Matlab and Simulink: Modeling and Simulating Continuous, Discrete, and Hybrid Systems

Per Bjuréus
per@ic.eecs.berkeley.edu

Agenda

8 Introduction
  4 Matlab
  4 Simulink
8 ODE example: the bouncing ball
  4 Modeling and solving in Matlab
  4 Modeling and solving in Simulink
8 Hybrid Systems in Simulink
8 Summary
Matlab Tool

8 Matlab is a Tool
  4 Matlab = Matrix Laboratory
  4 Function evaluation engine
  4 Open Environment
  4 Large set of toolboxes
    ➢ Numerical methods
    ➢ Visualization
    ➢ Statistics
    ➢ Signal processing
    ➢ Fuzzy Logic
    ➢ Etc., etc.

Matlab Visualization

8 Easy visualization
8 Example:
  4 surfl(peaks)
  4 colormap copper
  4 shading interp
  4 axis off
  4 zoom(1.5)
Matlab Language

8 Matlab is a Language
   4 Imperative
      ➢ Scripts
      ➢ Control flow
      ➢ Side effects
   4 Functional
      ➢ Functions
      ➢ Input/Output
      ➢ (No) side effects
8 Debugger

Simulink

8 Simulink is a GUI
   4 Draw and Connect
   4 Plug-in toolboxes
   4 User defined libraries
8 Models
   4 Continuous
   4 Discrete
   4 Hybrid
8 Simulink is a Simulator
   4 Solving ODE
Example

8 Bouncing ball example

4 Ordinary Differential Equation (ODE)

Drop a ball from 15m height, acceleration is \(-9,8\)m/s\(^2\). Second order system.

State equation, choosing the displacement and velocity as state variables \((y_1, y_2)\):

\[
\begin{align*}
y_1 &= x \\
y_2 &= \dot{x}
\end{align*}
\]

The time rates of change (derivatives) of the two state variables are:

\[
\begin{align*}
\dot{y}_1 &= y_2 \\
\dot{y}_2 &= -9,8
\end{align*}
\]

This is our Ordinary Differential Equation (ODE).

ODE representation

8 ODE representation for \(n\) states and \(m\) inputs

4 State vector \(x\) (size \(n\))

4 Input vector \(u\) (size \(m\))

4 System matrix \(A\) (always square, \(n\) by \(n\))

4 Input matrix \(B\) (\(n\) by \(m\))

\[
x = Ax + Bu
\]

\[
x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{bmatrix}
\]
8 Bouncing ball example

4 Matlab modeling

The ODE can be written in Matlab as a function using a matrix notation:

```matlab
function dydt = f(t,y)
    dydt = [y(2); -9.8];
```

Which is the core of our ODE function. We also need to initialize the state variables:

```matlab
function [tspan,y0,options] = init
    tspan = [0; 10]; y0 = [0; 0]; options = [];
end
```

`tspan` and `options` are used by the solver, which for instance is called with the command:

```matlab
[y, t] = ode45( odefile )
```

8 Solvers

There are seven ODE solvers to choose from:

4 `ode45`: Non-stiff differential equations, medium order method.
4 `ode23`: Non-stiff differential equations, low order method.
4 `ode113`: Non-stiff differential equations, variable order method.
4 `ode23t`: Moderately stiff differential equations, trapezoidal rule.
4 `ode15s`: Stiff differential equations, variable order method.
4 `ode23s`: Stiff differential equations, low order method.
4 `ode23tb`: Stiff differential equations, low order method.
8 Bouncing ball example

4 Solving

We collect our functions in a file called `bounceode`, add some parameter handling (not shown here) and call the solver:

```
[t, y] = ode45(‘bounceode’, [0 2], [15 0]);
```

The result from the solver can be analyzed and visualized:

```
plot(t,y)
```

The blue plot is the displacement and the green plot is the velocity, i.e. our state-variables.

But… the ball doesn’t bounce – it falls through the floor.

---

8 Bouncing ball example

4 Zero crossing

We have to detect when the ball displacement (our first state variable) crosses zero to be able to model a bouncing ball.

The `events` function will provide the solver with zero-crossing detection, which will halt integration:

```
function [value,isterminal,direction] = events(t,y)
value = y;
isterminal = [1; 0];
direction = [-1; 0];
```

This function is added to our `bounceode` file.
Solving with Event Trapping

8 Bouncing ball example

4 Event trapping

So, we call the solver with event trapping:

```matlab
options = odeset('Events', 'on');
[t, y] = ode45('bounceode', [0 2], [15 0], options);
```

And we can see that the simulation halts when zero is reached.

Suppose that we want to see plot after the bounces.

We have to write a program that loops the solver, and reverses the velocity at each bounce.

The elasticity should be 0.8.

Solving in Matlab

8 Bouncing ball example

4 Solving with bounce (discontinuous system)

Here is a part of the program (initializations not shown) that solves for ten bounces:

```matlab
for i = 1:10
    % Solve until the first terminal event.
    [t, y, te, ye, ie] = ode45('bounceode', ...
        [tstart tfinal], y0, options);
    % Accumulate output.
    tout = [tout; t];
    yout = [yout; y];
    y0(1) = 0;
    y0(2) = -.8*y(end,2);
    tstart = t(end);
end
```
There's got to be a better way

8 *High learning threshold, mathematical theory is sometimes hard to penetrate.*
8 *The bouncing ball is a small example, describing larger systems is difficult.*
8 *The bouncing ball is just one system – what if we want to connect several systems?*
8 *The solution is (of course) Simulink.*

---

Modeling in Simulink

8 *Graphical representation*
8 *Simulating = Solving*

Select one constant block, two integrators, and a scope (drag-n-drop). Connect them. Set the initial condition of Displacement integrator to 15.
Click the simulation run button.
8 *Built-in Zero-Crossing detection*

Select **external reset on falling edge** for Velocity integrator – two new ports appear. Connect displacement to edge trigger port and connect velocity to external condition port.

**Error!**

---

8 *Warning message*

Warning: Block diagram 'mybounce' contains 1 algebraic loop(s).

Found algebraic loop containing block(s):

- 'mybounce/Velocity' (discontinuity)
- 'mybounce/Gain' (algebraic variable)

Discontinuities detected within algebraic loop(s), may have trouble solving detected algebraic loop path(s) through integrator block reset or IC ports. Results may not be intuitive (see documentation). Use integrator block state port to avoid algebraic loops
8 Resolving algebraic loops

Select show state port for Velocity integrator – one more port appears. Use the state port for feedback to external condition port. Simulate! OK!

8 Zeno behavior

4 Not detected in Simulink

Suppose we run the simulation for a longer period of time – it breaks down! At some point, the integration step becomes too large, and the position is negative in two consecutive integration steps, no zero crossing can be detected, so the simulation goes awry.

The ball will stop bouncing after a finite amount of time. This time can be calculated analytically. (=15.7467s)
By resetting the Displacement integrator each time the zero crossing is detected prevents two consecutive negative values. However, simulation never stops.

---

**Hybrid Systems**

8 *Discrete Components*

4 Represented by matrices in a similar fashion
4 Step size is fixed and known
4 Multi-rate systems allowed

8 *Mixed Discrete and Continuous Components*

4 Run continuous simulation up to next discrete event
   ➢ Event trapping enabled
4 Compute discrete states
   ➢ Algebraic loops solved with fixed-point iteration
Hybrid System Example

- Digital multi-rate cruise control
- Analog source and feedback

Stateflow

- Integrates StateCharts into Simulink
- Reacts to events
  - Zero-Crossing
  - Discrete output
  - StateChart communication
- Executes in zero time
Summary

8 Matlab
   4 Tool and language
   4 Imperative and functional

8 Simulink
   4 Compiles system into state matrices
   4 Detects and handles algebraic loops
   4 Breaks down on Zeno behavior
   4 Integrates discrete and continuous systems
   4 Offers link to StateCharts