

# EL1820 Modeling of Dynamical Systems

## Lecture 5 - Simulation

- Numerical methods for solving ODEs
- Numerical methods for DAEs
- Dealing with discontinuous systems

## Today's goal

You should be able to

- know the basic ideas behind numerical ODE solvers
- choose an integration routine suited to your problem
- understand the additional issues with DAEs
- help the solver to treat discontinuous systems

## Simulation

Modeling of dynamical systems often results in ODEs or DAEs

$$\dot{x} = f(x, t) \quad (\text{ODE}) \qquad 0 = F(\dot{x}, x, t) \quad (\text{DAE})$$

Given an initial state  $x(0) = x_0$ , the solution to the model equations

$$x(t; x(0) = x_0)$$

predicts how the true system evolves over time, when starting from  $x_0$

By *simulating* the system, we mean solving the model equations

## Numerical solution of ODEs

**Observation** Analytical solution to ODEs

$$\dot{x} = f(x, t) \qquad x(0) = x_0$$

is almost never possible. Need to use numerical methods!

Numerical solution: a sequence

$$\{(\hat{x}_k, t_k)\}$$

of approximations  $\hat{x}_k$  to the exact solution  $x(t_k; x(0) = x_0)$

## From ODE to difference equation

We can represent the solution  $x(t + h)$  by a Taylor expansion

$$x(t + h) = x(t) + h\dot{x}(t) + \frac{h^2}{2!}\ddot{x}(t) + \dots$$

When  $h$  is small, we can write

$$x(t + h) = x(t) + h\dot{x}(t) + \mathcal{O}(h^2) \approx x(t) + hf(x, t)$$

This approximation results in *Euler's method* (1742)

$$\hat{x}_{k+1} = \hat{x}_k + hf(\hat{x}_k, t_k)$$

$$t_{k+1} = t_k + h$$

Numerical solution  $\{(\hat{x}_k, t_k)\}$  by iterating the difference equations

## Euler's method as finite differences

We can also view Euler's method as a finite difference approximation

$$\dot{x}(t_k) \approx \frac{x(t_{k+1}) - x(t_k)}{t_{k+1} - t_k}$$

*i.e.*, as an approximation of  $\dot{x} = f(x)$  by

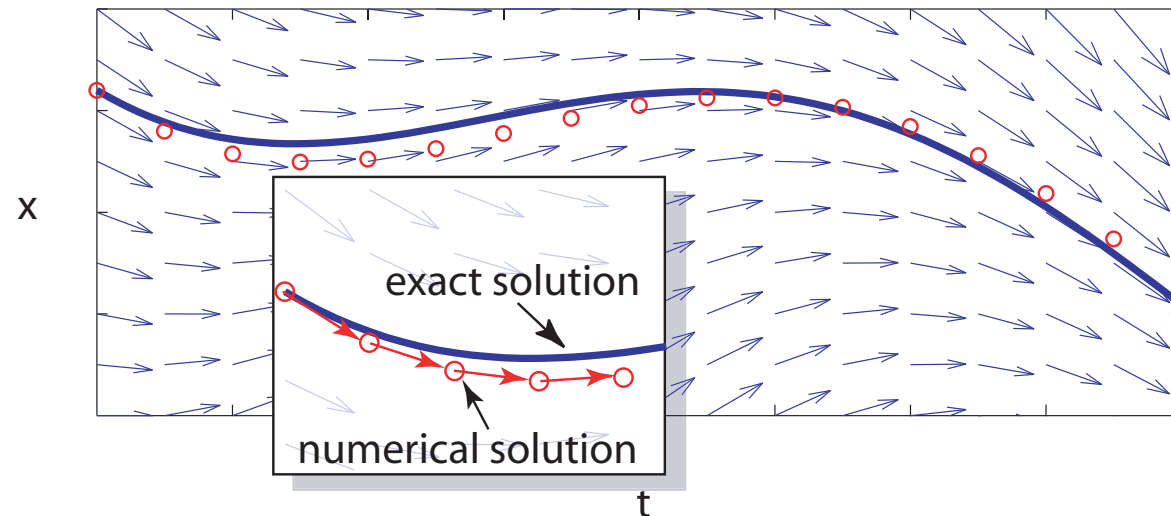
$$\frac{\hat{x}(t_{k+1}) - \hat{x}(t_k)}{t_{k+1} - t_k} = f(\hat{x}_k, t_k)$$

Assuming  $t_{k+1} - t_k = h$  and rearranging terms gives Euler's method

$$\hat{x}_{k+1} = \hat{x}_k + hf(\hat{x}_k, t_k)$$

$$t_{k+1} = t_k + h$$

# Exact vs. numerical solution



## Exact vs. numerical solution

- numerical solution contains errors
- errors depend on method, stepsize and underlying problem

Important to understand the accuracy of numerical methods!

## Accuracy: local errors and propagation

Numerical accuracy characterized by *global error* (“total” error)

$$g_n = \hat{x}_n - x(t_n; x(0) = x_0)$$

Global error is due to *local errors* (“per step” errors)

$$l_n = \hat{x}_n - x(t_n; x(t_{n-1}) = \hat{x}_{n-1})$$

and error propagation

$$g_n = \mathcal{F}(g_{n-1}) + l_n$$

Errors grow unboundedly if propagation is unstable!

## Numerical accuracy: order of local errors

Numerical method is of *order*  $p$  if local error is  $\mathcal{O}(h^{p+1})$  as  $h \rightarrow 0$

**Example** What is the order of Euler's method?

Use the definition of local error

$$l_{n+1} = \hat{x}_{n+1} - x(t_{n+1}; x(t_n) = \hat{x}_n)$$

and the equations for Euler's method

$$\hat{x}_{n+1} = \hat{x}_n + hf(\hat{x}_n)$$

$$x(t_{n+1}; x(t_n) = \hat{x}_n) = \hat{x}_n + h\dot{x}(t_n) + \frac{h^2}{2!}\ddot{x}(t_n) + \dots$$

Rearranging terms gives  $l_n = \mathcal{O}(h^2)$ , so the method is of first order

## Numerical accuracy: stability

Method is *numerically unstable* if error grows unboundedly

Study numerical accuracy of Euler's method on test equation

$$\dot{x} = -\lambda x \qquad x(0) = x_0$$

with  $\lambda > 0$ . Solution  $x(t) = \exp(-\lambda t)x_0$  converges to zero as  $t \rightarrow \infty$

Euler's method with  $h_k = h$  from initial condition  $x_0$ ,

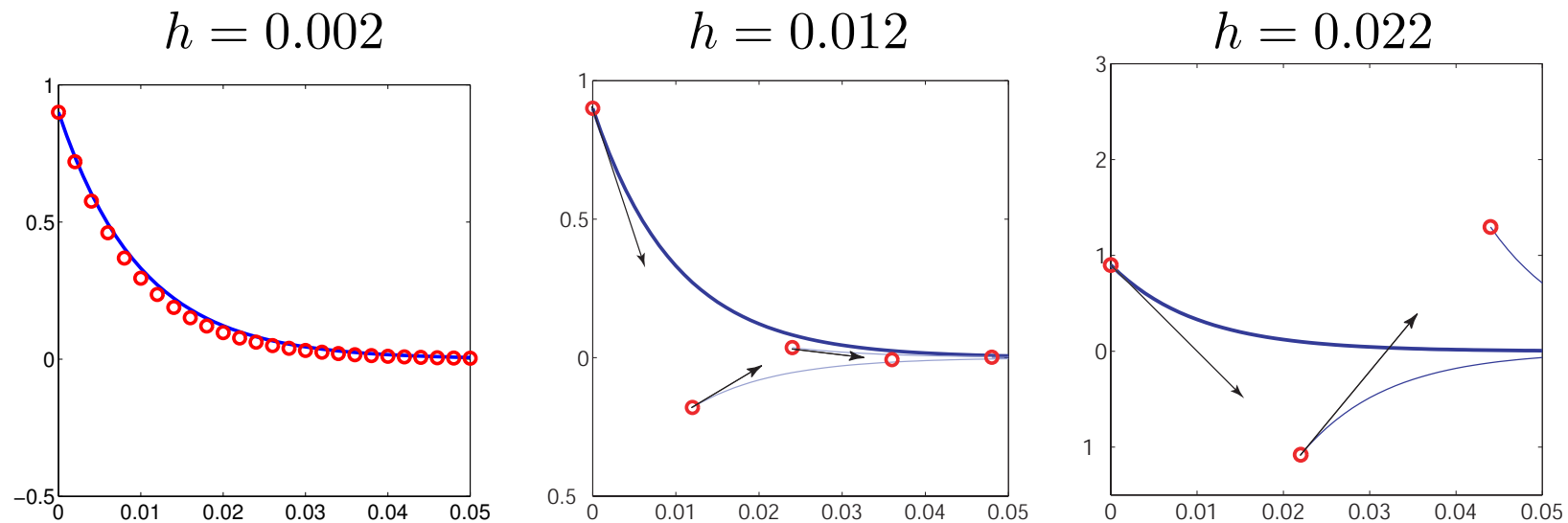
$$\hat{x}_{k+1} = \hat{x}_k - h\lambda\hat{x}_k = (1 - \lambda h)\hat{x}_k = (1 - \lambda h)^{k+1}x_0$$

converges to zero if  $|1 - \lambda h| < 1$ , diverges otherwise

- need to choose sufficiently small stepsize to ensure stability!

## Numerical stability example

**Example** Euler's method on test equation:  $\dot{x} = -100x$



Numerically stable for  $h < 0.02$  (but inaccurate long before)

## Step-size selection

Trade-off between efficiency and accuracy

- larger step sizes require less computations,
- but step size restricted by stability and accuracy concerns

Modern integration methods use adaptive step-size selection

**Example** In Euler's method, local error is approx  $(h_k^2/2)f'_k$ , so

$$h_k \leq \sqrt{2 \frac{\text{tol}}{\|f'_k\|}}$$

ensures that local error is less than tol ( $f'_k$  estimated numerically)

## Observations on Euler's method

Euler's method

- approximate derivative by finite difference
- difference equation iterated to find numerical solution to ODE

Simple, but inefficient

- low accuracy order and
- restricted numerical stability

forces us to use small step sizes

Can we do better?

## Implicit Euler

Implicit Euler uses the backward difference approximation

$$\dot{x}(t_{k+1}) \approx \frac{x(t_{k+1}) - x(t_k)}{h}$$

to obtain the iteration

$$\begin{aligned}\hat{x}_{k+1} &= \hat{x}_k + hf(\hat{x}_{k+1}, t_{k+1}) \\ t_{k+1} &= t_k + h\end{aligned}$$

Note that  $\hat{x}_{k+1}$  is implicitly defined

- need to solve nonlinear equation at each time step
- only interesting if we can use longer time steps than explicit Euler

## Solving the implicit equation

Implicit equation

$$\hat{x}_{k+1} = \hat{x}_k + hf(\hat{x}_{k+1}, t_{k+1})$$

typically solved by Newton's method or simply iterating the equation

Initial guess for  $\hat{x}_{k+1}$  obtained using explicit method

Often called *predictor-corrector* scheme:

- explicit method produces prediction of  $\hat{x}_{k+1}$
- implicit method computes a corrected estimate

## Numerical stability of implicit Euler

Consider again the test equation

$$\dot{x} = -\lambda x \qquad x(0) = x_0$$

Implicit Euler with  $h_k = h$  from initial condition  $x_0$

$$\hat{x}_{k+1} = \hat{x}_k - h\lambda\hat{x}_{k+1} \Rightarrow \hat{x}_{k+1} = \frac{1}{1 + \lambda h} \hat{x}_k = \frac{1}{(1 + \lambda h)^{k+1}} x_0$$

converges to zero if  $|1 + \lambda h| > 1$

- implicit Euler is numerically stable for all step sizes!

## Observations on implicit Euler

### Implicit Euler

- more computations in each step (need to solve implicit equation)
- but can take significantly longer steps

Numerically stable for all step-sizes, but

- step size still limited by low (first-order) accuracy

Can we do better?

## Improving accuracy: Runge-Kutta methods

The Runge-Kutta (RK) methods are simple, yet more accurate

### Basic idea

$$x(t_{k+1}) = x(t_k) + \int_{t_k}^{t_{k+1}} f(x(s), s) ds \approx x(t_k) + hf(x(t_{\text{mid}}), t_{\text{mid}})$$

where  $t_{\text{mid}} = t_k + h/2$  and  $x(t_{\text{mid}})$  is computed using an Euler step:

$$\begin{aligned}\hat{x}_{\text{mid}} &= \hat{x}_k + \frac{h}{2}f(\hat{x}_k, t_k) & t_{\text{mid}} &= t_k + h/2 \\ \hat{x}_{k+1} &= \hat{x}_k + hf(\hat{x}_{\text{mid}}, t_{\text{mid}}) & t_{k+1} &= t_{\text{mid}} + h/2\end{aligned}$$

This Runge-Kutta method has accuracy of order 2

RK methods with higher accuracy by evaluating  $f$  in more points

## Increasing accuracy: multistep methods

Higher accuracy can also be obtained using multi-step methods

$$\hat{x}_{k+1} = \hat{x}_k + \sum_{t=0}^T \beta_t f_{k+1-t} \quad f_l = f(\hat{x}(t_l), t_l)$$

can be implicit  $\beta_0 \neq 0$  or explicit  $\beta_0 = 0$

One of the most popular multistep methods is due to Adams

- explicit/implicit often combined in predictor-corrector scheme
- implicit methods better stability properties (but not unconditional)

# Stiffness

Systems with drastically different timescales

- transient of fast dynamics irrelevant for long-term solution, still
- fast dynamics restricts step-size (to avoid numerical instability)

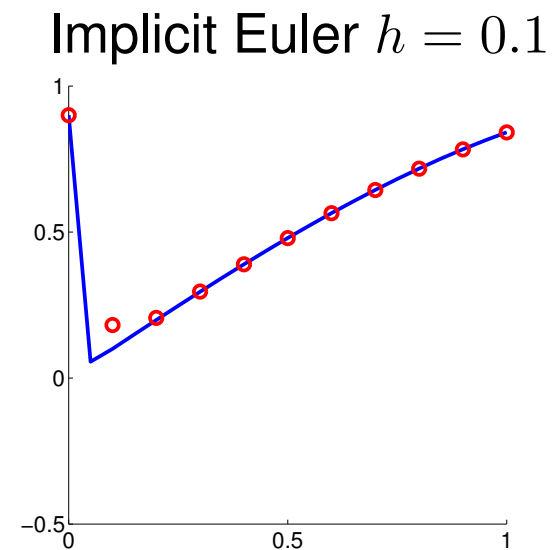
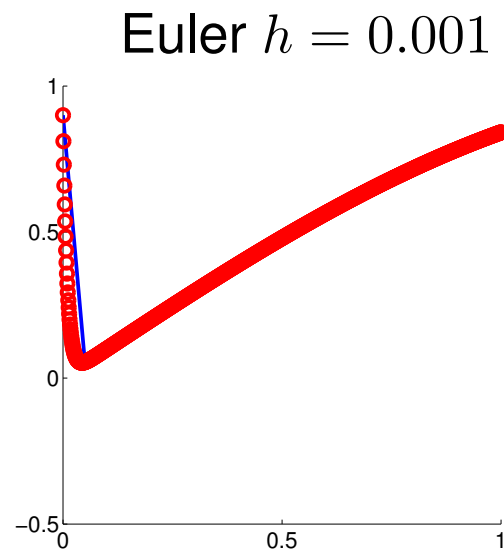
**Prototype example:**

$$\dot{x}(t) = -\lambda \{x(t) - \sin(t)\} + \cos(t)$$

with solution  $x(t) = \exp(-\lambda t)x_0 + \sin(t)$ . **Stiff when  $\lambda > 100$**

## Stiffness example

Prototype example for  $\lambda = 100$



Implicit Euler can easily use 100 times larger step sizes!

## Stiff systems abound

Physical modeling easily results in stiff models

**Example** DC motor drive from Lecture 3

$$\begin{aligned}\frac{d}{dt}i &= \frac{1}{L} (v_S - Ri - k_m\omega) \\ \frac{d}{dt}\omega &= \frac{1}{J} (k_m i - \gamma\omega)\end{aligned}$$

with  $L = 0.01, k_m = \gamma = 0.1, R = 1, J = 100$

Runge-Kutta needs 25 times more integration steps than stiff solver

## Choosing integration routine

What integration routine to choose depends on several factors

- required accuracy
- complexity of evaluating  $f(x, t)$
- whether or not problem is stiff

General recommendations

- use stiff solvers only for stiff problems
- use high-order methods only if high accuracy is required
- use RK for low-complexity  $f$ , and Adams for high-complexity  $f$

## Understanding the Simulink manual

- `ode45` is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a one-step solver; that is, in computing  $y(t_n)$ , it needs only the solution at the immediately preceding time point,  $y(t_{n-1})$ . In general, `ode45` is the best solver to apply as a first try for most problems
- `ode113` is a variable-order Adams-Bashforth-Moulton PECE solver. It can be more efficient than `ode45` at stringent tolerances. `ode113` is a multistep solver; that is, it normally needs the solutions at several preceding time points to compute the current solution
- `ode15s` is a variable order solver based on the numerical differentiation formulas (NDFs). These are related to but are more efficient than the backward differentiation formulas, BDFs (also known as Gears method). Like `ode113`, `ode15s` is a multistep method solver. If you suspect that a problem is stiff, or if `ode45` failed or was very inefficient, try `ode15s`

# EL1820 Modeling of Dynamical Systems

## Lecture 6 - Simulation

- Numerical methods for solving ODEs
- Numerical methods for DAEs
- Dealing with discontinuous systems

## Numerical solution of DAEs

Consider the DAE

$$F(\dot{z}, z, t) = 0$$

Approximate derivative  $\dot{z}$  by backward difference (as in implicit Euler)

$$\dot{z}(t_k) \approx \frac{z_k - z_{k-1}}{h}$$

Compute  $z_k$  by solving

$$F\left(t_k, z_k, \frac{z_k - z_{k-1}}{h}\right) = 0$$

using Newton's method. This is (roughly) the way DASSL works

## Numerical solution of DAEs on normal form

To illustrate the issues with numerical solution of DAEs, consider

$$\begin{aligned}\frac{d}{dt}w_2 &= -w_1 + u_1 \\ 0 &= -w_2 + u_2\end{aligned}$$

This is an index-2 DAE in normal form. Backward difference gives

$$\begin{aligned}w_{1,k} &= \frac{w_{2,k} - w_{2,k-1}}{h} + u_{1,k} = \frac{u_{2,k} - u_{2,k-1}}{h} + u_{1,k} \\ w_{2,k} &= u_{2,k}\end{aligned}$$

Differentiating a (possibly noisy) input signal may cause difficulties!

- modeling systems try to reduce index before simulation

## Modern DAE solvers

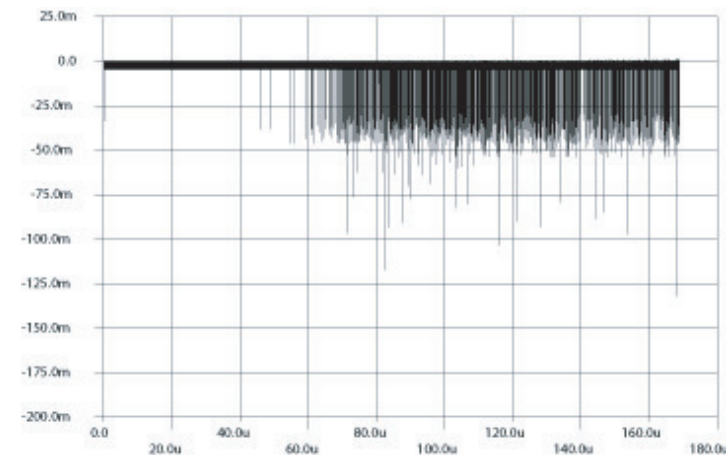
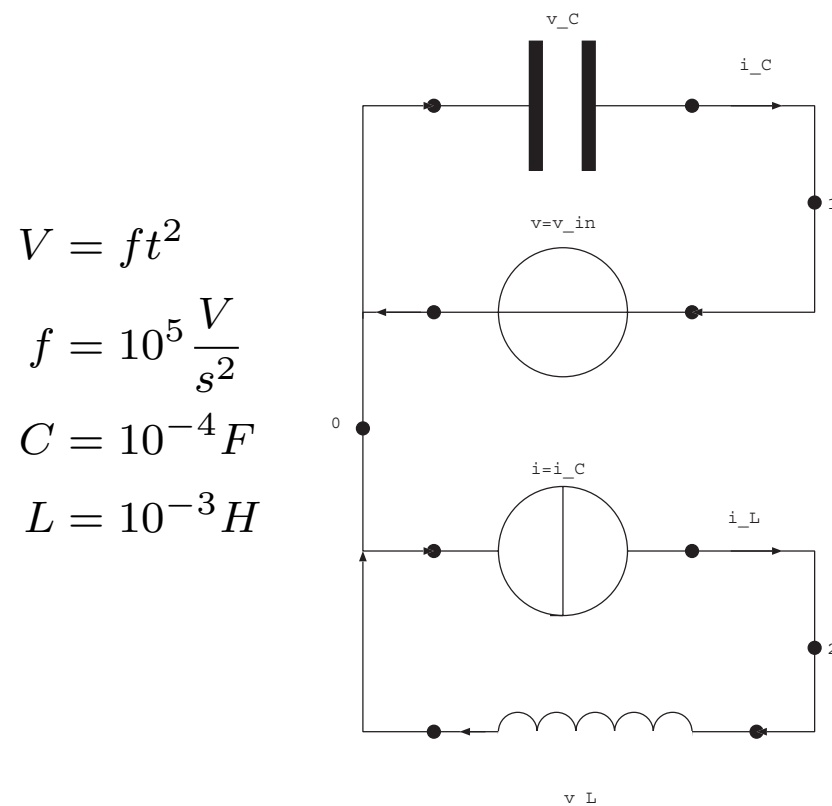
Modern DAE solvers work differently

- try to reduce index prior to numerical integration
- enforce algebraic constraints in difference approximation
  - projection techniques
  - integration routines that preserve invariants (*e.g.*, energy)

We can still expect progress on numerical methods for solving DAEs

# High-index DAEs hard to simulate

**Example.** Varactor circuit



Despite 1.3 million integration steps, solver does not converge!

# EL1820 Modeling of Dynamical Systems

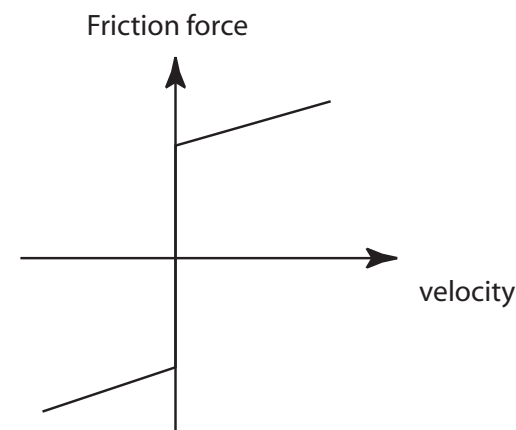
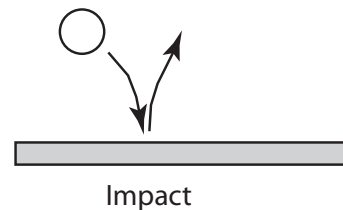
## Lecture 6 - Simulation

- Numerical methods for solving ODEs
- Numerical methods for DAEs
- Dealing with discontinuous systems

# Modeling with Discrete Events

Discrete events

- useful abstraction of continuous behaviour
- idealized models of fast dynamics or saturated high-gain elements



## Example: Bouncing ball

Consider the following simple model of a bouncing ball

Free fall when  $y(t) > 0$

$$\ddot{y}(t) = -g$$

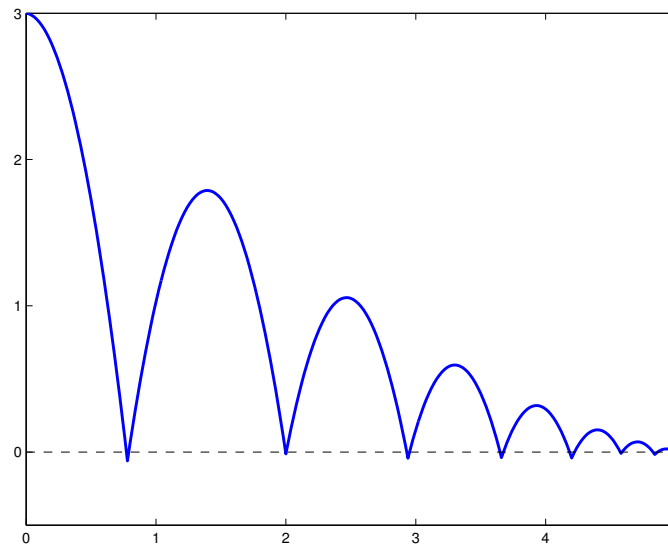
Collision when  $y(t) = 0$

$$\dot{y}(t) = -cy(t^-) \qquad y(t) = y(t^-)$$

Here  $\dot{y}(t^-)$  denotes the velocity just before impact

## Example: bouncing ball

Simulation using backward Euler with fixed step-size



Result is clearly incorrect

- ball passes through the floor at several occasions
- reason: collisions happen between time steps of ODE solver

## Handling discontinuities

Important for ODE solver to locate times of discrete events!

Standard implementation:

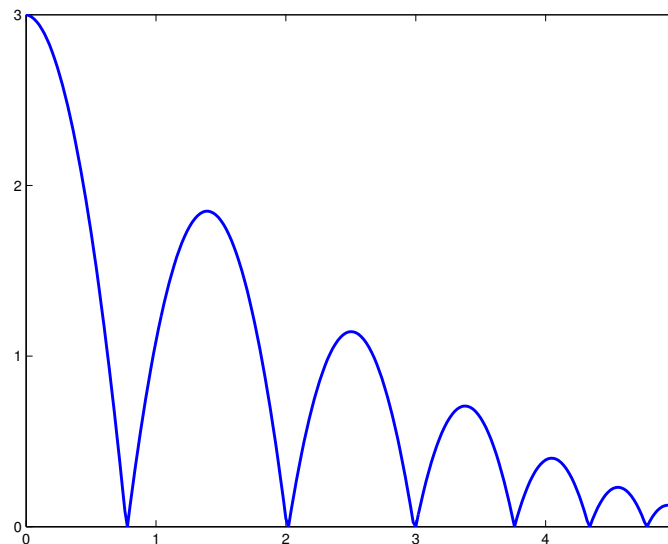
- user informs solver about events using “guard functions”
- discrete events happen when guard function crosses zero
- solver monitors guards, forces integration step at event times

## Example: bouncing ball

Inform solver about collision event using guard

$$g(y(t)) = y(t)$$

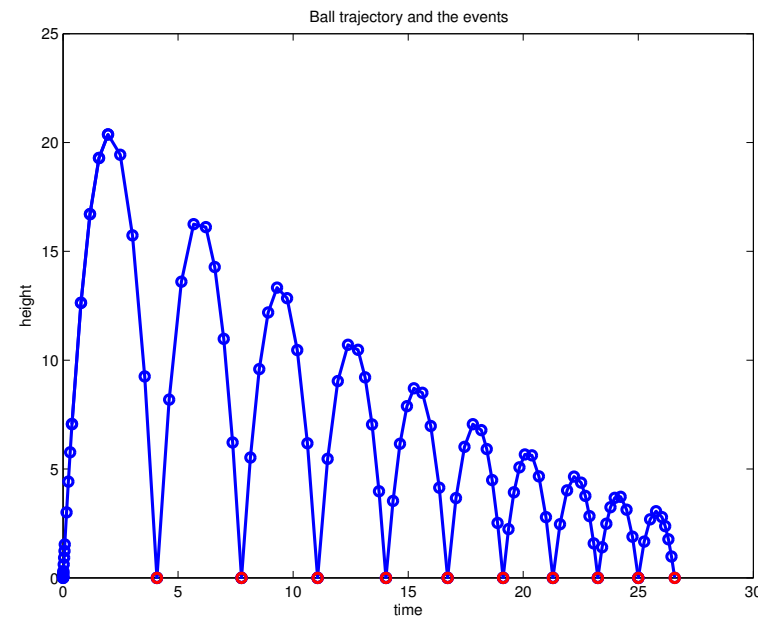
Result using backward Euler (same parameters as previously)



# Bouncing ball using state-of-the art solver

State-of-the art solvers combine many features

- variable step-length for efficient simulation
- accurate event detection
- interpolation to provide values between integration steps



# Summary

## Simulation

- solving the modelling equations

## Numerical simulation

- approximation of modelling equations at discrete time steps

## A wide range of methods

- Implicit/explicit, single/multi-step, . . .

## Need to understand key issues

- stability and accuracy
- stiff systems
- dealing with discrete events

## **Next lecture**

Review of probability and statistics