NCSU ASME Technical Sessions
Spring 2001, Fall 2001, Spring 2002

Simulink: a Graphical Tool for Dynamic System Simulation

G. D. Buckner
Department of Mechanical and Aerospace Engineering
North Carolina State University

What is Simulink?

- a graphical, interactive software tool for modeling, simulating, and analyzing dynamic systems

- enables rapid construction of "virtual prototypes" to explore design concepts at any level of detail with minimal effort

- ideally suited to linear, nonlinear, continuous-time and discrete-time systems

- commonly used in control system design, DSP design, communication system design, and other simulation applications

- a graphical plug-in for MATLAB®, offering additional access to a range of non-graphical analysis and design tools

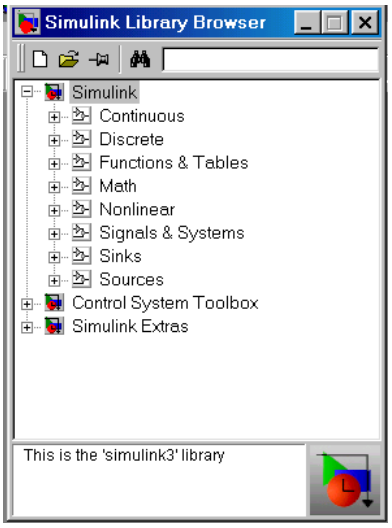
- more information: **www.mathworks.com**

Traditional approaches to system design typically include building a prototype followed by extensive testing and revision. This method can be both time-consuming and expensive. As an effective and widely accepted alternative, simulation is now the preferred approach to engineering design. Simulink is a powerful simulation software tool that enables you to quickly build and test virtual prototypes so that you can explore design concepts at any level of detail with minimal effort. By using Simulink to iterate and refine designs before building the first prototype, engineers can benefit from a faster, more efficient design process.

Highlights
Simulink provides an interactive, block-diagram environment for modeling and simulating dynamic systems. It includes an extensive library of predefined blocks that you can use to build graphical models of your systems using drag-and-drop operations. Supported model types include linear, nonlinear, continuous-time, discrete-time, multirate, conditionally executed, and hybrid systems. Models can be grouped into hierarchies to create a simplified view of components or subsystems. High-level information is presented clearly and concisely, while detailed information is easily hidden in subsystems within the model hierarchy.

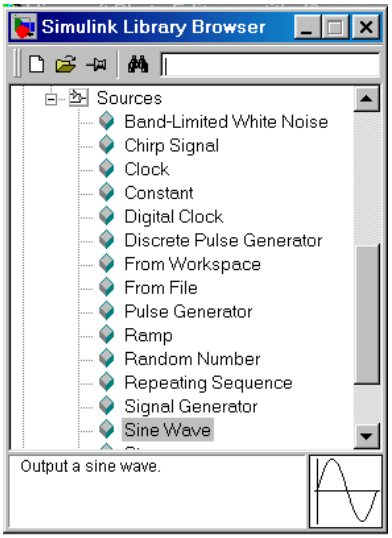
How do you create a program in Simulink?

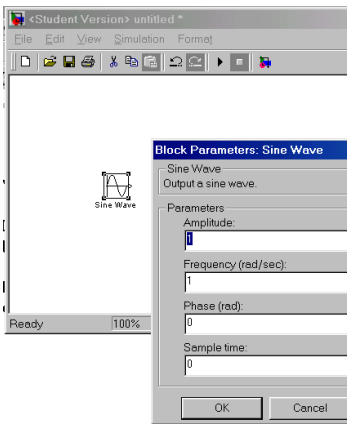
- Begin with the **Simulink Library Browser** - a collection of tools for building dynamic models
- Programming tools are organized into functional folders: **Sources** (function generators, etc), **Sinks** (oscilloscopes, data files, etc), **Math** functions, etc.
- Click the **New Model** icon to create a blank project



Start with a very simple program:

- Open the **Sources** folder, and select the **Sine Wave** block
- Drag the **Sine Wave** block onto your blank project, and double-click to setup its waveform parameters (Amplitude: 10, Frequency: 2*pi)





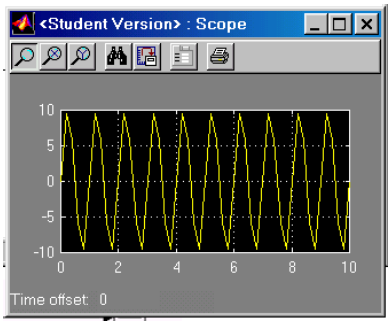
- Open the **Sinks** folder, and select the **Scope** block (an oscilloscope)
- Drag the **Scope** block onto your blank project

- Click and drag your mouse cursor to create a connection between the **Sine Wave** output and the **Scope** input:

- Click the **Start Simulation** toolbar control: 

- Double-click the **Scope** block to view the simulation output:

  It doesn't look like a perfect sinewave because Simulink defaults to a variable-step solver, but in this trivial problem we have no states to solve

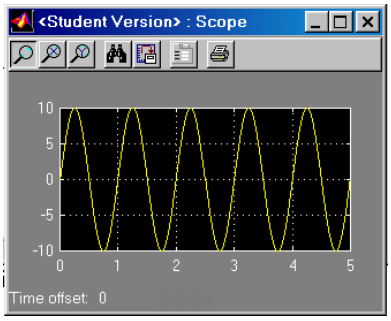


- To change the simulation parameters, select the **Simulation/Parameters** menu bar item.

- Change the **Maximum Step Size** to 0.01 sec, and the **Stop Time** to 5.0 sec.

- Run the program again, and the **Scope** output now looks like:

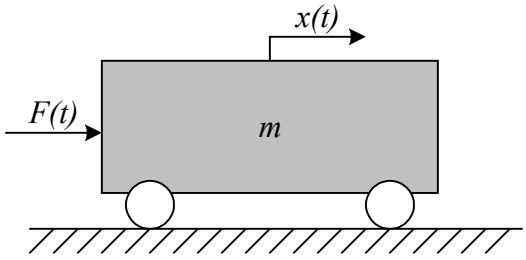
  Looks better due to smaller step sizes



Now consider a more realistic simulation:

- A cart with mass *m* rolling on a frictionless surface.

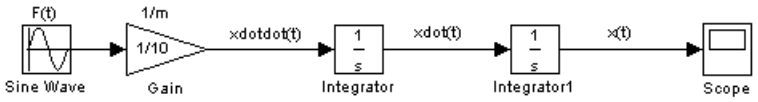
- The equations of motion for this system can be found using Newton's 2nd Law:



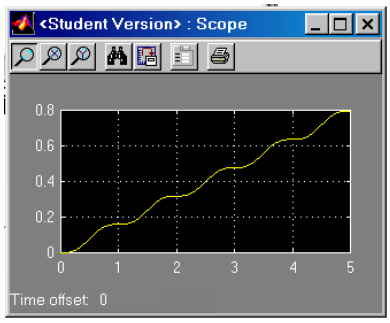
$$m\ddot{x} = \sum F_x$$

$$\ddot{x} = \frac{F(t)}{m} \qquad (1)$$

- Simulink integrates differential equations like (1) using **Integrator** blocks (in the **Continuous** folder). Since this system is second order, click and drag two **Integrator** blocks onto your project, along with one **Gain** block (in the **Math** folder), and connect them as shown:



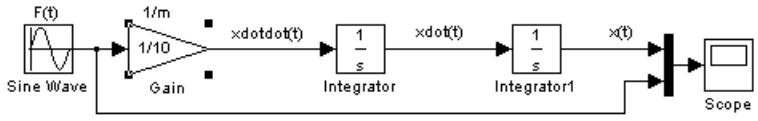
- Assuming the cart mass is 10, run the program, and the **Scope** output now looks like this:



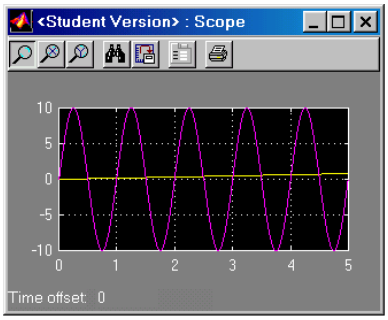
  This is a plot of *x(t)* vs. time

  Notice that the cart has net motion to the right due to the sinusoidal forcing

- If you want to display the forcing function F(t) on the same plot, click and drag a **Mux** block (a multiplexer, in the **Signals & Systems** folder). Connect the **Mux** to the **Scope** as shown:



- Run the program, and the **Scope** output now looks like this:



  Both *F(t)* and *x(t)* vs. time

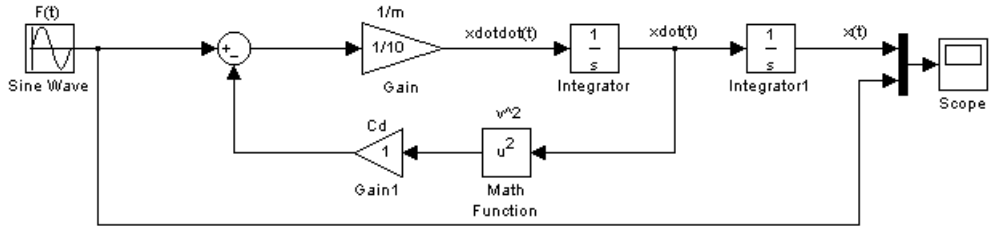
Let's add air drag (nonlinear damping):

- The equations of motion for this system are now:

$$m\ddot{x} = \sum F_x$$

$$\ddot{x} = \frac{F(t) - C_D \dot{x}^2}{m}$$

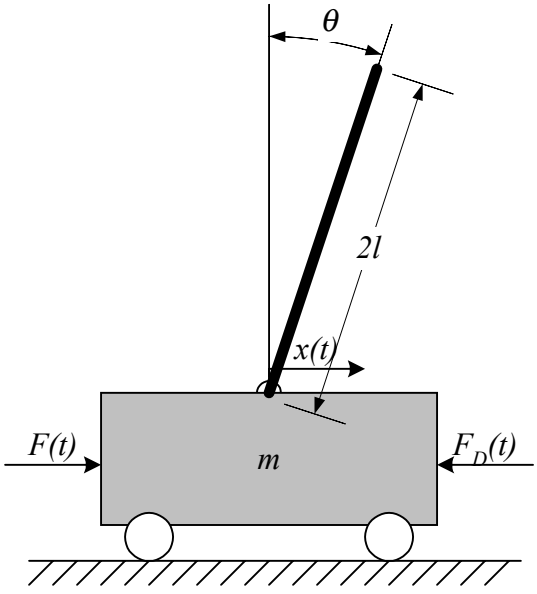
- To add this nonlinearity, click and drag a **Sum** block, a **Gain** block, and a **Math Function** block, (all 3 in the **Math** folder) and connect them as shown:



- Notes:
  - To rotate a block, type Ctrl+R on the keyboard
  - To select the **Square** function on the **Math Function** block, double-click it and choose this function from the list
  - To change the +/- signs on the **Sum** block, double-click it

Consider a challenging control simulation:

- A cart with mass *m,* rolling on a frictionless surface, with air drag, balancing a pole of mass m_p.

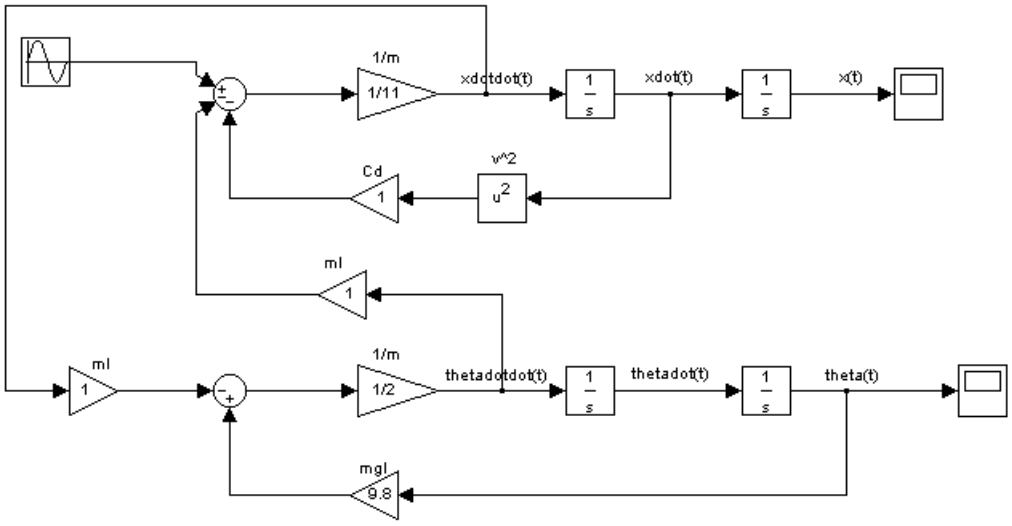


- Control Objective: manipulate cart force *F(t)* to balance the pole (keep *θ(t)* near zero)

- The equations of motion for this system can be found using Newton's 2nd Law, and making some simplifications:

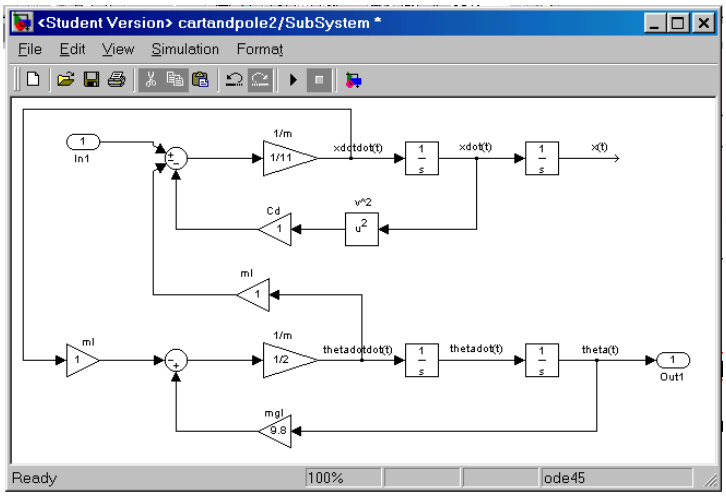
$$\left(m+m_p\right)\ddot{x}-C_D\dot{x}^2+m_p l\ddot{\theta}=F(t)$$
$$m_p l^2\ddot{\theta}+m_p l\ddot{x}=m_p gl\theta$$

- Assuming $m=10$, $m_p=1$, $l=1$, $J=1$, and $C_D=1$, the Simulink implementation of this complicated, nonlinear (and very unstable) system is:

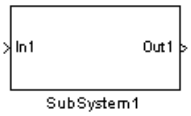


- This might look like a mess, but imagine programming this system in Fortran or C!

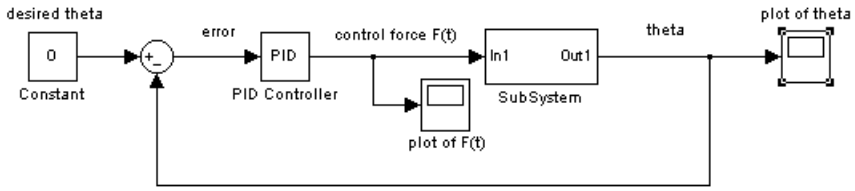
- One way to simplify the look of this system is to use a **SubSystem** block (in the **Signals & Systems** folder). Click and drag a **SubSystem** block onto your project, then select the entire cart/pole model and move it into the **SubSystem** (you can use Cut/Paste). Add an **In1** and **Out1** block (both in the **Signals & Systems** folder) as shown:



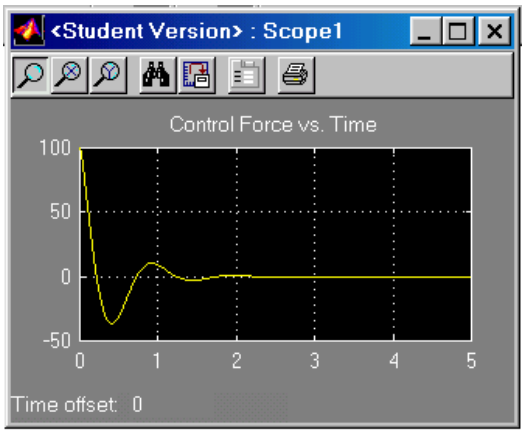
- Now, when you close this **SubSystem**, you will see only a block representing the entire cart/pole model. The input **In1** is the control force *F(t)*, the output **Out1** is the angular position of the pole:

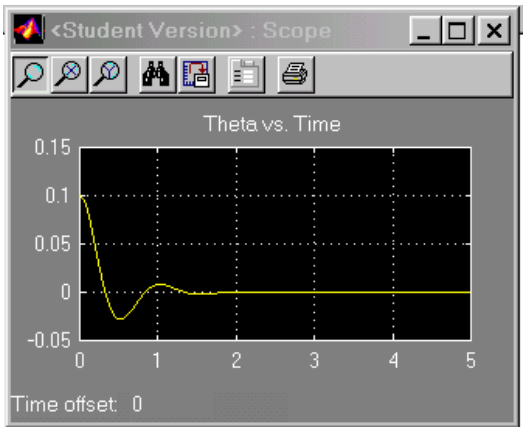


- Designing a controller to stabilize this system is now much easier. One "low-tech" approach is to select a **PID Controller** block and combine it into your project as shown:



- By "tweaking" the proportional and derivative gains a bit (*Kp=-1000, Kd=-100*), this system can be stabilized from an initial deflection of 0.1 radians, as shown in the following figures.


<Student Version> : Scope1
Control Force vs. Time
100
50
0
-50
0
1
2
3
4
5
Time offset: 0


<Student Version> : Scope
Theta vs. Time
0.15
0.1
0.05
0
-0.05
0
1
2
3
4
5
Time offset: 0