EE4323 – Industrial Control Systems
Module 3: Modeling & Simulation of Nonlinear Dynamic Systems

James H. Taylor
Department of Electrical & Computer Engineering
University of New Brunswick
Fredericton, New Brunswick, Canada E3B 5A3
e-mail: jtaylor@unb.ca
web site: www.ee.unb.ca/jtaylor/

22 March 2017

Modeling & Simulation Overview

- **Motivation**
- **Basic concepts**
- **Modeling; model categories**
- **Matching methods to models**
- **Predictor/corrector algorithms**
- **Runge-Kutta methods**
- **Stiff systems of equations**
- **Systems with discontinuities**
- **General considerations**

Basic simulation references:

- S. M. Pizer, *Numerical Computing and Mathematical Analysis*, Science Research Associates Inc., Chicago, 1975.
- A. C. Hindmarsh, "Large Ordinary Differential Equation Systems and Software", *IEEE Control Systems Magazine*, December 1982.
- C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.

References on handling discontinuity:

- Taylor, J. H. and Kebede, D., "Modeling and Simulation of Hybrid Systems", *Proc. IEEE CDC*, New Orleans, LA, December 1995.
- Taylor, J. H. and Zhang, J., "Rigorous Hybrid Systems Simulation with Continuous-time Discontinuities and Discrete-time Components", *Proc. IEEE 15th Mediterranean Conference on Control and Automation (MED'07)* Athens, Greece, 27-29 June 2007.

Motivation

- Simulations often tell more about the behavior of a system than any other information (e.g., simulations vs eigenvalues).
- In many (most) cases there are no practical analysis methods available.
- **However:** if the differential equation can be solved analytically, it's less error-prone, and it yields direct access to parametric effects.
- You can interface simulations with hardware and/or human operators to determine unmodelable or poorly modelable aspects effecting the system behavior.
- Simulation is commonly used for:
  - Understanding plant behavior
  - Design trade-off studies
  - Design verification
  - Design optimization (but this can be tricky)

Use of modeling and simulation is growing as fast as:

$$\frac{\textbf{computer\_power}}{\text{cost}}$$

in many fields ...

Model Types

Some fundamental dynamic model categories:

- Ordinary Differential Equations (ODEs) – "Lumped-parameter Systems"
- Partial Differential Equations (PDEs) – "Distributed-parameter Systems"
- Differential / Algebraic Equations (DAEs)
- Subcategories of ODEs:
  - Continuous ("nice")
  - Stiff
  - Discontinuous
- Any of the above may be interfaced with Discrete-time Components (DTCs) to create a **Hybrid System Model**; typically modeling and simulation is only a little more complicated (but we're not going to cover that)

Model Types (Cont'd)

Model type is not crisp: some considerations:

- The model type is dictated in part by "the physics" and in part by what you want to study/observe
- PDEs can be converted into ODEs if you don't need ultimate fidelity and bandwidth (e.g., a flexible shaft $\rightarrow$ lumped inertias and springs)
- DAEs can be converted into ODEs (e.g., an algebraic loop may be eliminated by introducing high-frequency roll-off)
- . . . or a stiff ODE can be converted into a DAE
- Modeling is as much an art as a science, and requires good judgement (and often some trial-and-error and iteration)

Basic Concepts

- Hereafter we consider first-order vector ODEs; $\dot{x} = f(x,t)$ where $x =$ state vector, $t =$ time

- This form is often directly obtainable from a higher-order ODE (e.g., Newton $\rightarrow \ddot{\xi} = \phi(\xi, \dot{\xi}, t) \Rightarrow x^T = \left[\begin{array}{cc} \xi & \dot{\xi} \end{array} \right]$)

- This yields

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= \phi(x_1, x_2, t) \end{aligned}$$

- This includes $\dot{x} = f(x,u,t)$ once $u(t) =$ input is defined; in MATLAB simulation models $u(t)$ is defined within the model `model.m`

- The **complete problem**:

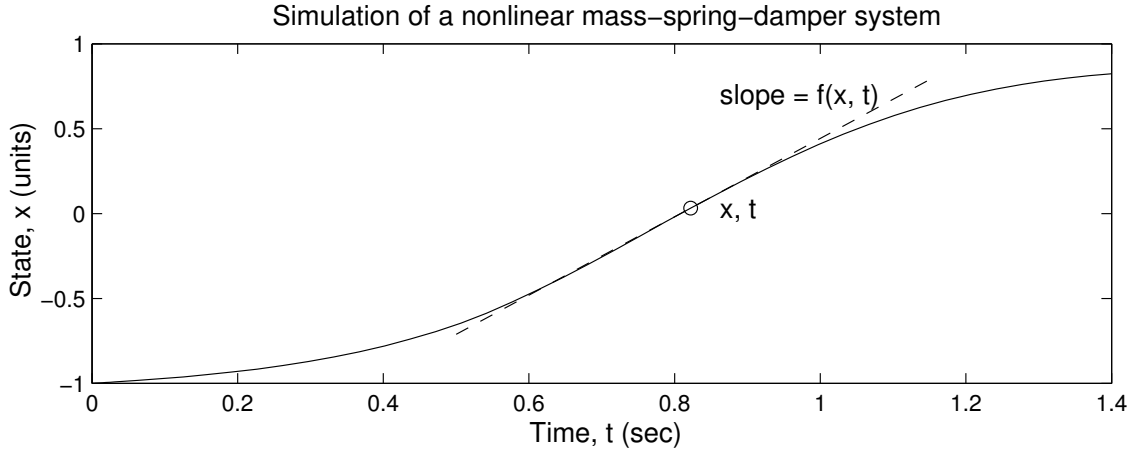
$$\dot{x} = f(x,t) \ , \tag{1}$$

$$x(t_0) = x_0 \tag{2}$$

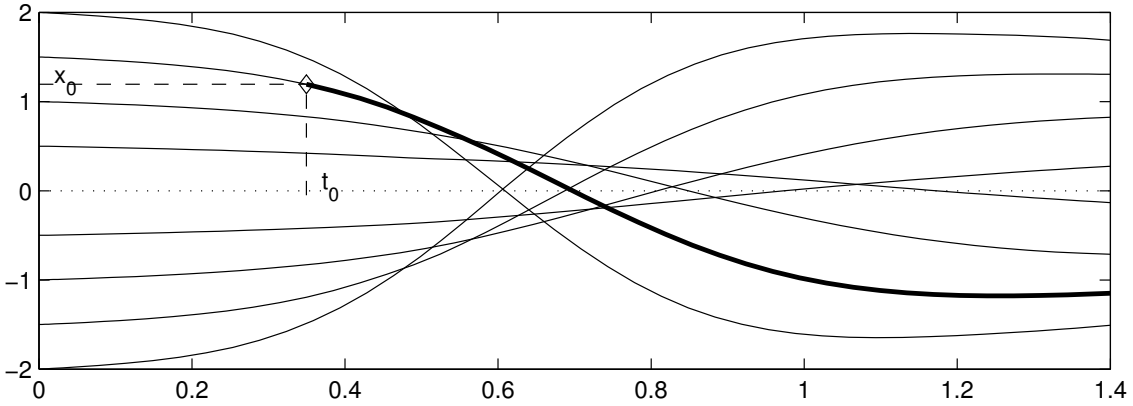
  defines an *initial value problem* to solve, usually over a finite time interval $t_0 \leq t \leq t_F$.

Conceptual framework

Equation (1) defines a **flow field** in the x, t plane – at each point, $\dot{x}$ defines where the solution curve is headed:



This flow field is the domain of an infinite number of solutions to the ODE; Equation (2) defines which solution curve is the one sought:

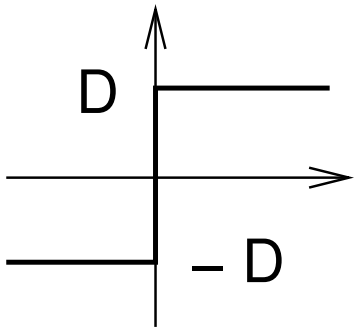


Note: Always remember that the model $f(x, t)$ is **never** globally valid – watch for simulations that pass out of its region of validity (envelope)!

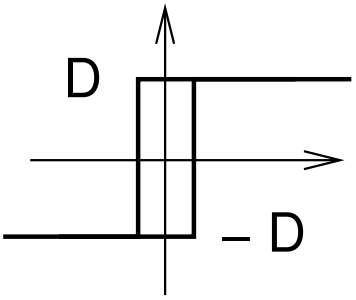
Mathematical well-posedness

Before you can say anything **rigorous** about solutions to Eqns. (1) and (2) you have to impose some conditions:

- differentiable right-hand sides - **not**



- single-valued right-hand sides - **not**



- Lipschitz conditions - e.g., $|f(x',t) - f(x'',t)| \leq L|x' - x''|$ for all x'' in some region of x'

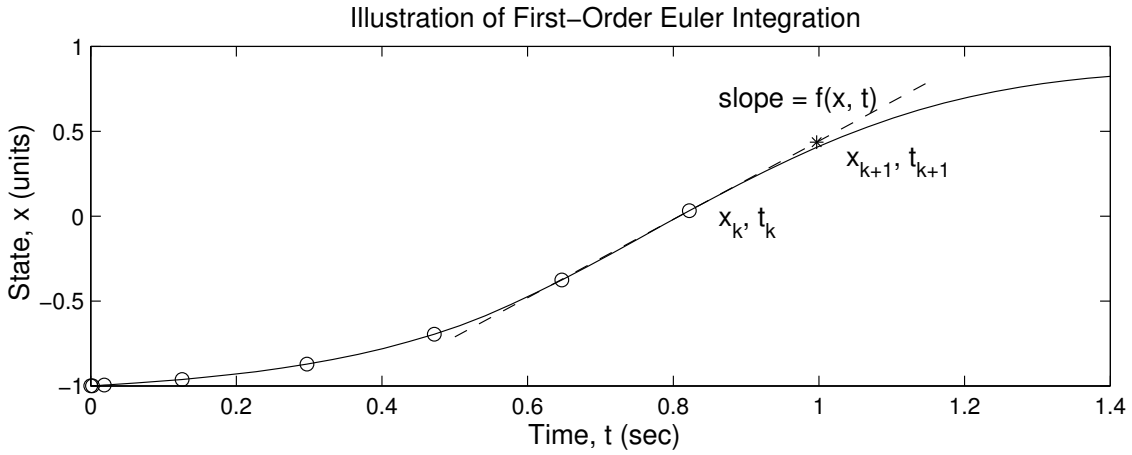
Unfortunately, most engineering models are not so "nice", and our approach must recognize this and take precautions!

The Essence of Numerical Integration: the Euler algorithm

Algorithm: given $x(t_k) \rightarrow$

$$x(t_k + h) = x(t_k) + h\dot{x}(t_k) \tag{3}$$

Geometrical interpretation:



Neglecting the terms $\frac{1}{2}h^2\ddot{x}(t_k)+\ldots$ leads to bf truncation error:

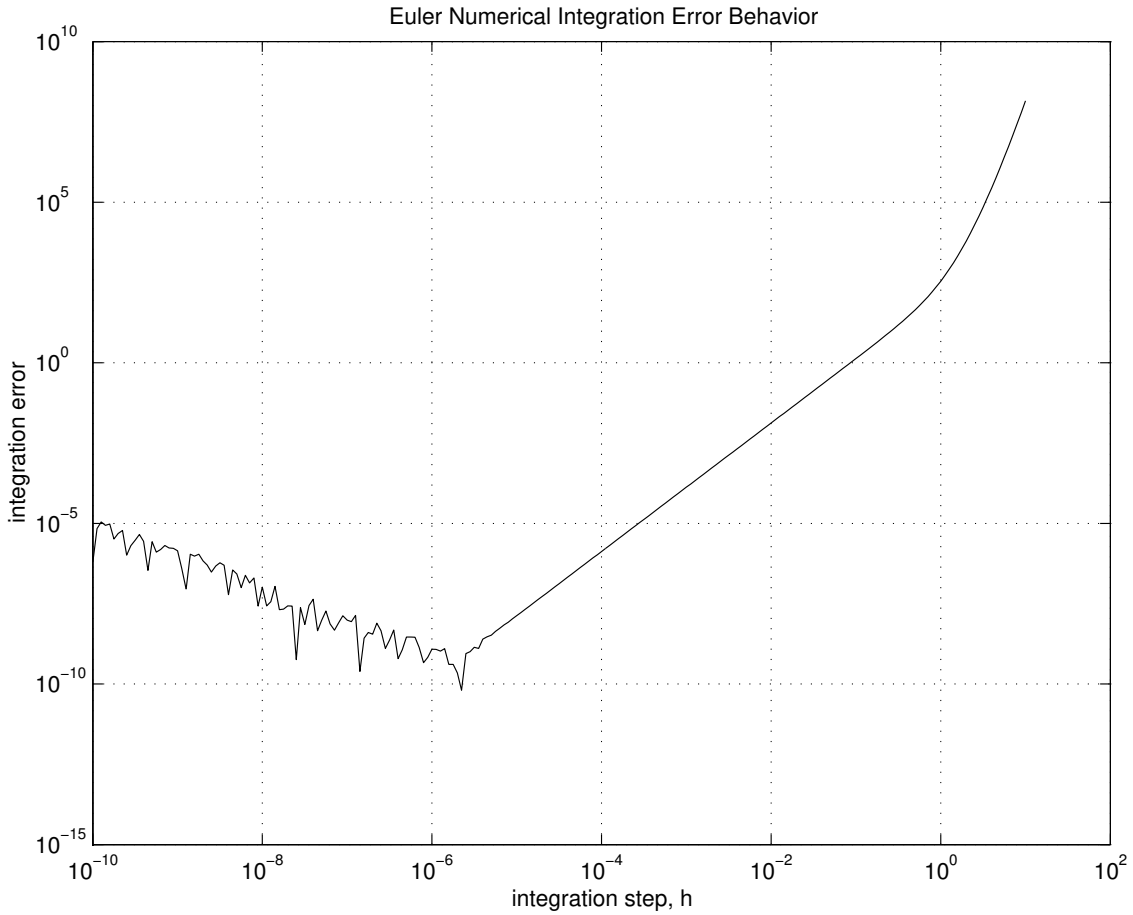
$$\epsilon_{trunc} \approx \frac{1}{2}h^2\ddot{x} = \mathcal{O}(h^2) \tag{4}$$

$\Rightarrow$ one must take small steps to keep ϵ_{trunc} small.

Error Analysis for the Euler Algorithm

However: In addition to truncation error we have round-off error : $x(t_k + h) \approx x(t_k) + h\dot{x}(t_k)$ – adding small increments to large numbers (as inevitably happens as h is made small) loses significant digits.

- **Note the step-size trade-off:**



- **Note the implicit differentiability assumption in analyzing truncation error.**

The Euler-Heun Predictor/Corrector – The "Trapezoidal Rule"

Algorithm: given $x(t_k)$

$$\begin{aligned} x_p(t_k + h) &= x(t_k) + h\dot{x}(t_k) \\ \dot{x}_p &= f(x_p, t_k + h) \\ x_c(t_k + h) &= x(t_k) + \frac{1}{2}h[\dot{x}(t_k) + \dot{x}_p] \end{aligned} \tag{5}$$

- **Geometrical interpretation: the area being added in taking a step is a trapezoid, not a rectangle – however, the vertex** $\dot{x}_p$ **is only approximate since** $x_p(t_k)$ **is an approximation**
- **The error is** $\mathcal{O}(h^3)$
- **This is not very effective because the predictor is first order, the corrector is second** $\Rightarrow$ **we cannot estimate or mop up errors (later).**

Modified Euler-Heun Predictor/Corrector

Algorithm: given $x(t_k)$ and $x(t_{k-1})$

$$\begin{aligned} x_p(t_k + h) &= x(t_{k-1}) + 2h\dot{x}(t_k) \\ \dot{x}_p &= f(x_p, t_k + h) \\ x_c(t_k + h) &= x(t_k) + \frac{1}{2}h(\dot{x}(t_k) + \dot{x}_p) \end{aligned} \tag{6}$$

- **The predictor is now "symmetric"**
- **Note: you don't have $x(t_{-1}) \Rightarrow$ you need to "start" this algorithm some other way.**

Modified Euler-Heun P/C – Error Estimation and Mop-up

$$\begin{aligned} \epsilon_p &= -\frac{1}{3}h^3 x^{(3)} + \mathcal{O}(h^4) \\ \epsilon_c &= \frac{1}{12}h^3 x^{(3)} + \mathcal{O}(h^4) \end{aligned}$$

Therefore, x_p and x_c "bracket" the true value; in fact,

$$x_c - x_p = (\frac{1}{12} + \frac{1}{3})h^3 x^{(3)} + \mathcal{O}(h^4) \approx 5\,\epsilon_c$$

So, we have a good error magnitude estimate:

$$|\epsilon_c| \approx \frac{1}{5}|x_c - x_p| \tag{7}$$

. . . and we can "mop up" the truncation error to obtain the final integration value $x_f(t_k + h)$:

$$x_f = \frac{1}{5}(4x_c + x_p) \tag{8}$$

which has error $\mathcal{O}(h^4)$ – a significant improvement.

Higher-order P/C Methods – Methods of Adams *et al*

- **General form: given $x_{k-m}, \ldots x_{k-1}, x_k$ and corresponding past derivatives,**

$$\begin{aligned} x_{p,k+1} &= \sum_{i=0}^{m} (a_i x_{k-i} + h b_i \dot{x}_{k-i}) \\ \dot{x}_{p,k+1} &= f(x_{p,k+1}, t_{k+1}) \\ x_{c,k+1} &= \sum_{i=-1}^{m} (c_i x_{k-i} + h d_i \dot{x}_{k-i}) \end{aligned} \tag{9}$$

- **Coefficients: $\rightarrow$ stability plus minimum truncation error (make (9) exact for $x = t^m$) – example: $m = 4$:**

$$\begin{aligned} x_{p,k+1} &= x_k + \frac{h}{24}(55\dot{x}_k - 59\dot{x}_{k-1} + 37\dot{x}_{k-2} - 9\dot{x}_{k-3}) \\ x_{c,k+1} &= x_k + \frac{h}{24}(9\dot{x}_{k+1} + 19\dot{x}_k - 5\dot{x}_{k-1} + \dot{x}_{k-2}) \end{aligned} \tag{10}$$

  (Adams-Bashford / Adams-Moultin)

- **Starting method: use a high-order Runge-Kutta method until enough points are available.**

- **Problem: with discrete-time subsystems, you have to re-start every sample time**

- **Serious problem: whenever $\dot{x}$ is discontinuous the previous derivatives are meaningless**

Runge-Kutta methods

Algorithm (fourth order): given x_k and $\dot{x}_k$

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

$$\rightarrow x_{k+1} = x_k + \frac{h}{6}(\dot{x}_k + 2\dot{x}^{(1)}_{k+\frac{1}{2}} + 2\dot{x}^{(2)}_{k+\frac{1}{2}} + \dot{x}_{k+1}) \qquad (11)$$

Geometrical interpretation: take tentative steps to "explore the flow field" ahead of the current accepted point $t_k\,,\ x_k$

Motivation:

- **Need for a method with error $\mathcal{O}(h^5)$ to start high-order P/C algorithms**
- **Makes more sense for systems with sampled-data components and/or non-differentiable right-hand sides.**

Variable Step-size Algorithms

Given an integration algorithm with error estimates, we have a mechanism for "optimizing" the step size.

Rough idea: assume an error ϵ_{step} is available at each step.

- **from a P/C method based on $x_{p,k+1}$ and $x_{c,k+1}$**
- **from a R-K method by looking at the differences among the exploratory steps used to obtain x_{k+1}**

Compare the error estimate with a tolerance on maximum acceptable error; adjust h accordingly (e.g., h may be halved or doubled if the error estimate is too large or well below the tolerance). For a P/C algorithm this is somewhat complicated:

- **doubling requires saving more past values;**
- **halving requires interpolation;**

for R-K methods this is less of a burden.

This is handled invisibly by a good variable step-size algorithm. Modern variable step-size algorithms are quite sophisticated (for example, the integration step may be adjusted more subtly)

Selection of Algorithm Class

Summary of basic considerations:

- **Predictor / Corrector methods have a firm, classical numerical basis, including error analysis and techniques such as "mopping up" error – if your problems/models satisfy the required conditions of continuity, then good P/C algorithms will do an excellent job**

- **However, engineering problem models are so often "not nice" that Runge-Kutta methods have become dominant; they have also been developed to the point that they are nearly as effective as the P/C methods for "nice" models**

- **Predictor / Corrector methods must not be used if your system is comprised of a mixture of continuous- and discrete-time components – each time the discrete-time component(s) are executed the past derivatives are meaningless**

Simulating in MATLAB

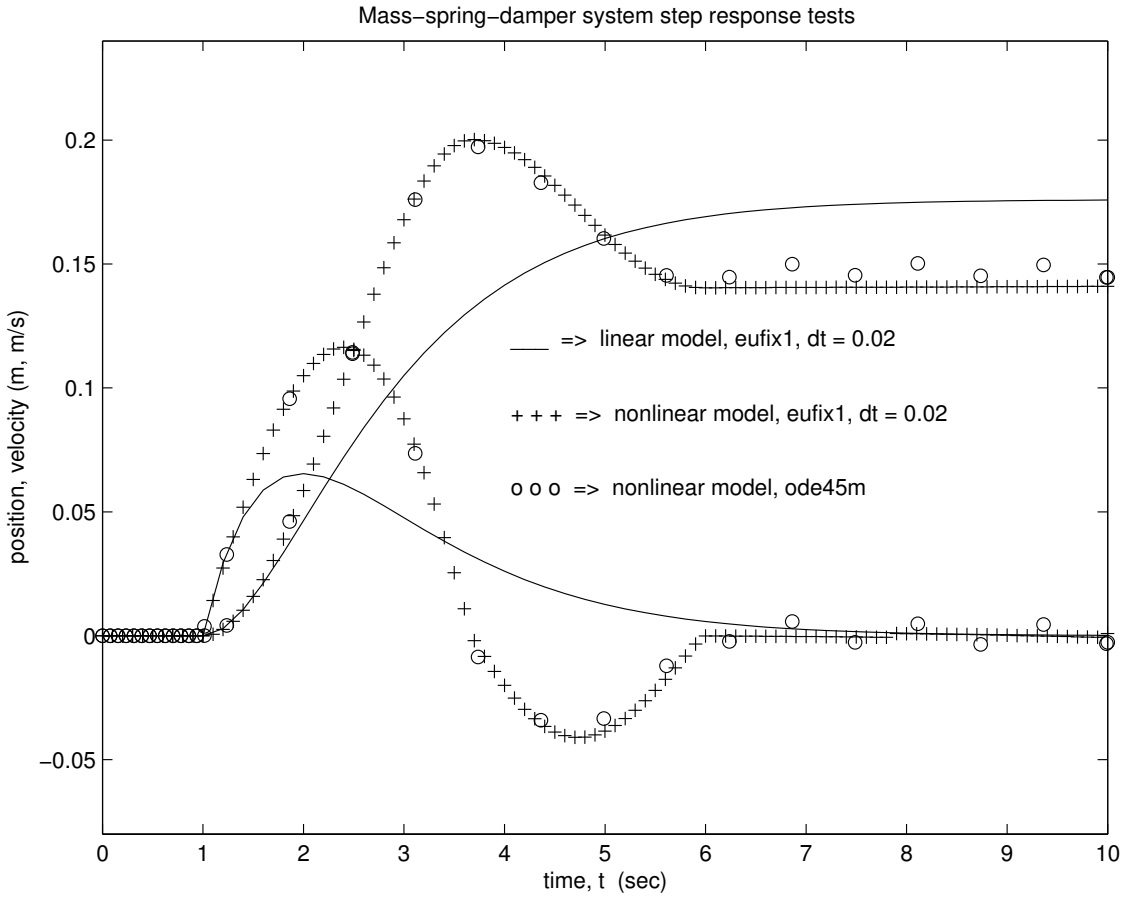
You need to create a model and a script:

```
function xdot = nlmsd(t,x)
% nonlinear model of mass-spring-damper system
%
M = 5;     %  kg
Bv = 1;   %  N-sec/m
Bc = 0.2;  %  N
K1 = 5;    %  N/m
K3 = 50;   %  N/m^3
if t < 1   % implement a step function input
   F = 0;
else
   F = 0.88;
end
xdot(1) = x(2);
xdot(2) = ( F - K1*x(1) - K3*x(1)^3 - Bv*x(2) - Bc*sign(x(2)) )/M;
xdot = xdot(:);  % make xdot a column vector (for matlab 5.n)
```

```
% matlab script to run step-response tests for linear and nonlinear mo
%
clear; clf
sym = [ '-', '+', '*', 'x' ]; dt = 0.02;
[t1,x1] = eufix1('msd',0, 10, [ 0 ; 0 ], dt);
[t2,x2] = eufix1('nlmsd',0, 10, [ 0 ; 0 ], dt);
plot(t1,x1,sym(1))
hold on
plot(t2,x2,sym(2))
axis([0 10 -.08 .24]);
title('Mass-spring-damper system step response tests')
xlabel('time, t  (sec)')
ylabel('position, velocity (m, m/s)')
text(4,.12,[' \_\_\_  =>  linear model, eufix1, dt = ',num2str(dt)])
text(4,.09,[' +++  =>  nonlinear model, eufix1, dt = ',num2str(dt)])
[t3,x3] = ode45('nlmsd',0, 10, [ 0 ; 0 ], .1);
```

```
plot(t3,x3,sym(4))
text(4,.06,' xxx  =>  nonlinear model, ode45m')
```

Here's what you will see:



A few things to note:

- **The ode45 uses far fewer points; it's a higher order algorithm**
- **... but it "chatters" when the motion "sticks"**

"Stiff Systems"

- **Informal definition: a model is "stiff" if it combines very fast and very slow dynamics.**
- **First question: are the very fast dynamics needed?**
- **Problem: the previous methods won't converge if the system is too stiff (e.g., an integration step cannot be found that is satisfactory for all states).**
- **Solution # 1: modify the algorithm to improve convergence – Gear's algorithm and variants (see Hindmarsh, Gear).**
- **Solution # 2: convert the fast dynamics into algebraic constraints by assuming they are "instantaneous" – in other words, the fast states are at "steady state" over most of the integration interval, so all you need to do is find their equilibrium condition at each long integration step for the slow dynamics. This produces a DAE set. For example:**

$$\begin{aligned} x &= [\, x_{fast}\ x_{slow}\,]^T \\ \dot{x}_{fast} &= f_{fast}(x_{fast}, x_{slow}, t) \\ \dot{x}_{slow} &= f_{slow}(x_{fast}, x_{slow}, t) \end{aligned} \tag{12}$$

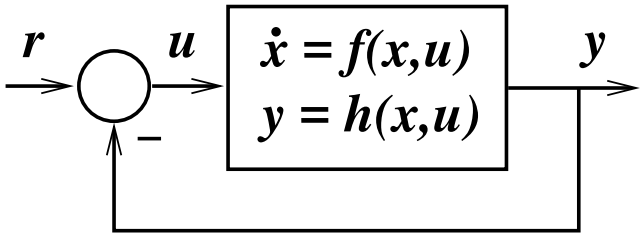
yields the DAE set

$$\begin{aligned} \dot{x}_{slow} &= f_{slow}(x_{fast}, x_{slow}, t) \\ 0 &= f_{fast}(x_{fast}, x_{slow}, t) \end{aligned} \tag{13}$$

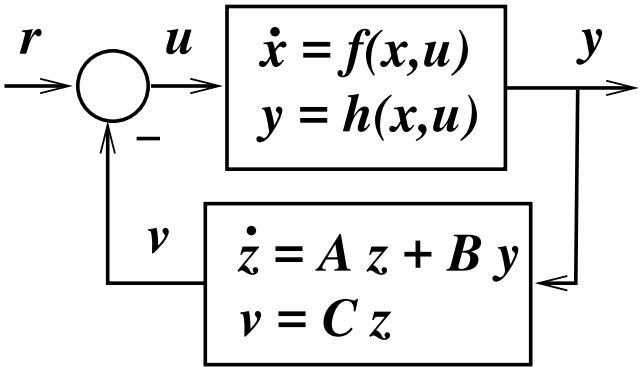
In other words, you are assuming that the fast dynamics are "infinitely fast".

Systems with Algebraic Loops

control systems:



(a) System with algebraic loop



(b) Loop broken with finite–bandwidth dynamics

In case (a) we have a differential/algebraic system of equations, namely $\dot{x} = f(x, u)$ subject to the constraint $0 = r(t) - u - h(x, u)$. No reputable differential equation solver will accept such a modelcannot be sorted for proper execution.

Systems with Algebraic Loops (Cont'd)

The problem is easiest to understand in the linear case: Given

$$\dot{x}(t) = Ax + Bu \tag{14}$$
$$y(t) = Cx + Du \tag{15}$$
$$u(t) + r(t) - y(t) \tag{16}$$

where $r(t)$ is the input to the control loop. Here is the "loop": You must have u to evaluate y but you need y to evaluate u .... The loop is broken if there is no D matrix, which means that the open loop transfer function is strictly proper (see slide 24 in Module 2). All *practical* models must be strictly proper, otherwise they would have infinite bandwidth.. In case (b) the finite-bandwidth dynamics added to the model (perhaps modeling sensor dynamics) eliminates the loop and produces an ODE: $\frac{K_{sensor}}{1+s\tau}$ will suffice to break the loop.

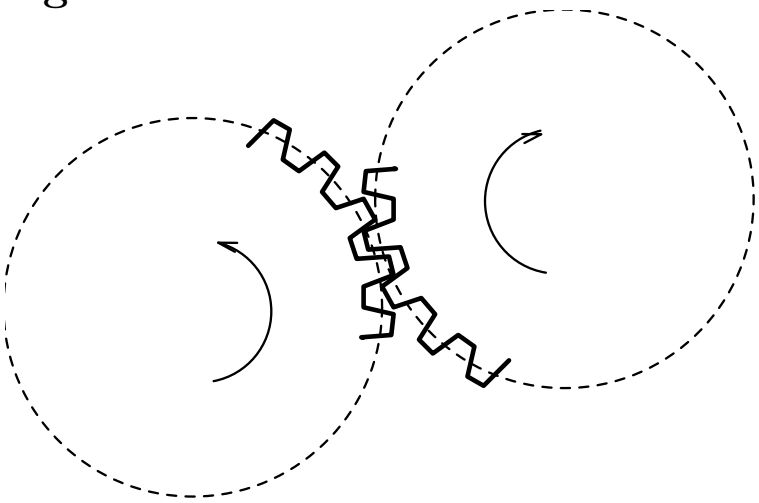
Systems with Discontinuities

- **Nonlinear systems with discontinuous ODEs (or worse yet, those with multi-valued nonlinearities) are very difficult:**
  - **nonphysical things may happen, e.g., gears engage with "overlap", relays switch at incorrect times**
  - **a numerical integrator may even become confused and miss switching events**
- **Variable step-size algorithms may reduce the first problem, but at the expense of longer simulation time ($\rightarrow$ "creeping solutions")**
- **The correct solution is to include an "state-event handler" in the simulation environment**
- **Mr. Dawit Kebede (Masters student) worked with me to develop this advanced approach in MATLAB (see references); this software is available on my web site**

Systems with Discontinuities (Cont'd)

Physical and practical motivation:

- **Modeling & simulation of physical objects contacting is difficult . . .**



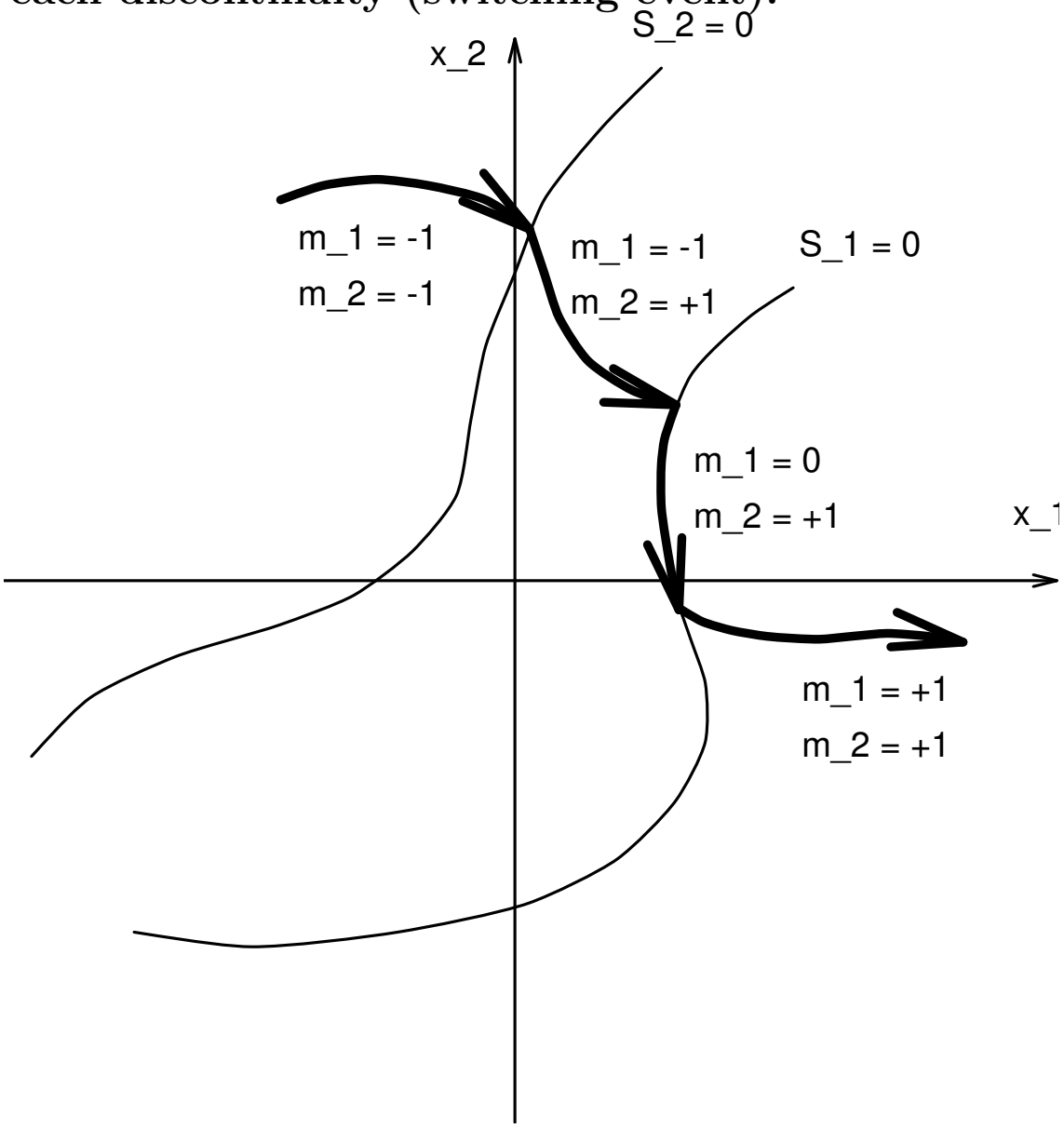
- **Modeling & simulation of friction is *very* difficult . . .**

- **Prediction of limit cycles, chaos, deadlocks, . . . may be highly questionable . . .**

Rigorous simulation is important for process understanding, prototyping, system design validation, . . .

Systems with Discontinuities (Cont'd)

In general we have the "mode" of the model changing at each discontinuity (switching event):



A switching event may be a simple boundary crossing, as shown for mode m_2, or involve motion along a boundary for some time (m_1).

Systems with Discontinuities (Cont'd)

- **System model**

$$\begin{aligned} \dot{x} &= f(x, u, m, t) \\ y &= h(x, u, m, t) \end{aligned} \tag{17}$$

  x **= state,** u **= input,** m **= mode and** t **= time**

- **State events are characterized by *zero-crossings*,**

$$S(x, m, t) = 0 \tag{18}$$

- **. . . and may require instantaneous state reset,**

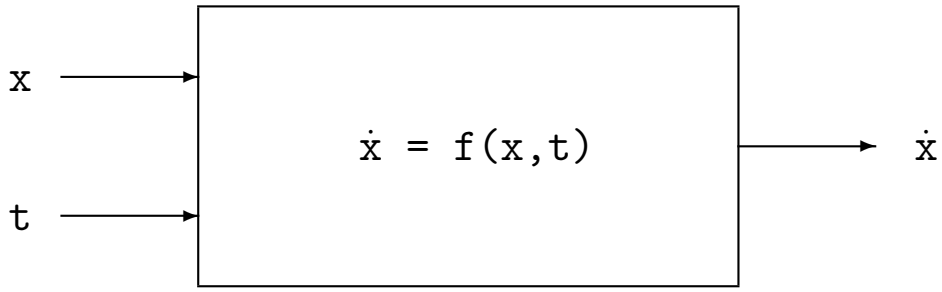
$$x^+ = x(t_e^+) = r(x(t_e^-), m, t_e^-) \tag{19}$$

Systems with Discontinuities (Cont'd)

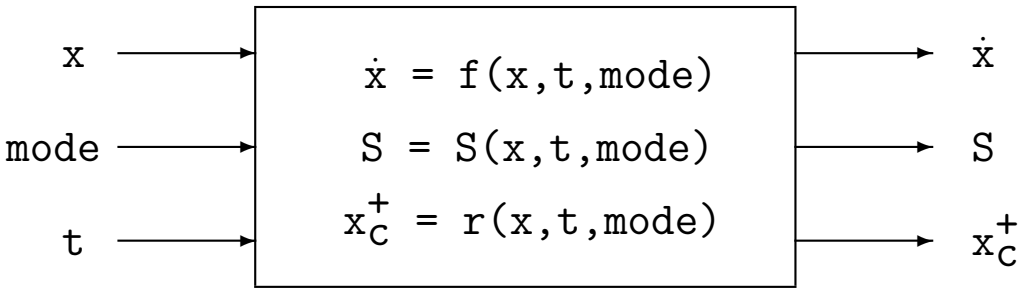
To handle discontinuities with good generality:

- **The model has to be told what "`mode`" it is in at the present time (e.g., engaged/disengaged)**
- **The model has to indicate when the discontinuity occurs → switching (zero-crossing) function `S`**
- **The model has to take care of state reset x_c^+ (if needed, e.g., to preserve momentum)**

The following model scheme takes care of all this:

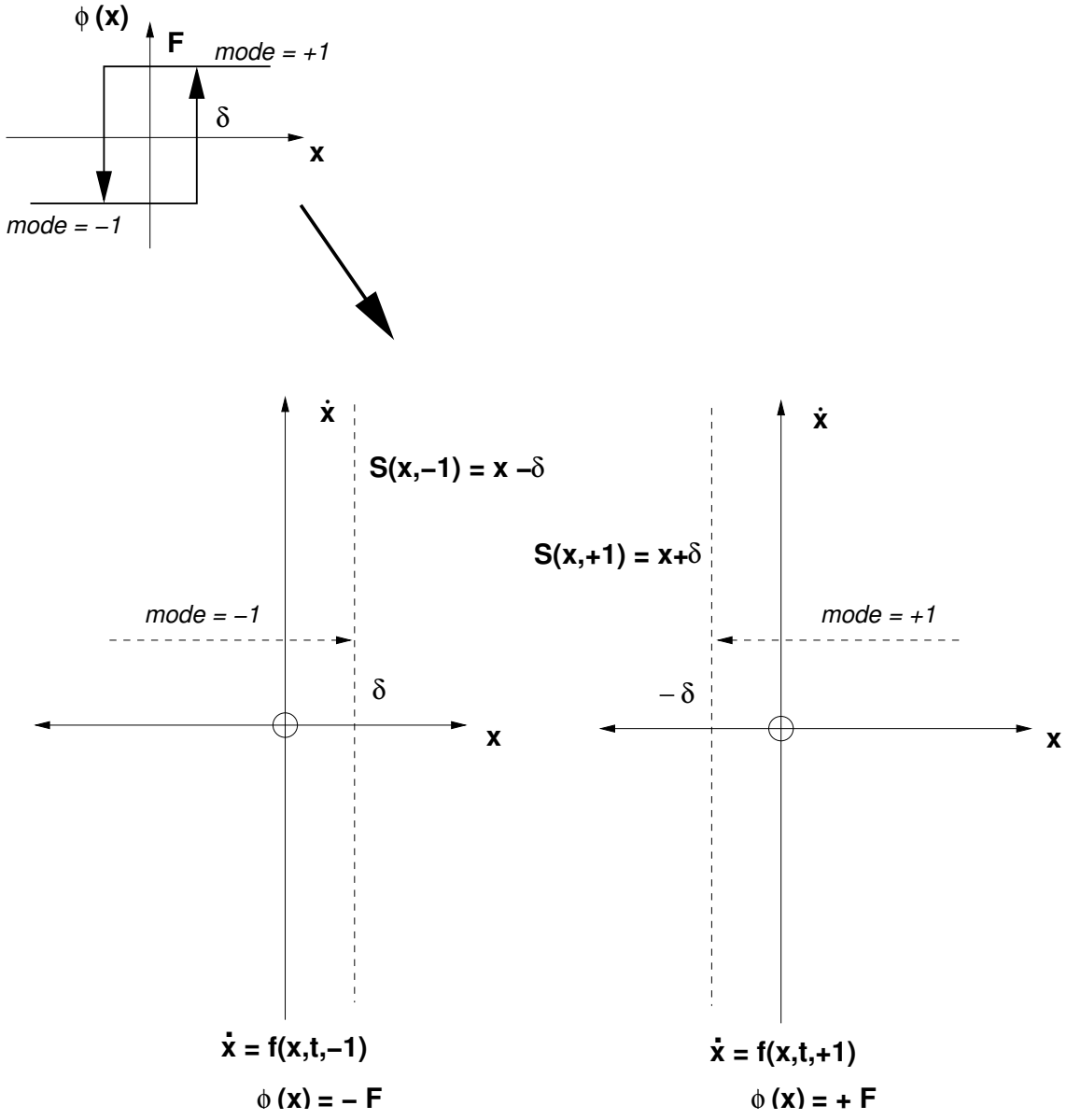


(a) Standard MATLAB model schema



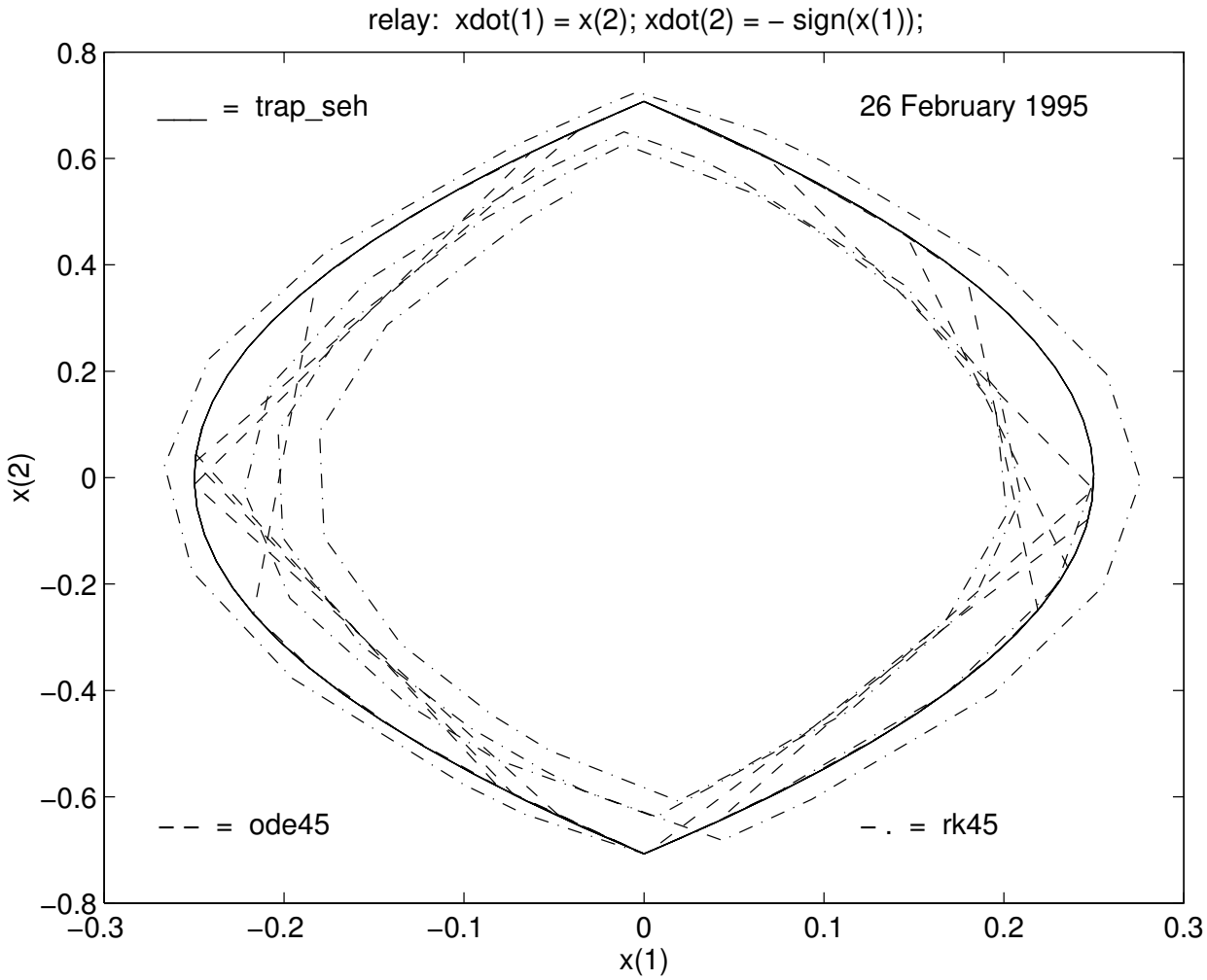
(b) Extended MATLAB model schema

Modes for Systems with Discontinuities



The `mode` "remembers" the state of the relay.

Systems with Discontinuities (Cont'd)



`ode45` is an excellent MATLAB integrator, `rk45` was embedded in SimuLink, and `trap_seh` was the first state-event handler (seh) we created using a trapezoidal rule integrator. The MathWorks replaced `rk45` after we published this result, and no longer allows using the SimuLink algorithm from a command file.

First Test Case – Updated

We will look at the first test case more closely:

- **First, a basic MATLAB model:**

```
function xdot = relay(t,x)
xdot(1) = x(2);
xdot(2) = -sign(x(1));
xdot = xdot(:);
```

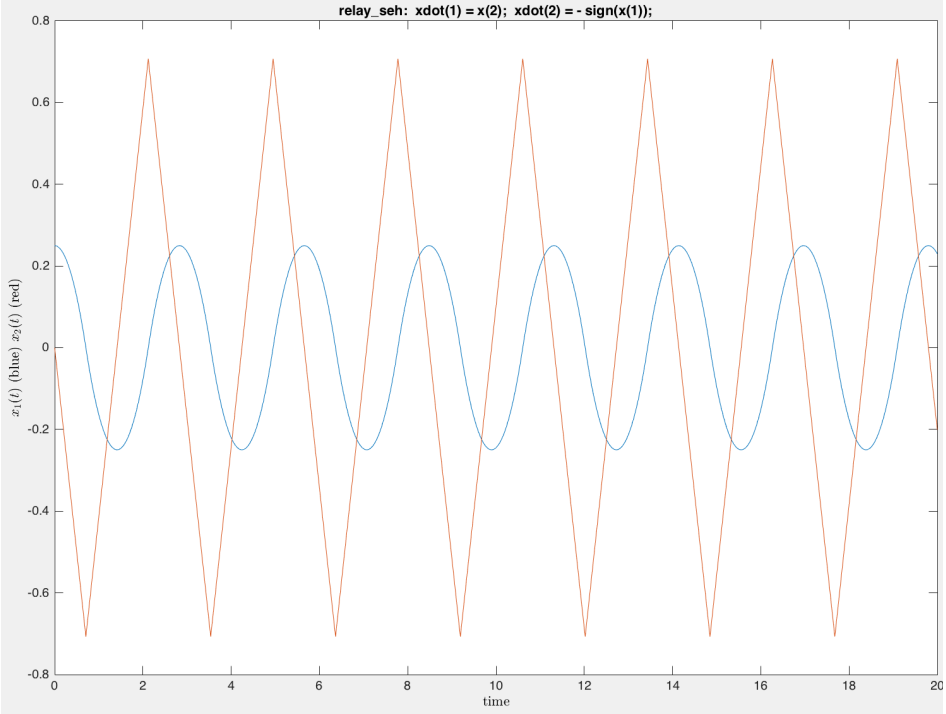
- **Next, a model implementing state-event handling:**

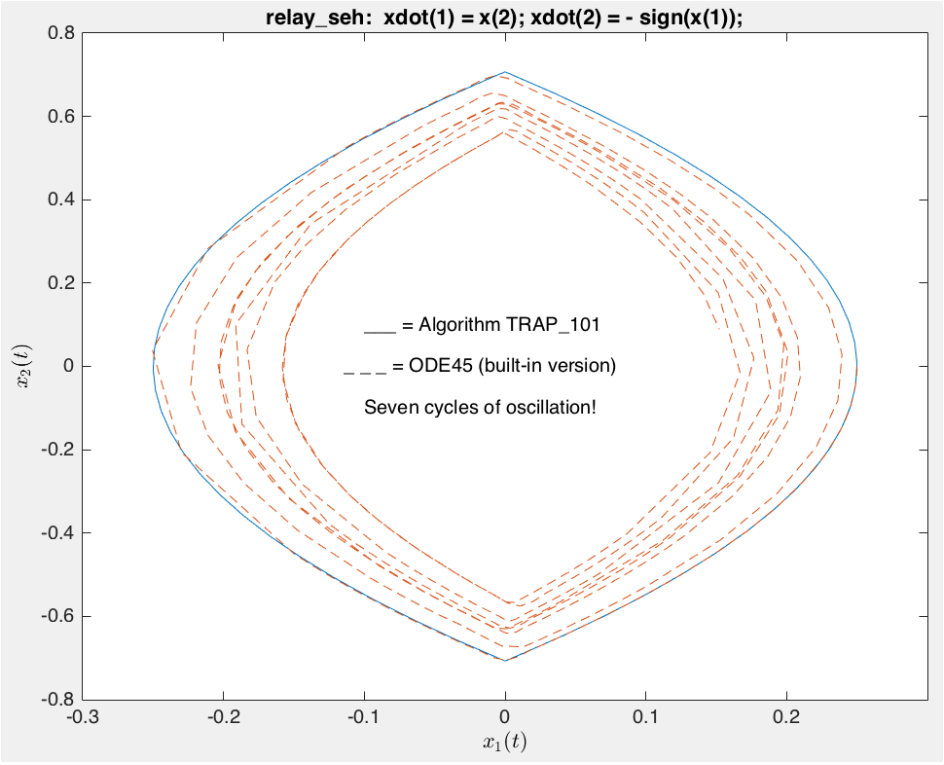
```
function [xdot,S,reset] = relay(t,x,mode)
% model of the relay switching system, d\^2x/dt\^2 = - sign(x)
% switches when S = x(1) passes through zero; and that triggers
% the state event (causes the mode to switch mode from
% -1 to 1 or vice versa).
%
if isempty(mode), % initialization section
   % Set S to be consistent with the IC, so that mode
   % will be initialized correctly -- e.g., if x(1) = 0,
   % then x(2) governs mode, i.e. x(2) > 0 => mode = +1 etc.
   if x(1) == 0, S = x(2);
   else S = x(1);
   end
   xdot = []; reset = []; % to prevent new warning msgs!
   return
end
% Define the mode-dependent model and switching flag:
xdot(1) = x(2);
xdot(2) = -mode;
S = x(1);
reset = []; % to prevent new warning msgs!
```

State-event handling software is here: http://www.ece.unb.ca/jtaylor/HS_software.html – there are two integration routines, several examples, and a user's guide

First Test Case – Updated (Cont'd)

Here are the simulation results:

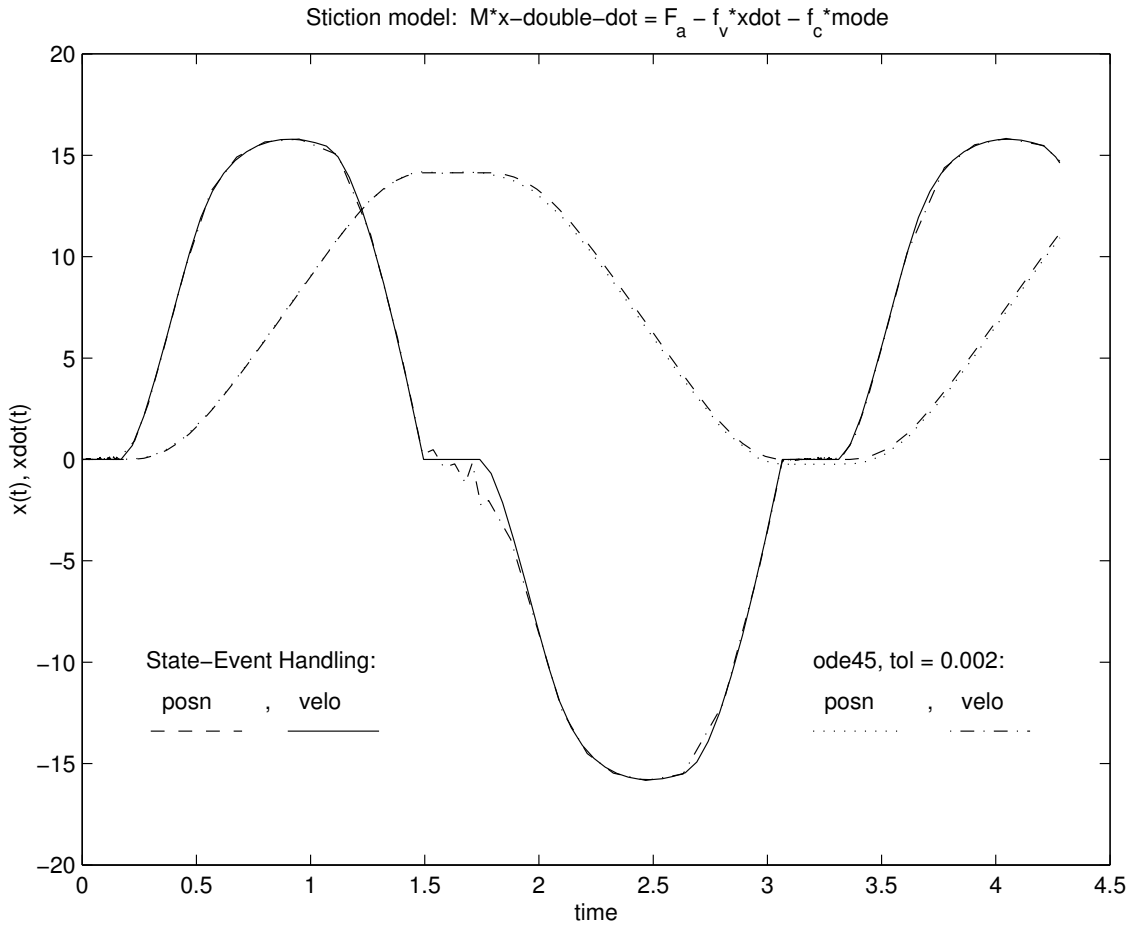




Notice that the behavior of `ode45` is different now!

Systems with Discontinuities (Cont'd)

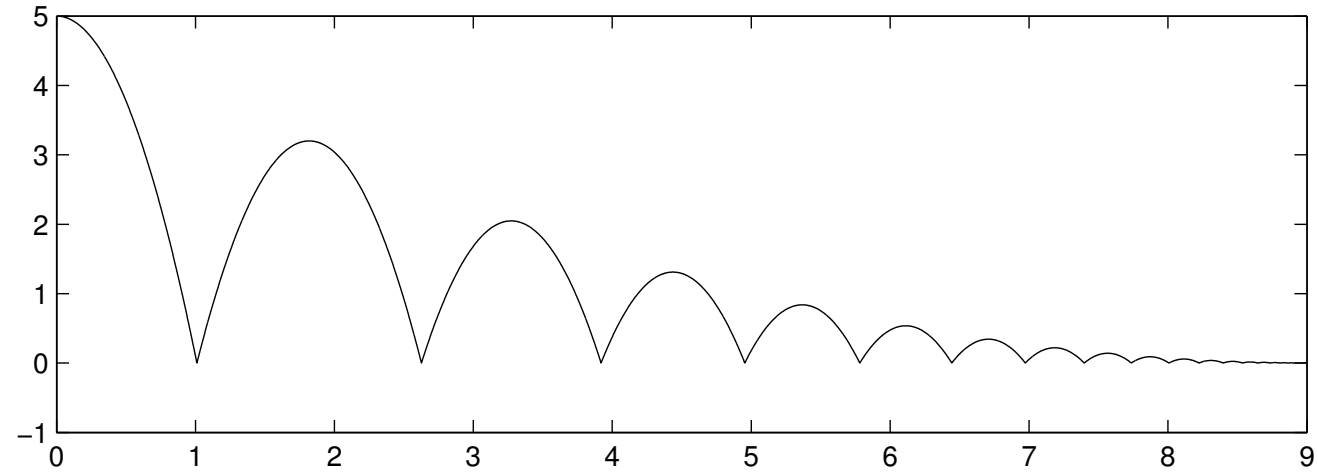
Example 2: Mass with viscous and Coulomb friction:

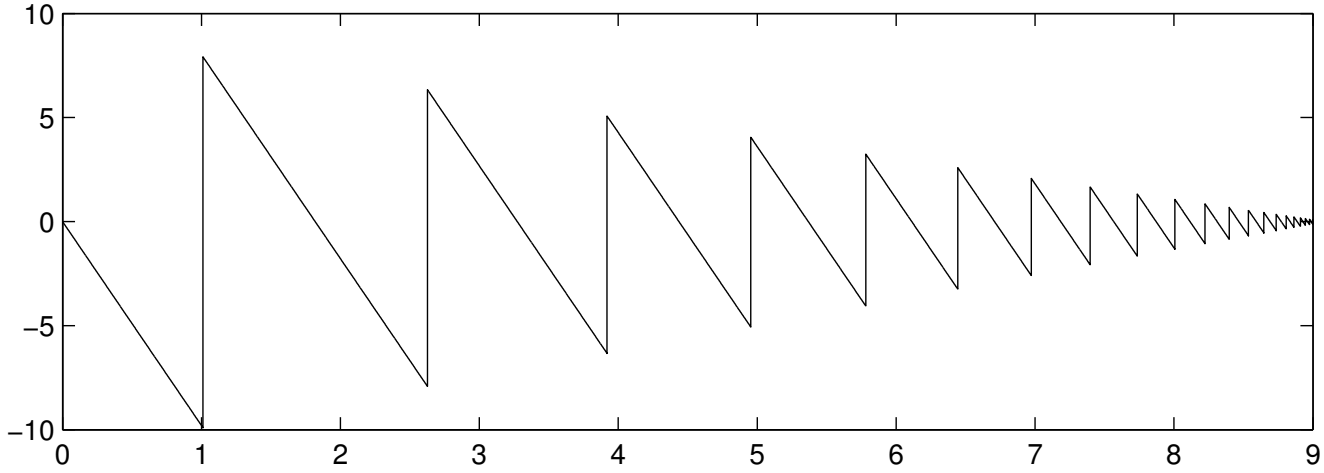


Notice the "chattering" on the `ode45` solution as it tries to deal with the stiction effect; it is very visible due to the large tolerence set for `ode45`, `tol = 0.002` vs the default 10^{-6}; this was done to make the simulation time about the same as for the `eufix_seh` algorithm.

Systems with Discontinuities (Cont'd)

Example 3: Bouncing ball:





When the ball hits the floor we reset the velocity from v^- to $v^+ = -0.8 * v^-$. You cannot perform state resets in MATLAB. We can now simulate systems with discontinuities with complete rigor using state-event handling

Conclusions / General Considerations

- **Class of problem ⇒ algorithm selection:**
  - **multi-valued nonlinearities (e.g., hysteresis, backlash) require a state-event handler (Euler *might* work).**
  - **non-differentiable nonlinearities require Euler or Runge-Kutta (ode45) at least (a predictor/ corrector which uses past derivatives doesn't make sense); a state-event handler may be highly desirable.**
  - **variable step-size algorithms are almost always preferable unless part of the system is discrete-time.**
  - **a state-event handler is highly desirable for discontinuous systems**
- **For preliminary explorations, the fourth-order Runge-Kutta with variable step size is highly recommended.**
- **Never take the first results on faith!**