Modeling & Simulation of Nonlinear Dynamic Systems

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

22 July 2002

JUV RESCCE'02 Summer School
Danang, Viet Nam
05-09 August 2002

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:

- J. H. Taylor, "A Modeling Language for Hybrid Systems", *Proc. Joint Symposium of CACSD,* Tucson, AZ, March 1994.
- Taylor, J. H. and Kebede, D., "Modeling and Simulation of Hybrid Systems", *Proc. IEEE CDC*, New Orleans, LA, December 1995.

Warning: The literature is not standardized with respect to algorithm names **or** the definition of stability!

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:**
  - **Design trade-off studies**
  - **Design verification**
  - **Design optimization (but this can be tricky)**

Use of modeling and simulation is growing as fast as:

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

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 Algorithms (DTAs); typically modeling and simulation is no more complicated**

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 → 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** $\rightarrow \ddot{\xi} = \phi(\xi, \dot{\xi}, t) \Rightarrow x^T = \begin{bmatrix} \xi & \dot{\xi} \end{bmatrix}$**)**

- **This includes** $\dot{x} = f(x,u,t)$ **once** $u(t)$ **= input is defined**

- **The complete problem:**

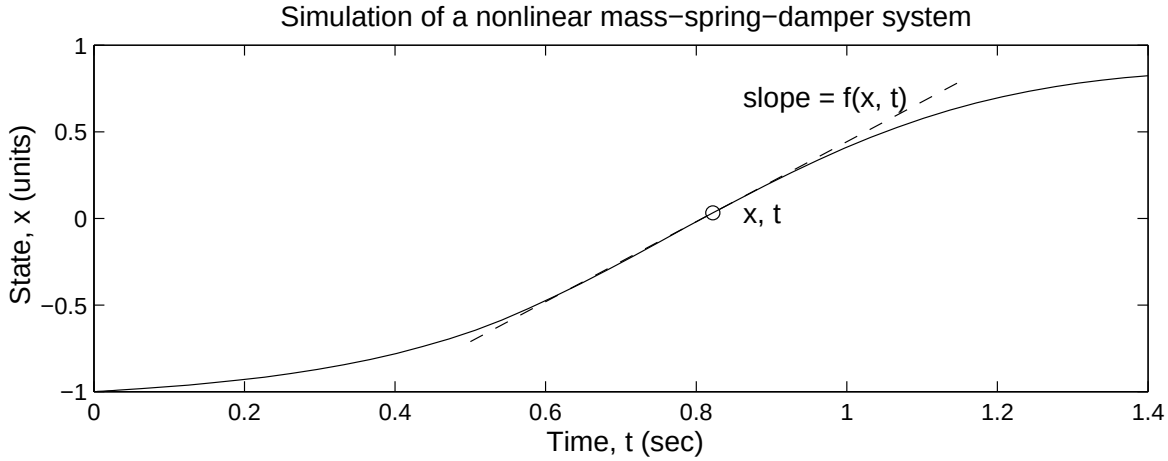
$$\dot{x} = f(x,t) \ , \tag{1}$$

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

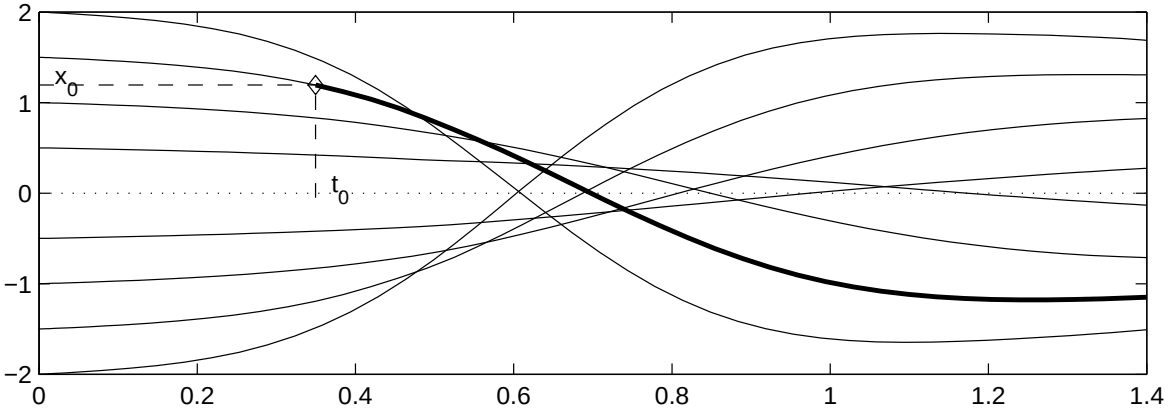
defines an *initial value problem* **to solve, usually over a finite time interval** $t_0 \leq t \leq t_F$**.**

Conceptual framework

Equation (1) defines a <u>flow field</u> 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:

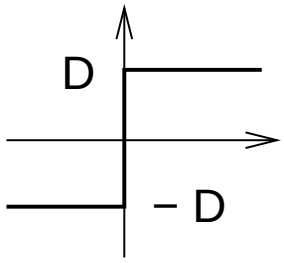


<u>Note:</u> Always remember that the model $f(x, t)$ is generally not globally valid – watch for simulations that pass out of its region of validity!

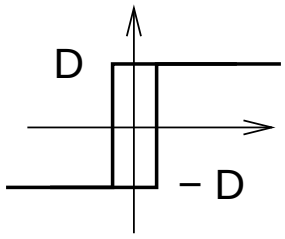
Mathematical well-posedness

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'**

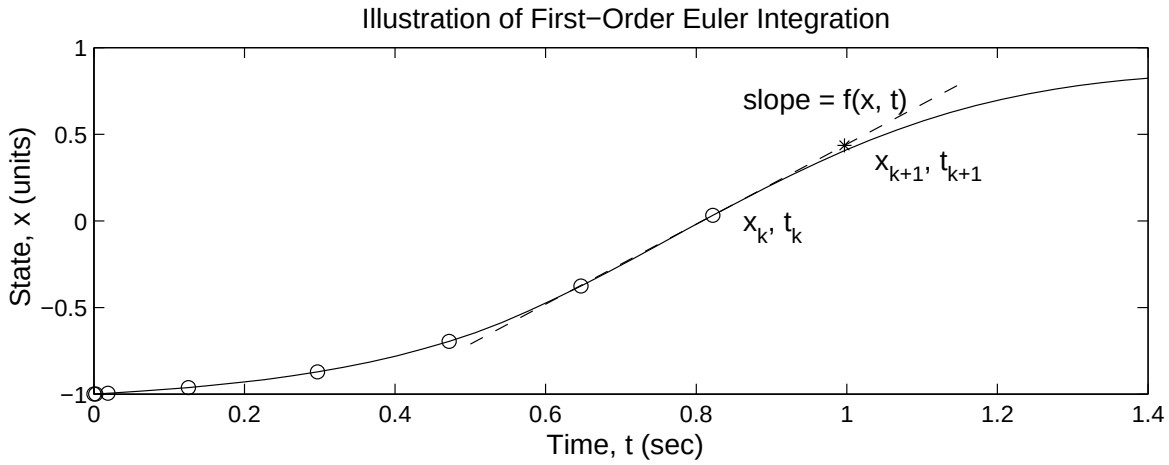
Unfortunately, many engineering models are not so nice, and our approach must recognize this and take precautions!

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 <u>truncation error:</u>**

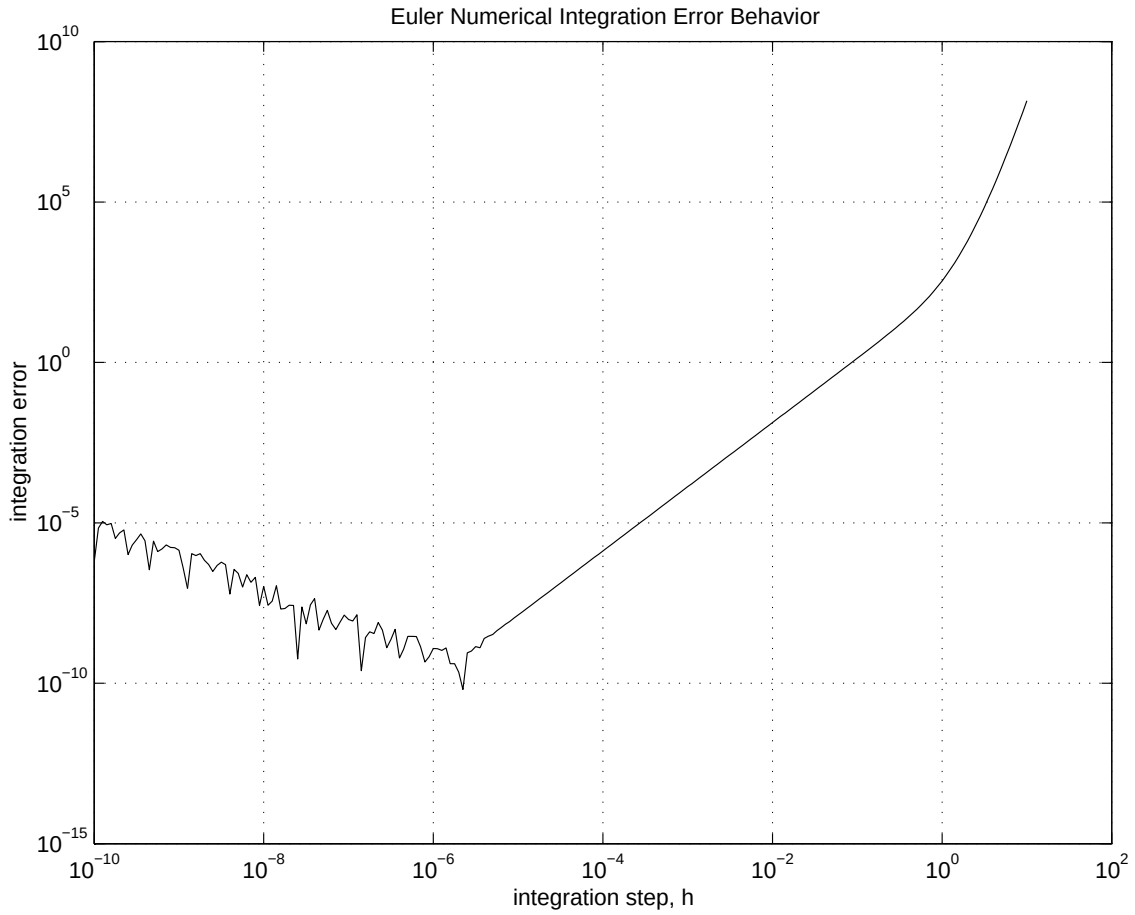
$$\epsilon_{trunc} \approx \frac{1}{2}h^2\ddot{x} = \mathcal{O}(h^2) \tag{4}$$

$\Rightarrow$ **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, strictly speaking, not the trapezoidal rule because** x_p **is not the true value of the integrand.**
- **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.**
- **stability:**
  - **The predictor alone is seriously unstable**
  - **The combined P/C is relatively stable for $-0.601 < hJ_i < 0$ and $0.701 < hJ_i$; the worst case is "only mildly unstable . . . common property . . . " (Pizer)**

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

Truncation error (usually dominates):

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

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,**

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

- **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**

- **<u>Serious</u> 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" <u>ahead of</u> 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**

"Stiff Systems"

- **Informal definition: a model is "stiff" if it combines very fast and very slow dynamics.**
- **First question: are the very fast dynamics <u>needed</u>?**
- **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:**

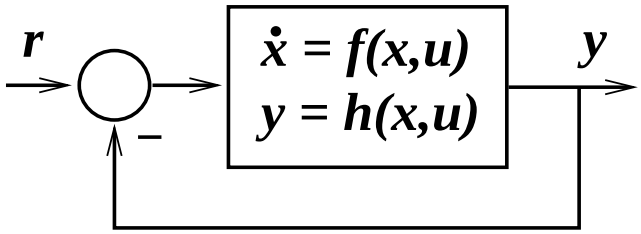
$$\begin{aligned} x &= [\, x_{fast}\ x_{slow} \,]^T \\ \dot{x}_{fast} &= f_{fast}(x_{fast}, x_{slow}, t) \\ \dot{x}_{slow} &= f_{slow}(x_{fast}, x_{slow}, t) \end{aligned} \tag{12}$$

yields the DAE set

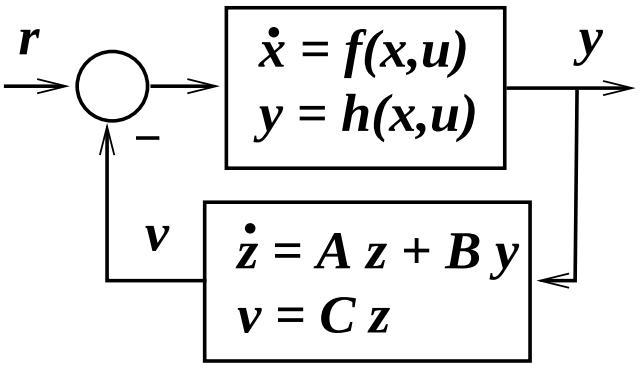
$$\begin{aligned} \dot{x}_{slow} &= f_{slow}(x_{fast}, x_{slow}, t) \\ 0 &= f_{fast}(x_{fast}, x_{slow}, t) \end{aligned} \tag{13}$$

Systems with Algebraic Loops

Differential/algebraic systems of equations occur naturally in controls:



(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** $0 = r(t) - u - h(x, u)$

In case (b) the finite-bandwidth dynamics added to the model (perhaps modeling sensor dynamics) eliminates the loop and produces an ODE

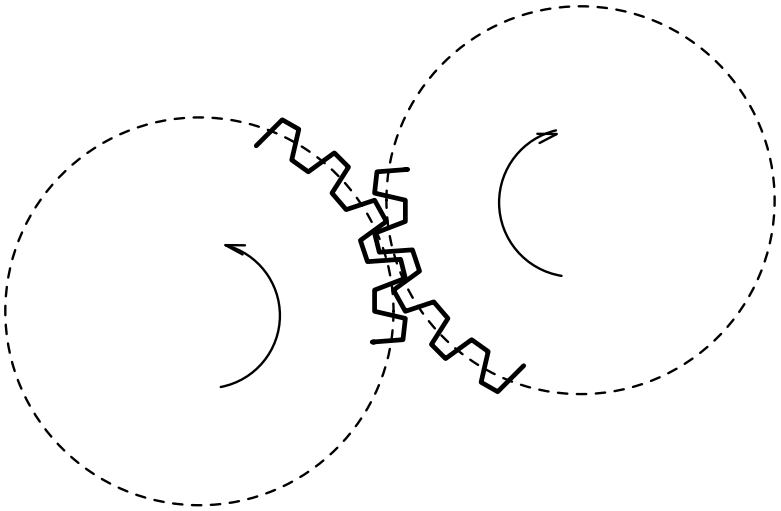
Systems with Discontinuities

- **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 (→ "creeping solutions")**
- **The correct solution is to include an "state-event handler" in the simulation environment**
- **ACSL has a rudimentary state-event handler; we developed a more advanced approach in MATLAB (Taylor & Kebede, see references; this software is available on my web site)**

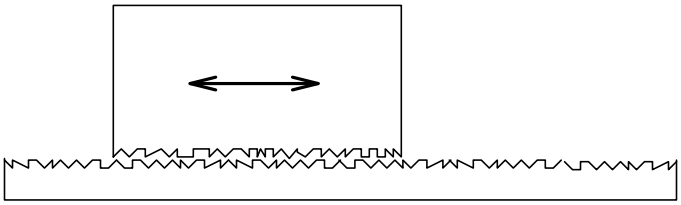
Systems with Discontinuities (Cont'd)

Physical and practical motivation:

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



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

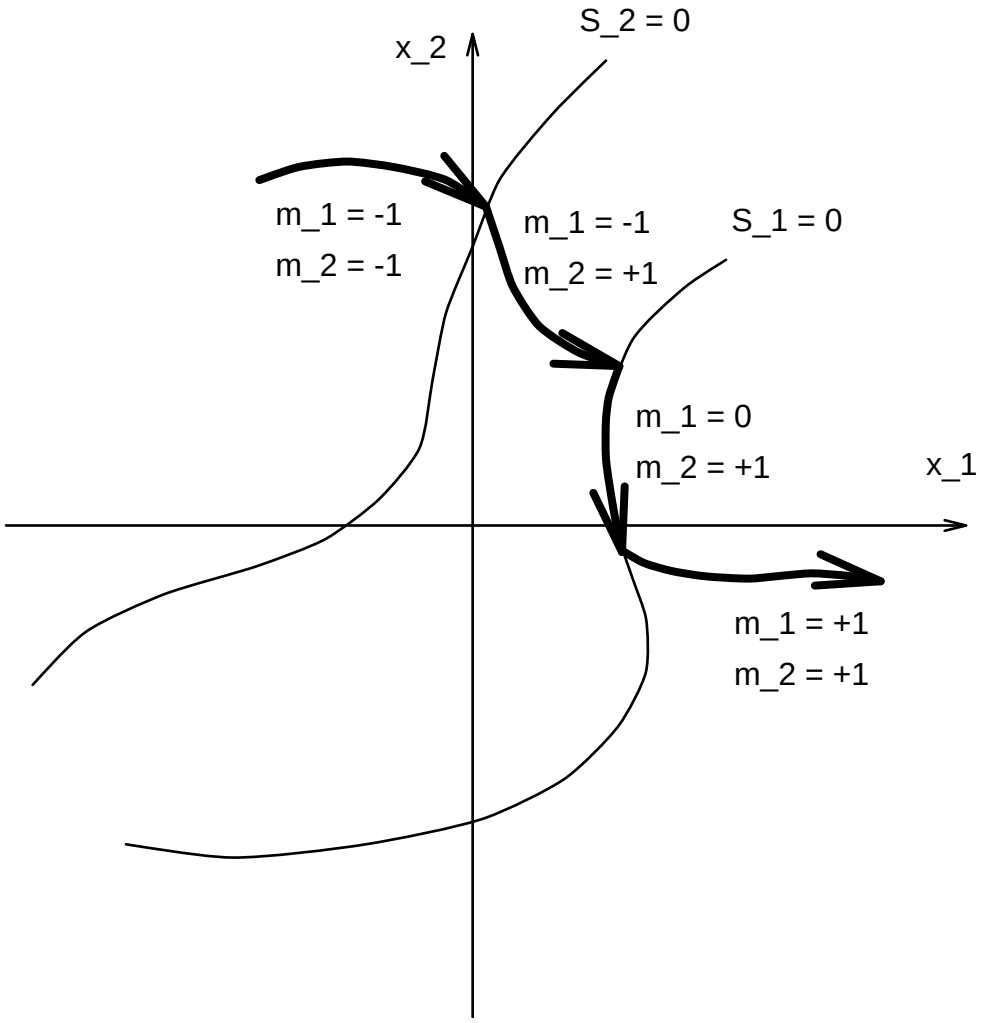


- **Prediction of limit cycles, chaos, deadlocks, . . . may be highly questionable . . .**

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):



Systems with Discontinuities (Cont'd)

What do we need for a complete characterization?

- **System model**

$$\begin{aligned} \dot{x} &= f(x, u, m, t) \\ y &= h(x, u, m, t) \end{aligned} \tag{14}$$

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

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

$$S(x, m, t) = 0 \tag{15}$$

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

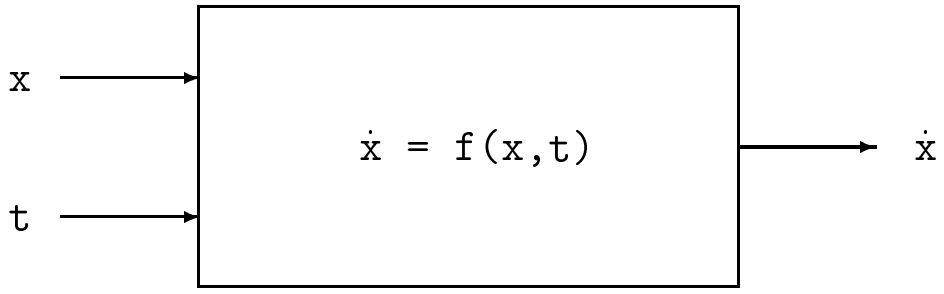
$$x^+ = x(t_e^+) = r(x(t_e^-), m, t_e^-) \tag{16}$$

Systems with Discontinuities (Cont'd)

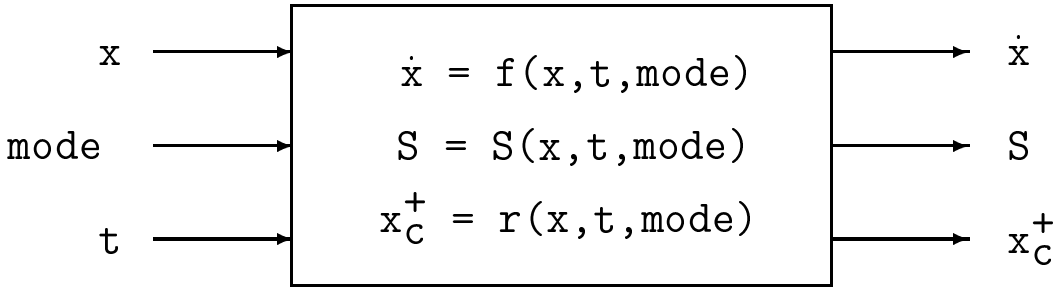
To handle discontinuities with good generality:

- **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 this:



(a) Standard MATLAB model schema



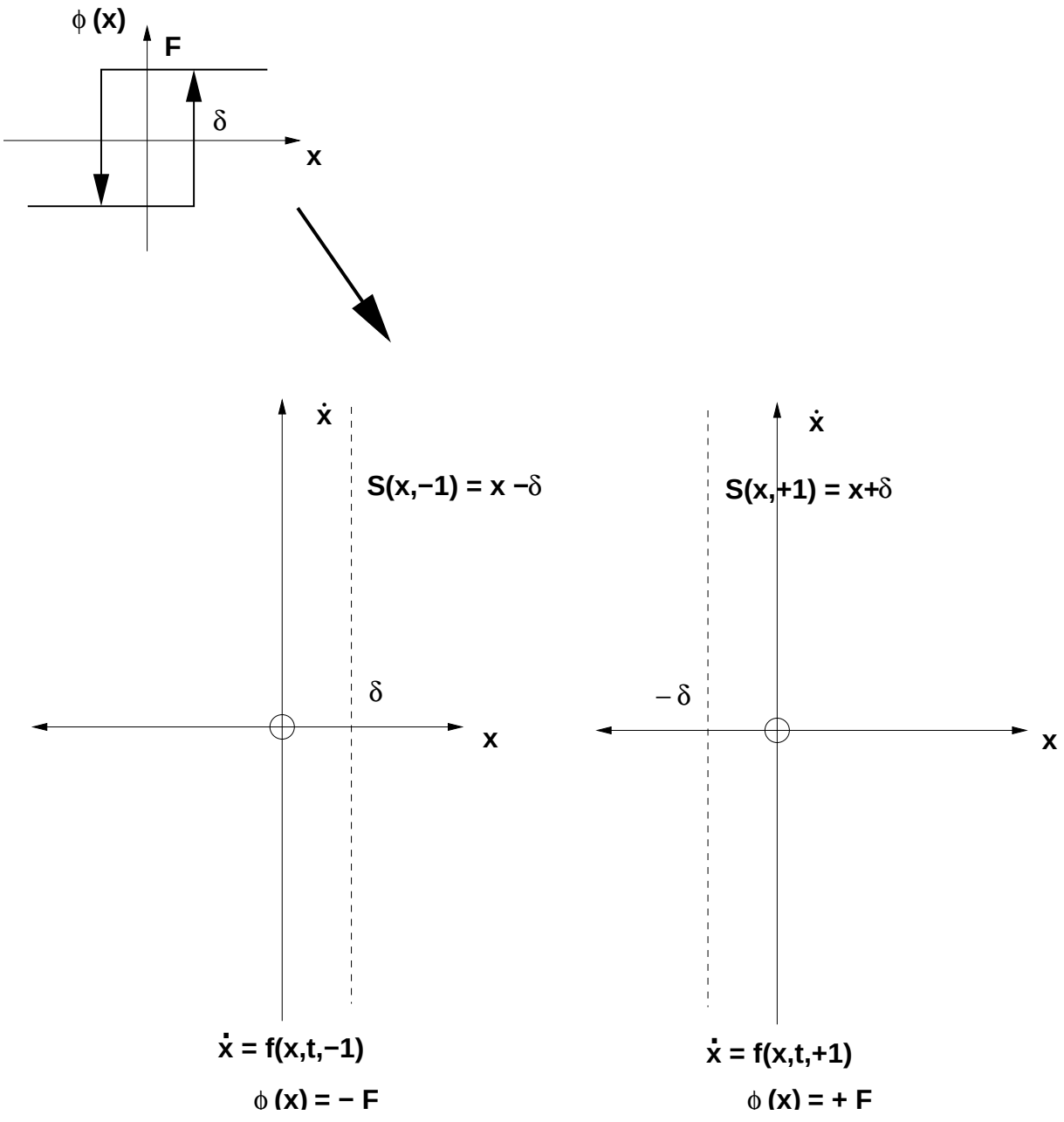
(b) Extended MATLAB model schema

Hybrid Integration Algorithm Framework

```
function [tout,yout] = method(ydot,t0,tf,y0,tol,trace)
%
%  integrates a system of ordinary differential equations
%  using a "good" algorithm, with a hybrid interpolation
%  scheme to catch state events (points where phi changes
%  sign).  Includes provision to handle state reset.
%
%% Initialization
< model called to determine mdim = number of modes >
%% Main integration loop
while (t < tfinal) & (t + h > t)
   % save data from last "accepted" point:
      y_old = y(:); t_old = t; phi_old=phi(:);
   % Update the solution (TRIAL point!)
      tt = t + h;
      < yt = y at tt using a "good" algorithm >
      < phi at tt also determined >
   % Check for state events (SEs)
      for im = 1:mdim
         if sign(phi(im)) == -sign(phi_old(im))
            %% OK - we have detected a state event:
            % set up fzero-like zero-finding scheme:
            < h*(im), yh*(:,im) found for each crossing >
         end
      end
   % now, check for earliest/simultaneous SEs:
```

```
      < im**, h**, yh**(:) ; later points discarded >
   % see if reset is called for:
      [junk,trash,reset] = feval(dyfun, t, y, rmode);
      if reset ~= [],
         < yr = y at tt+ according to model >
      end
   % now, at last, update mode:
      < mode(im**) = sign(phi(im**)) >
end % of main WHILE loop
tout = tout(1:k);
yout = yout(1:k,:);
```

Modes for Systems with Discontinuities

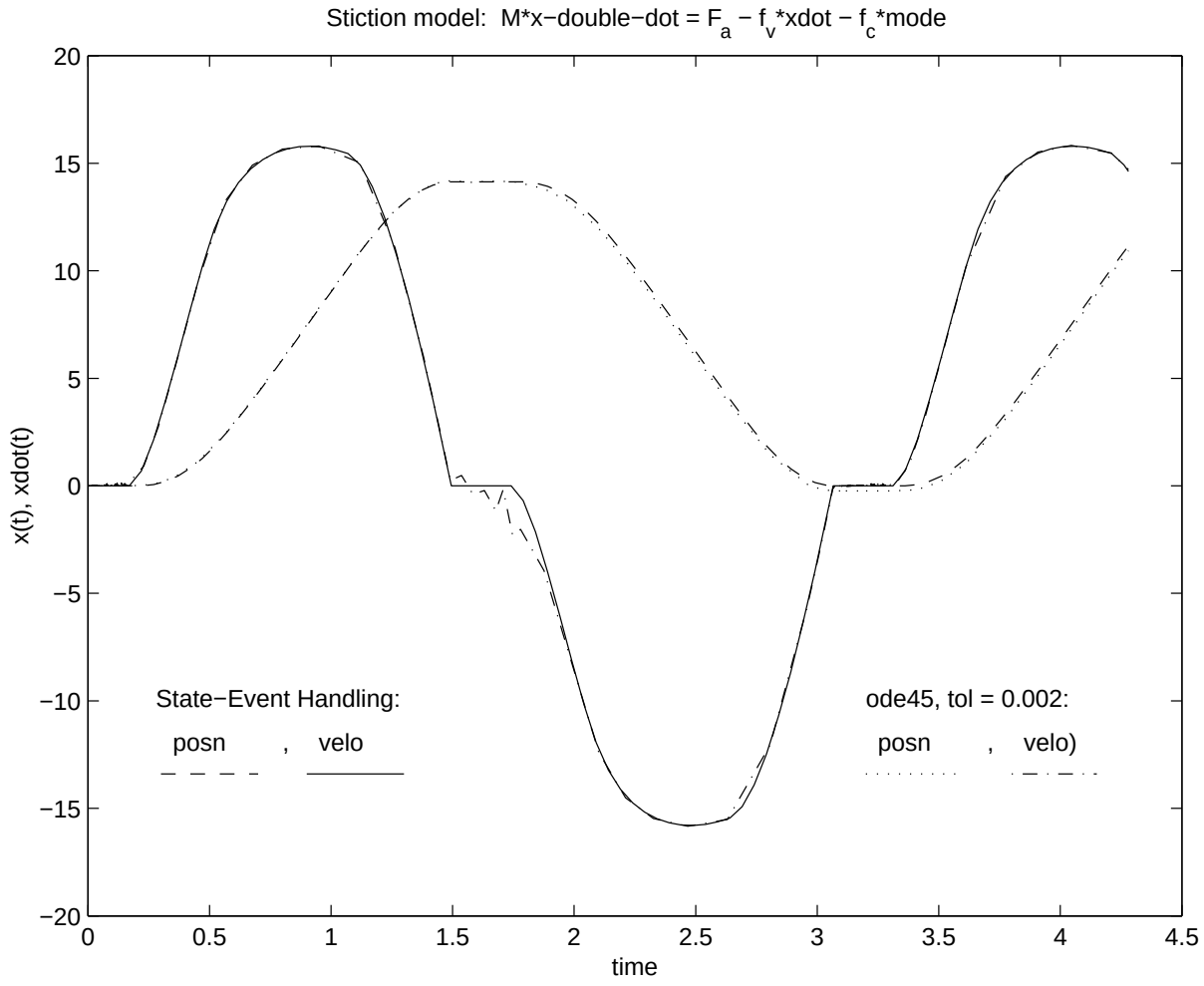


The `mode` "remembers" the state of the relay.

Systems with Discontinuities (Cont'd)

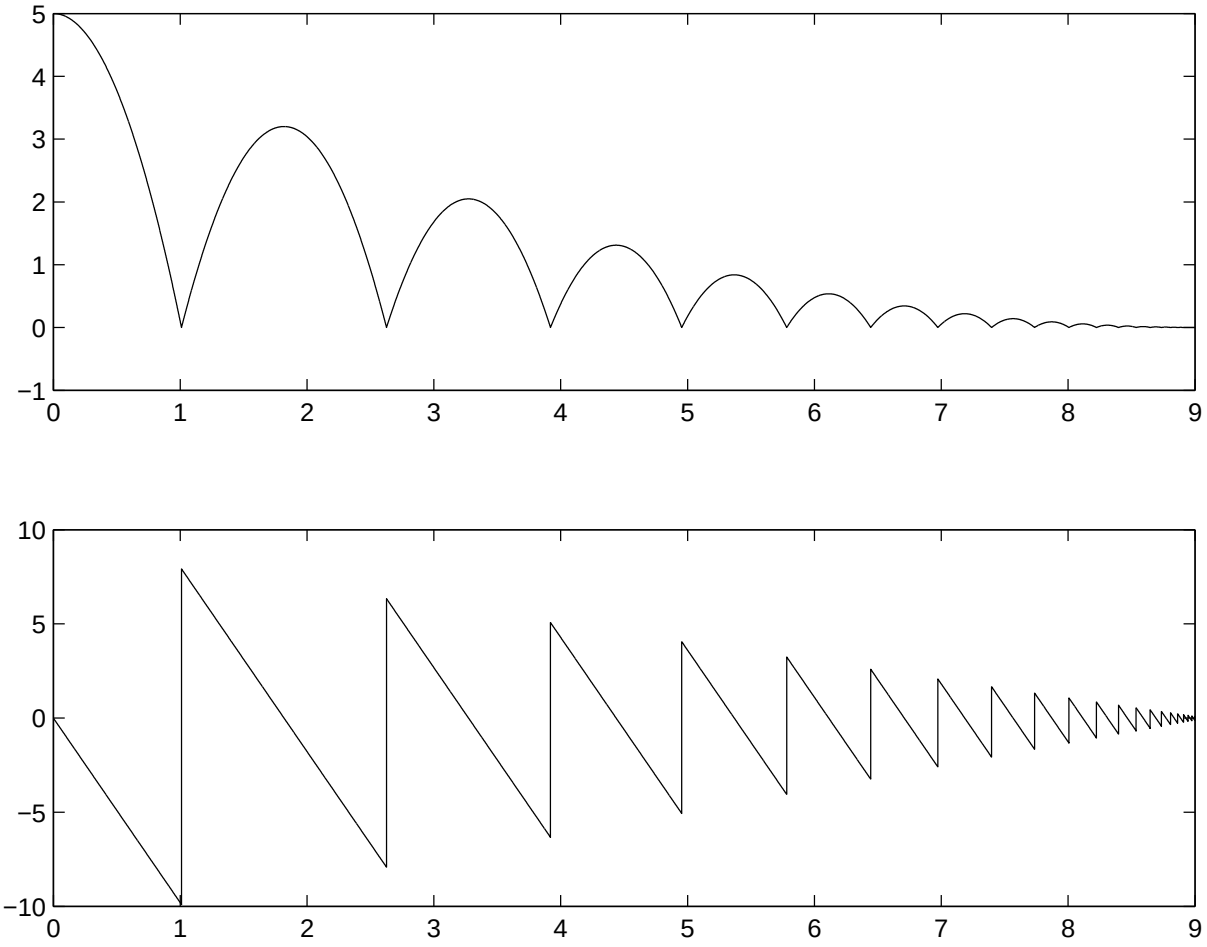
Examples:

- **Mass with viscous and Coulomb friction:**



Systems with Discontinuities (Cont'd)

- **Bouncing ball (with velocity reset):**



We can now simulate systems with discontinuities with complete rigor!

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 at least (using past derivatives doesn't make sense); a state-event handler is highly desirable.**
  - **variable step-size algorithms are almost always preferable unless part of the system is discrete-time.**

  For a first exploration, fourth-order Runge Kutta with variable step size is recommended.
- **Safety checks:**
  - **try a different algorithm**
  - **halve the step size; try again**
  - **use double precision; try again**

  Never take the first results on faith!