ x(a) = \alpha \quad x(b) = \beta \end{cases}$$

We need: $x(a) = \alpha \quad \dot{x}(a) = \gamma$

Guessing the missing initial condition $x'(a) = A$

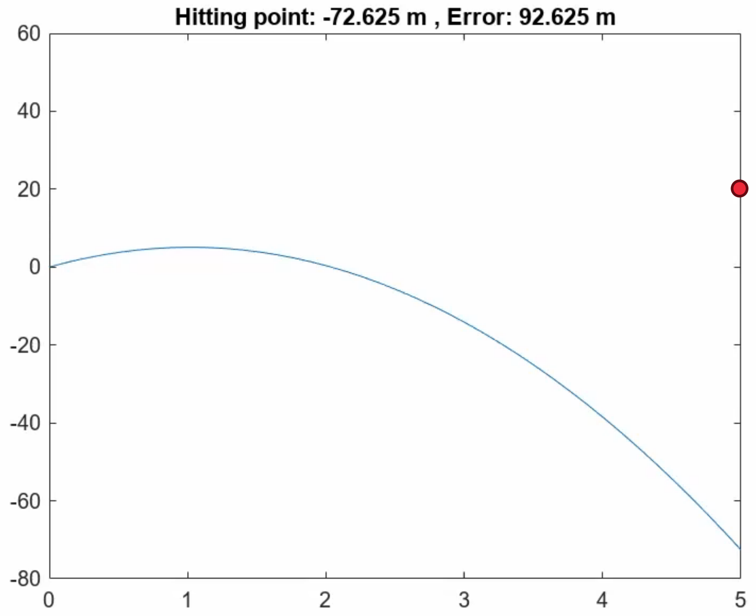
Based on the resulted $x(b)$, iteratively find the correct A



Example

- Free fall of a 1kg mass under standard gravity, known conditions
  – Position at time 0s is 0 m
  – Position at time 5s is 20 m

$$\frac{d^2x}{dt^2} = -g$$




```
%% Shooting method for boundary value problem
%
%  Free fall of a mass of 1Kg
%  d(dx/dt)/dt = -mg;
%  dx1 = x2
%  dx2 = -mg (where m = 1)
%
clear all;
close all;

style = ['- ';': ';'--';'-.';'. '];
styleIndex = 1;

timeSpan = [0 5]; % time span: [T0 TFinal]
P0 = 0;           % position at time T0
target = 20;      % position at time TFinal
A = 10;           % initial guess of the velocity at time 0
dA = 6;           % initial step size to change the A
tol = 0.1;        % tolerance of solution position at TFinal
error = 10;       % initial error value (some large number)

while(error > tol)

    init = [P0; A];% initial ponsitions: [x1 x2]
    g = 9.81;      % gravitational constant
    f = @(t,x) [x(2); -g];
    [t,y] = ode45(f,timeSpan,init);
    error = abs(y(end,1) - target);
    if y(end,1) < target
        A = A + dA;
    else
        A = A - dA;
        dA = dA/2;
    end
    figure(1), plot(t,y(:,1),style(styleIndex,:));
    styleIndex = styleIndex + 1;
    if styleIndex > length(style)
        styleIndex = 1;
    end
    title(['Hitting point: ' num2str(y(end,1))...
        ' m , Error: ' num2str(error) ' m']);
    axis([0 5 -80 60]);
    hold on;    % break point here
    pause(2);
end
```

Finite difference method

- Derivatives in the ODE are replaced by finite divided differences
- Recall that a derivative be approximated:

$$\frac{d}{dt}\big(x(t)\big) \approx \frac{x(t+h)-x(t)}{h}$$

- The ODE can become a set of algebraic equations with intermediate points
- Solution is obtained on the intermediate points

	Forward difference	Backward difference	Central difference
$\frac{dx(t)}{dt}$	$\frac{x(t+h)-x(h)}{h}$	$\frac{x(h)-x(t-h)}{h}$	$\frac{x(t+h)-x(t-h)}{2h}$
$\frac{d^2x(t)}{dt^2}$	$\frac{x(t+2h)-2x(t+h)+x(h)}{h^2}$	$\frac{x(t)-2x(t-h)+x(x-2h)}{h^2}$	$\frac{x(t+h)-2x(t)+x(t-h)}{h^2}$

Central difference is more accurate, but on boundaries it may not be available

Example

	Forward difference	Backward difference	Central difference
$\frac{dx(t)}{dt}$	$\frac{x(t+h)-x(h)}{h}$	$\frac{x(h)-x(t-h)}{h}$	$\frac{x(t+h)-x(t-h)}{2h}$
$\frac{d^2x(t)}{dt^2}$	$\frac{x(t+2h)-2x(t+h)+x(h)}{h^2}$	$\frac{x(t)-2x(t-h)+x(x-2h)}{h^2}$	$\frac{x(t+h)-2x(t)+x(t-h)}{h^2}$

- The previous ODE: $\frac{d^2x}{dt^2} = -g$ becomes, using central difference

$$\frac{x(t+h) - 2x(t) + x(t-h)}{h^2} = -g$$

Or

$$x(t-h) - 2x(t) + x(t+h) = -gh^2$$

- Boundary conditions:
  - Position at time 0s is 0 m
  - Position at time 5s is 20 m

- Let $h = 1s$
  - @t=0 $x(0) = 0$
  - @t=1 $x(0) - 2x(1) + x(2) = -gh^2$
  - @t=2 $x(1) - 2x(2) + x(3) = -gh^2$
  - @t=3 $x(2) - 2x(3) + x(4) = -gh^2$
  - @t=4 $x(3) - 2x(4) + x(5) = -gh^2$
  - @t=5 $x(5) = 20$

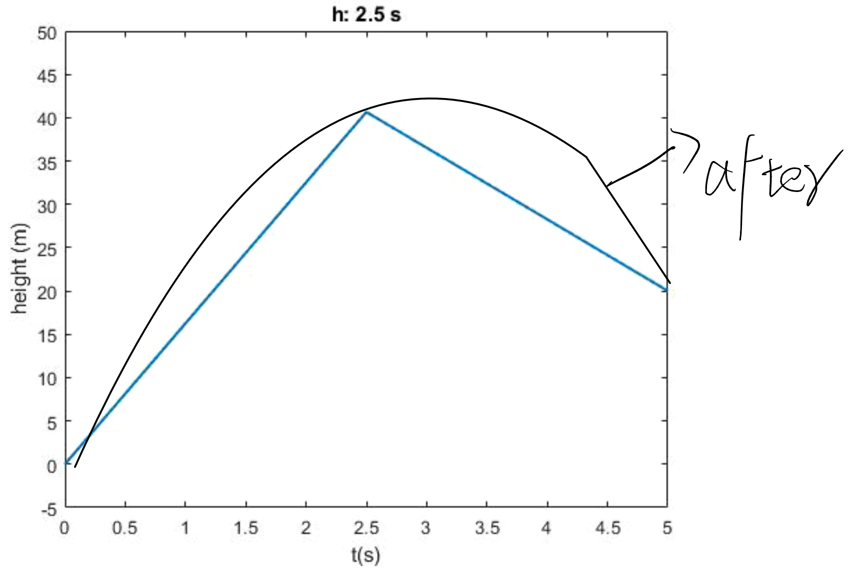
4 equations, 4 unknowns

$$\begin{bmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -2 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} = \begin{bmatrix} -gh^2 - x(0) \\ -gh^2 \\ -gh^2 \\ -gh^2 - x(5) \end{bmatrix}$$

For *M* intermediate points

$$\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & \ddots & & \\ & 1 & \ddots & 1 & \\ & & \ddots & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(M-1) \\ x(M) \end{bmatrix} = \begin{bmatrix} -gh^2 - x(0) \\ -gh^2 \\ \vdots \\ -gh^2 \\ -gh^2 - x(M+1) \end{bmatrix}$$

Example



$$\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & \ddots & & \\ & 1 & \ddots & 1 & \\ & & \ddots & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(M-1) \\ x(M) \end{bmatrix} = \begin{bmatrix} -gh^2 - x(0) \\ -gh^2 \\ \vdots \\ -gh^2 \\ -gh^2 - x(M+1) \end{bmatrix}$$

```
%% Finite divided differences for boundary value problem
%
% Free fall of a mass of 1 kg
% d(dx/dt)/dt=-mg
% This method replaces the derivatives with finite divided differences
clear; close all;

timeSpan = [0 5];          % time span: [T0 TFinal]
g  = 9.81;                 % the gravitational constant
P0 = 0;                    % position at time TO
PF = 20;                   % position at time TFinal
N  = 3;                    % number of total time steps including the start and end
                           % of the span. the number of intermediate points are N-2
% solving for different Ns
for N = 3:1:11
    M = N-2;               % M is the number of intermediate time steps

    t = linspace(timeSpan(1), timeSpan(2), N);
    h = t(2)-t(1);         % the timestep

    % generate the equations in matrix form

    A = zeros(M,M);        % defining the coefficient matrix
    b = ones(M,1)*-g*h^2;% defining the RHS vector

    A(1:1+M:M*M) = -2;     % adjusting the diagonal and off-diagonal elements
    A(M+1:1+M:M*M) = 1;
    A(2:1+M:M*M-M) = 1;

    b(1) = b(1)-P0;        % adjusting the first and last row of the RHS vector
    b(end) = b(end)-PF;    % by the known values on the boundaries

    x_int = A\b;           % now the ODE is an algebraic set of equations
    x = [P0;x_int;PF];     % adding known solutions from the boundaries
    plot(t,x,'-');
    title(['h: ' num2str(h) ' s' ]);
    axis([0 5 -5 50]);
    xlabel('t(s)'); ylabel('height (m)');
    hold on;
    pause(2);
end
```

Smaller step size → better accuracy → larger matrices

Boundary value problems – MATLAB bvp solver

- MATLAB boundary value problem solver using finite difference methods
- Let's formulate the following boundary value problem in MATLAB

$$\frac{d^2x(t)}{dt} + 3\frac{dx(t)}{dt} + 6x = 5$$
$$t \in [1,3]$$

- Boundary conditions:

$$\begin{cases} x(1) = 3 \\ x(3) + 2\frac{dx(3)}{dt} = 5 \end{cases}$$

- Let's formulate it into a system of 1st order differential equations:

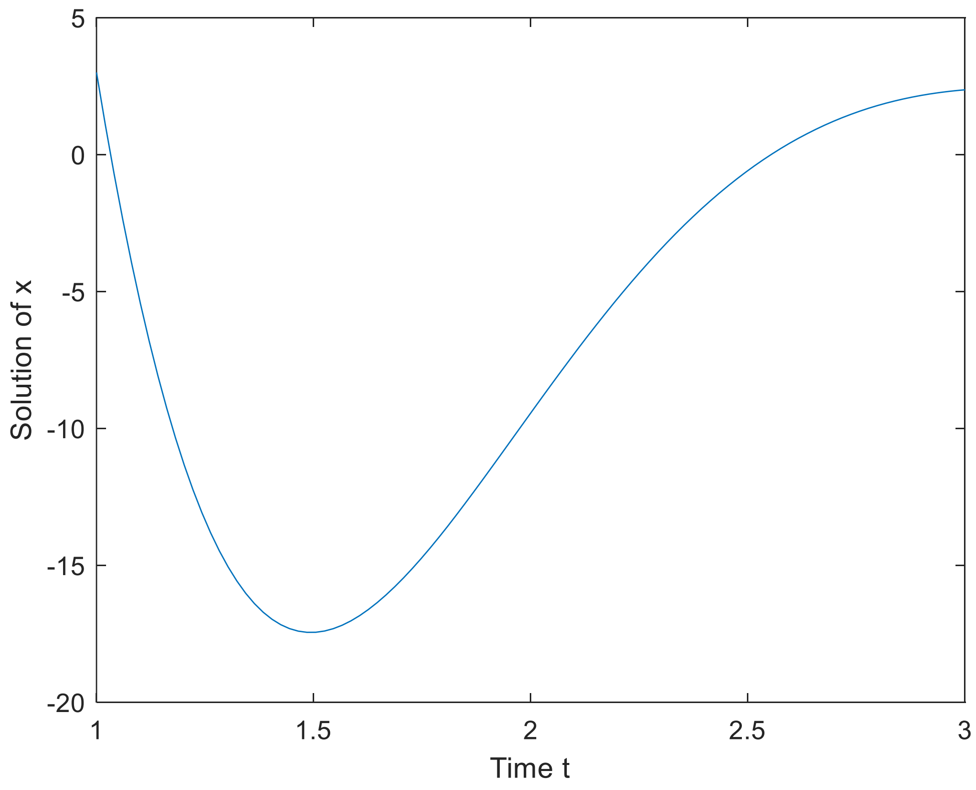
$$\frac{d}{dt}\begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} = \begin{pmatrix} x_2(t) \\ 5 - 3x_2(t) - 6x_1(t) \end{pmatrix}$$

- Boundary conditions:

$$\begin{cases} f(x_1, x_2) = x_1 - 3 = 0 \\ g(x_1, x_2) = x_1 + 2x_2 - 5 = 0 \end{cases}$$

Matlab implementation

```
%% solving boundary value problems using Matlab bvp calls

odeFunc = @(t,x) [x(2); 5 - 3*x(2) - 6*x(1)];

bc = @(xL,xR) [xL(1)-3; xR(1)+2*xR(2)-5];

init = bvpinit(linspace(1,3,10),[0 0]);
sol = bvp4c(odeFunc, bc, init);
t = linspace(1,3,100);
BS = deval(sol,t);
plot(t,BS(1,:));
xlabel('Time t'); ylabel('Solution of x')
```



$$\frac{d}{dt}\begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} = \begin{pmatrix} x_2(t) \\ 5 - 3x_2(t) - 6x_1(t) \end{pmatrix}$$

Boundary conditions: $\begin{cases} f(x_1, x_2) = x_1 - 3 = 0 \\ g(x_1, x_2) = x_1 + 2x_2 - 5 = 0 \end{cases}$

Aalto University
School of Electrical
Engineering

Remarks on boundary value problem

- In contrast to initial value problem, which always has a unique solution, BVP may have:
  - No solution
  - A finite number of solutions
  - Infinitely many solutions

Summary

- Solving ordinary differential equations as an initial value problem
  - Numerical methods for solving ODEs
  - Adaptive time stepping
  - Stiff equations
  - Formulation of initial value problem in MATLAB
- Solving ordinary differential equation as a boundary value problem
  - Shooting method
  - Finite difference method
  - Formulation of boundary value problem in MATLAB

Next session

- Simulation using Simulink
- Don’t forget to install MATLAB and have your laptop ready before the lecture

Readings

- Ward Cheney, David Kincaid, Numerical Mathematics and Computing, 2008, Chapter 10.1, 10.2, 14.1
- MATLAB documentation on choosing ODE solvers
- MATLAB documentation on the following commands: ode45, ode15s, bvp4c, odeset