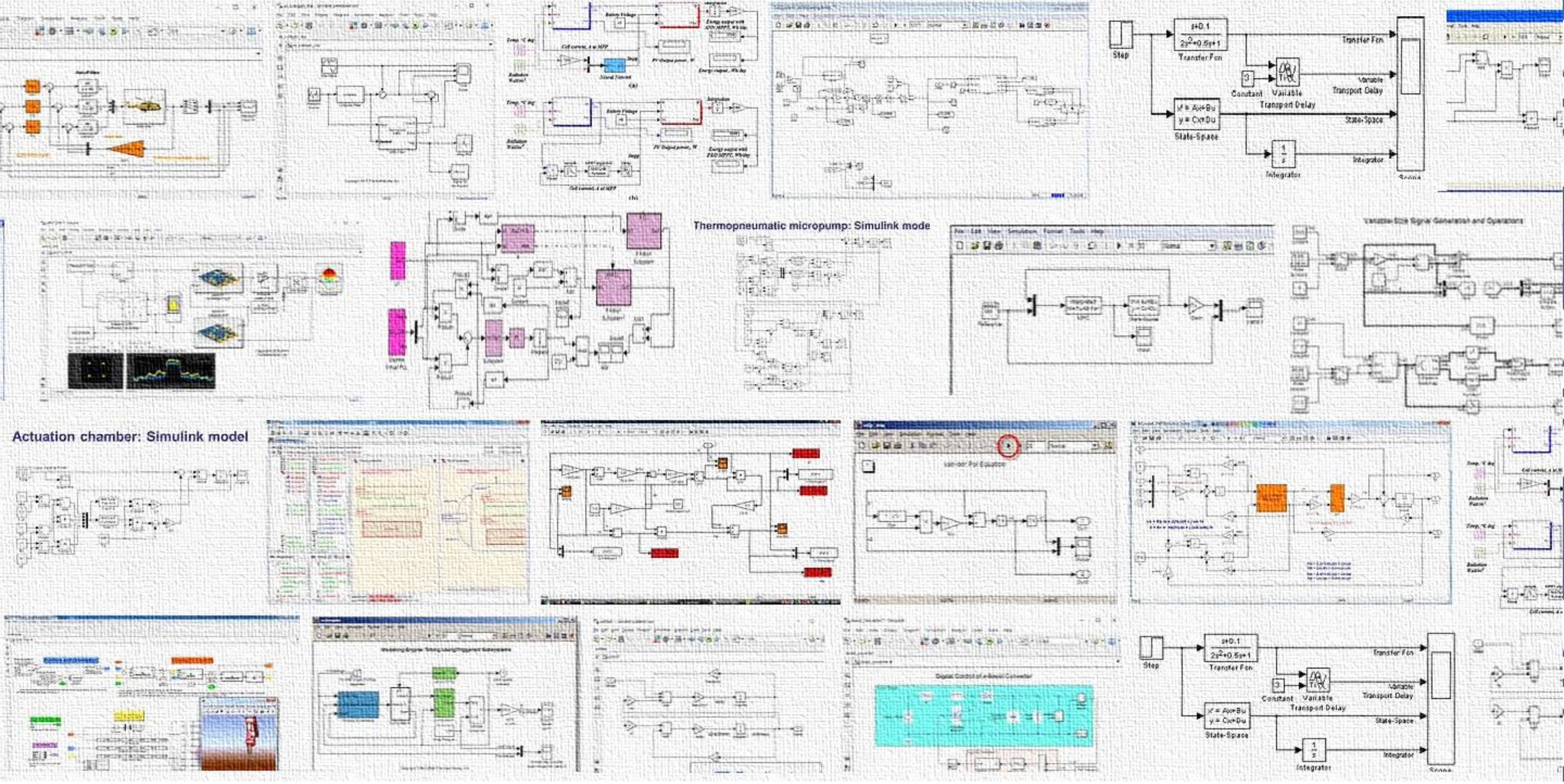


INTRODUCTION TO SIMULINK

Teaching Design
Support Group

SESSION OUTLINE

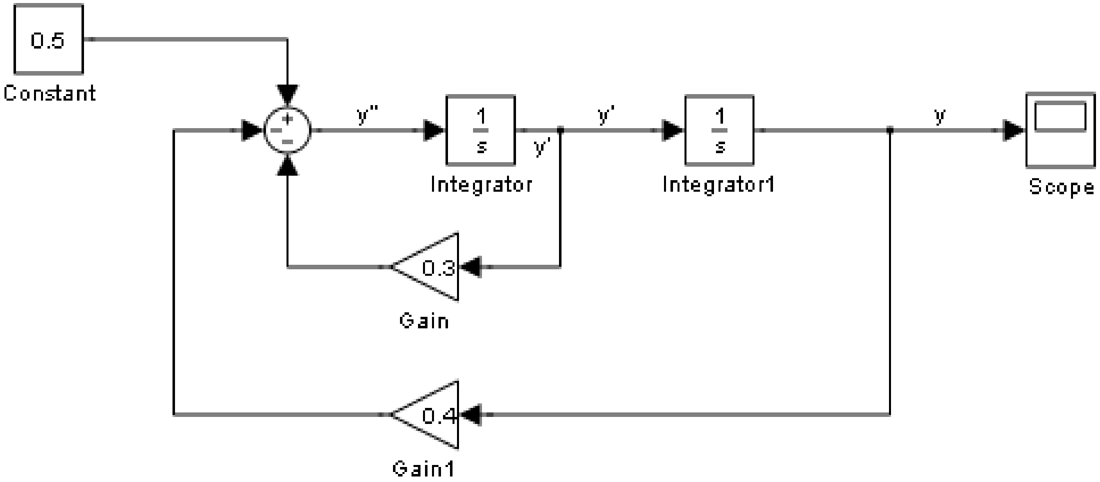
- Short Talk: Simulink Background & Basics
- Exercise 1: Firing a cannon
- Short Talk: Linking MATLAB & Simulink,
  Good Coding Practice
  Other Features
- Exercise 2: Systems of ODEs
- Exercise 3: Model of Bouncing Ball

WHAT IS SIMULINK?

- Simulink is a visual programming interface built into MATLAB
- A way to solve equations numerically using a graphical user interface

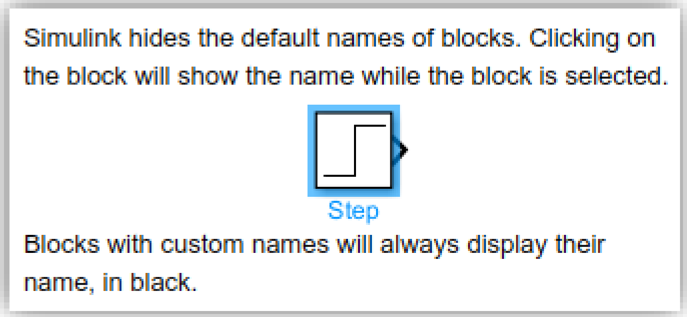
A simple example:

$$\frac{d^2y}{dt^2} = 0.5 - 0.3\frac{dy}{dt} - 0.4y$$



WHAT DOES A SIMULINK MODEL LOOK LIKE?

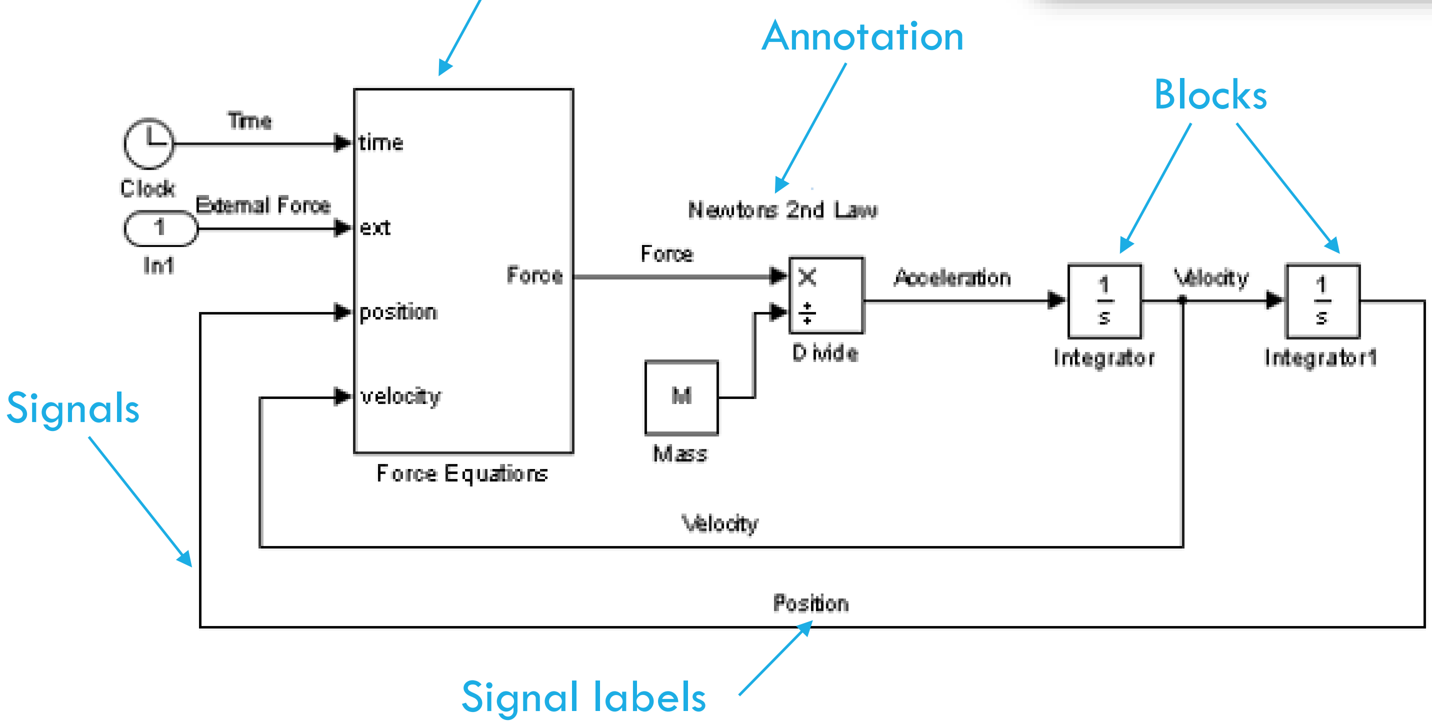
Simulink models are either .slx or .mdl files.



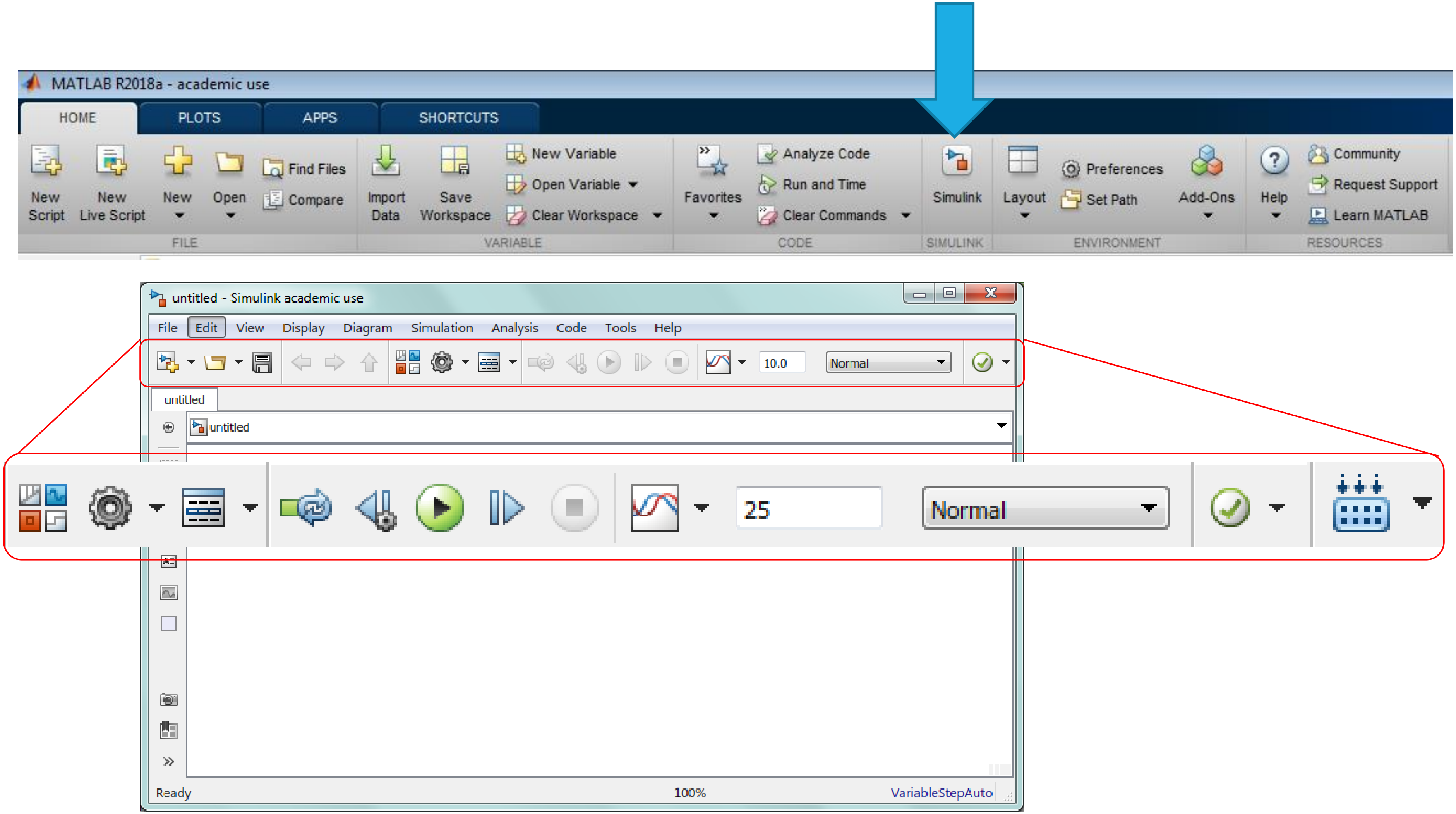
Simulink hides the default names of blocks. Clicking on the block will show the name while the block is selected.

Blocks with custom names will always display their name, in black.

Subsystems



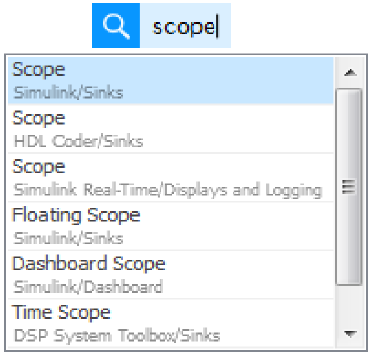
LAUNCHING SIMULINK

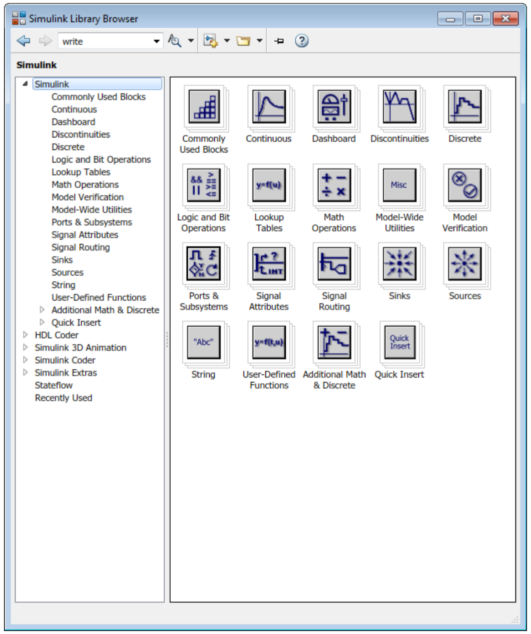


WORKING WITH BLOCKS

There are two ways to add blocks to a model:

- Library Browser
- Quick Search



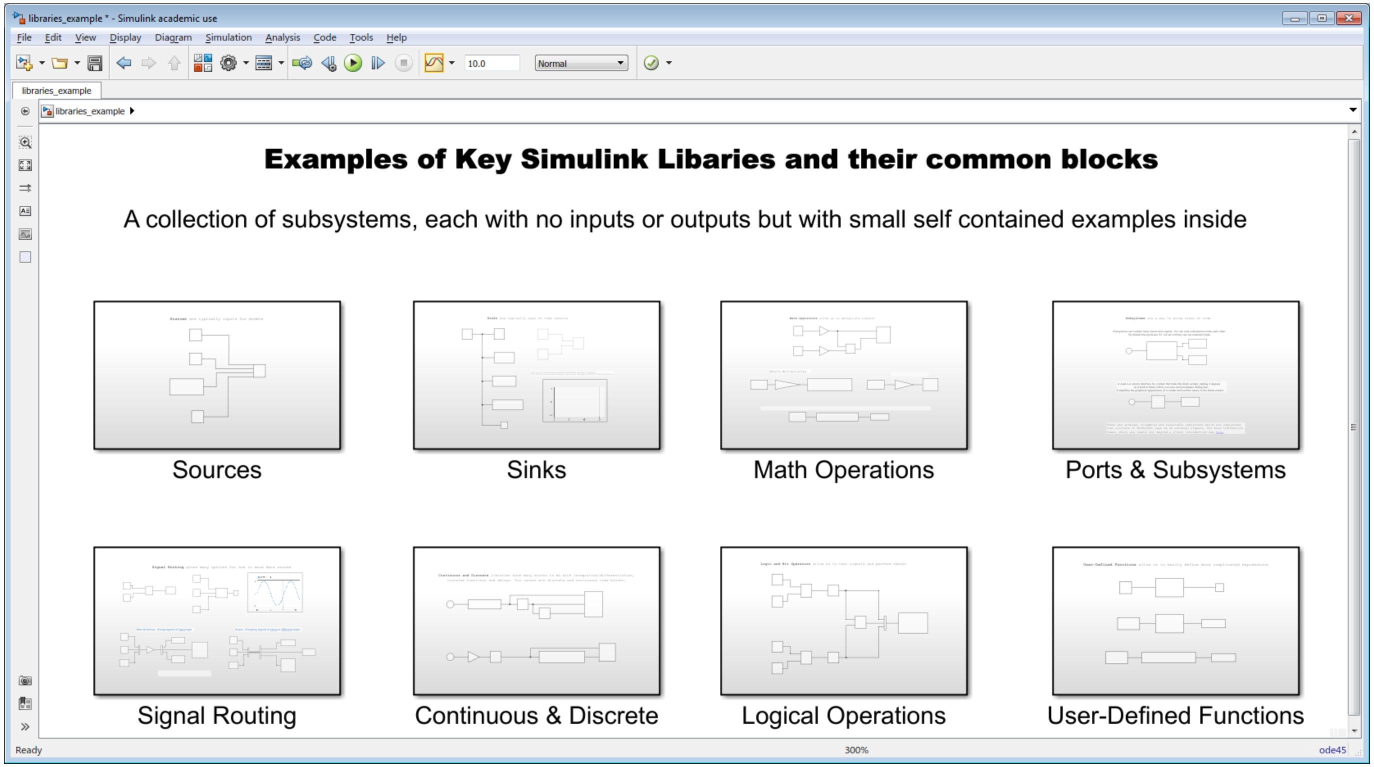


Each block has its own settings:

- **Block Parameters** – values when using the block [Double Click]
- **Block Properties** – how the block looks [Right Click]

IMPORTANT LIBRARIES

Simulink blocks are sorted into Libraries. Often with Simulink if you want to do something new, you need to search through the documentation to find the name of the block you need.

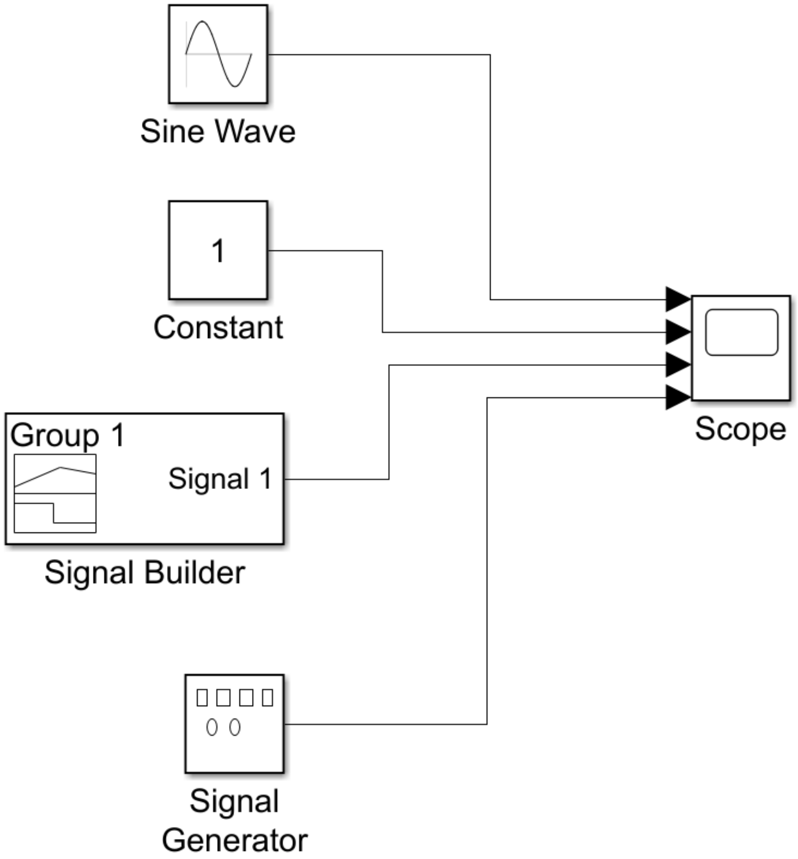


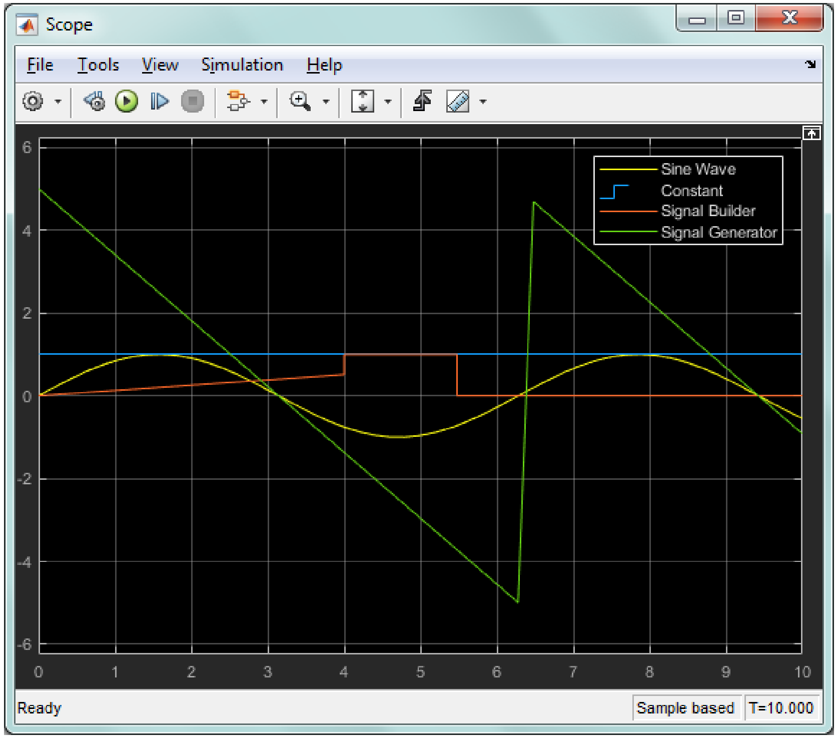
Examples of Key Simulink Libaries and their common blocks

A collection of subsystems, each with no inputs or outputs but with small self contained examples inside

SOURCES

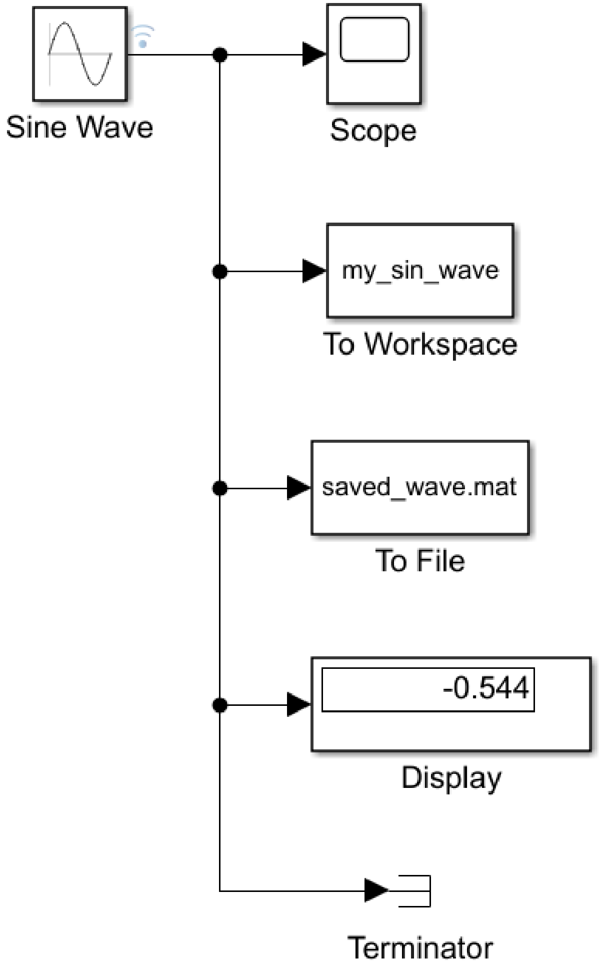
Sources are typically inputs for models

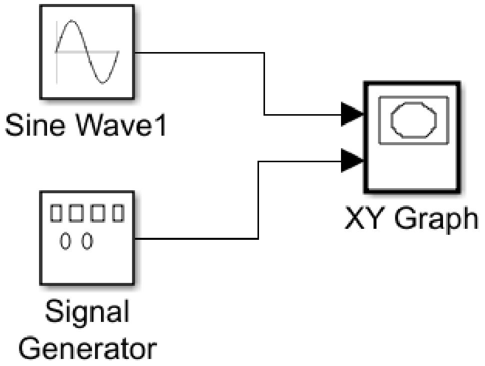


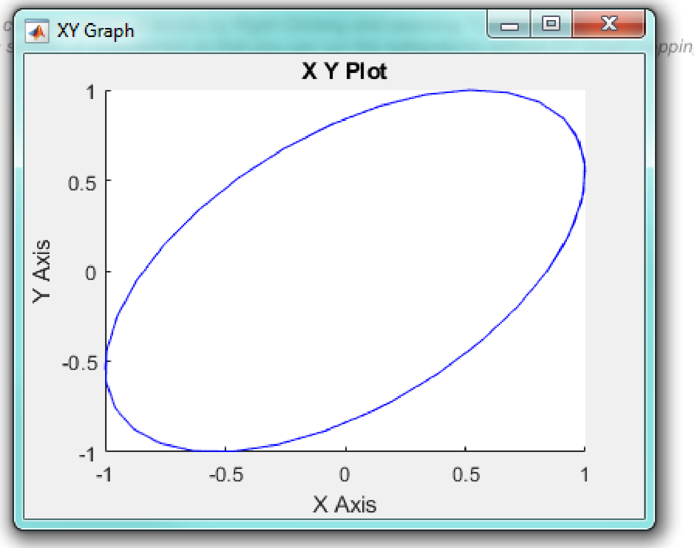


SINKS

Sinks are typically ways to view results

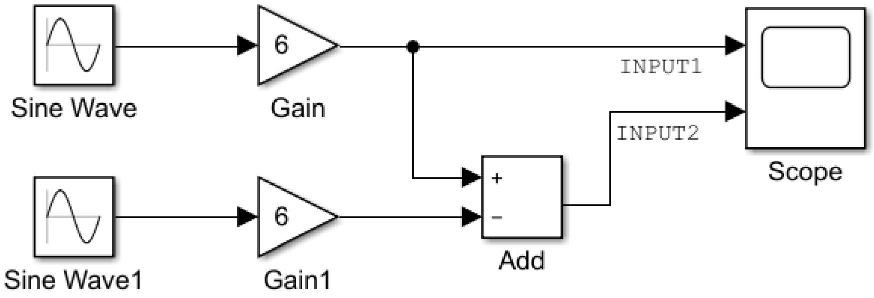




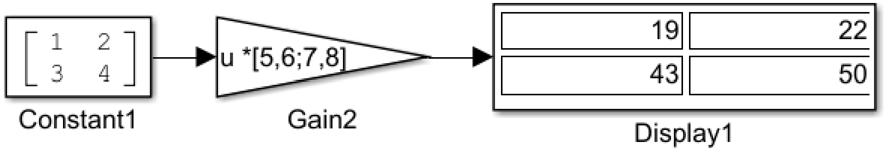


MATH OPERATIONS

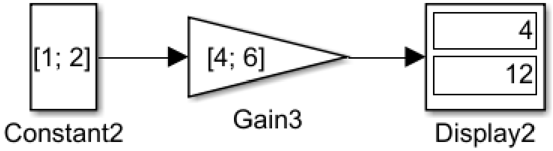
Math Operations allow us to manipulate signals



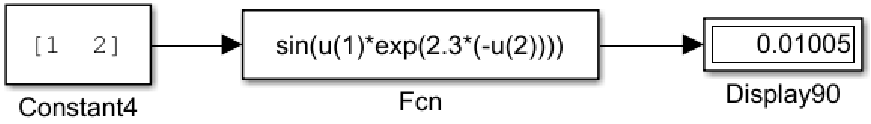
Matrix Multipliation



Array Multiplication

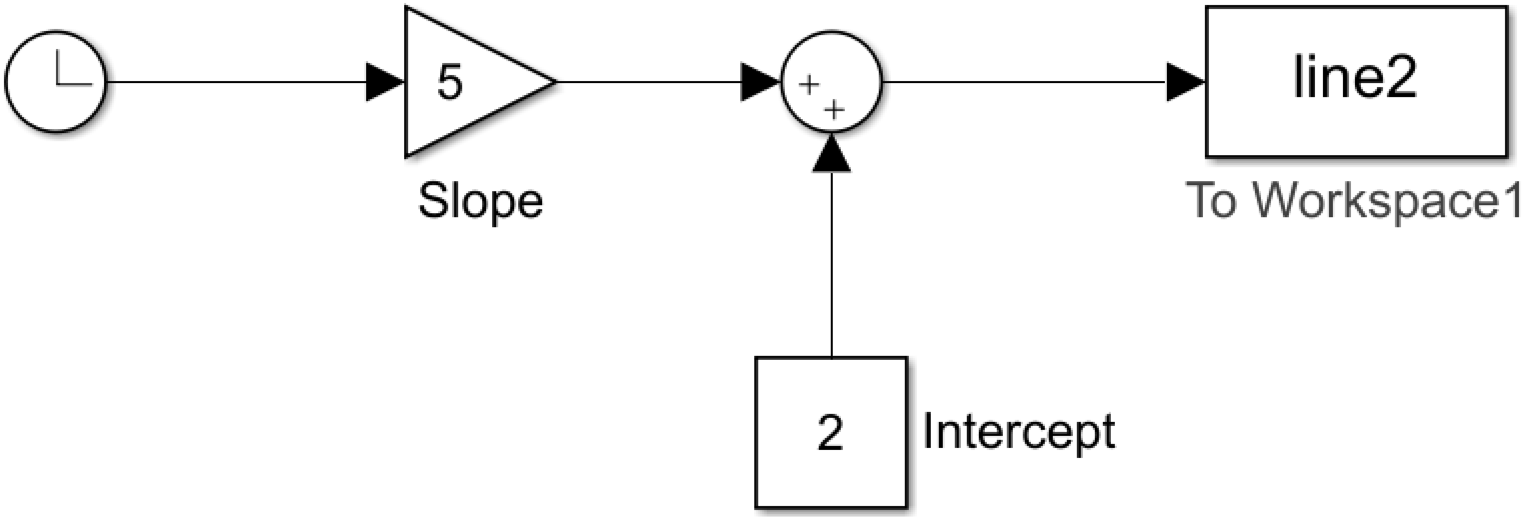


Technically a User-Defined Function, but "Fcn" is useful when thinking about math operations



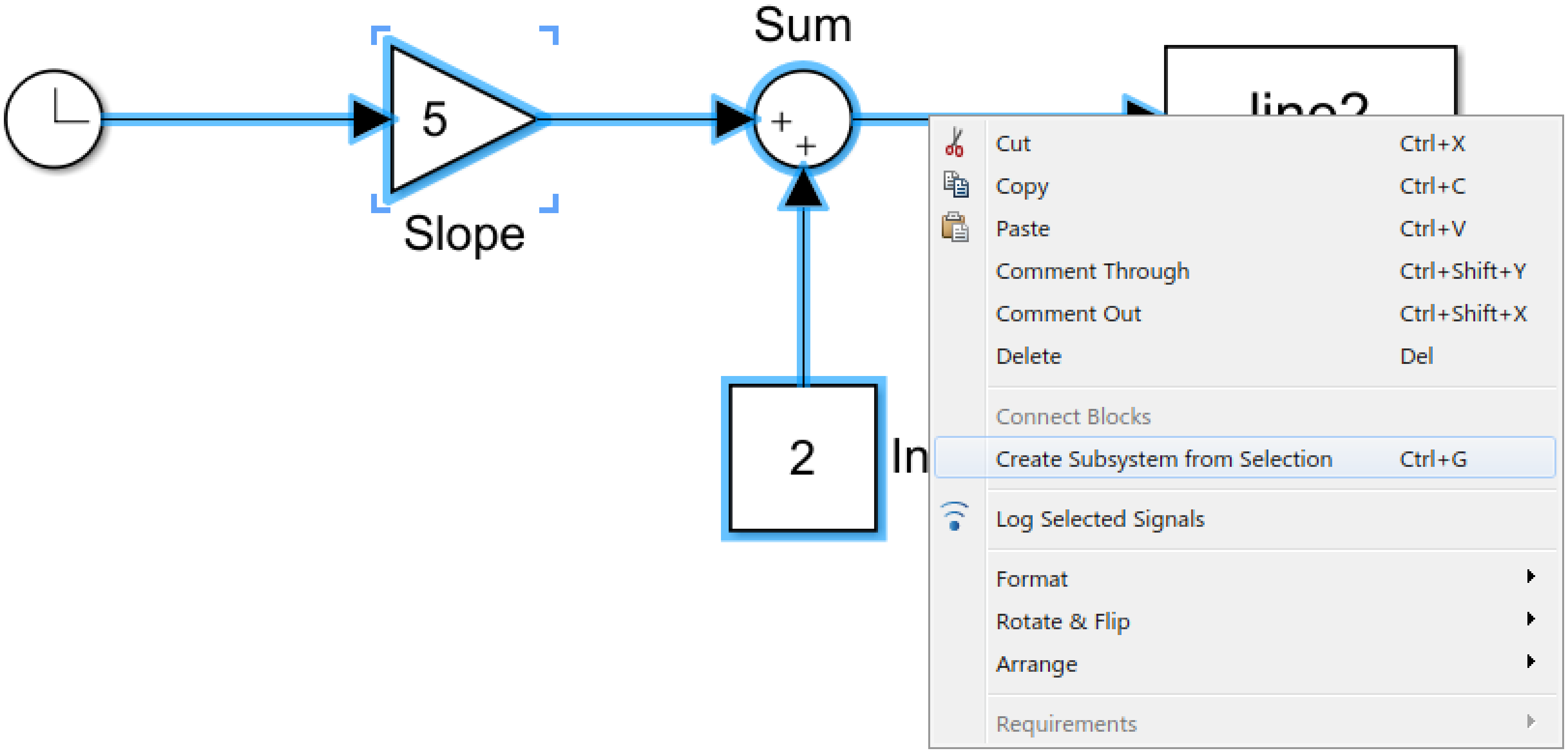
SUBSYSTEMS

Subsystems are a way to group areas of code.



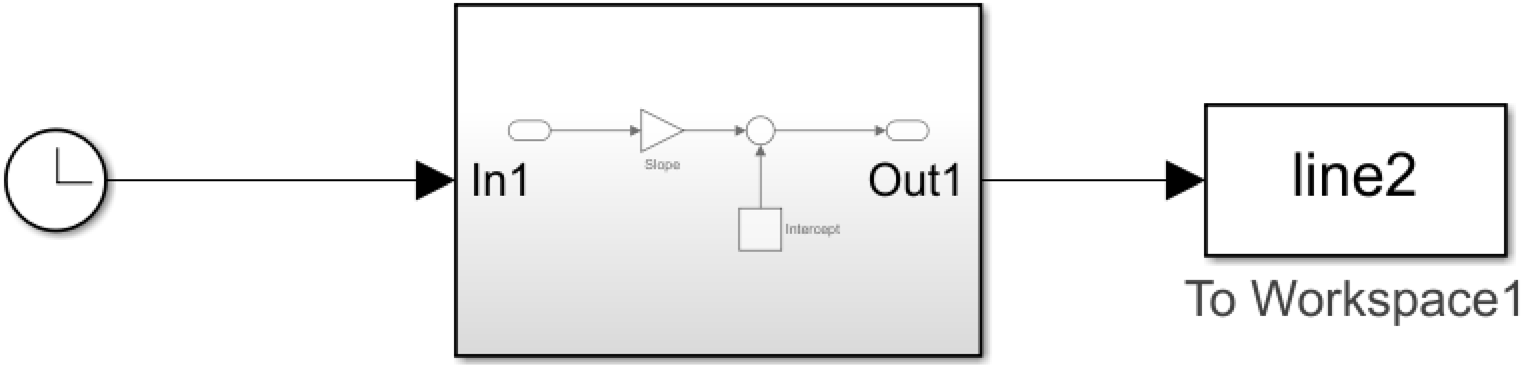
SUBSYSTEMS

Subsystems are a way to group areas of code.



SUBSYSTEMS

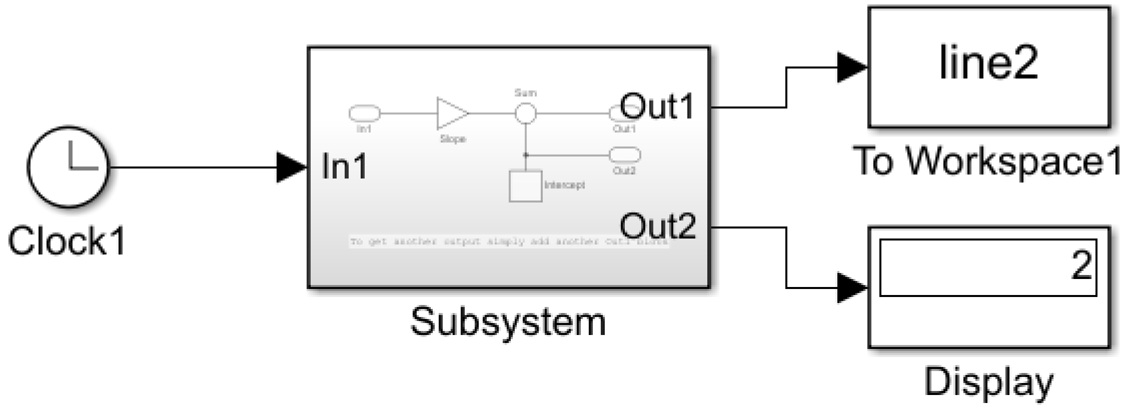
Subsystems are a way to group areas of code.

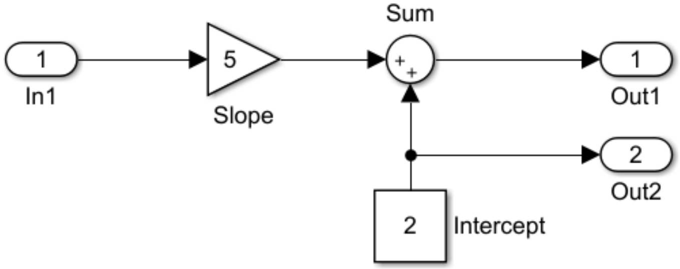


SUBSYSTEMS

Subsystems are a way to group areas of code.

Subsystems can contain many inputs and outputs. You can nest subsystems inside each other. By default the inputs are In1, In2 etc but they can be renamed inside.



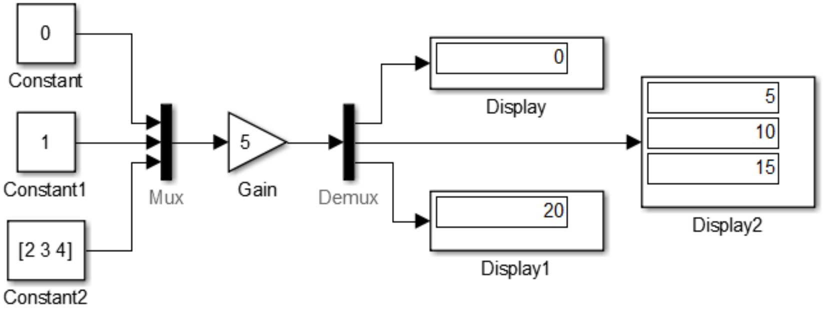


SIGNAL ROUTING

Mux & Demux : Group signals of same type

Mux block **groups multiple** signals
Demux block **separates out** individual signals

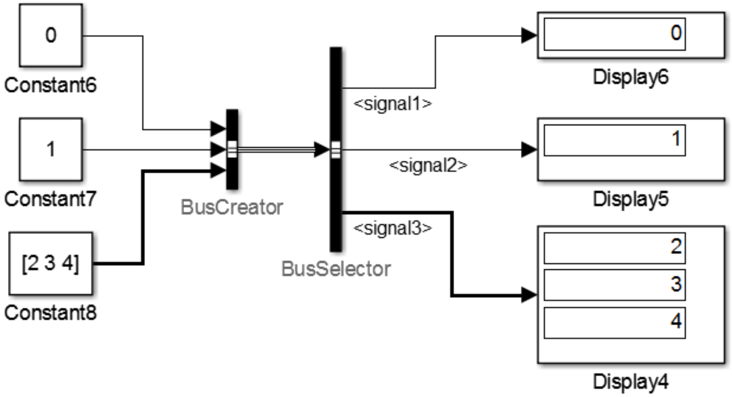
Can generally be used in mathematical operation blocks

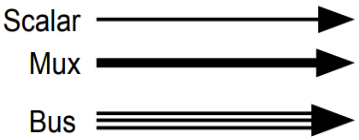


Buses : Grouping signals of same or different data types

A **bus** is like cable tidy. Not always possible to feed a bus into a mathematical block.

Use **BusCreator** and **BusSelector.**

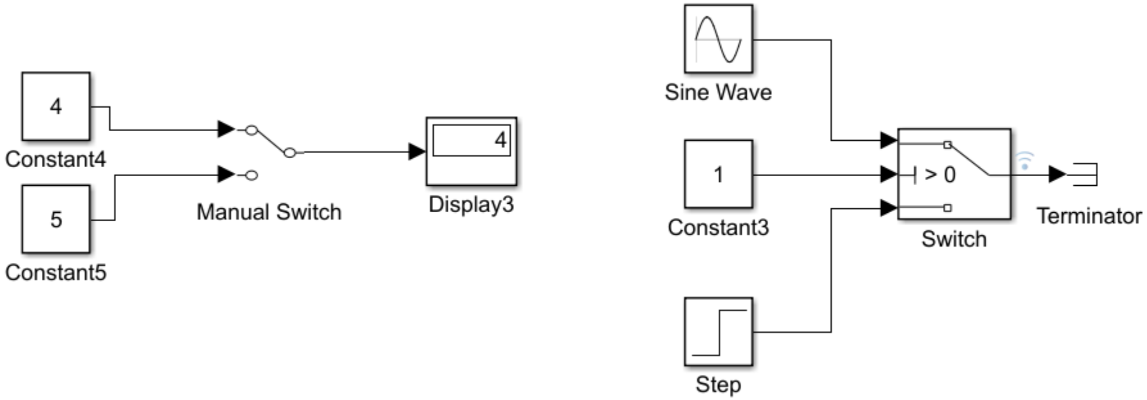




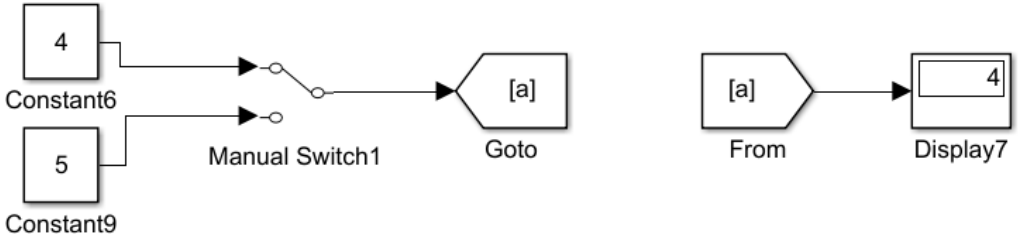
Display > Signals & Ports > Wide Nonscalar Lines

SIGNAL ROUTING

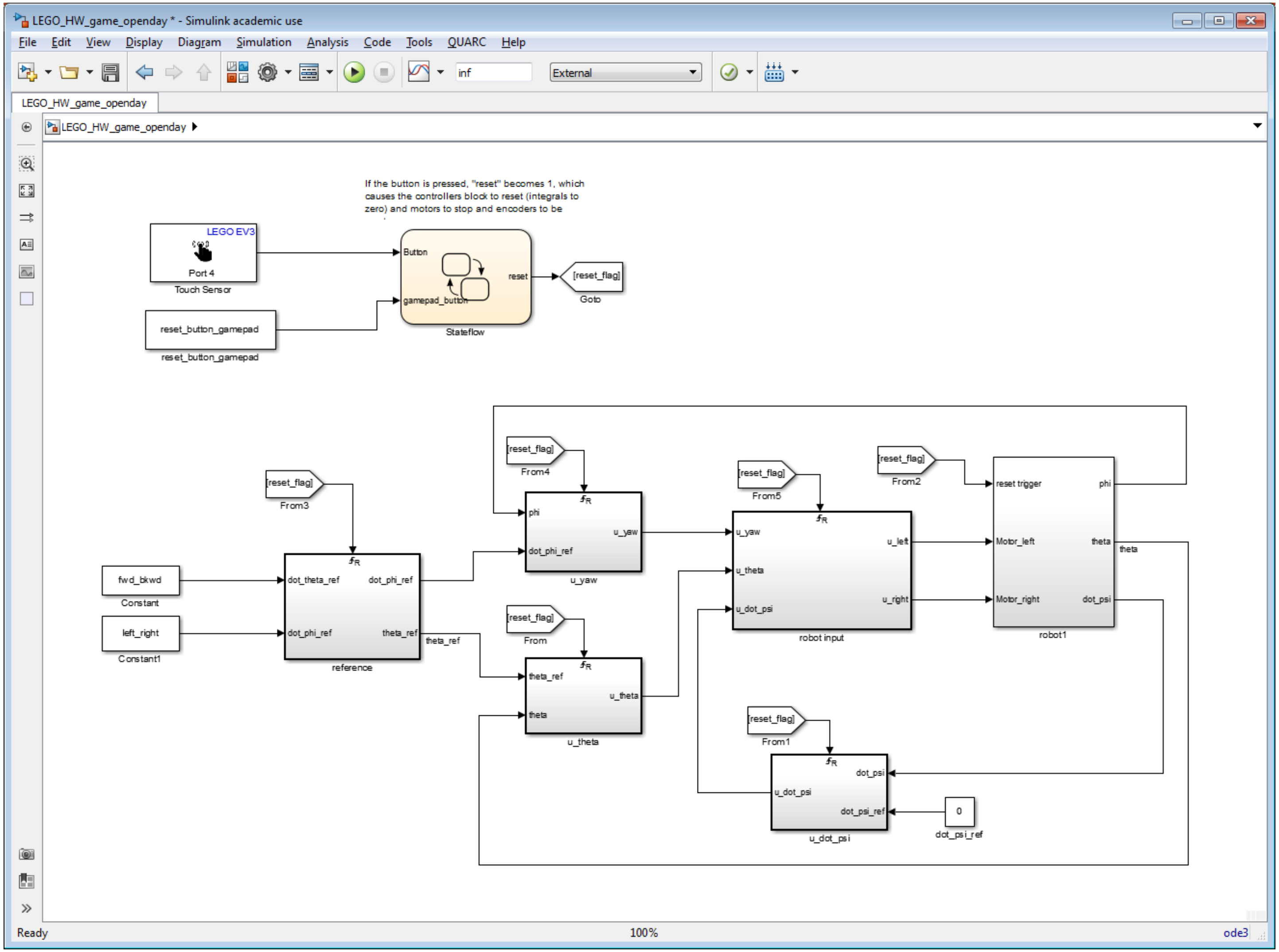
Switches



Goto / From



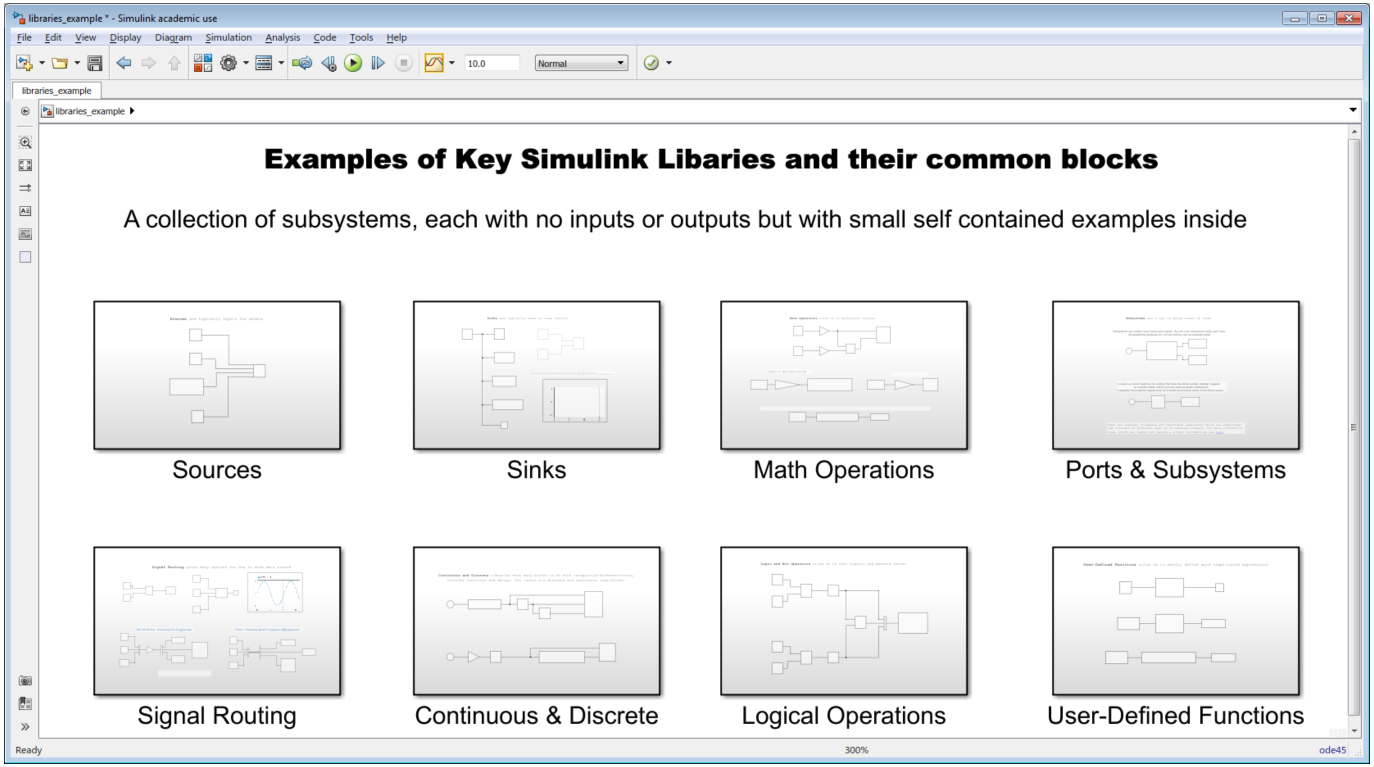
EXAMPLE OF GOTO/FROM



If the button is pressed, "reset" becomes 1, which causes the controllers block to reset (integrals to zero) and motors to stop and encoders to be

IMPORTANT LIBRARIES

Simulink blocks are sorted into Libraries. Often with Simulink if you want to do something new, you need to search through the documentation to find the name of the block you need.



libraries_example * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
libraries_example
libraries_example

Examples of Key Simulink Libaries and their common blocks

A collection of subsystems, each with no inputs or outputs but with small self contained examples inside

Sources

Sinks

Math Operations

Ports & Subsystems

Signal Routing

Continuous & Discrete

Logical Operations

User-Defined Functions

Ready 300% ode45

EXERCISES

- **Exercise 1: Firing a cannon**

a) Creating a Simple Simulink Cannon Model

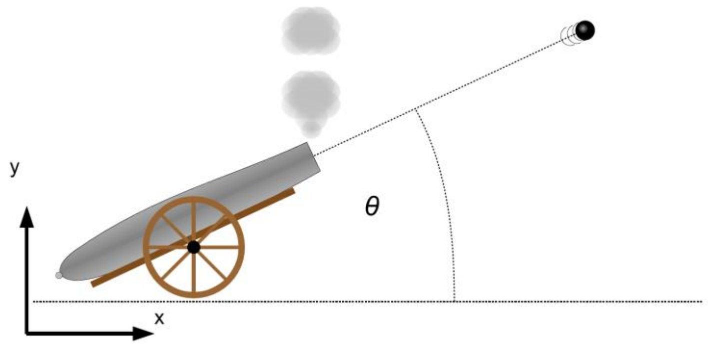
b) Introduction to Subsystems by Modelling Air Resistance

c) Adjusting Model Parameters

d) Further Plotting

e) Using MATLAB Expressions

f) Combining MATLAB and Simulink



Ask for help! Talk to us about where you need to use Simulink in your work/research!

SESSION OUTLINE

- ✓ ~~Short Talk: Simulink Background & Basics~~
- ✓ ~~Exercise 1: Firing a cannon~~
- ➢ Short Talk: Linking MATLAB & Simulink,
  Good Coding Practice
  Other Features
- ➢ Exercise 2: Systems of ODEs
- ➢ Exercise 3: Model of Bouncing Ball

SIMULINK & MATLAB TOGETHER

There are many ways in which MATLAB and Simulink interact:

- Run Simulink models programmatically in MATLAB (exercise 1. f)

```
s = sim('cannon5','StopTime','25','MaxStep','0.01');
```

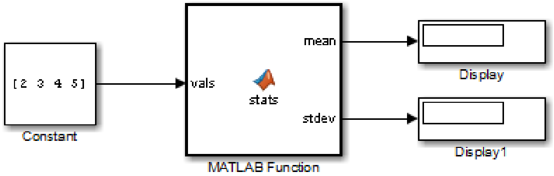
SIMULINK & MATLAB TOGETHER

There are many ways in which MATLAB and Simulink interact:

- Run Simulink models programmatically in MATLAB (exercise 1. f)

```
s = sim('cannon5','StopTime','25','MaxStep','0.01');
```

- MATLAB Function Block in Simulink



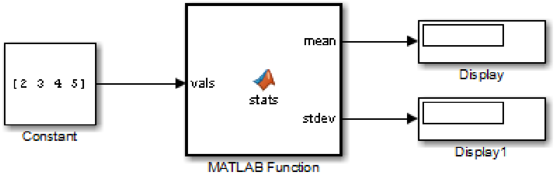
SIMULINK & MATLAB TOGETHER

There are many ways in which MATLAB and Simulink interact:

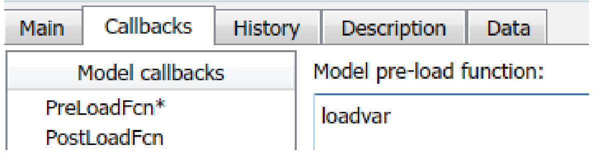
- Run Simulink models programmatically in MATLAB (exercise 1. f)

```
s = sim('cannon5','StopTime','25','MaxStep','0.01');
```

- MATLAB Function Block in Simulink



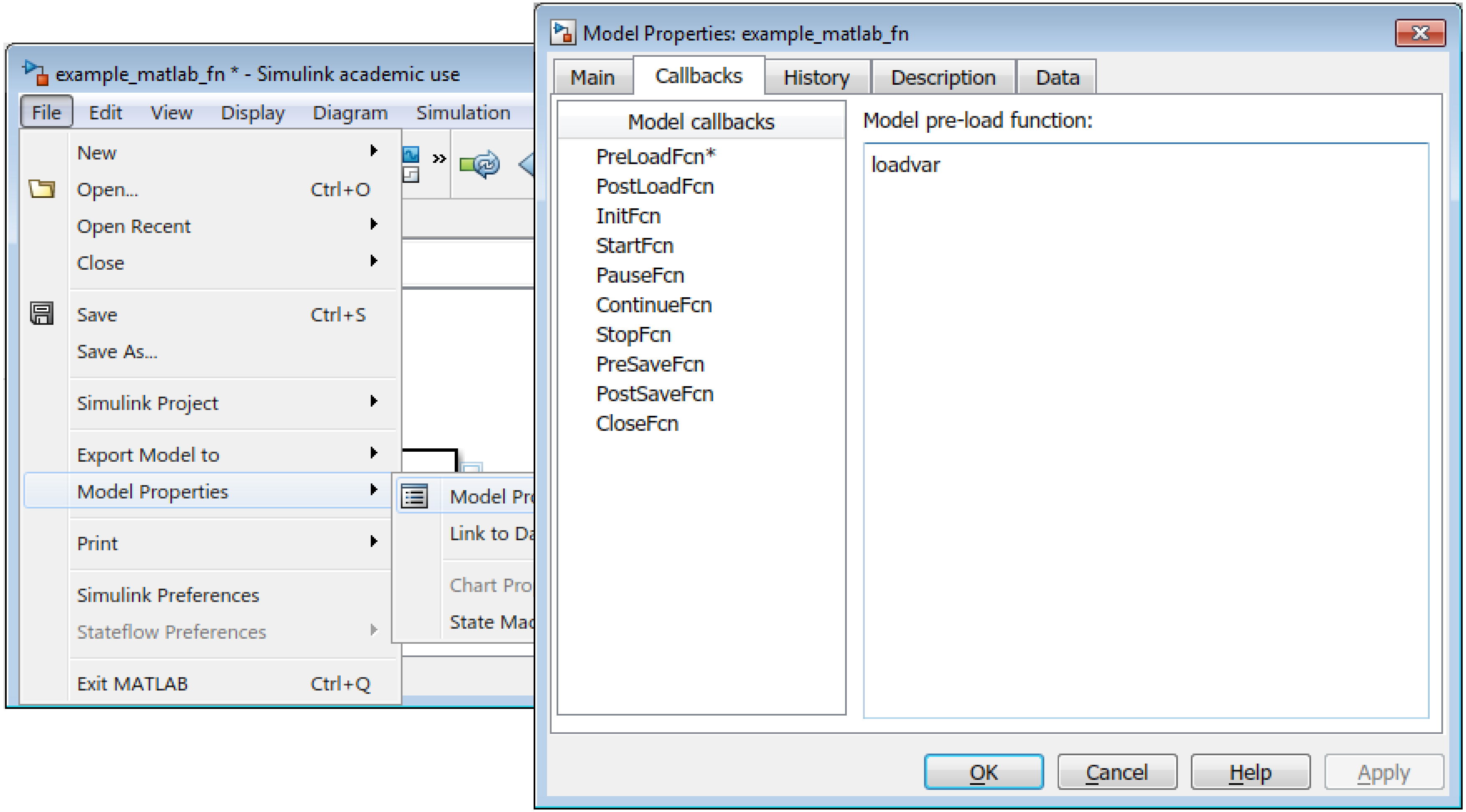
- Simulink Model Callbacks



Model pre-load function:

loadvar

MODEL CALLBACKS



Model pre-load function:

```
loadvar
```

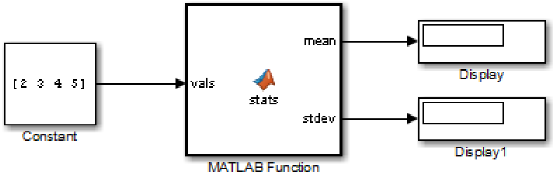
SIMULINK & MATLAB TOGETHER

There are many ways in which MATLAB and Simulink interact:

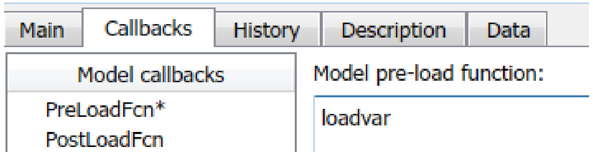
- Run Simulink models programmatically in MATLAB, exercise 1. f)

```
s = sim('cannon5','StopTime','25','MaxStep','0.01');
```

- MATLAB Function Block in Simulink



- Simulink Model Callbacks



Model callbacks

PreLoadFcn*

PostLoadFcn

Model pre-load function:

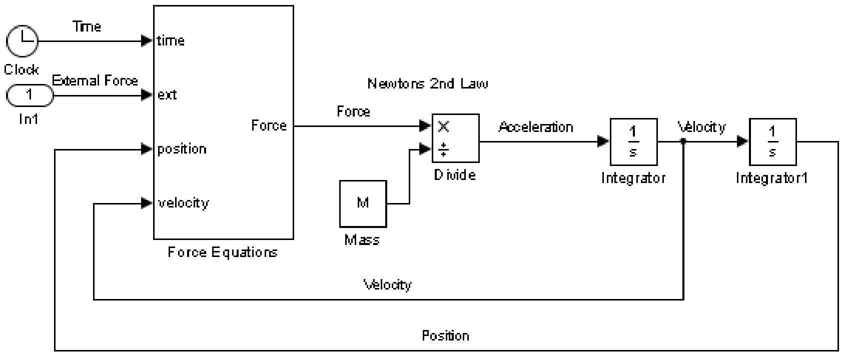
loadvar

- Create Simulink models programmatically

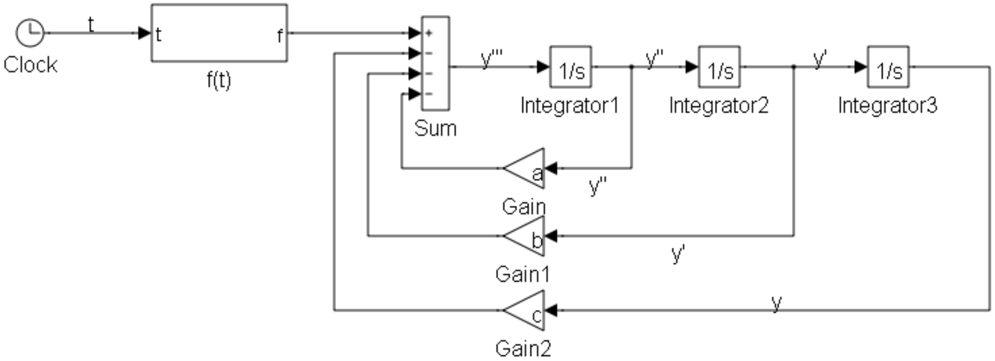
```
set_param('cannon5','PreLoadFcn','loadvar')
```

EXAMPLES IN NOTES

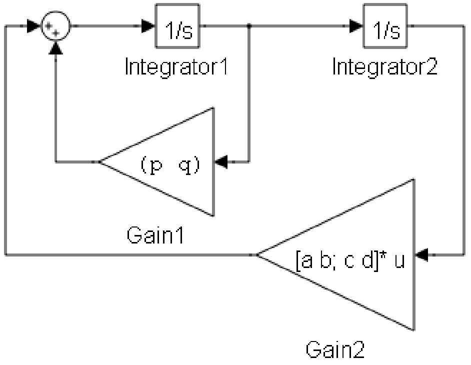
1: Dynamic Systems



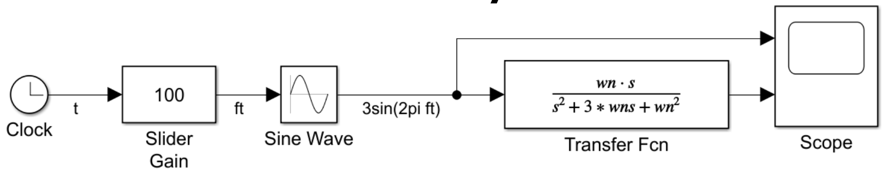
2: Ordinary Differential Equations (ODEs)



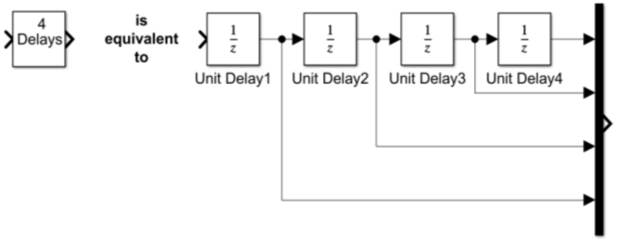
3: Simultaneous ODEs



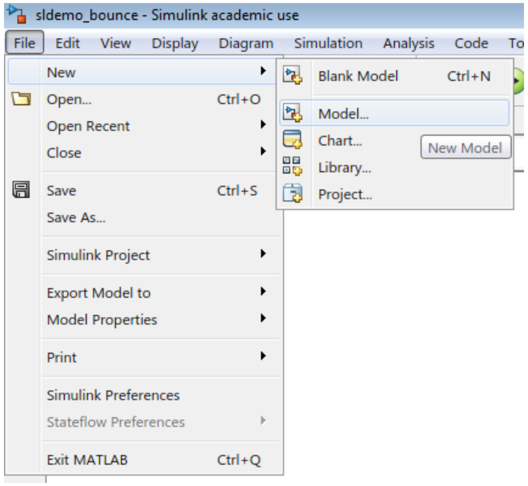
4: Linear Systems

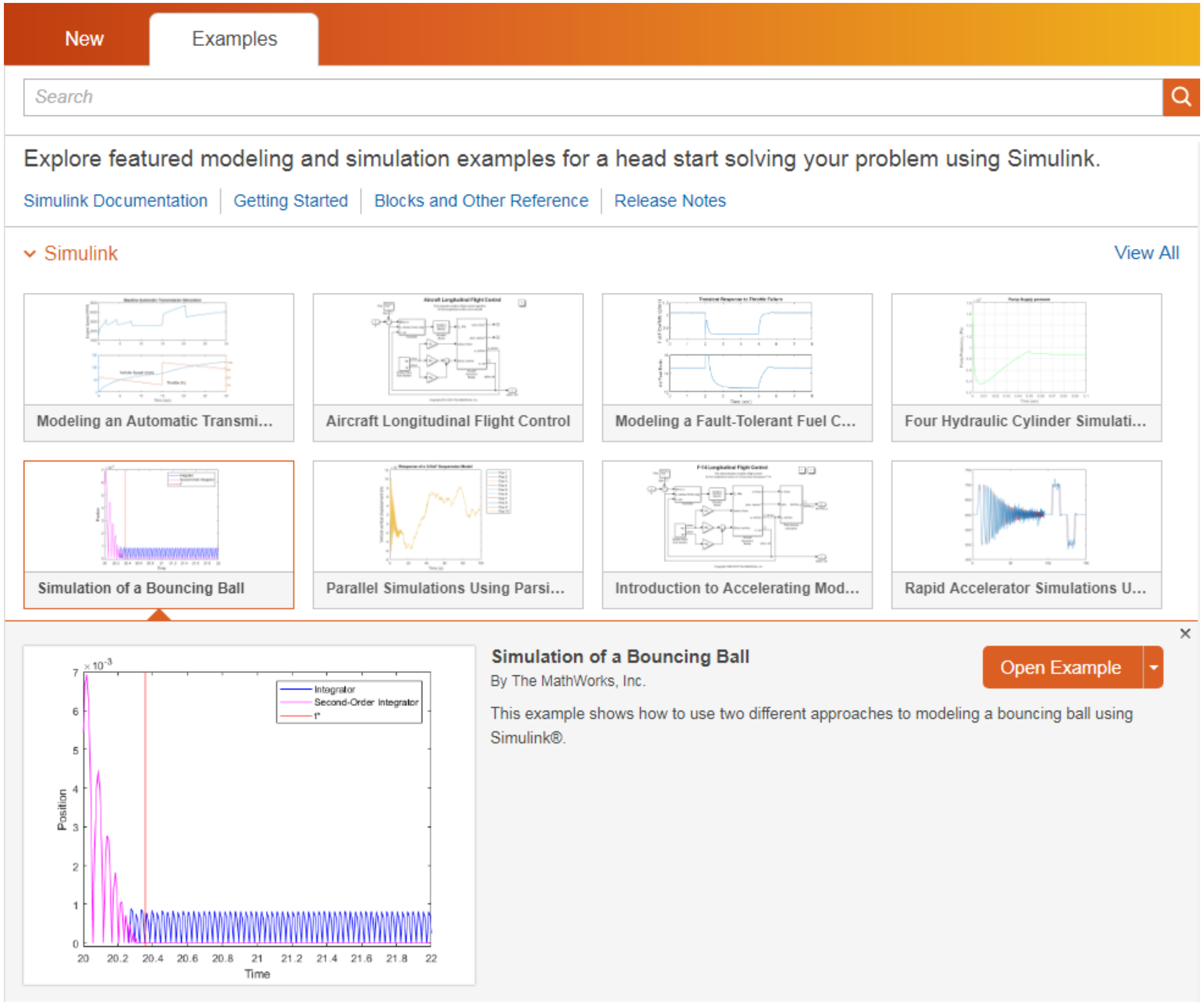


5: Discrete Systems



EXAMPLES





Explore featured modeling and simulation examples for a head start solving your problem using Simulink.

Simulink Documentation
Getting Started
Blocks and Other Reference
Release Notes
Simulink
View All
Modeling an Automatic Transmi...
Aircraft Longitudinal Flight Control
Modeling a Fault-Tolerant Fuel C...
Four Hydraulic Cylinder Simulati...
Simulation of a Bouncing Ball
Parallel Simulations Using Parsi...
Introduction to Accelerating Mod...
Rapid Accelerator Simulations U...
× 10^-3
Position
Time
Integrator
Second-Order Integrator
t*
Open Example

Simulation of a Bouncing Ball

By The MathWorks, Inc.

This example shows how to use two different approaches to modeling a bouncing ball using Simulink®.

GOOD CODING PRACTICE IN SIMULINK

✓ Keep your model **readable**

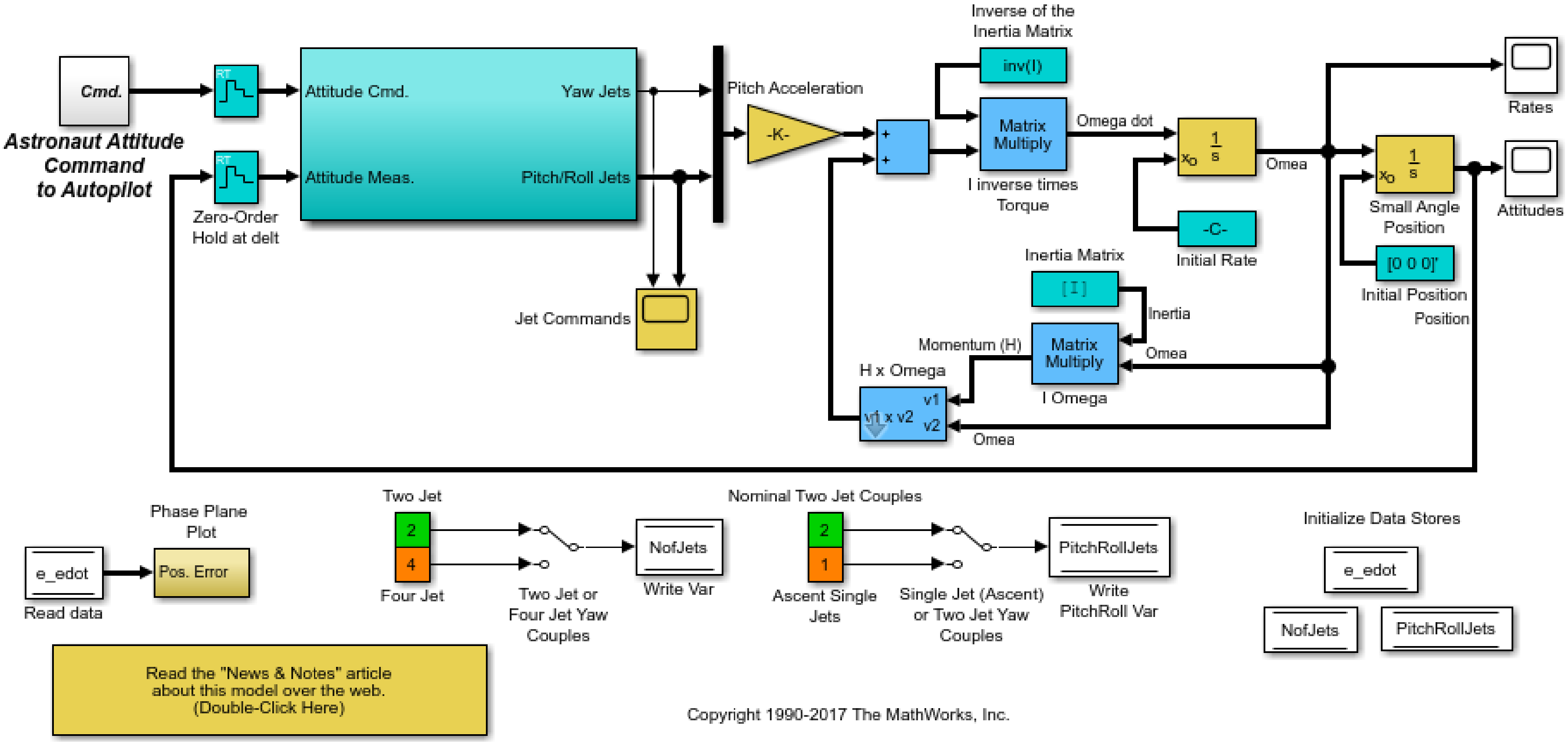
Make use of **annotations**, but keep them brief

Make use of the **colours** for different blocks

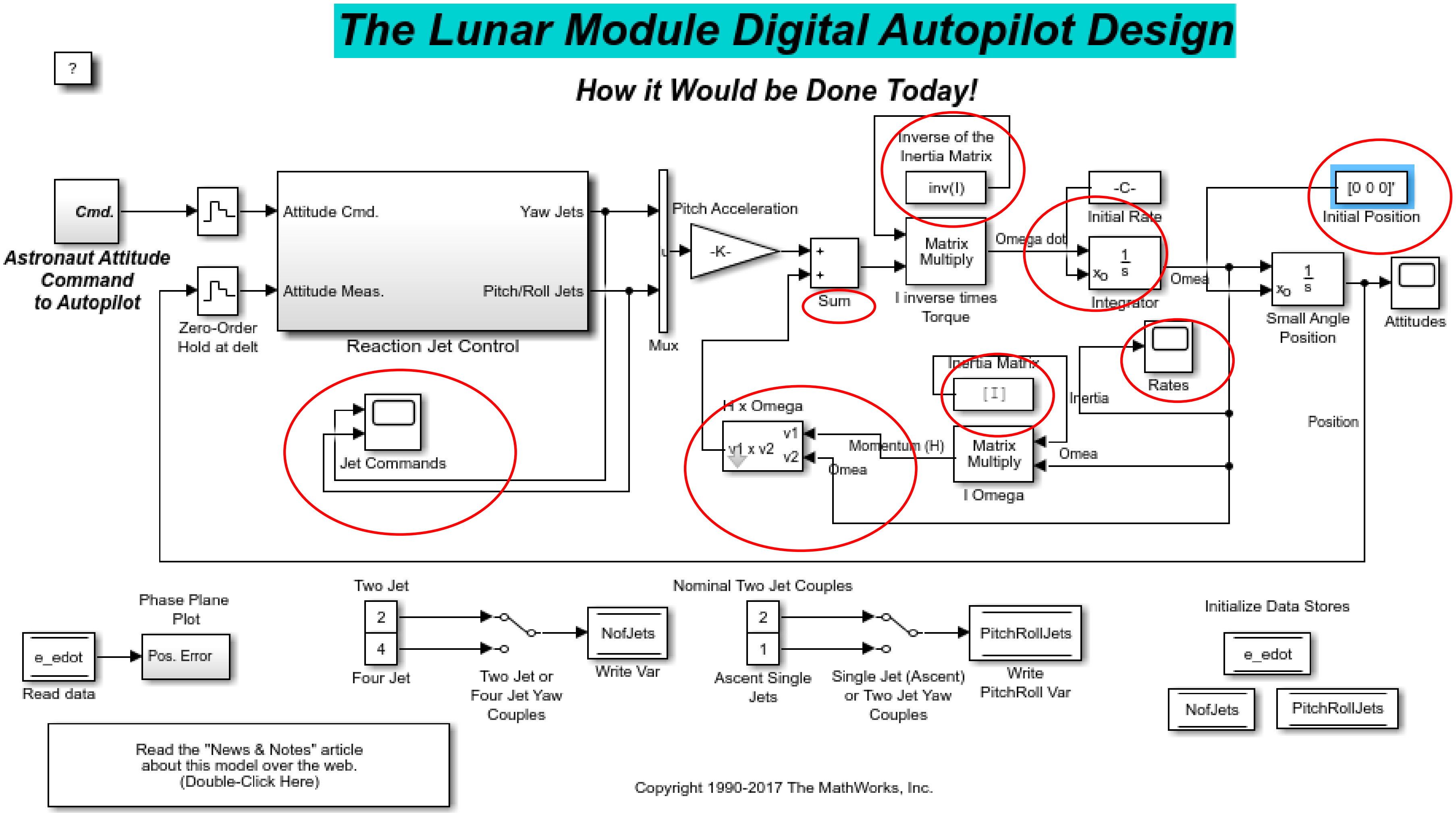
As your model evolves, don’t be afraid to **reorganise**

The Lunar Module Digital Autopilot Design

How it Would be Done Today!



aero_dap3dof



The Lunar Module Digital Autopilot Design

How it Would be Done Today!

Read the "News & Notes" article
about this model over the web.
(Double-Click Here)

aero_dap3dof

GOOD CODING PRACTICE IN SIMULINK

✓ Keep your model **readable**

- Make use of **annotations**, but keep them brief
- Make use of the **colours** for different blocks
- As your model evolves, don’t be afraid to **reorganise**

✓ Try **not to ‘bury’** key values in blocks (they are easily forgotten!)

- Combat this by adding automatic annotation to plots or blocks
- Set up key parameters in a MATLAB script

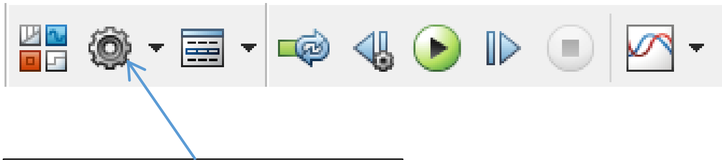
GOOD CODING PRACTICE IN SIMULINK

- ✓ Keep your model **readable**
  - Make use of **annotations**, but keep them brief
  - Make use of the **colours** for different blocks
  - As your model evolves, don’t be afraid to **reorganise**
- ✓ Try **not to ‘bury’** key values in blocks (they are easily forgotten!)
  - Combat this by adding automatic annotation to plots or blocks
  - Set up key parameters in a MATLAB script
- ✓ Use **tools when debugging**
  - Make use of the **Display** options to find dimension mismatch problems
  - Use **temporary scopes** or displays to investigate errors
- ✓ You can’t know everything Simulink does – **use documentation & google!**

THE SOLVER

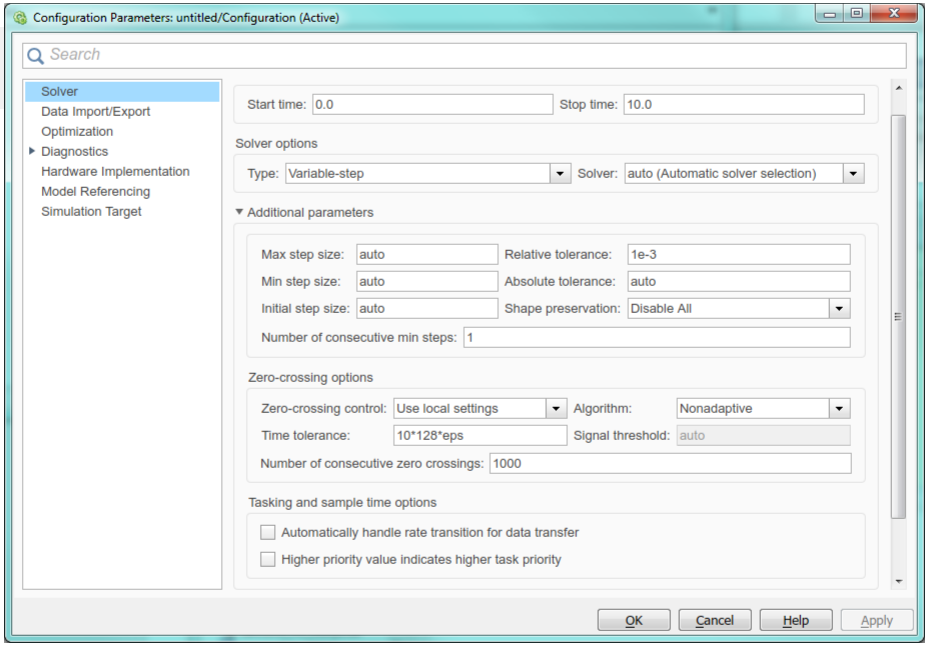
So, we’ve been modelling systems, that is solving equations.
Yet we haven’t had to code up any numerical analysis techniques.

The useful thing is that Simulink takes care of that for us.
But we still need to know what it is doing behind the scenes!



Model Settings

All of the settings related to how to numerically solve the equations of the model are found in “Model Configuration Parameters” Cog.



Configuration Parameters: untitled/Configuration (Active)

Start time: 0.0 Stop time: 10.0

Solver options

Type: Variable-step Solver: auto (Automatic solver selection)

Additional parameters

Max step size: auto Relative tolerance: 1e-3

Min step size: auto Absolute tolerance: auto

Initial step size: auto Shape preservation: Disable All

Number of consecutive min steps: 1

Zero-crossing options

Zero-crossing control: Use local settings Algorithm: Nonadaptive

Time tolerance: 10*128*eps Signal threshold: auto

Number of consecutive zero crossings: 1000

Tasking and sample time options

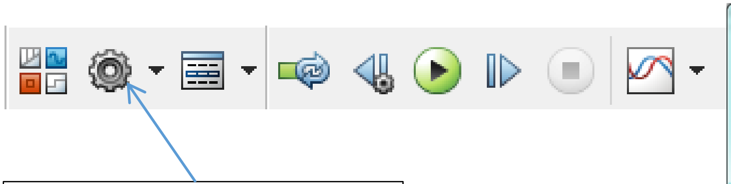
□ Automatically handle rate transition for data transfer

□ Higher priority value indicates higher task priority

THE SOLVER

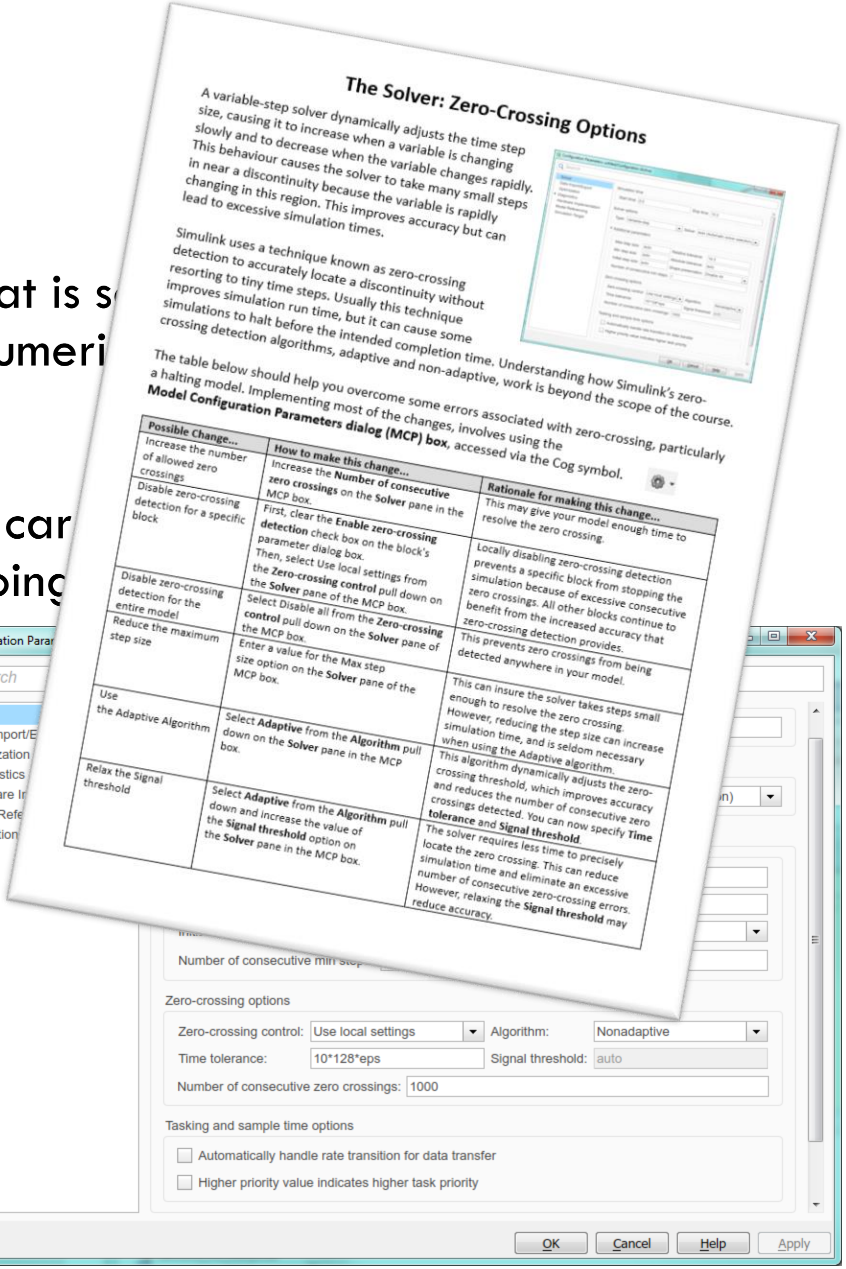
So, we've been modelling systems, that is s
Yet we haven't had to code up any numeri

The useful thing is that Simulink takes car
But we still need to know what it is doing



Model Settings

All of the settings related to how to numerically solve the equations of the model are found in "Model Configuration Parameters" Cog.



The Solver: Zero-Crossing Options

A variable-step solver dynamically adjusts the time step size, causing it to increase when a variable is changing slowly and to decrease when the variable changes rapidly. This behaviour causes the solver to take many small steps in near a discontinuity because the variable is rapidly changing in this region. This improves accuracy but can lead to excessive simulation times.

Simulink uses a technique known as zero-crossing detection to accurately locate a discontinuity without resorting to tiny time steps. Usually this technique improves simulation run time, but it can cause some simulations to halt before the intended completion time. Understanding how Simulink's zero-crossing detection algorithms, adaptive and non-adaptive, work is beyond the scope of the course.

The table below should help you overcome some errors associated with zero-crossing, particularly a halting model. Implementing most of the changes, involves using the **Model Configuration Parameters dialog (MCP) box**, accessed via the Cog symbol.

Possible Change...	How to make this change...	Rationale for making this change...
Increase the number of allowed zero crossings	Increase the **Number of consecutive zero crossings** on the **Solver** pane in the MCP box.	This may give your model enough time to resolve the zero crossing.
Disable zero-crossing detection for a specific block	First, clear the **Enable zero-crossing detection** check box on the block's parameter dialog box. Then, select Use local settings from the **Zero-crossing control** pull down on the **Solver** pane of the MCP box.	Locally disabling zero-crossing detection prevents a specific block from stopping the simulation because of excessive consecutive zero crossings. All other blocks continue to benefit from the increased accuracy that zero-crossing detection provides.
Disable zero-crossing detection for the entire model	Select Disable all from the **Zero-crossing control** pull down on the **Solver** pane of the MCP box.	This prevents zero crossings from being detected anywhere in your model.
Reduce the maximum step size	Enter a value for the Max step size option on the **Solver** pane of the MCP box.	This can insure the solver takes steps small enough to resolve the zero crossing. However, reducing the step size can increase simulation time, and is seldom necessary when using the Adaptive algorithm.
Use the Adaptive Algorithm	Select **Adaptive** from the **Algorithm** pull down on the **Solver** pane in the MCP box.	This algorithm dynamically adjusts the zero-crossing threshold, which improves accuracy and reduces the number of consecutive zero crossings detected. You can now specify **Time tolerance** and **Signal threshold**.
Relax the Signal threshold	Select **Adaptive** from the **Algorithm** pull down and increase the value of the **Signal threshold** option on the **Solver** pane in the MCP box.	The solver requires less time to precisely locate the zero crossing. This can reduce simulation time and eliminate an excessive number of consecutive zero-crossing errors. However, relaxing the **Signal threshold** may reduce accuracy.

Configuration Parar

Number of consecutive min step

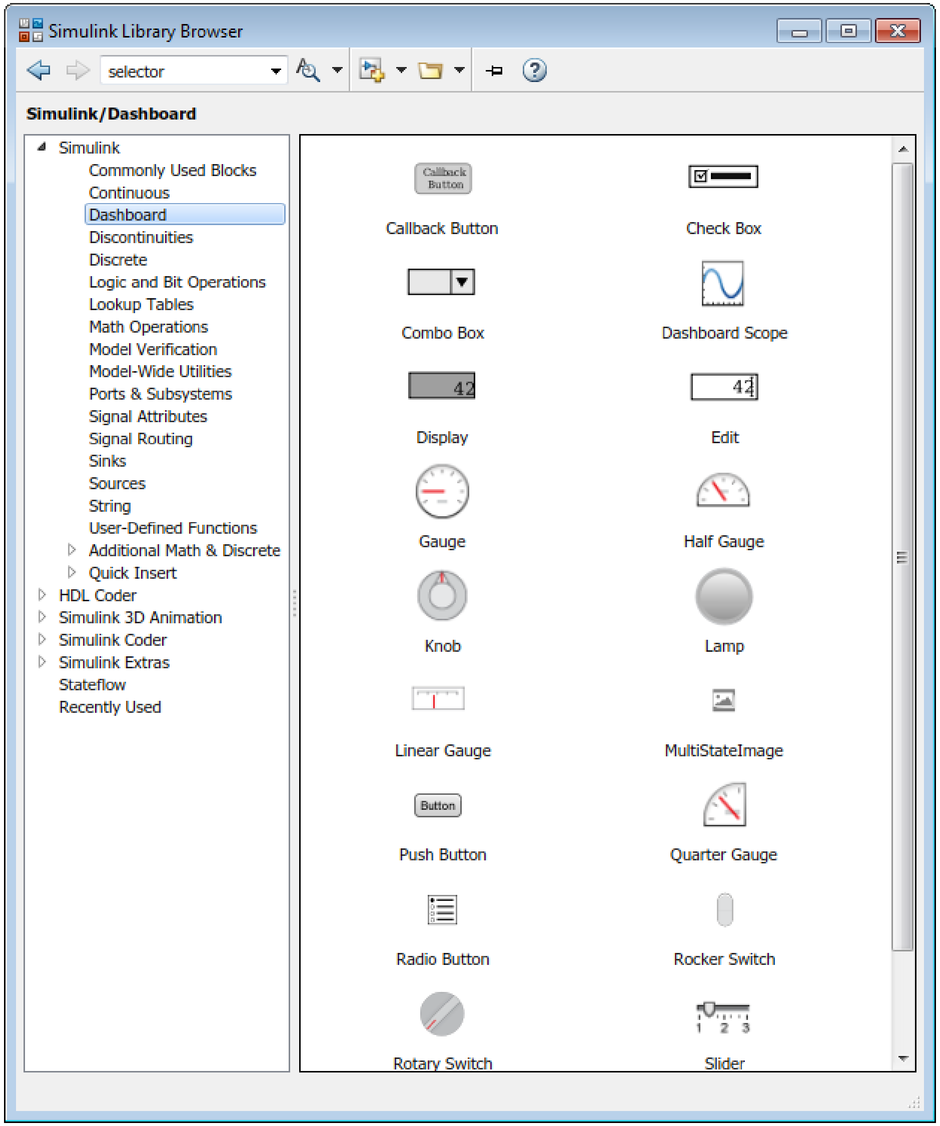
Zero-crossing options

Zero-crossing control: Use local settings
Algorithm: Nonadaptive
Time tolerance: 10*128*eps
Signal threshold: auto
Number of consecutive zero crossings: 1000

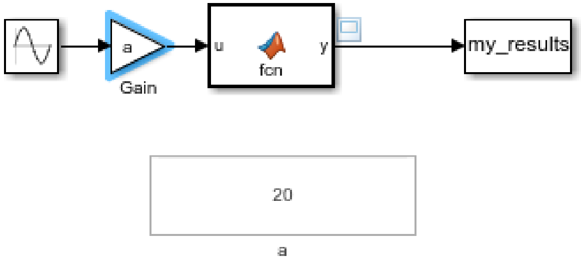
Tasking and sample time options

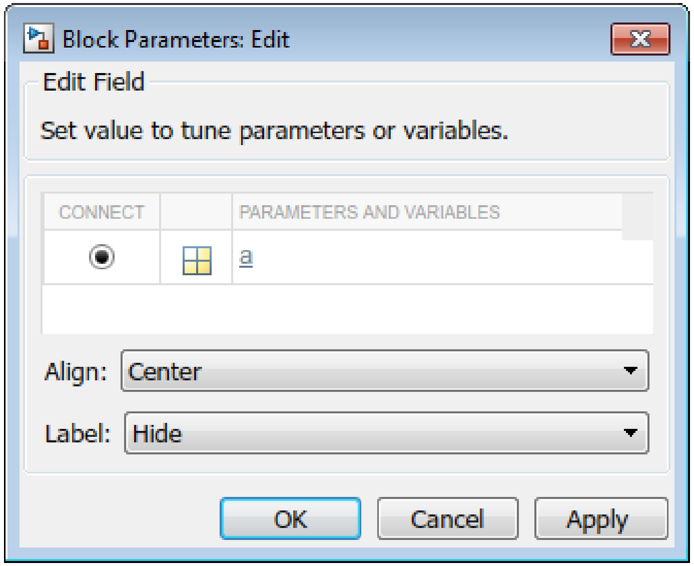
☐ Automatically handle rate transition for data transfer
☐ Higher priority value indicates higher task priority

OTHER FEATURES - DASHBOARD



Control and visualize your Simulink models during simulation and while paused.





Edit Field

Set value to tune parameters or variables.

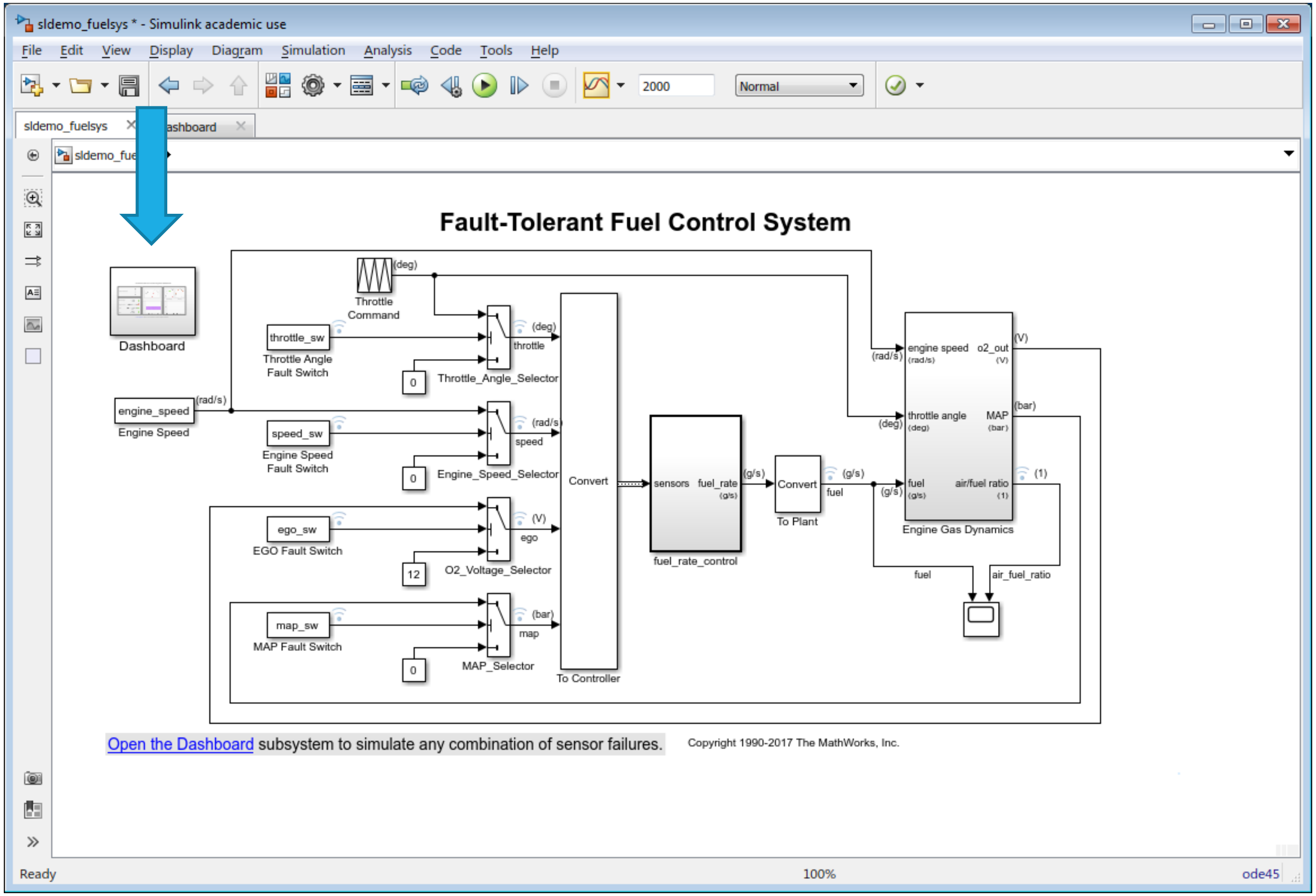
CONNECT		PARAMETERS AND VARIABLES
●	⊞	a

Align: Center

Label: Hide

OTHER FEATURES - DASHBOARD EXAMPLE

sldemo_fuelsys



Open the Dashboard subsystem to simulate any combination of sensor failures.

Copyright 1990-2017 The MathWorks, Inc.

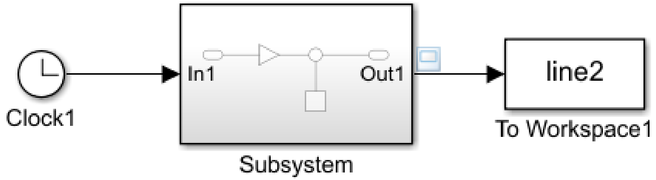
```
open_system([matlabroot '\toolbox\simulink\simdemos\automotive\fuelsys\sldemo_fuelsys'])
```

OTHER FEATURES - MASKS

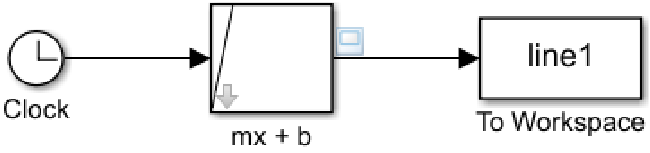
A **mask** is created from a subsystem. A **masked subsystem:**

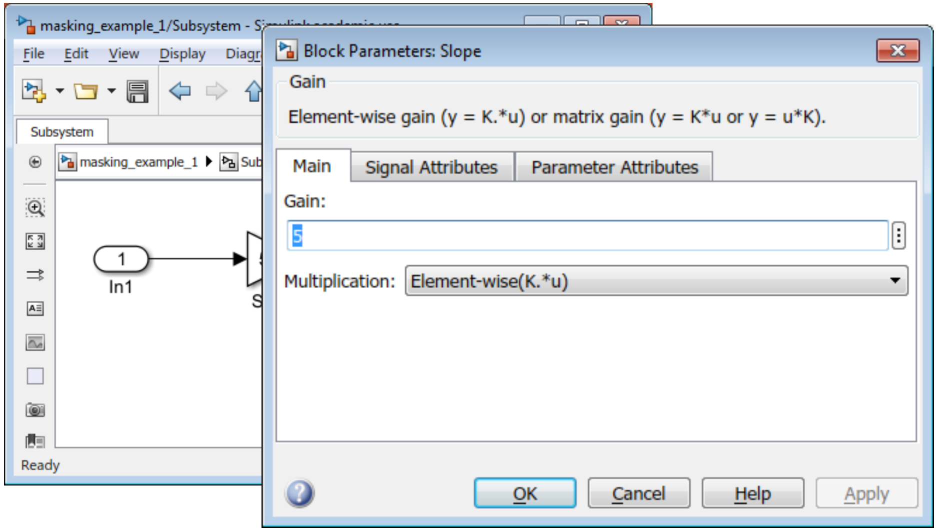
- Offers a custom interface for that subsystem
- Hides the content
- Makes system appear as a "built in" block (custom icon & parameter dialog box.)

Subsystem



Mask





Block Parameters: Slope

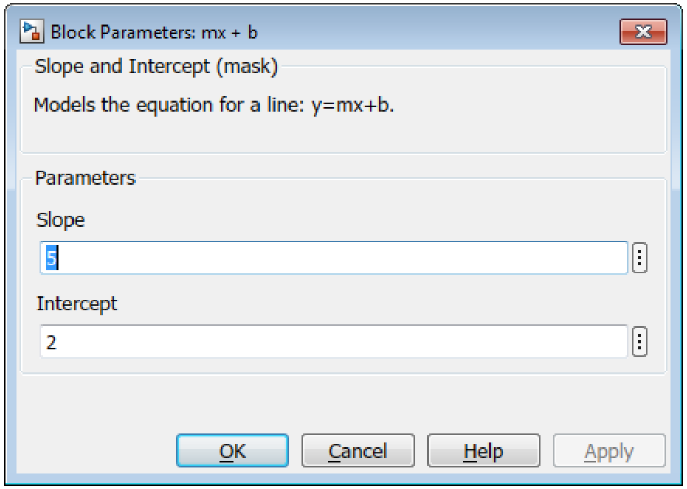
Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Main | Signal Attributes | Parameter Attributes

Gain: 5

Multiplication: Element-wise(K.*u)



Block Parameters: mx + b

Slope and Intercept (mask)

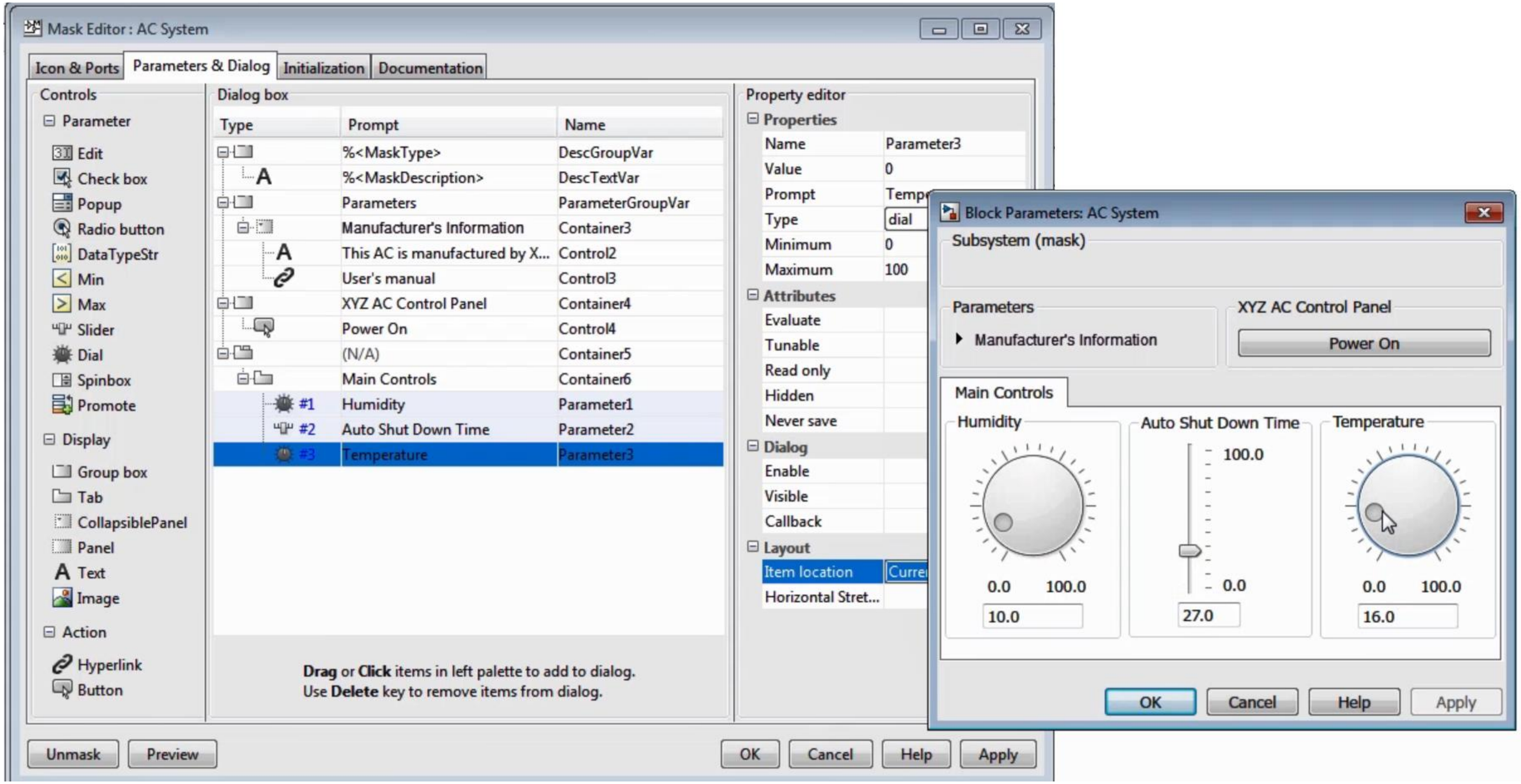
Models the equation for a line: y=mx+b.

Parameters

Slope: 5

Intercept: 2

OTHER FEATURES – MASKS WITH DASHBOARD



Type	Prompt	Name
	%<MaskType>	DescGroupVar
A	%<MaskDescription>	DescTextVar
	Parameters	ParameterGroupVar
	Manufacturer's Information	Container3
A	This AC is manufactured by X...	Control2
	User's manual	Control3
	XYZ AC Control Panel	Container4
	Power On	Control4
	(N/A)	Container5
	Main Controls	Container6
#1	Humidity	Parameter1
#2	Auto Shut Down Time	Parameter2
#3	Temperature	Parameter3

Drag or Click items in left palette to add to dialog.
Use Delete key to remove items from dialog.

Properties	
Name	Parameter3
Value	0
Prompt	Tempe
Type	dial
Minimum	0
Maximum	100
Attributes	
Evaluate	
Tunable	
Read only	
Hidden	
Never save	
Dialog	
Enable	
Visible	
Callback	
Layout	
Item location	Curre
Horizontal Stret...	

Subsystem (mask)

Parameters
▸ Manufacturer's Information

XYZ AC Control Panel
Power On

Main Controls

Humidity: 0.0 100.0 — 10.0

Auto Shut Down Time: 100.0 0.0 — 27.0

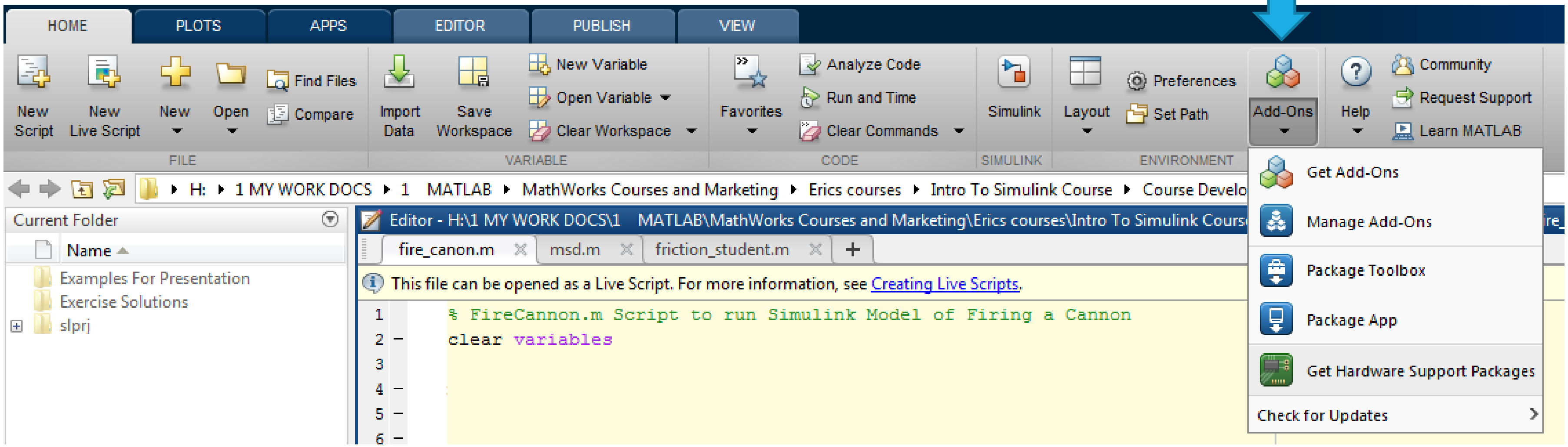
Temperature: 0.0 100.0 — 16.0

For intro to masks, see this video. For how to create a mask using the Mask Editor, see this video.

OTHER FEATURES – HARDWARE SUPPORT

Simulink can be used to interface with lots of different hardware.
It is used to develop algorithms to run standalone on devices.

Devices use custom blocks gained through **Hardware Support Packages:**

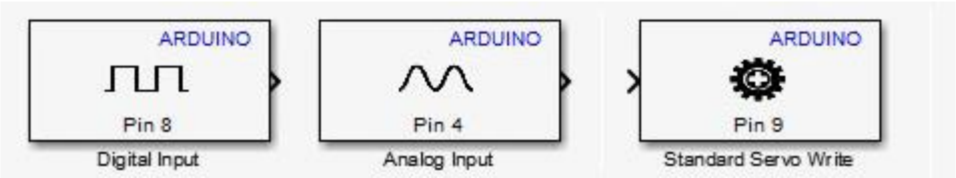


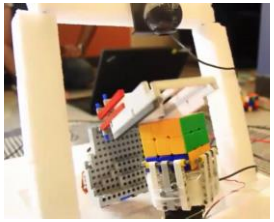
This file can be opened as a Live Script. For more information, see Creating Live Scripts.

```
% FireCannon.m Script to run Simulink Model of Firing a Cannon
clear variables
```

OTHER FEATURES – HARDWARE SUPPORT

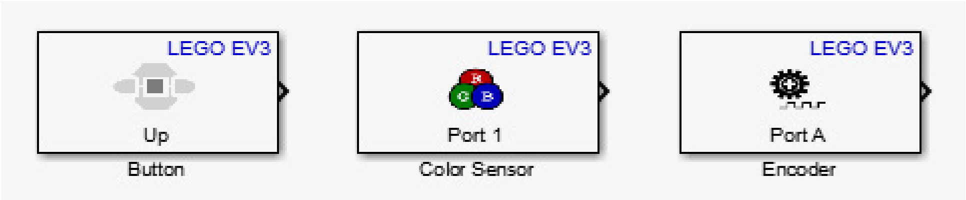
Example Arduino blocks:

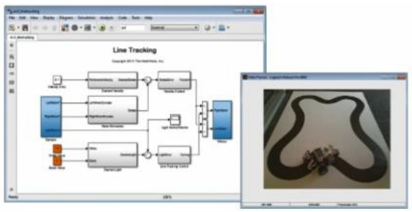




Rubik's Cube Solver

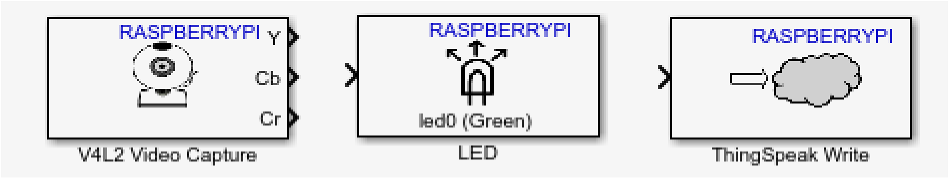
Example EV3 blocks:

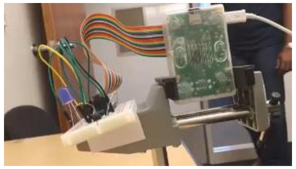




Line Following Robot

Example Raspberry Pi blocks:



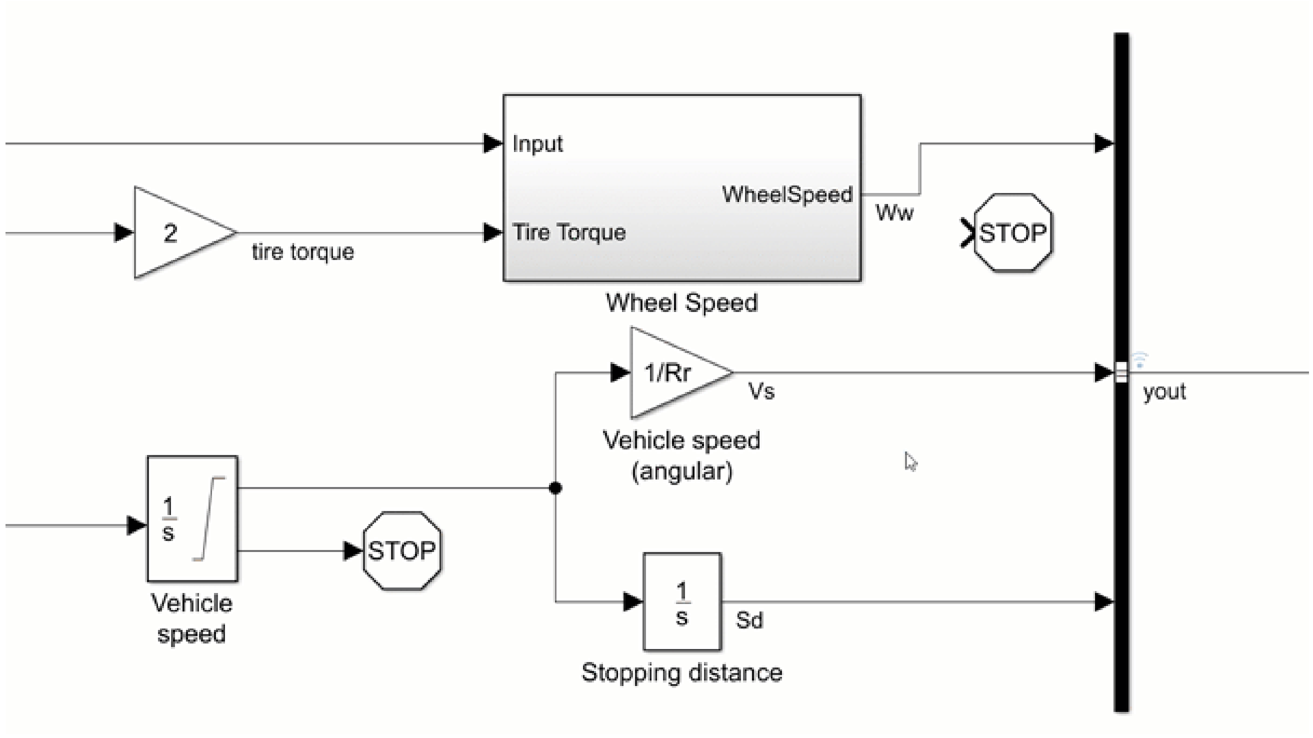


Tweeting Security System

SIMULINK VERSIONS

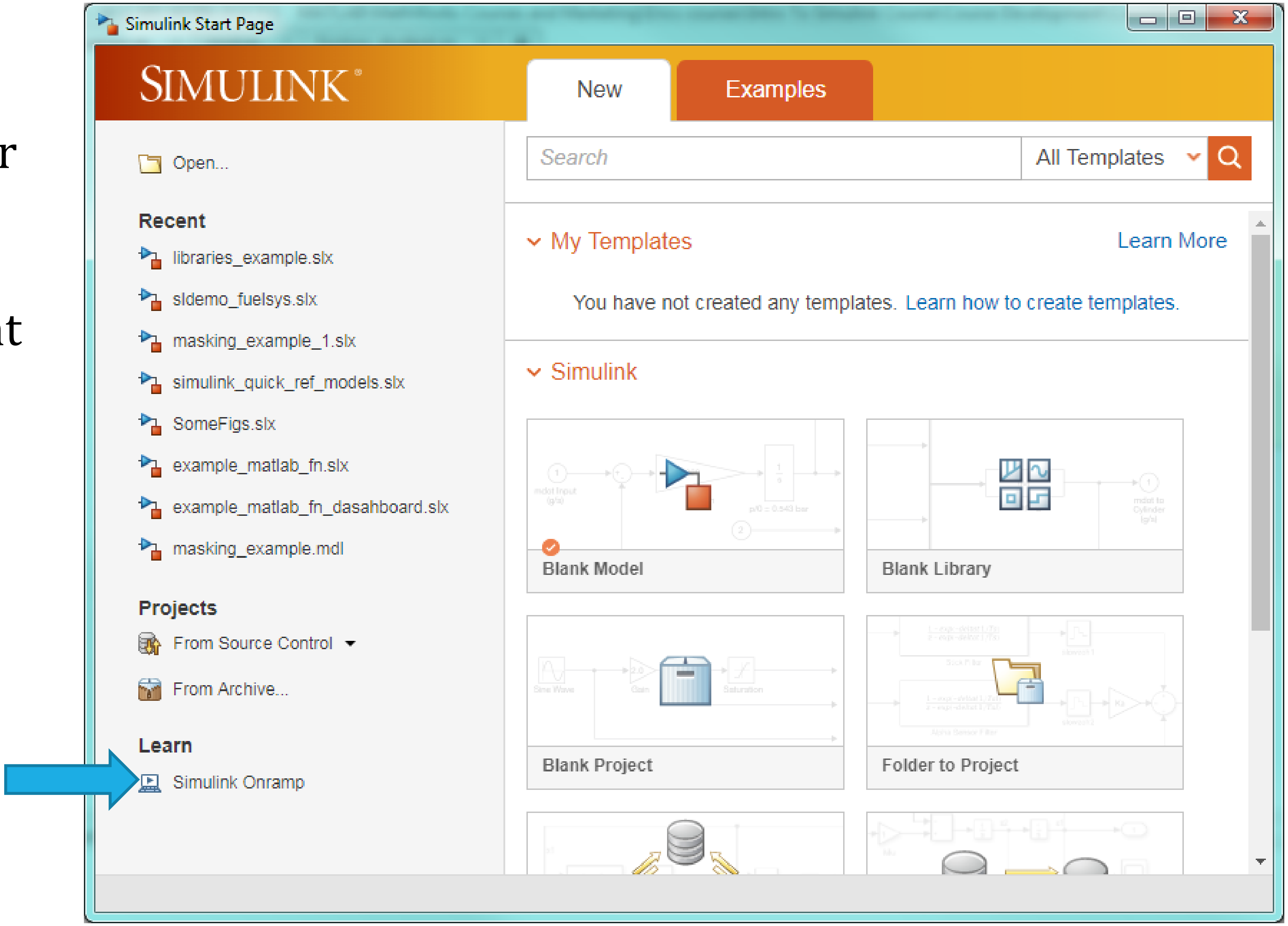
Simulink models created in newer versions must be exported before use in older versions.

From 2018b major improvements:
Automatic Port Creation, Edit on Block Icon **& Simulink On Ramp**



SIMULINK ONRAMP

- Download for 2018b
- Free
- Great Content
- ~3 hours



Simulink Start Page

SIMULINK

New | Examples

Open...

Recent

- libraries_example.slx
- sldemo_fuelsys.slx
- masking_example_1.slx
- simulink_quick_ref_models.slx
- SomeFigs.slx
- example_matlab_fn.slx
- example_matlab_fn_dasahboard.slx
- masking_example.mdl

Projects

- From Source Control
- From Archive...

Learn

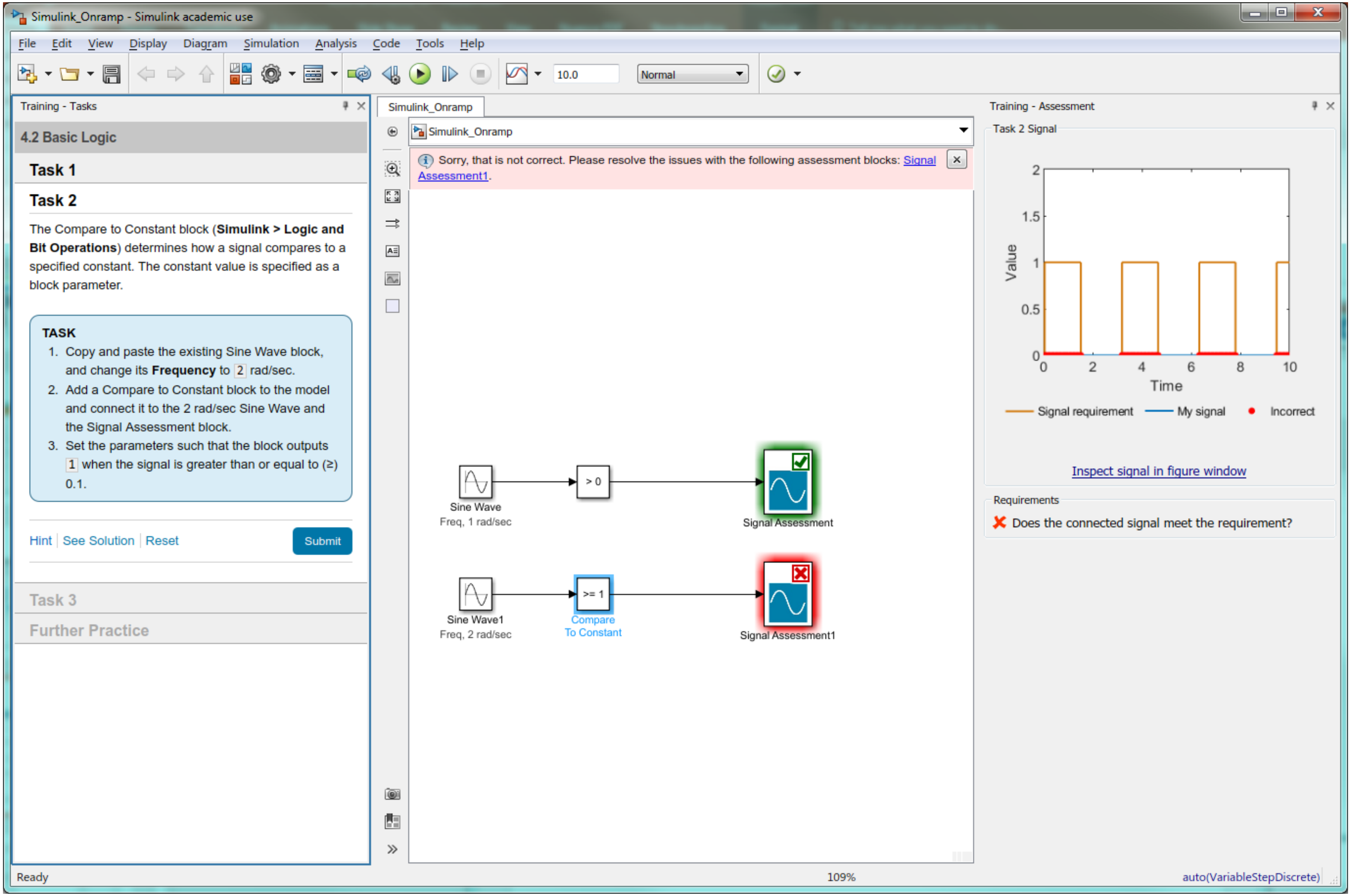
- Simulink Onramp

Search | All Templates

My Templates — Learn More

You have not created any templates. Learn how to create templates.

Simulink



4.2 Basic Logic

Task 1

Task 2

The Compare to Constant block (**Simulink > Logic and Bit Operations**) determines how a signal compares to a specified constant. The constant value is specified as a block parameter.

TASK

1. Copy and paste the existing Sine Wave block, and change its **Frequency** to 2 rad/sec.
2. Add a Compare to Constant block to the model and connect it to the 2 rad/sec Sine Wave and the Signal Assessment block.
3. Set the parameters such that the block outputs 1 when the signal is greater than or equal to (≥) 0.1.

Hint | See Solution | Reset   Submit

Task 3

Further Practice

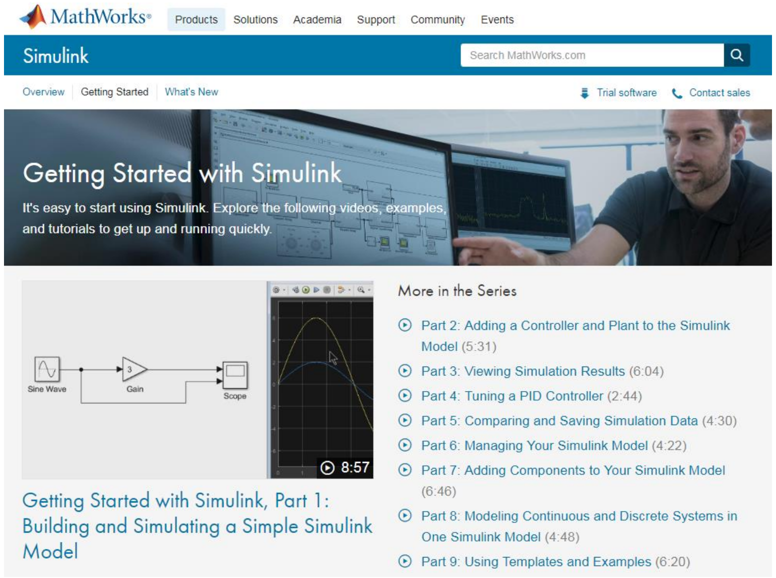
Sorry, that is not correct. Please resolve the issues with the following assessment blocks: Signal Assessment1.

Requirements

✗ Does the connected signal meet the requirement?

WHERE NEXT?

- Try Exercise 2 or 3
- Use the Examples on launch
- Explore the features mentioned in this talk
- Ask for support installing MATLAB
- Visit the Simulink Getting Started webpage for videos
- If you have your own laptop, try Simulink Onramp



Getting Started with Simulink

It's easy to start using Simulink. Explore the following videos, examples, and tutorials to get up and running quickly.

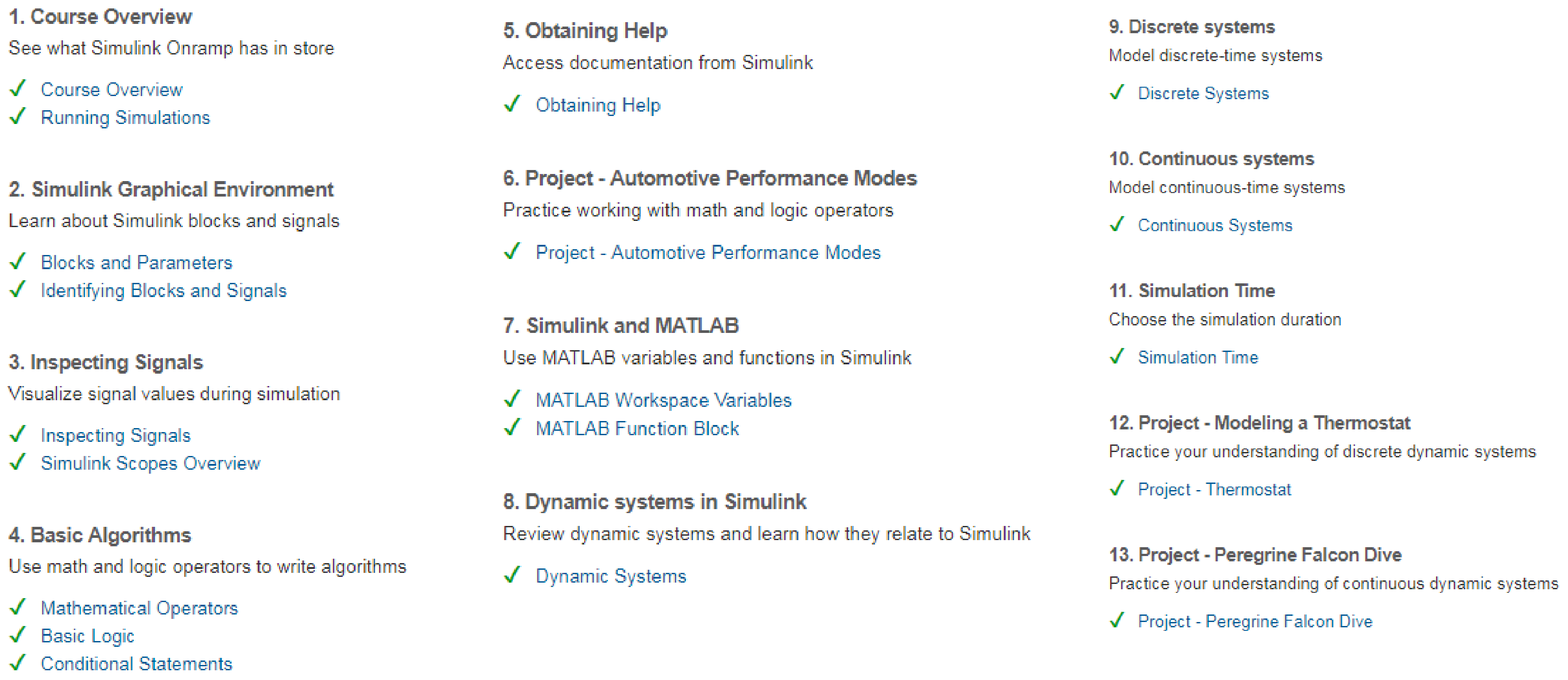
Getting Started with Simulink, Part 1: Building and Simulating a Simple Simulink Model

More in the Series

- Part 2: Adding a Controller and Plant to the Simulink Model (5:31)
- Part 3: Viewing Simulation Results (6:04)
- Part 4: Tuning a PID Controller (2:44)
- Part 5: Comparing and Saving Simulation Data (4:30)
- Part 6: Managing Your Simulink Model (4:22)
- Part 7: Adding Components to Your Simulink Model (6:46)
- Part 8: Modeling Continuous and Discrete Systems in One Simulink Model (4:48)
- Part 9: Using Templates and Examples (6:20)

Visit bit.ly/simulinkfeedback to answer a few short questions to let us know what you thought of the course ☺

ON RAMP COURSE CONTENT



1. Course Overview

See what Simulink Onramp has in store

- ✓ Course Overview
- ✓ Running Simulations

2. Simulink Graphical Environment

Learn about Simulink blocks and signals

- ✓ Blocks and Parameters
- ✓ Identifying Blocks and Signals

3. Inspecting Signals

Visualize signal values during simulation

- ✓ Inspecting Signals
- ✓ Simulink Scopes Overview

4. Basic Algorithms

Use math and logic operators to write algorithms

- ✓ Mathematical Operators
- ✓ Basic Logic
- ✓ Conditional Statements

5. Obtaining Help

Access documentation from Simulink

- ✓ Obtaining Help

6. Project - Automotive Performance Modes

Practice working with math and logic operators

- ✓ Project - Automotive Performance Modes

7. Simulink and MATLAB

Use MATLAB variables and functions in Simulink

- ✓ MATLAB Workspace Variables
- ✓ MATLAB Function Block

8. Dynamic systems in Simulink

Review dynamic systems and learn how they relate to Simulink

- ✓ Dynamic Systems

9. Discrete systems

Model discrete-time systems

- ✓ Discrete Systems

10. Continuous systems

Model continuous-time systems

- ✓ Continuous Systems

11. Simulation Time

Choose the simulation duration

- ✓ Simulation Time

12. Project - Modeling a Thermostat

Practice your understanding of discrete dynamic systems

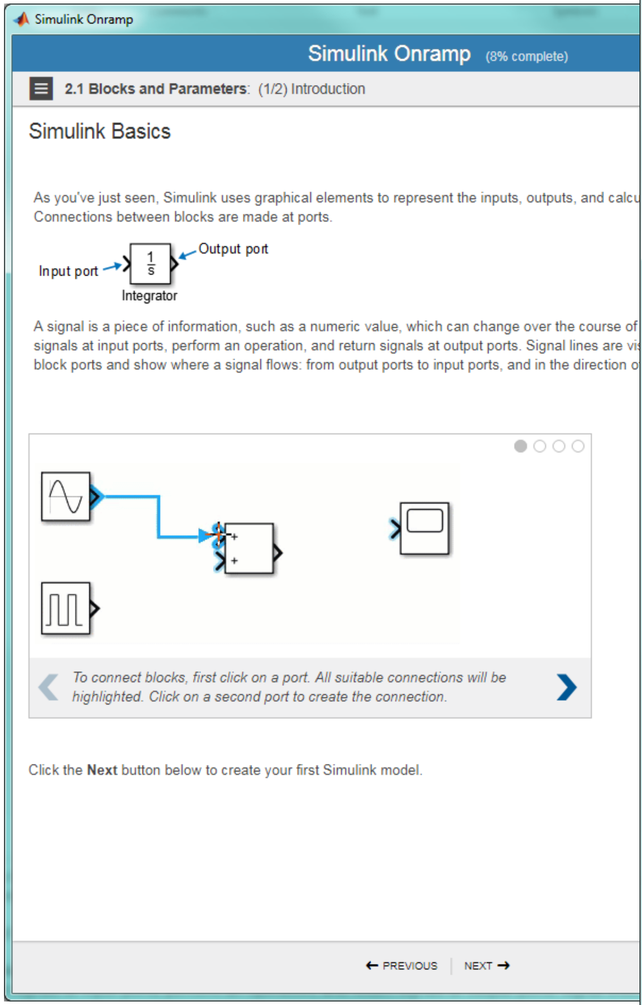
- ✓ Project - Thermostat

13. Project - Peregrine Falcon Dive

Practice your understanding of continuous dynamic systems

- ✓ Project - Peregrine Falcon Dive

ONRAMP TEXT & QUIZZES



2.1 Blocks and Parameters: (1/2) Introduction

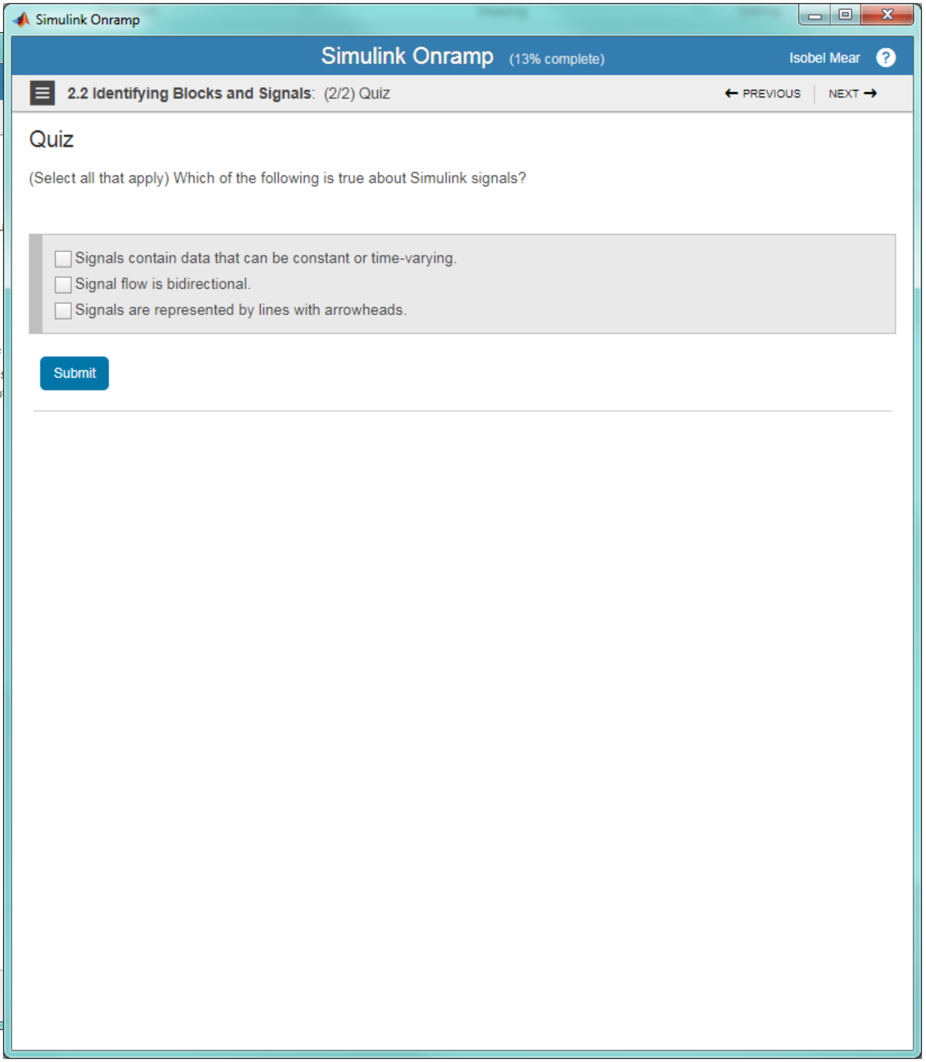
Simulink Basics

As you've just seen, Simulink uses graphical elements to represent the inputs, outputs, and calcu
Connections between blocks are made at ports.

A signal is a piece of information, such as a numeric value, which can change over the course of
signals at input ports, perform an operation, and return signals at output ports. Signal lines are vi
block ports and show where a signal flows: from output ports to input ports, and in the direction o

To connect blocks, first click on a port. All suitable connections will be highlighted. Click on a second port to create the connection.

Click the **Next** button below to create your first Simulink model.



2.2 Identifying Blocks and Signals: (2/2) Quiz

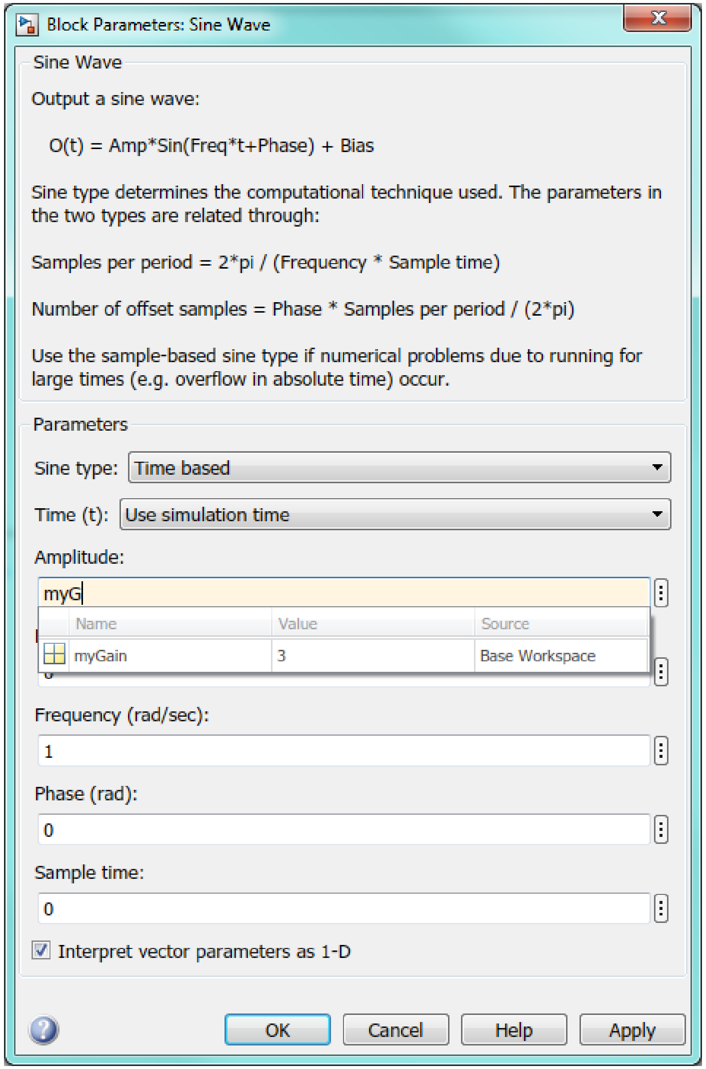
Quiz

(Select all that apply) Which of the following is true about Simulink signals?

- □ Signals contain data that can be constant or time-varying.
- □ Signal flow is bidirectional.
- □ Signals are represented by lines with arrowheads.

Submit

2018B – LINKING VARIABLE NAMES



Sine Wave

Output a sine wave:

O(t) = Amp*Sin(Freq*t+Phase) + Bias

Sine type determines the computational technique used. The parameters in the two types are related through:

Samples per period = 2*pi / (Frequency * Sample time)

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude:

myG

Name	Value	Source
myGain	3	Base Workspace

Frequency (rad/sec):

1

Phase (rad):

0

Sample time:

0

☑ Interpret vector parameters as 1-D

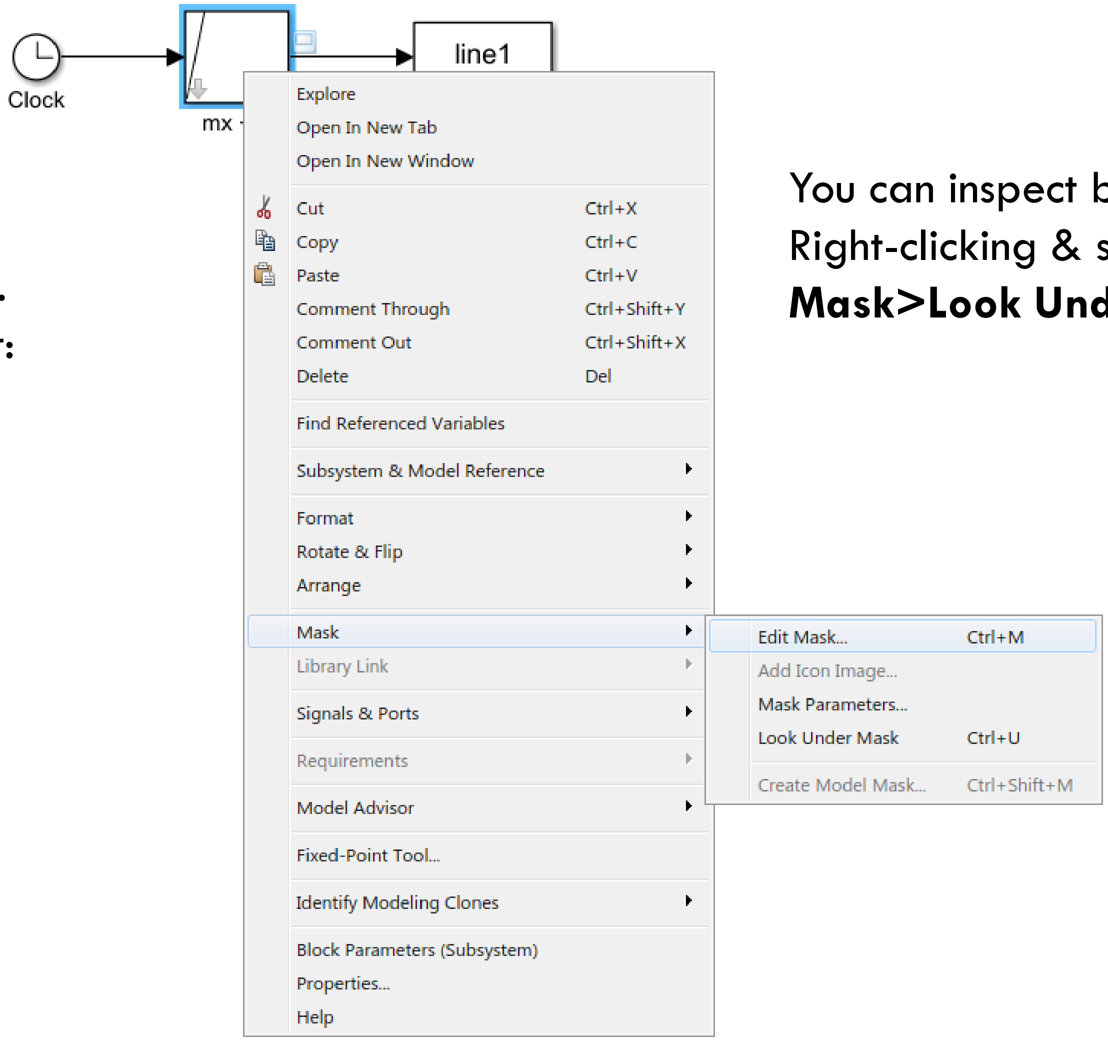
WHERE IS MATLAB/SIMULINK USED?

Examples of companies using MATLAB & SIMULINK

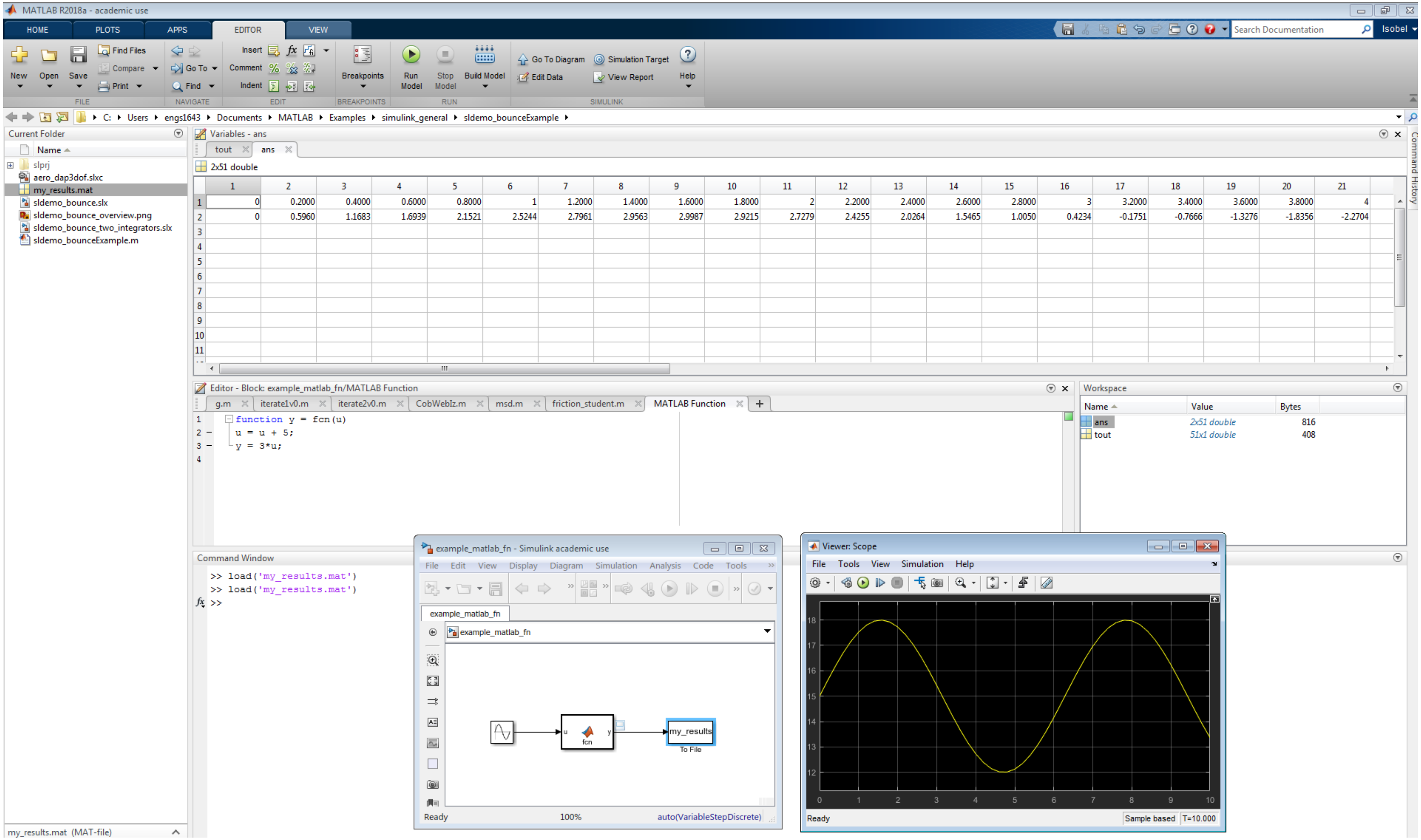
ROLLS ROYCE
BT
TESCO
BANK OF ENGLAND
BAE SYSTEMS
KOREAN AIR
ARM
FUJITSU
MBDA MISSILE SYSTEMS
ALSTOM
RaspberryPi
hp
NASA
JAGUAR
SIEMENS
dyson
LAND ROVER
[dstl]
AIRBUS
AstraZeneca
NISSAN
mazda
Honeywell
THALES
John Lewis
vodafone
bp
gsk GlaxoSmithKline
BOEING
Deutsche Bank
HSBC
SAMSUNG
REALTEK

OTHER FEATURES - MASKS

Create a mask by selecting a subsystem. Right-clicking & select: **Mask>Create Mask**



You can inspect by Right-clicking & select: **Mask>Look Under Mask**



2x51 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	0.2000	0.4000	0.6000	0.8000	1	1.2000	1.4000	1.6000	1.8000	2	2.2000	2.4000	2.6000	2.8000	3	3.2000	3.4000	3.6000	3.8000	4
2	0	0.5960	1.1683	1.6939	2.1521	2.5244	2.7961	2.9563	2.9987	2.9215	2.7279	2.4255	2.0264	1.5465	1.0050	0.4234	-0.1751	-0.7666	-1.3276	-1.8356	-2.2704
3																					
4																					
5																					
6																					
7																					
8																					
9																					
10																					
11																					

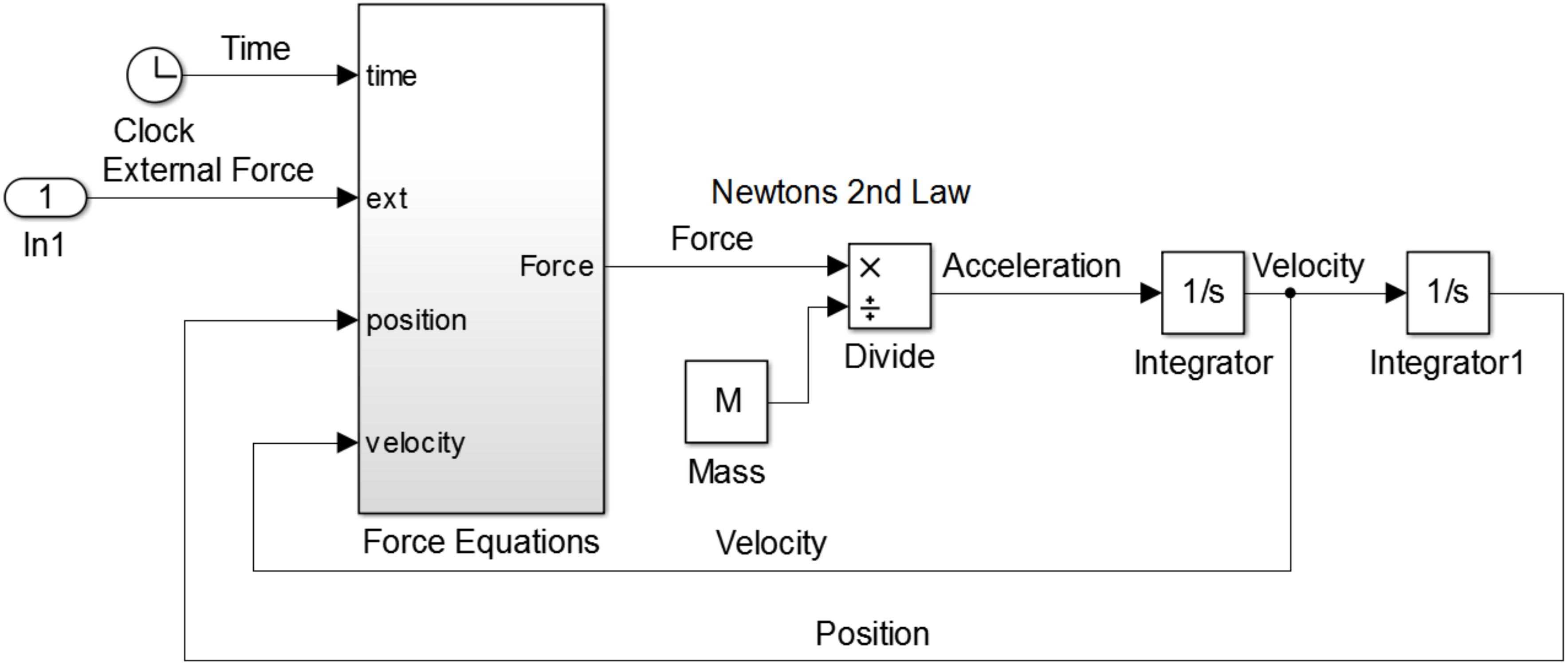
```
function y = fcn(u)
u = u + 5;
y = 3*u;
```

Name	Value	Bytes
ans	2x51 double	816
tout	51x1 double	408

```
>> load('my_results.mat')
>> load('my_results.mat')
fx >>
```

An Introduction to Using Simulink

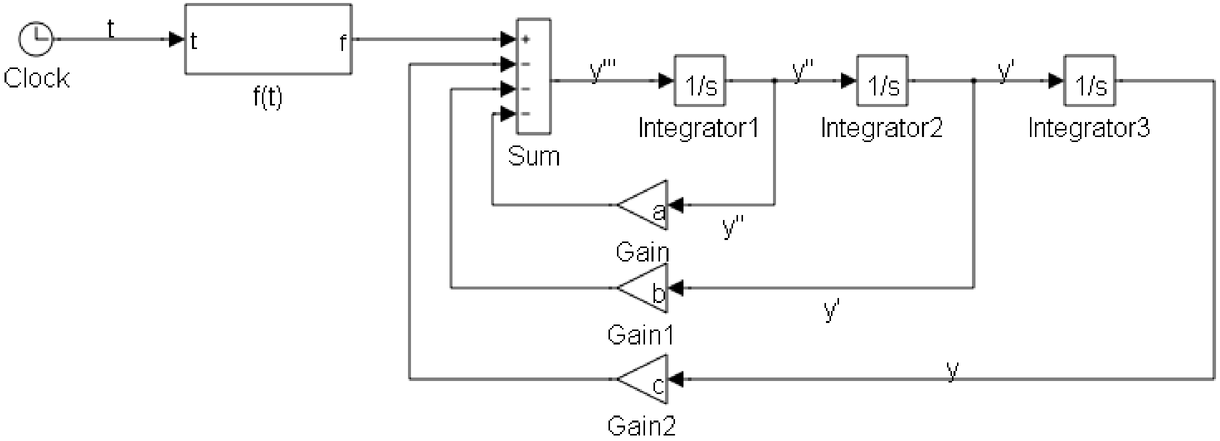
Dynamic System



Ordinary Differential Equations

$$\dddot{y} + a\ddot{y} + b\dot{y} + cy = f(t)$$

$$\dddot{y} = f(t) - a\ddot{y} - b\dot{y} - cy$$



Simultaneous ODE

$$\dot{x} = 1 - 0.2x - y$$

$$\dot{y} = x$$

$$x(0) = y(0) = 0$$

Simultaneous ODE

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -0.2 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

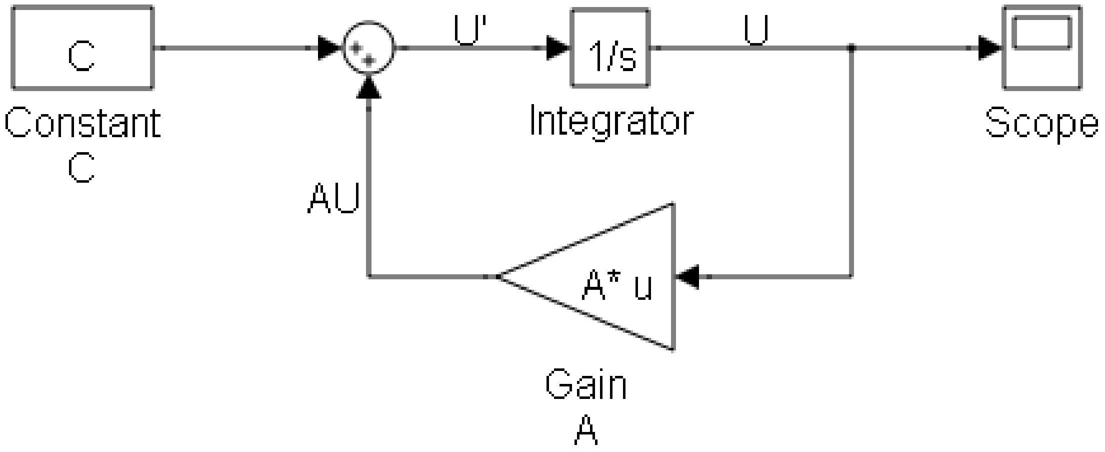
$$x(0) = y(0) = 0$$

Simultaneous ODE

$$\dot{U} = C + AU$$

$$U = \begin{bmatrix} x \\ y \end{bmatrix} \quad C = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathrm{A} = \begin{bmatrix} -0.2 & -1 \\ 1 & 0 \end{bmatrix}$$

$$\dot{U} = C + AU$$


C
Constant
C
U'
1/s
Integrator
U
Scope
AU
A* u
Gain
A