# Simscape™
## User's Guide

# MATLAB&SIMULINK®

**R**2022**a**

MathWorks®

# How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

**Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

**Patents**

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

# **Contents**

# **Model Construction**

## 1

# Isothermal Liquid Models

## 2

# Thermal Liquid Models

## 3

# Two-Phase Fluid Models

# 4

# Gas System Models

# 5

**6**

**7**

# Variable Initialization and Operating Points

**8**

# Linearization and Trimming

**9**

# Simscape Run-Time Parameters

## 10

# Code Generation

## 12

# Data Logging

## 13

# Model Statistics

## 14

# Physical Units

# 15

# Scalable Compilation

# 16

# Add-On Product License Management

**17**

# Network Couplers

**18**

# Model Construction

# Basic Principles of Modeling Physical Networks

## Overview of the Physical Network Approach to Modeling Physical Systems

Simscape software is a set of block libraries and special simulation features for modeling physical systems in the Simulink® environment. It employs the Physical Network approach, which differs from the standard Simulink modeling approach and is particularly suited to simulating systems that consist of real physical components.

Simulink blocks represent basic mathematical operations. When you connect Simulink blocks together, the resulting diagram is equivalent to the mathematical model, or representation, of the system under design. Simscape technology lets you create a network representation of the system under design, based on the Physical Network approach. According to this approach, each system is represented as consisting of functional elements that interact with each other by exchanging energy through their ports.

These connection ports are nondirectional. They mimic physical connections between elements. Connecting Simscape blocks together is analogous to connecting real components, such as pumps, valves, and so on. In other words, Simscape diagrams mimic the physical system layout. If physical components can be connected, their models can be connected, too. You do not have to specify flow directions and information flow when connecting Simscape blocks, just as you do not have to specify this information when you connect real physical components. The Physical Network approach, with its Through and Across variables and nondirectional physical connections, automatically resolves all the traditional issues with variables, directionality, and so on.

The number of connection ports for each element is determined by the number of energy flows it exchanges with other elements in the system, and depends on the level of idealization. For example, a fixed-displacement hydraulic pump in its simplest form can be represented as a two-port element, with one energy flow associated with the inlet (suction) and the other with the outlet. In this representation, the angular velocity of the driving shaft is assumed constant, making it possible to neglect the energy exchange between the pump and the shaft. To account for a variable driving torque, you need a third port associated with the driving shaft.

An energy flow is characterized by its variables. Each energy flow is associated with two variables, one Through and one Across (see "Variable Types" on page 1-3 for more information). Usually, these are the variables whose product is the energy flow in watts. They are called the basic, or conjugate, variables. For example, the basic variables for mechanical translational systems are force and velocity, for mechanical rotational systems—torque and angular velocity, for hydraulic systems—flow rate and pressure, for electrical systems—current and voltage.

The following example illustrates a Physical Network representation of a double-acting hydraulic cylinder.
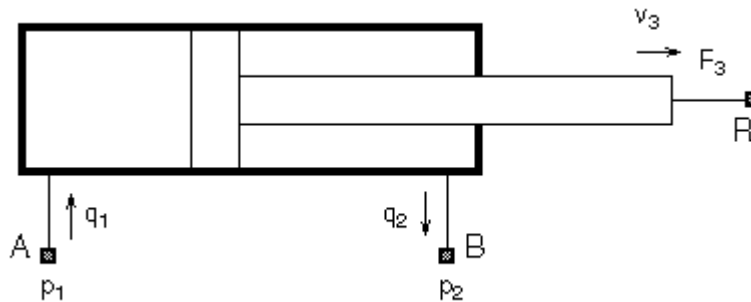
The element is represented with three energy flows: two flows of hydraulic energy through the inlet and outlet of the cylinder and a flow of mechanical energy associated with the rod motion. It therefore has the following three connector ports:

*   A — Hydraulic conserving port associated with pressure $p_1$ (an Across variable) and flow rate $q_1$ (a Through variable)
*   B — Hydraulic conserving port associated with pressure $p_2$ (an Across variable) and flow rate $q_2$ (a Through variable)
*   R — Mechanical translational conserving port associated with rod velocity $v_3$ (an Across variable) and force $F_3$ (a Through variable)

See "Connector Ports and Connection Lines" on page 1-6 for more information on connector port types.

## Variable Types

Physical Network approach supports two types of variables:

*   Through — Variables that are measured with a gauge connected in series to an element.
*   Across — Variables that are measured with a gauge connected in parallel to an element.

The following table lists the Through and Across variables associated with each type of physical domain in Simscape software:

| Physical Domain | Across Variable | Through Variable |
| --- | --- | --- |
| Electrical | Voltage | Current |
| Gas | Absolute pressure and temperature | Mass flow rate and energy flow rate |
| Hydraulic | Gauge pressure | Volumetric flow rate |
| Isothermal liquid | Absolute pressure | Mass flow rate |
| Magnetic | Magnetomotive force (mmf) | Flux |
| Mechanical rotational | Angular velocity | Torque |
| Mechanical translational | Translational velocity | Force |

| Physical Domain | Across Variable | Through Variable |
|---|---|---|
| Moist Air | Absolute pressure, temperature, specific humidity (water vapor mass fraction), and trace gas mass fraction | Mixture mass flow rate, mixture energy flow rate, water vapor mass flow rate, and trace gas mass flow rate |
| Thermal | Temperature | Heat flow |
| Thermal liquid | Absolute pressure and temperature | Mass flow rate and energy flow rate |
| Two-phase fluid | Absolute pressure and specific internal energy | Mass flow rate and energy flow rate |

## Building the Mathematical Model

Through and Across variables associated with all the energy flows form the basis of the mathematical model of the block.
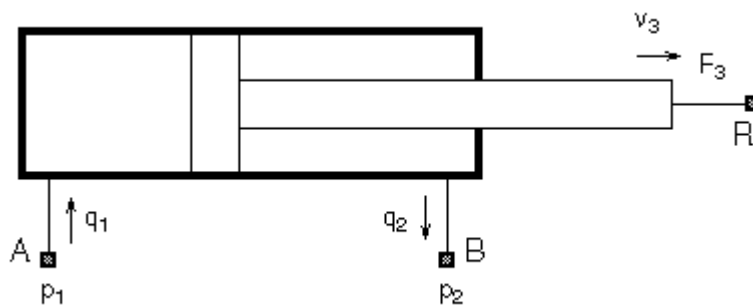


For example, the model of a double-acting hydraulic cylinder shown in the previous illustration can be described with a simple set of equations:

$$F_3 = p_1 \cdot A_1 - p_2 \cdot A_2$$

$$q_1 = A_1 \cdot v_3$$

$$q_2 = A_2 \cdot v_3$$

where

| | |
|---|---|
| $q_1, q_2$ | Flow rates through ports A and B, respectively (Through variables) |
| $p_1, p_2$ | Gauge pressures at ports A and B, respectively (Across variables) |
| $A_1, A_2$ | Piston effective areas |
| $F_3$ | Rod force (Through variable) |
| $v_3$ | Rod velocity (Across variable) |

The model could be considerably more complex, for example, it could account for friction, fluid compressibility, inertia of the moving parts, and so on. For all these different mathematical models, however, the element configuration (that is, the number and type of ports and the associated Through and Across variables) would remain the same, meaning that the Physical Network approach lets you

substitute models of different levels of complexity without introducing any changes to the schematic. For example, you can start developing your system by using the Resistive Tube block from the Foundation library, which accounts only for friction losses. At a later stage in development, you may want to account for fluid compressibility. You can then replace it with a Hydraulic Pipeline block, available with Simscape Fluids™ block libraries, or, depending on your application, even with a Segmented Pipeline block if you also need to account for fluid inertia. This modeling principle is called incremental modeling.

## Direction of Variables

Each variable is characterized by its magnitude and sign. The sign is the result of measurement orientation. The same variable can be positive or negative, depending on the polarity of a measurement gauge.

Elements with only two ports are characterized with one pair of variables, a Through variable and an Across variable. Since these variables are closely related, their orientation is defined with one direction. For example, if an element is oriented from port A to port B, it implies that the Through variable ($TV$) is positive if it "flows" from A to B, and the Across variable is determined as $AV = AV_A - AV_B$, where $AV_A$ and $AV_B$ are the element node potentials or, in other words, the values of this Across variable at ports A and B, respectively.
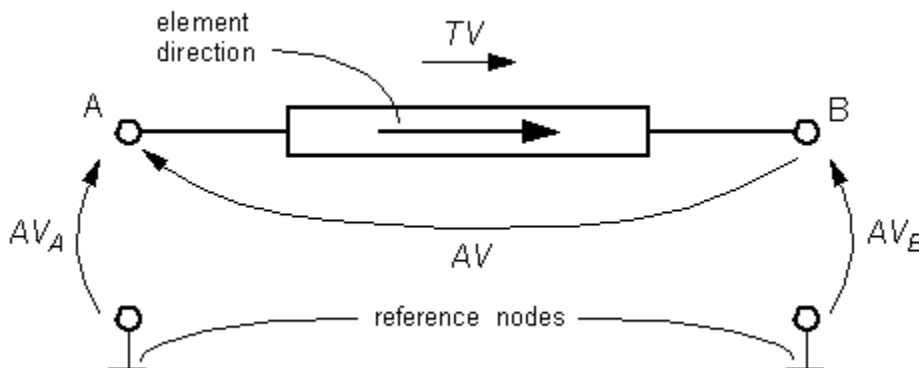


This approach to the direction of variables has the following benefits:

- Provides a simple and consistent way to determine whether an element is active or passive. Energy is one of the most important characteristics to be determined during simulation. If the variables direction, or sign, is determined as described above, their product (that is, the energy) is positive if the element consumes energy, and is negative if it provides energy to a system. This rule is followed throughout the Simscape software.

- Simplifies the model description. Symbol A → B is enough to specify variable polarity for both the Across and the Through variables.

- Lets you apply the oriented graph theory to network analysis and design.

As an example of variables direction rules, let us consider the Ideal Force Source block. In this block, as in many other mechanical blocks, port C is associated with the source reference point (case), and port R is associated with the rod.
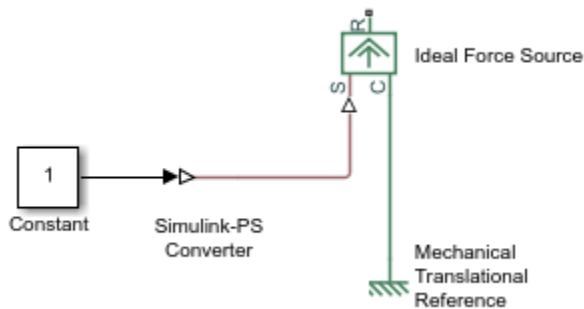
The block positive direction is from port C to port R. This means that the force is positive if it acts in the direction from C to R, and causes bodies connected to port R to accelerate in the positive direction. The relative velocity is determined as $v = v_C - v_R$, where $v_R$, $v_C$ are the absolute velocities at ports R and C, respectively, and it is negative if velocity at port R is greater than that at port C. The power generated by the source is computed as the product of force and velocity, and is negative if the source provides energy to the system.

Definition of positive direction is different for different blocks. Check the block source or the block reference page if in doubt about the block orientation and direction of variables.

All the elements in a network are divided into active and passive elements, depending on whether they deliver energy to the system or dissipate (or store) it. Active elements (force and velocity sources, flow rate and pressure sources, etc.) must be oriented strictly in accordance with the line of action or function that they are expected to perform in the system, while passive elements (dampers, resistors, springs, pipelines, etc.) can be oriented either way.

## Connector Ports and Connection Lines

Simscape blocks may have the following types of ports:

- Physical conserving ports — Nondirectional ports (for example, hydraulic or mechanical) that represent physical connections and relate physical variables based on the Physical Network approach.

- Physical signal ports — Unidirectional ports transferring signals that use an internal Simscape engine for computations.

Each of these ports and connections between them are described in greater detail below.

### Physical Conserving Ports

Simscape blocks have special conserving ports ▣. You connect conserving ports with physical connection lines, distinct from normal Simulink lines. Physical connection lines have no inherent directionality and represent the exchange of energy flows, according to the Physical Network approach.

- You can connect conserving ports only to other conserving ports of the same type.

- The physical connection lines that connect conserving ports together are nondirectional lines that carry physical variables (Across and Through variables, as described above) rather than signals. You cannot connect physical lines to Simulink ports or to physical signal ports.

- Two directly connected conserving ports must have the same values for all their Across variables (such as pressure or angular velocity).
- You can branch physical connection lines. When you do so, components directly connected with one another continue to share the same Across variables. Any Through variable (such as flow rate or torque) transferred along the physical connection line is divided among the multiple components connected by the branches. How the Through variable is divided is determined by the system dynamics.

  For each Through variable, the sum of all its values flowing into a branch point equals the sum of all its values flowing out.

Each type of physical conserving ports used in Simscape blocks uniquely represents a physical modeling domain. For a list of port types, along with the Through and Across variables associated with each type, see the table in "Variable Types" on page 1-3.

For improved readability of block diagrams, each Simscape domain uses a distinct default color and line style for the connection lines. For more information, see "Domain-Specific Line Styles" on page 1-34.

### Physical Signal Ports

Physical signal ports ▷ carry signals between Simscape blocks. You connect them with regular connection lines, similar to Simulink signal connections. Physical signal ports are used in Simscape block diagrams instead of Simulink input and output ports to increase computation speed and avoid issues with algebraic loops. Physical signals can have units associated with them. You specify the units along with the parameter values in the block dialogs and Property Inspector, and Simscape software performs the necessary unit conversion operations when solving a physical network.

Simscape Foundation library contains, among other sublibraries, a Physical Signals block library. These blocks perform math operations and other functions on physical signals, and allow you to graphically implement equations inside the physical network.

Physical signal lines also have a distinct style and color in block diagrams, similar to physical connection lines. For more information, see "Domain-Specific Line Styles" on page 1-34.

## See Also

## Related Examples
- "Creating and Simulating a Simple Model" on page 1-16

## More About
- "Connecting Simscape Diagrams to Simulink Sources and Scopes" on page 1-8

# Connecting Simscape Diagrams to Simulink Sources and Scopes

Simscape block diagrams use physical signals instead of regular Simulink signals. Therefore, you need converter blocks to connect Simscape diagrams to Simulink sources and scopes.

Use the Simulink-PS Converter block to connect Simulink sources or other Simulink blocks to the inputs of a Simscape physical network. You can also use it to specify the input signal units. For more information, see the Simulink-PS Converter block reference page.
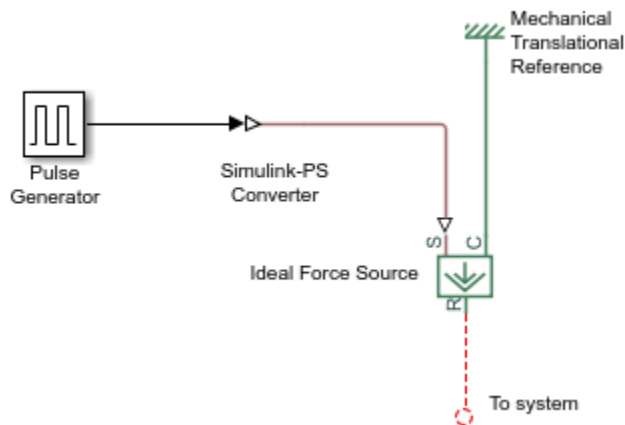


Use the PS-Simulink Converter block to connect outputs of a Simscape physical network to Simulink scopes or other Simulink blocks. You can also use it to specify the desired output signal units. For more information, see the PS-Simulink Converter block reference page.
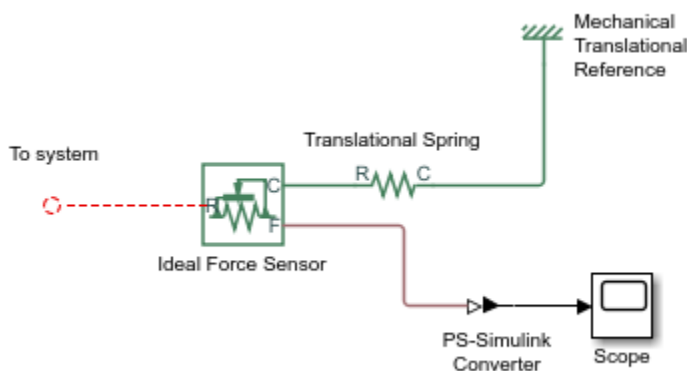


For a detailed example of using converter blocks to connect Simscape diagrams to Simulink sources and scopes, see "Creating and Simulating a Simple Model" on page 1-16.

You can also use data logging, instead of sensors and scopes, to view and analyze simulation results. For a detailed example, see "Log, Navigate, and Plot Simulation Data" on page 13-20.

Another way to access variables during simulation is to use a Probe block. Adding this block to your model lets you select variables from another block in the model and output them as Simulink signals.

## See Also

Simulink-PS Converter | PS-Simulink Converter | Probe

## Related Examples

- "Creating and Simulating a Simple Model" on page 1-16
- "Log, Navigate, and Plot Simulation Data" on page 13-20

## More About

- "Physical Signal Ports" on page 1-7
- "About Simulation Data Logging" on page 13-2

# Simscape Block Libraries

| **In this section...** |
|---|
| |
| |

## Library Structure Overview

Simscape block library contains two libraries that belong to the Simscape product:

- Foundation library — Contains basic physical elements and building blocks, as well as sources and sensors, organized into sublibraries according to technical discipline and function performed.
- Utilities library — Contains essential environment blocks for creating Physical Networks models.

In addition, if you have installed any of the add-on products, you will see the corresponding libraries under the main Simscape library. Add-on products are products in the Physical Modeling family that use Simscape platform and, as a result, share common functionality, such as physical units management or editing modes.

Simscape Foundation libraries contain a comprehensive set of basic elements and building blocks organized by physical domain: electrical, mechanical rotational and translational, isothermal liquid, gas, and so on. Within each domain, the blocks are grouped into elements, sources, and sensors. In each of the fluid domains, the Utilities sublibrary contains blocks that specify fluid properties. For more information, see "Fluid System Modeling" on page 1-14. The Physical Signals block library lets you perform math operations on physical signals.

Using the basic building blocks in the Foundation libraries, you can create more complex components that span different physical domains. You can then group this assembly of blocks into a subsystem and parameterize it to reuse and share these components.

In addition to Foundation libraries, there is also a Simscape Utilities library, which contains utility blocks, such as:

- Solver Configuration block, which contains parameters relevant to numerical algorithms for Simscape simulations. Each Simscape diagram (or each topologically distinct physical network in a diagram) must contain a Solver Configuration block.
- Simulink-PS Converter block and PS-Simulink Converter block, to connect Simscape and Simulink blocks. Use the Simulink-PS Converter block to connect Simulink outports to Physical Signal inports. Use the PS-Simulink Converter block to connect Physical Signal outports to Simulink inports.
- Probe block, which lets you select variables from another Simscape block in the model and output them as Simulink signals.

For examples of using these blocks in a Simscape model, see the tutorial "Creating and Simulating a Simple Model" on page 1-16 and the "Circuit Breaker with Probe Block" on page 20-88 example.

You can combine all these blocks in your Simscape diagrams to model physical systems. You can also use the basic Simulink blocks in your diagrams, such as sources or scopes. See "Connecting Simscape Diagrams to Simulink Sources and Scopes" on page 1-8 for more information on how to do this.

Simscape block libraries contain a comprehensive selection of blocks that represent engineering components such as valves, resistors, springs, and so on. These prebuilt blocks, however, may not be sufficient to address your particular engineering needs. When you need to extend the existing block libraries, use the Simscape language to define customized components, or even new physical domains, as textual files. Then convert your textual components into libraries of additional Simscape blocks that you can use in your model diagrams. For more information on how to do this, see "Typical Simscape Language Tasks".

## Accessing the Block Libraries

You can access the blocks through the Simulink Library Browser. To display the Library Browser, type `slLibraryBrowser` in the MATLAB® Command Window. Then expand the **Simscape** entry in the contents tree.
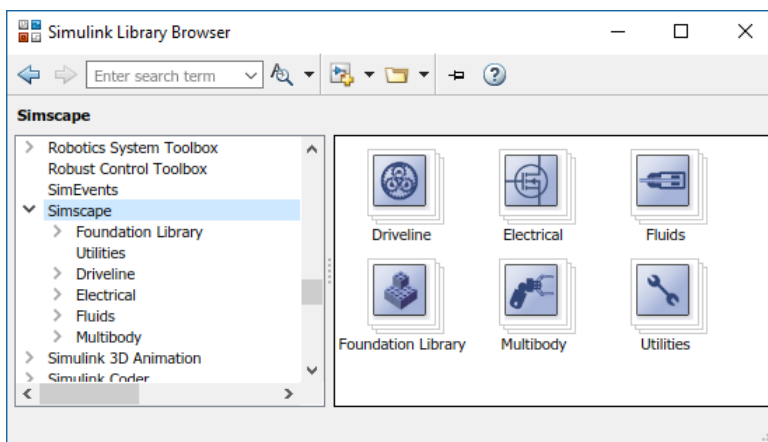


When you type `simscape` in the MATLAB Command Window, the main Simscape library opens in a separate window.

The Simscape library consists of two top-level libraries, Foundation and Utilities. In addition, if you have installed any of the add-on products of the Physical Modeling family, you will see the corresponding libraries under Simscape library, as shown in the following illustration. Some of these libraries contain second-level and third-level sublibraries. You can expand each library by double-clicking its icon.
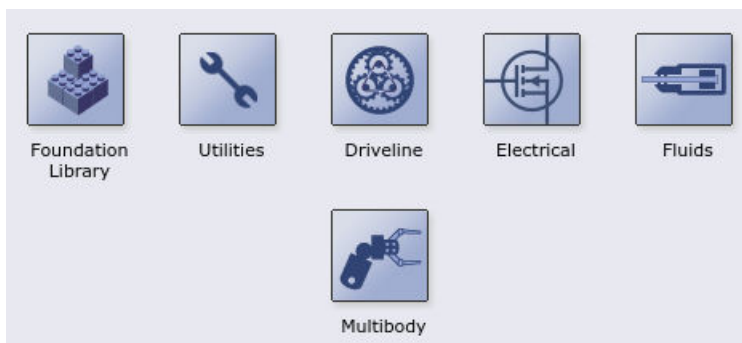
# Essential Physical Modeling Techniques

## Building Your Model

The rules that you must follow when building a physical model with Simscape software are described in "Basic Principles of Modeling Physical Networks" on page 1-2. This section briefly reviews these rules.

- Build your physical model by using a combination of blocks from the Simscape Foundation and Utilities libraries. Simscape software lets you create a network representation of the system under design, based on the Physical Network approach. According to this approach, each system is represented as consisting of functional elements that interact with each other by exchanging energy through their ports.

- Each Simscape diagram (or each topologically distinct physical network in a diagram) must contain a Solver Configuration block from the Simscape Utilities library.

- For fluid system modeling, specify the working fluid for each topologically distinct circuit, as described in "Specifying the Working Fluid" on page 1-14.

- To connect regular Simulink blocks (such as sources or scopes) to your physical network diagram, use the converter blocks, as described in "Using the Physical Signal Ports" on page 1-13.

- Use the incremental modeling approach. Start with a simple model, run and troubleshoot it, then add the desired special effects. For example, you can start developing your system by using the Pipe (IL) block from the Foundation library, which accounts only for friction losses and fluid compressibility. If you also need to account for fluid inertia, you can turn on this effect after your simplified model works as expected. At a later stage in development, you may want to account for other effects, such as pipe wall compliance, various cross-sectional geometries, elevation, or curvature. You can then replace the Foundation library block with one of the blocks available with the Simscape Fluids block libraries, such as Pipe (IL), Partially Filled Pipe (IL), or Pipe Bend (IL). For all these different mathematical models, the element configuration (that is, the number and type of ports and the associated Through and Across variables) would remain the same, meaning that the Physical Network approach lets you substitute models of different levels of complexity without introducing any changes to the schematic.

Simscape blocks, in general, feature both Conserving ports ⊠ and Physical Signal inports and outports ▷.

## Using the Conserving Ports

The following rules apply to Conserving ports:

- There are different types of Physical Conserving ports used in Simscape block diagrams, such as hydraulic, electrical, mechanical translational, mechanical rotational, and so on. Each type has specific Through and Across variables associated with it. For more information, see "Variable Types" on page 1-3.

- You can connect Conserving ports only to other Conserving ports of the same type. Domain-specific line styles and colors help distinguish between different domains and facilitate the connection process. For more information, see "Domain-Specific Line Styles" on page 1-34.

- The Physical connection lines that connect Conserving ports together are nondirectional lines that carry physical variables (Across and Through variables, as described above) rather than signals. You cannot connect Physical lines to Simulink ports or to Physical Signal ports.

- Two directly connected Conserving ports must have the same values for all their Across variables (such as voltage or angular velocity).
- You can branch Physical connection lines. When you do so, components directly connected with one another continue to share the same Across variables. Any Through variable (such as current or torque) transferred along the Physical connection line is divided among the multiple components connected by the branches. How the Through variable is divided is determined by the system dynamics.

  For each Through variable, the sum of all its values flowing into a branch point equals the sum of all its values flowing out.

## Using the Physical Signal Ports

The following rules apply to Physical Signal ports:

- You can connect Physical Signal ports to other Physical Signal ports with regular connection lines, similar to Simulink signal connections. These connection lines carry physical signals between Simscape blocks.
- You can connect Physical Signal ports to Simulink ports through special converter blocks. Use the Simulink-PS Converter block to connect Simulink outports to Physical Signal inports. Use the PS-Simulink Converter block to connect Physical Signal outports to Simulink inports.
- Physical Signals can have units associated with them. Simscape block dialogs and Property Inspector let you specify the units along with the parameter values, where appropriate. Use the converter blocks to associate units with an input signal and to specify the desired output signal units.

For examples of applying these rules when creating an actual physical model, see the tutorial "Creating and Simulating a Simple Model" on page 1-16.

# Fluid System Modeling

## Simscape Fluid Domains

Simscape Foundation library contains several domains for modeling fluid systems. This table provides a summary of fluid domains, to help you select the appropriate domain for your application.

| Domain | Working Fluid | Thermodynamic Effects |
|---|---|---|
| Isothermal Liquid | Liquid, possibly with a small amount of entrained air | No |
| Thermal Liquid | Liquid | Yes |
| Two-Phase Fluid | Part liquid and part vapor | Yes |
| Gas | Gas: perfect, semiperfect, or real (single species) | Yes |
| Moist Air | Mixture of dry air, water vapor, and trace gas (or of up to three other semiperfect gas species) | Yes |

**Note** The hydraulic domain and Hydraulic block library also let you model fluid systems that do not account for thermodynamic effects and where the working fluid is liquid with a small amount of entrained air. However, MathWorks recommends that you use the Isothermal Liquid library for modeling isothermal hydraulic systems. For more information, see "Upgrading Hydraulic Models To Use Isothermal Liquid Blocks" on page 2-9.

## Specifying the Working Fluid

The rules that you must follow when building a physical model with Simscape software are described in "Basic Principles of Modeling Physical Networks" on page 1-2. This section briefly reviews the rules that are specific to fluid system modeling.

When modeling systems that contain fluid elements, it is very important to specify the working fluid correctly:

*   If you have isothermal liquid elements in your model, the working fluid is liquid, possibly with a small amount of entrained air. Attach an Isothermal Liquid Properties (IL) block to each topologically distinct circuit to define the working fluid properties. This block lets you specify whether the liquid bulk modulus is constant or pressure-dependent. You can also specify whether the fluid contains entrained air, and, if present, whether its amount is constant or pressure-dependent. The default working fluid is water, with constant bulk modulus and zero entrained air.

    If you have a Simscape Fluids license, you can also use the Isothermal Liquid Predefined Properties (IL) block to specify the working fluid in a circuit. This block lets you select from a list of predefined liquids and specify values for various fluid properties, as well as visualize fluid properties in the circuit as a function of pressure.

*   Similarly, if you have thermal liquid elements in your model, attach a Thermal Liquid Settings (TL) block to each topologically distinct circuit to define the working fluid properties. The block takes as inputs the necessary fluid properties, each specified as a tabulated function of pressure and temperature.

If you have a Simscape Fluids license, you can also use the Thermal Liquid Properties (TL) block to specify the working fluid in a thermal liquid circuit. This block lets you select the name of the fluid from a list that includes water, seawater, and various mixtures with uses in cooling and deicing.

- If you have two-phase elements in your model, the working fluid is part liquid and part vapor. Attach a Two-Phase Fluid Properties (2P) block to each topologically distinct circuit to specify the working fluid properties. This block lets you define the properties of liquid and vapor separately, each as a tabulated function of pressure and temperature.

  If you have a Simscape Fluids license, you can also use the Two-Phase Fluid Predefined Properties (2P) block to specify the working fluid in a two-phase fluid circuit. This block lets you select the name of the fluid from a list that includes water, ammonia, $CO_2$, and various other fluids at the specified ranges of pressure and temperature.

- If you have gas elements in your model, default gas properties are for dry air. Attach a Gas Properties (G) block to each topologically distinct circuit to change gas properties.

- If you have moist air elements in your model, default properties correspond to dry air, water vapor, and carbon dioxide (the optional trace gas). Attach a Moist Air Properties (MA) block to each topologically distinct circuit to change the air mixture properties.

- If you have hydraulic blocks in your model, the working fluid used in the hydraulic circuit defines their global parameters, such as fluid density, fluid kinematic viscosity, and fluid bulk modulus, specified as constant values. To specify the working fluid, attach a Custom Hydraulic Fluid block (or a Hydraulic Fluid block, available with Simscape Fluids block libraries) to each topologically distinct hydraulic circuit.

If you omit the fluid properties block in a circuit, the working fluid in that circuit assumes the domain default properties. The default fluid properties for each domain correspond to the default parameter values of the fluid properties block in the respective Foundation library.

# Creating and Simulating a Simple Model

| In this section... |
|---|
| "Building a Simscape Diagram" on page 1-16 |
| "Modifying Initial Settings" on page 1-21 |
| "Running the Simulation" on page 1-22 |
| "Adjusting the Parameters" on page 1-24 |

## Building a Simscape Diagram

In this example, you are going to model a simple mechanical system and observe its behavior under various conditions. This tutorial illustrates the essential steps to building a physical model on page 1-12 and makes you familiar with using the basic Simscape blocks.

**Note** For time-saving techniques and advanced ways of analyzing simulation data, see the "Essential Steps for Constructing a Physical Model" tutorial.

The following schematic represents a simple model of a car suspension. It consists of a spring and damper connected to a body (represented as a mass), which is agitated by a force. You can vary the model parameters, such as the stiffness of the spring, the mass of the body, or the force profile, and view the resulting changes to the velocity and position of the body.
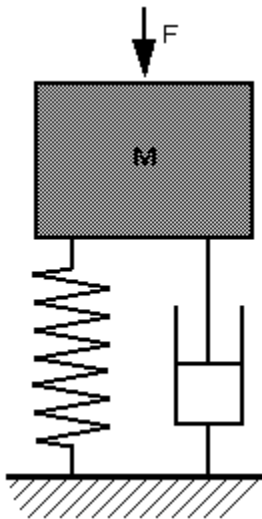


To create an equivalent Simscape diagram, follow these steps:

1   Open the Simulink Library Browser, as described in "Simscape Block Libraries" on page 1-10.

2   Create a new Simulink model using the **Blank Model** template. The software creates an empty model in memory and displays it in a new model editor window.

> **Note** Alternatively, you can type `ssc_new` at the MATLAB Command prompt, to create a new model prepopulated with certain required and commonly used blocks. For more information, see "Creating a New Simscape Model".

**3** By default, Simulink Editor hides the automatic block names in model diagrams. To display hidden block names for training purposes, clear the **Hide Automatic Block Names** check box. For more information, see "Manage Block Names and Ports".

**4** Open the Simscape > Foundation Library > Mechanical > Translational Elements library.

**5** Drag the Mass, Translational Spring, Translational Damper, and two Mechanical Translational Reference blocks into the model window.

**6** Orient the blocks as shown in the following illustration. To rotate a block, select it and press **Ctrl +R**.
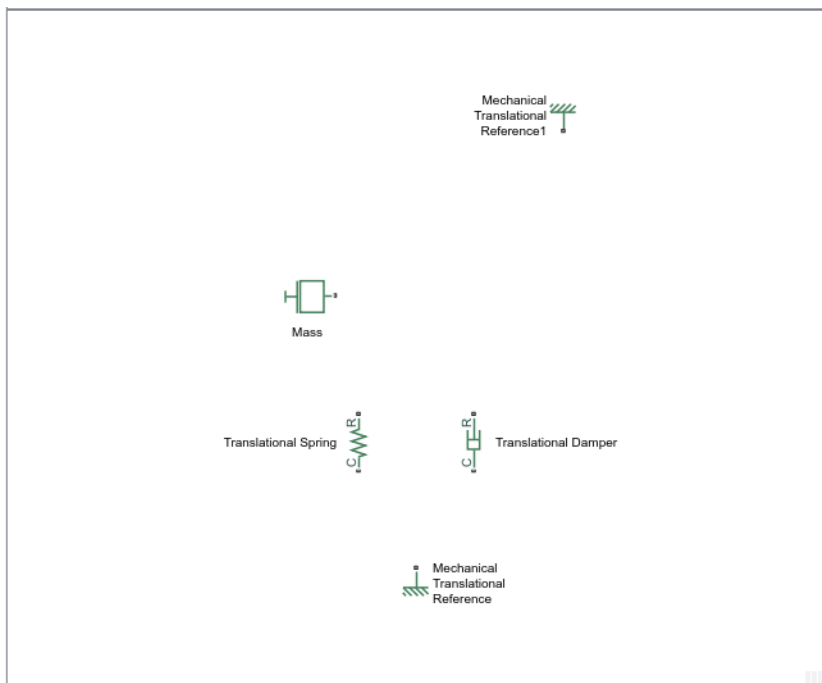


**7** Connect the Translational Spring, Translational Damper, and Mass blocks to one of the Mechanical Translational Reference blocks as shown in the next illustration.
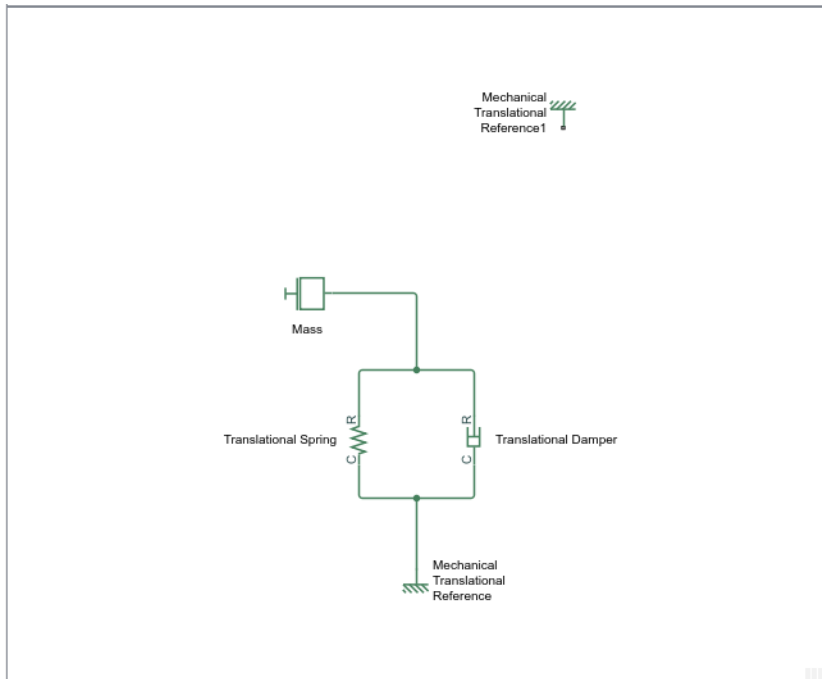
**8** To add the representation of the force acting on the mass, open the Simscape > Foundation Library > Mechanical > Mechanical Sources library and add the Ideal Force Source block to your diagram.

To reflect the correct direction of the force shown in the original schematic, flip the block orientation. With the Ideal Force Source block selected, on the **Format** tab at the top of the model window, under **Arrange**, click **Flip up-down**. Connect the block's port C (for "case") to the second Mechanical Translational Reference block, and its port R (for "rod") to the Mass block, as shown below.
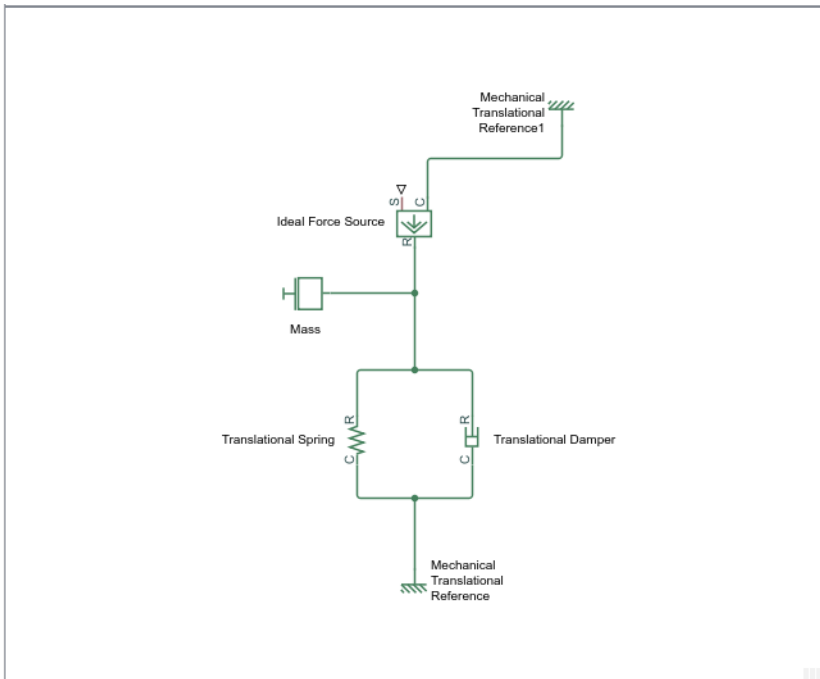
9  Add the sensor to measure speed and position of the mass. Place the Ideal Translational Motion Sensor block from the Mechanical Sensors library into your diagram and connect it as shown below.
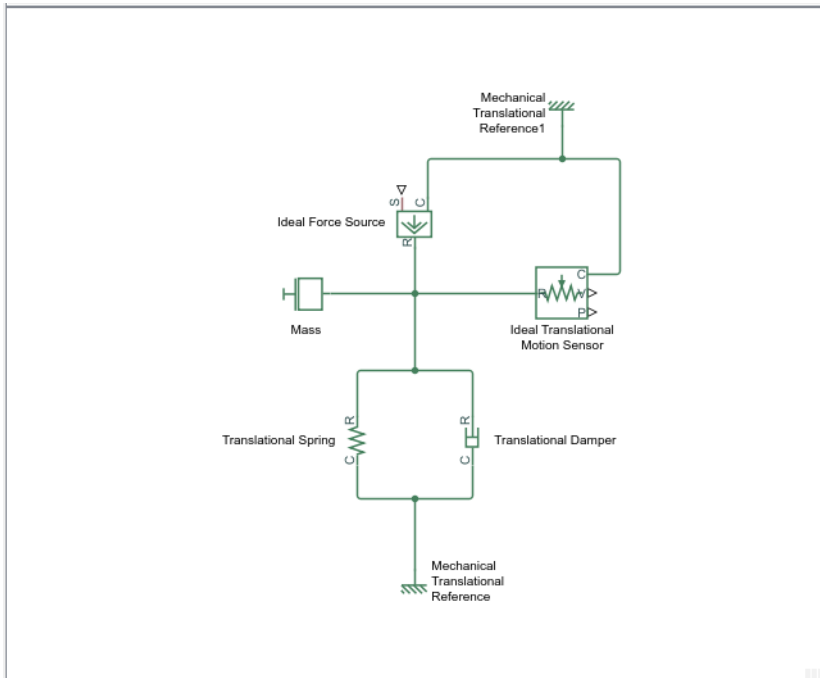


10  Now you need to add the sources and scopes. They are found in the Simulink libraries. Open the Simulink > Sources library and copy the Signal Editor block into the model. Then open the Simulink > Sinks library and copy two Scope blocks. Rename one of the Scope blocks to `Velocity` and the other to `Position`.
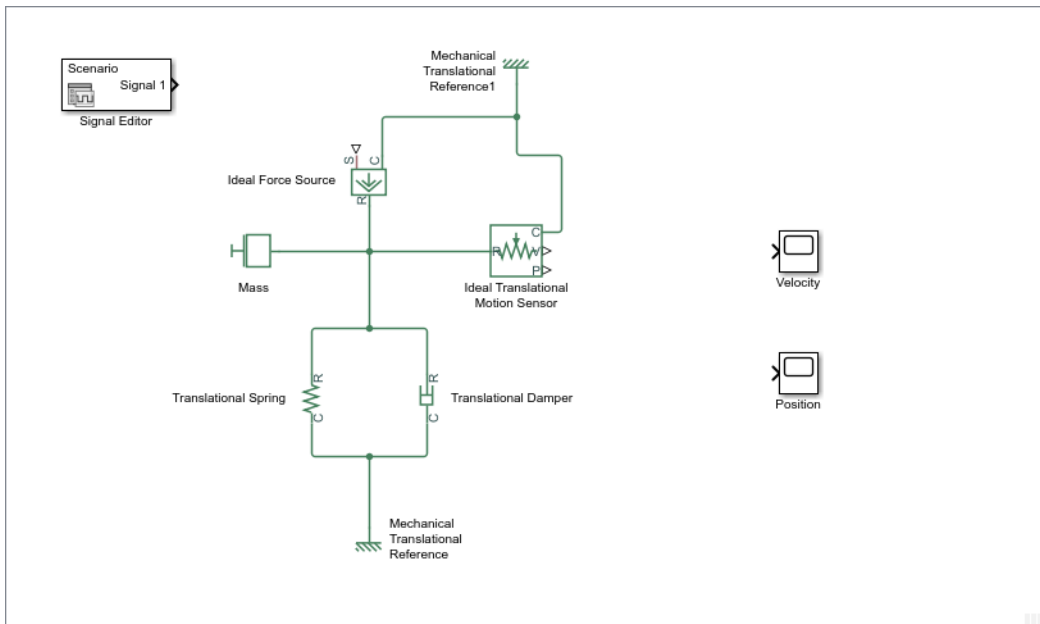
11 Every time you connect a Simulink source or scope to a Simscape diagram, you have to use an appropriate converter block, to convert Simulink signals into physical signals and vice versa. Open the Simscape > Utilities library and copy a Simulink-PS Converter block and two PS-Simulink Converter blocks into the model. Connect the blocks as shown below.
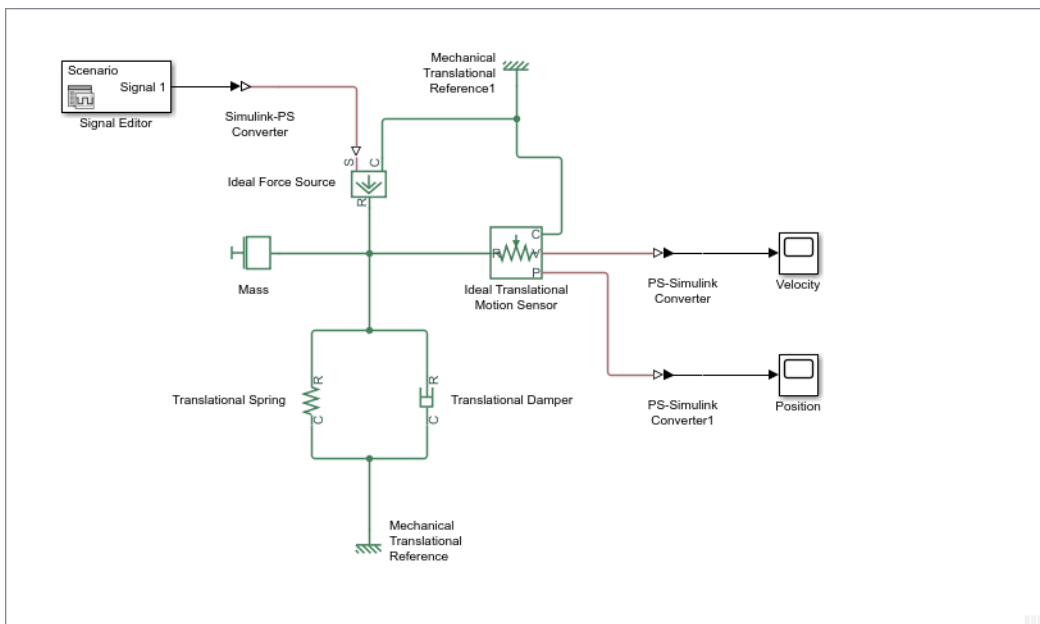


12 Each topologically distinct physical network in a diagram requires exactly one Solver Configuration block, found in the Simscape > Utilities library. Copy this block into your model and connect it to the circuit by creating a branching point and connecting it to the only port of the Solver Configuration block. Your diagram now should look like this.
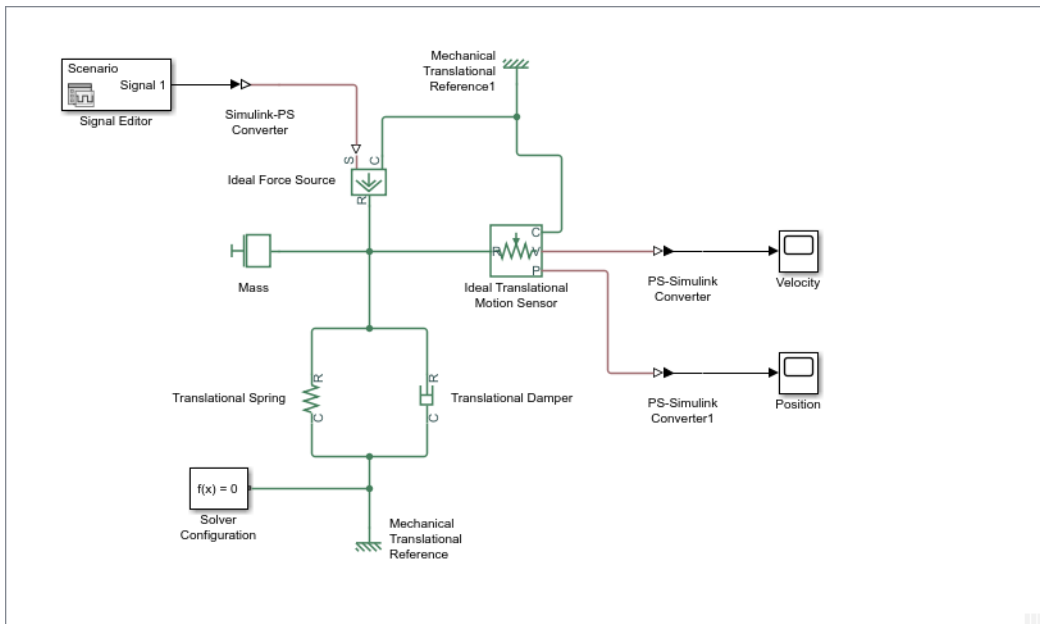
**13** Your block diagram is now complete. Save it as `mech_simple`.

## Modifying Initial Settings

After you have put together a block diagram of your model, as described in the previous section on page 1-16, you need to select a solver and provide the correct values for configuration parameters.

To prepare for simulating the model, follow these steps:

**1** Select a Simulink solver. In the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens, showing the **Solver** pane.

Under **Solver selection**, set **Solver** to `ode23t (mod.stiff/Trapezoidal)`.

Expand **Solver details** and set **Max step size** to `0.2`.

Also note that **Simulation time** is specified to be between 0 and 10 seconds. You can adjust this setting later, if needed.
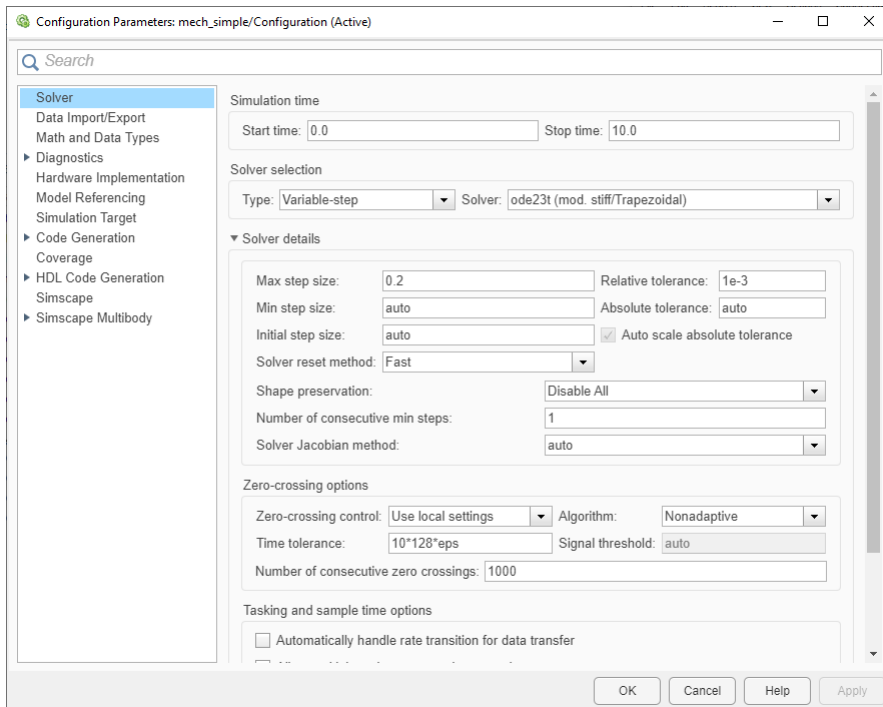
Click **OK** to close the Configuration Parameters dialog box.

**2** Save the model.

## Running the Simulation

After you've put together a block diagram and specified the initial settings for your model, you can run the simulation.

**1** The input signal for the force is provided by the Signal Editor block. Edit the signal properties as shown in the illustration below. The signal profile starts with a value of 0, then at 4 seconds there is a step change to 1, and then it changes back to 0 at 6 seconds.
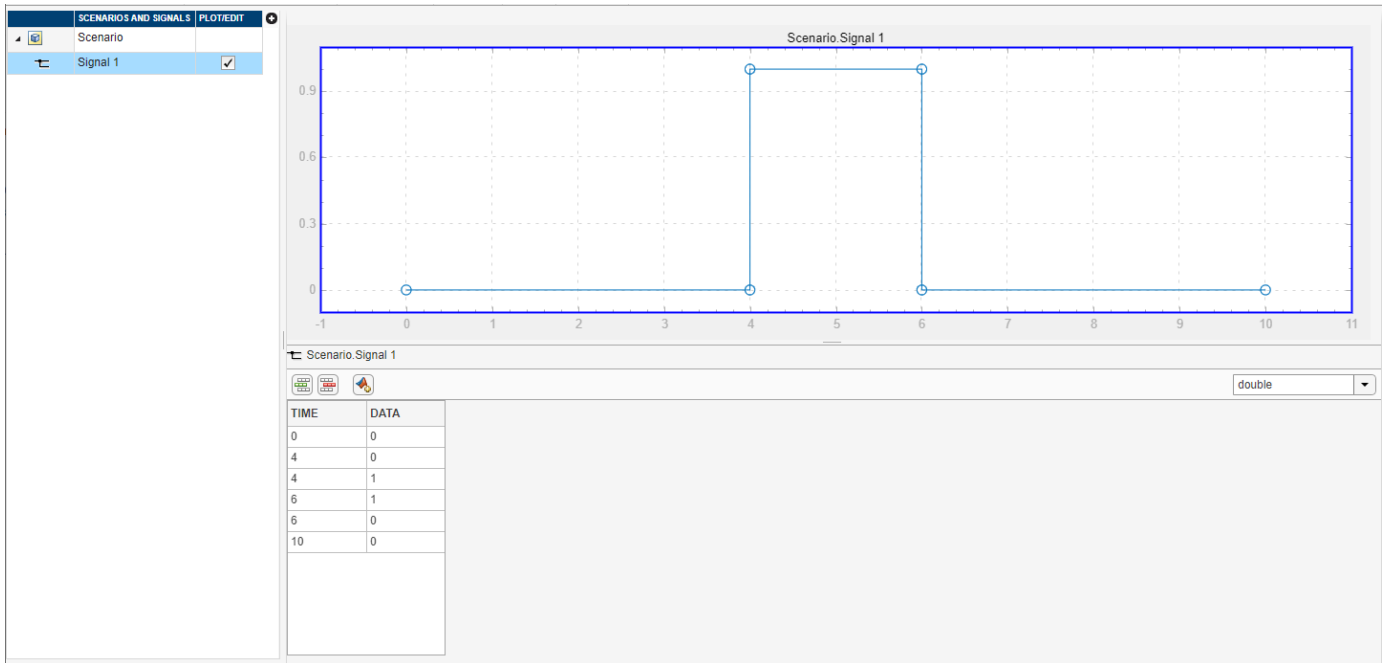
The Velocity scope outputs the mass velocity, and the Position scope outputs the mass displacement as a function of time. Double-click both scopes to open them.

**2** Click . The Simscape solver evaluates the model, calculates the initial conditions, and runs the simulation. For a detailed description of this process, see "How Simscape Simulation Works" on page 7-6. Completion of this step may take a few seconds. The message in the bottom-left corner of the model window provides the status update.

**3** Once the simulation starts running, the Velocity and Position scope windows display the simulation results, as shown in the next illustration.
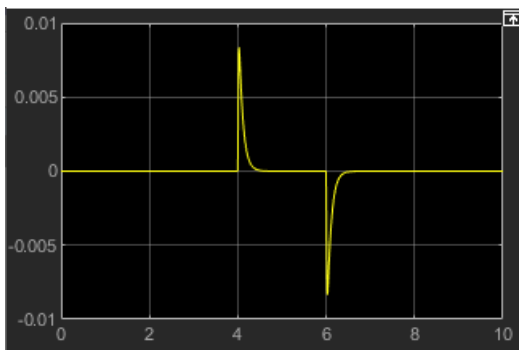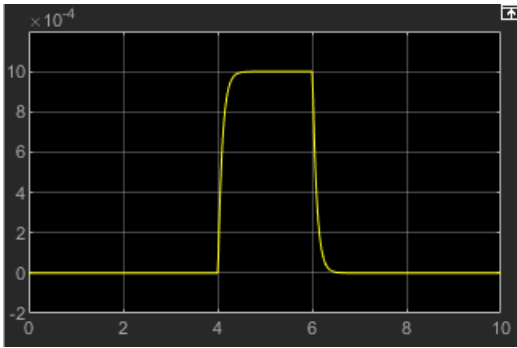
In the beginning, the mass is at rest. Then at 4 seconds, as the input signal changes abruptly, the mass velocity spikes in the positive direction and gradually returns to zero. The mass position at the same time changes more gradually, on account of inertia and damping, and stays at the new value as long as the force is acting upon it. At 6 seconds, when the input signal changes back to zero, the velocity gets a mirror spike, and the mass gradually returns to its initial position.

You can now adjust various inputs and block parameters and see their effect on the mass velocity and displacement.

## Adjusting the Parameters

After running the initial simulation, you can experiment with adjusting various inputs and block parameters.

Try the following adjustments:

**1**   Change the force profile on page 1-24.
**2**   Change the model parameters. on page 1-25
**3**   Change the mass position output units. on page 1-26

### Changing the Force Profile

This example shows how a change in the input signal affects the force profile, and therefore the mass displacement.

**1**   Double-click the Signal Editor block to open it.
**2**   Change the signal profile by moving its first vertical segment from 4 to 2 seconds, as shown below. Close the block dialog.
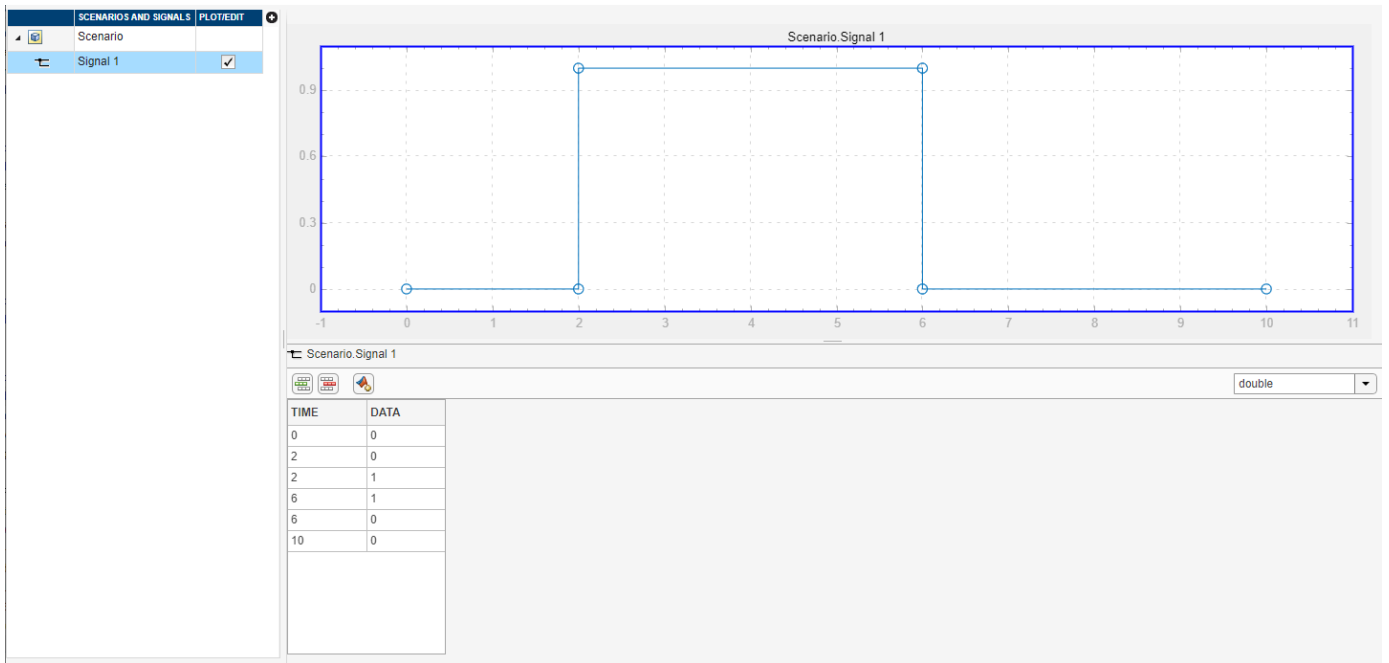
**3** Run the simulation. The simulation results are shown in the following illustration.
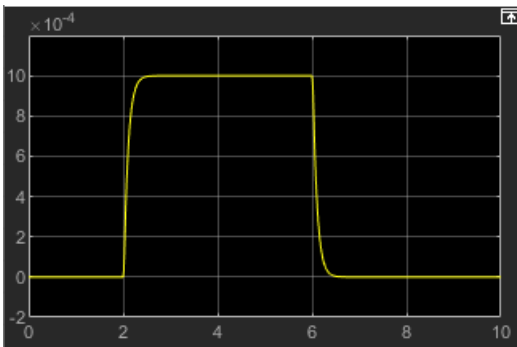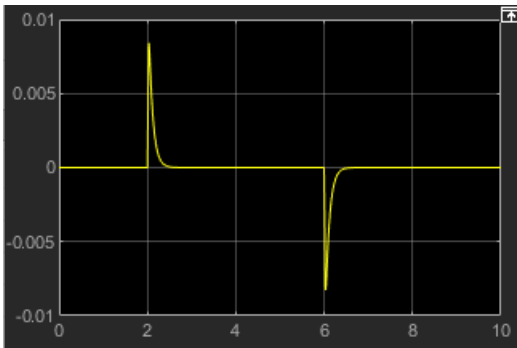




**Changing the Model Parameters**

In our model, the force acts on a mass against a translational spring and damper, connected in parallel. This example shows how changes in the spring stiffness and damper viscosity affect the mass displacement.

1    Double-click the Translational Spring block. Set its **Spring rate** to `2000 N/m`.

2    Run the simulation. The increase in spring stiffness results in smaller amplitude of mass displacement, as shown in the following illustration.
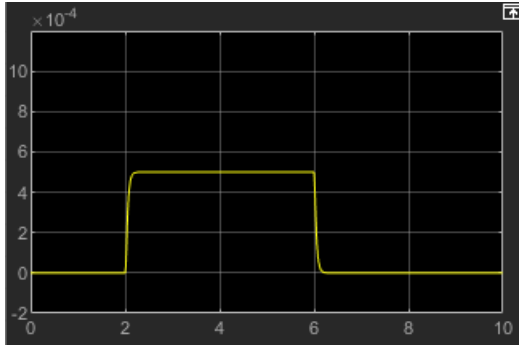


3    Next, double-click the Translational Damper block. Set its **Damping coefficient** to `500 N/(m/s)`.

4    Run the simulation. Because of the increase in viscosity, the mass is slower both in reaching its maximum displacement and in returning to the initial position, as shown in the following illustration.
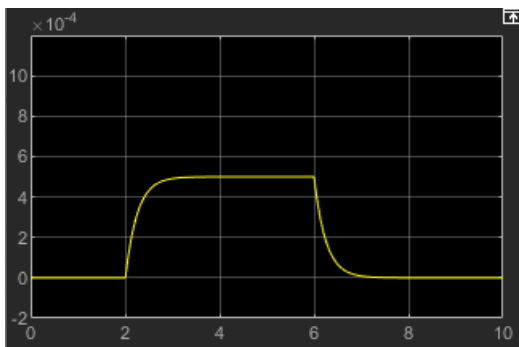


**Changing the Mass Position Output Units**

In our model, we have used the PS-Simulink Converter block in its default parameter configuration. Therefore, the `Position` scope outputs the mass displacement in the default length units, that is, in meters. This example shows how to change the output units for the mass displacement to millimeters.

1    Double-click the PS-Simulink Converter1 block. Type `mm` in the **Output signal unit** combo box and click **OK**.

2    Run the simulation. In the `Position` scope window, click ⬍ to autoscale the scope Y-axis. The mass displacement is now output in millimeters, as shown in the following illustration.
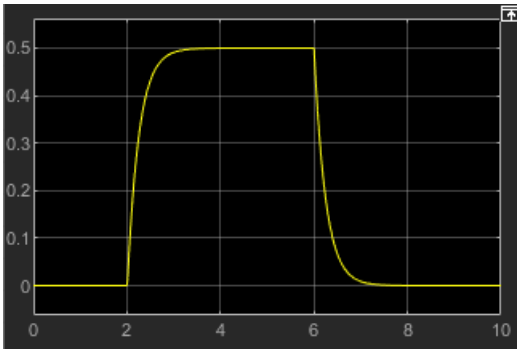
## See Also

## More About

- "Basic Principles of Modeling Physical Networks" on page 1-2
- "Modeling Best Practices" on page 1-28

# Modeling Best Practices

| **In this section...** |
|---|
| |
| |

## Grounding Rules

This section contains guidelines for using domain-specific reference blocks (such as Electrical Reference, Mechanical Translational Reference, and so on) in Simscape diagrams, along with examples of correct and incorrect configurations.

Add reference blocks to your models according to the following rules:

- "Each Domain Requires at Least One Reference Block" on page 1-28
- "Each Circuit Requires at Least One Reference Block" on page 1-28
- "Multiple Connections to the Domain Reference Are Allowed Within a Circuit" on page 1-30

### Each Domain Requires at Least One Reference Block

Within a physical network, each domain must contain at least one reference block of the appropriate type. For example, the electromechanical model shown in the following diagram has both Electrical Reference and Mechanical Rotational Reference blocks attached to the appropriate circuits.
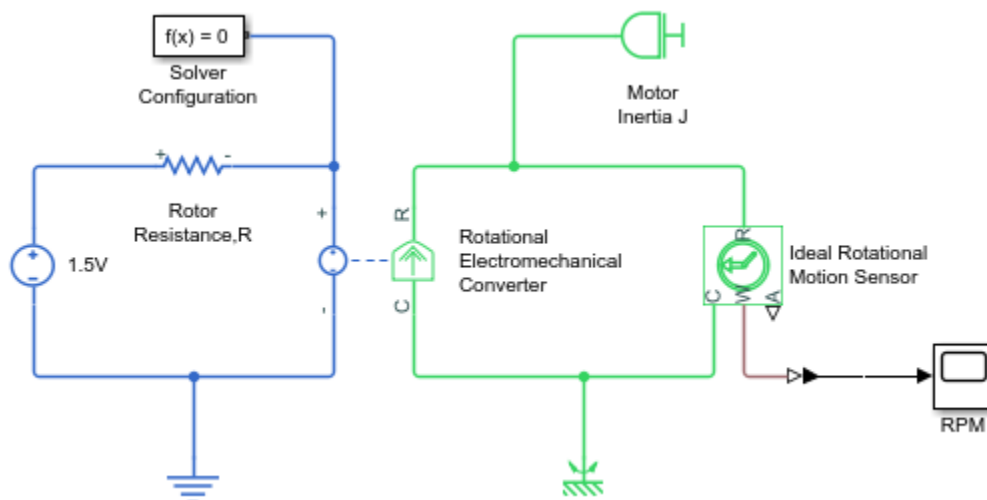


### Each Circuit Requires at Least One Reference Block

Each topologically distinct circuit within a domain must contain at least one reference block. Some blocks, such as an Ideal Transformer, interface two parts of the network but do not convey information about signal levels relative to the reference block. In the following diagram, there are two separate electrical circuits, and the Electrical Reference blocks are required on both sides of the Ideal Transformer block.
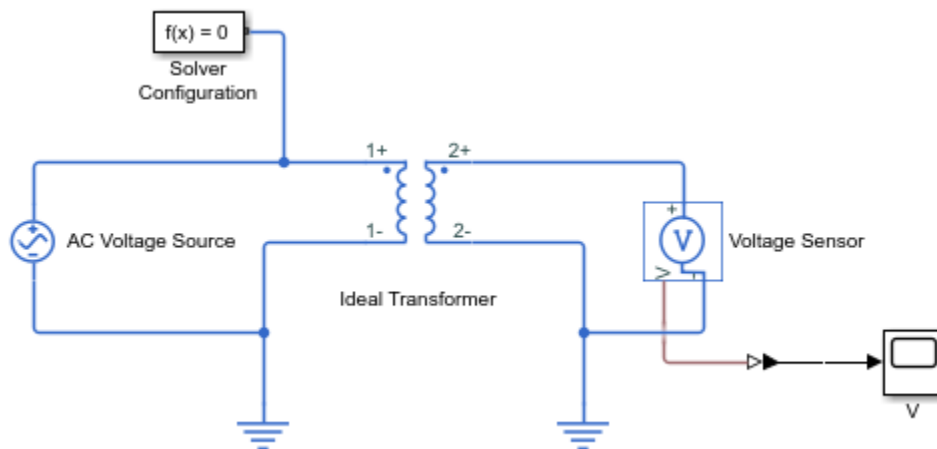
The next diagram would produce an error because it is lacking an electrical reference in the circuit of the secondary winding.
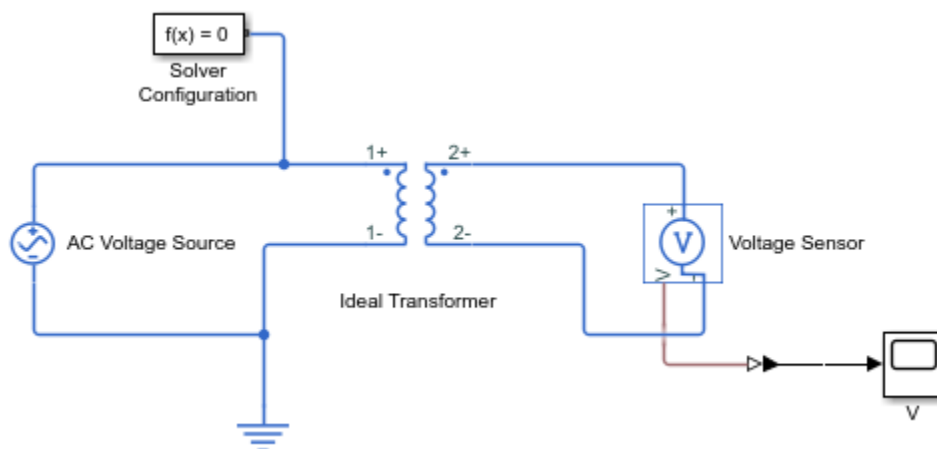


The following diagram, however, will not produce an error because the resistor defines the output voltage relative to the ground reference.
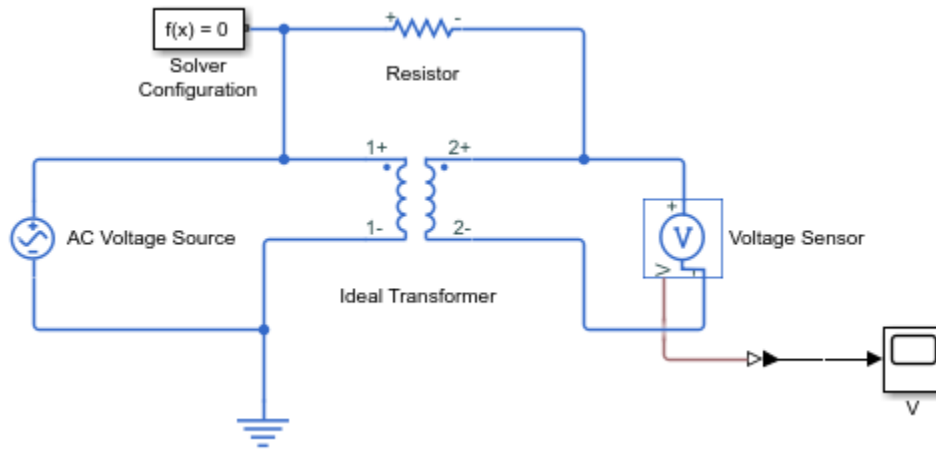
**Multiple Connections to the Domain Reference Are Allowed Within a Circuit**

More than one reference block may be used within a circuit to define multiple connections to the domain reference:

- Electrical conserving ports of all the blocks that are directly connected to ground must be connected to an Electrical Reference block.

- All translational ports that are rigidly clamped to the frame (ground) must be connected to a Mechanical Translational Reference block.

- All rotational ports that are rigidly clamped to the frame (ground) must be connected to a Mechanical Rotational Reference block.

- Conserving ports of all the fluids blocks that are referenced to atmosphere (for example, suction ports of hydraulic pumps, or return ports of valves, cylinders, pipelines, if they are considered directly connected to atmosphere) must be connected to the appropriate domain reference, such as the Hydraulic Reference block.

For example, the following diagram correctly indicates two separate connections to an electrical ground.
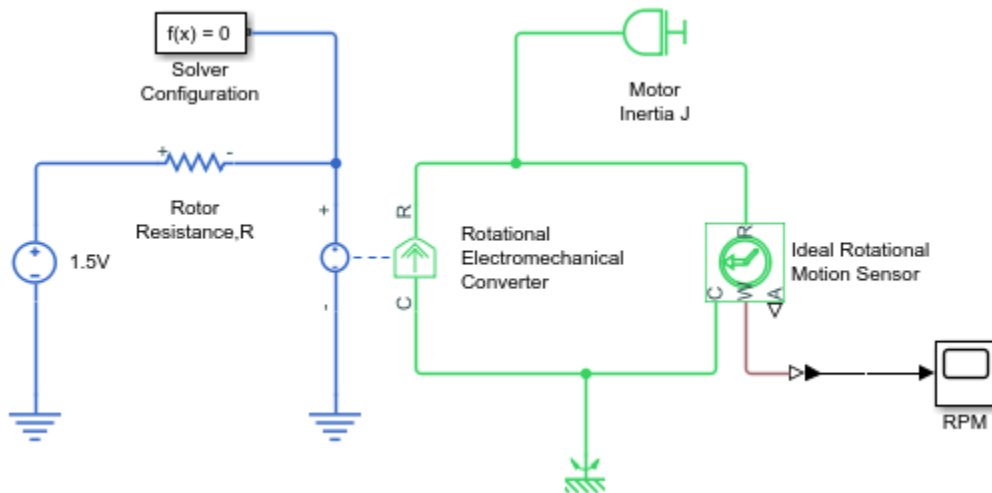
## Avoiding Numerical Simulation Issues

Certain configurations of physical modeling blocks can cause numerical difficulties or slow down your simulation. When this happens, Simscape solver issues a warning in the MATLAB workspace and, if it fails to initialize, a Simscape error.

In electrical circuits, common examples that can cause this behavior include voltage sources connected in parallel with capacitors, inductors connected in series with current sources, voltage sources connected in parallel, and current sources connected in series. Often, the cause of the numerical difficulty is immediately apparent. For example, two voltage sources in parallel must have identical voltage values; otherwise, the ports connecting them would not be physical conserving ports. In practical circuits, topologies such as parallel voltage sources are possible, and small difference in their instantaneous voltages is possible due to parasitic series resistance.

**Note** Mathematically, these topologies result in Index-2 differential algebraic equations (DAEs). Their solution requires two differentiations of the constraint equations and, as such, it is numerically better to avoid these component topologies where possible.

There are two approaches to resolving these difficulties. The first is to change the circuit to an equivalent simpler one. In the example of two parallel voltage sources, one source can be simply deleted. The same applies to two series current sources, the deleted one being replaced by a short circuit. For some circuit topologies, however, it is not possible to find an equivalent simpler one that resolves the problem, and the second approach is needed.

The second approach is to include small parasitic resistances in the component. In the Simscape Foundation library, the Capacitor and Inductor blocks include such parasitic terms, so that you can connect capacitances in parallel with voltage sources and inductors in series with current sources. If your circuit does not have any such topologies, then you can change the default parasitic terms to zero. Note that other blocks do not contain these parasitic terms, for example, the Mutual Inductor block. Therefore, if you wanted to connect a mutual inductor primary in series with a current source, you would need to introduce your own parasitic conductance across the primary winding.

**Example of Using a Parasitic Resistance to Avoid Numerical Simulation Issues**

The following diagram models a differentiator that might be used as part of a Proportional-Integral-Derivative (PID) controller. You can open this model by typing `ssc_opamp_differentiator` in the MATLAB Command Window.
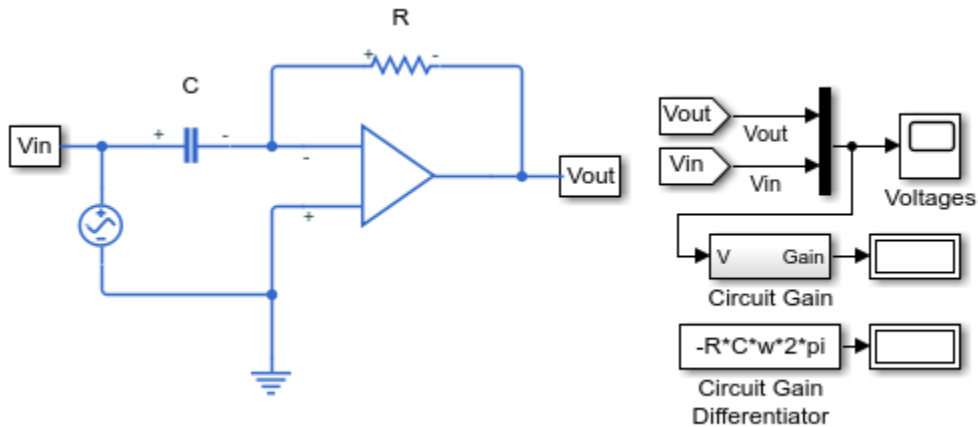


Simulate the model, and you will see that the output is minus the derivative of the input sinusoid.
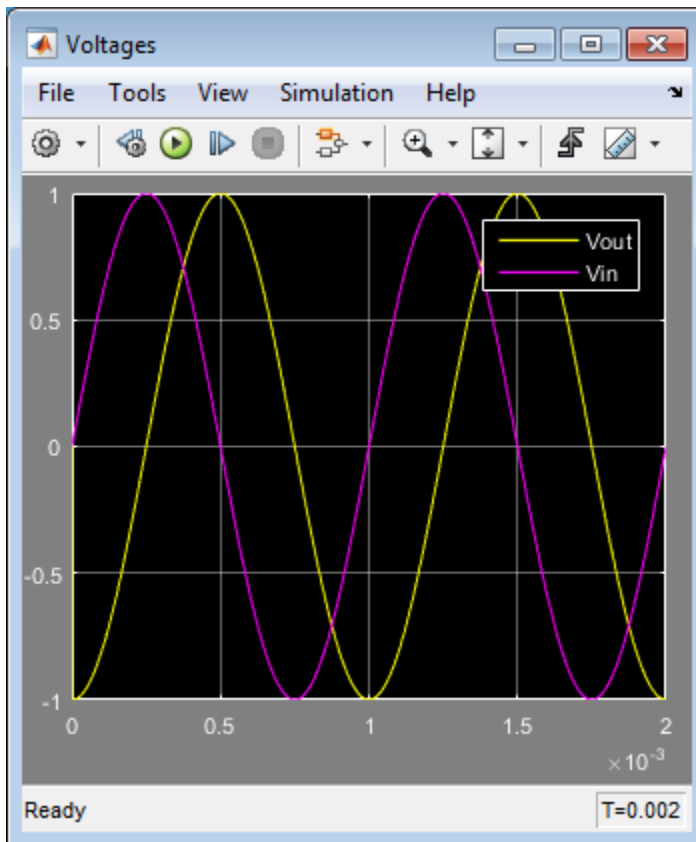
Now double-click the capacitor C block, and set the series resistance to zero. The model now runs very slowly and issues warnings about problems with transient initialization and step size control for transient solve.

The cause of the problems is that the circuit effectively connects the voltage source in parallel with the capacitor. This is because an ideal op-amp satisfies V+ = V- , where V+ and V- are the noninverting and inverting inputs, respectively. This is an example where it is not possible to replace the circuit with an equivalent simpler one, and a parasitic small resistance has to be introduced.

# Domain-Specific Line Styles

For improved readability of block diagrams, each Simscape domain uses a distinct default color and line style for the connection lines. Physical signal lines also have a distinct style and color.

Domain-specific line styles apply to the block icons as well. If all the block ports belong to the same domain, then the whole block icon assumes the line style and color of that domain. If a block has multiple port types, such as the Rotational Electromechanical Converter, then relevant parts of the block icon assume domain-specific line styles and colors.
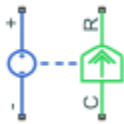


To view the line styles assigned to each domain, in the Simulink Toolstrip, on the **Debug** tab, select **Information Overlays > Simscape Legend**. The Simscape Line Styles Legend window opens, listing the line color assigned to each registered domain, the domain name, and the domain path. If you click a domain path link, the Simscape file for the corresponding domain opens in MATLAB Editor. For more information on domain paths and files, see "Foundation Domain Types and Directory Structure".



To turn off domain-specific line styles for a particular model, in the Simulink Toolstrip, on the **Debug** tab, select **Information Overlays > Simscape Domains**. This action toggles the **Simscape Domains** button off, and the block diagram display changes to black connection lines and block

icons, with physical ports visible at connection points. Repeatedly selecting the **Simscape Domains** button toggles the domain-specific line styles for this model on or off.

To turn off domain-specific line styles for all models, in the MATLAB Toolstrip, click **Preferences**. In the left pane of the Preferences dialog box, select **Simscape**, then clear the **Enable domain styles for all models** check box.
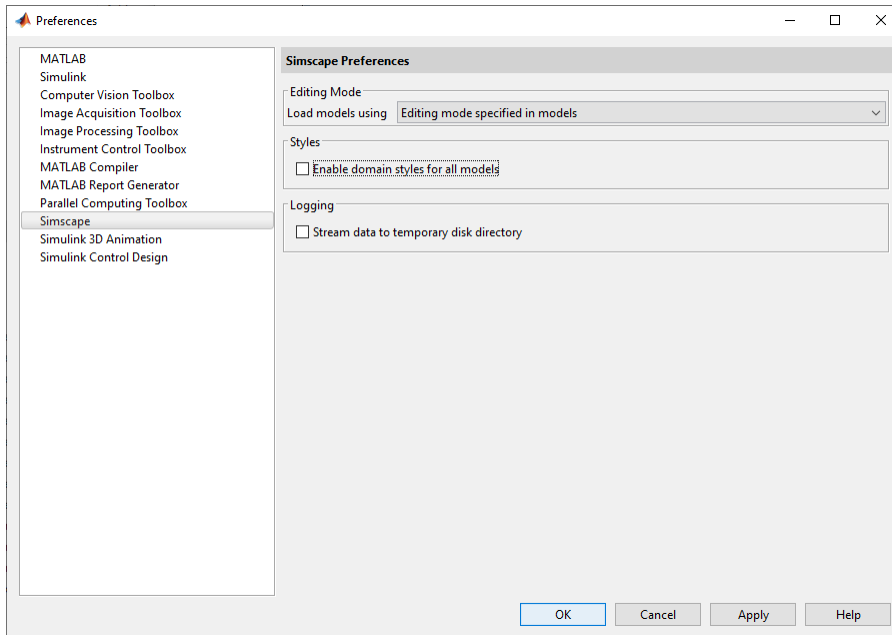
# Design Rigid Interface Specifications for Conserving Connections

| **In this section...** |
| --- |
| "Create Connection Bus Objects Using the Bus Editor" on page 1-36 |
| "Apply a Rigid Interface Specification to a Simscape Bus Block" on page 1-40 |

You can design rigid interface specifications for conserving connections and lock down connection names and types by using `Simulink.ConnectionBus` and `Simulink.ConnectionElement` objects. Then, when you apply a rigid specification to a Simscape Bus or Connection Port block, the block ports become typed by the interface and do not accept connections to a different domain type. This functionality helps you ensure the correct connection types within your model architecture. For example, if you design a 6-pin rigid interface and apply it to two Simscape Bus blocks in your model, these blocks can be connected to each other, but not to a block with a 9-pin interface.

Before you can apply a rigid interface specification to a Simscape Bus or Connection Port block, you need to have a `Simulink.ConnectionBus` object in your base workspace or a data dictionary. You can construct or modify these objects:

- Using MATLAB commands. For more information, see "Create Connection Bus and Connection Element Objects Using MATLAB Commands".
- Using the Simulink **Bus Editor**.
- Using the **Model Explorer**.

## Create Connection Bus Objects Using the Bus Editor

Before you can apply a rigid interface specification to a Simscape Bus or Connection Port block, you need to have a `Simulink.ConnectionBus` object in your base workspace or a data dictionary.

In this example, define a rigid interface with one mechanical translational and one electrical port by creating a `ConnectionBus` object named `MechElec`.

1   Open the Bus Editor. At the MATLAB command prompt, enter `buseditor`.

> **Tip** You can also open the Bus Editor directly from a Simscape Bus or Connection Port block dialog box. Under **Type Assistant**, set **Mode** to `Connection Bus object` and click the **Edit** button.

2   In the Bus Editor, click the **Connection Bus** button.

A `ConnectionBus` object with a default name appears in the middle pane and its default properties appear in the **Simulink.ConnectionBus** dialog pane.

3   Specify the name for the `ConnectionBus` object using the **Name** property. In this example, name the `ConnectionBus` object `MechElec`.

Optionally, add a description for the `ConnectionBus` object using the **Description** property.
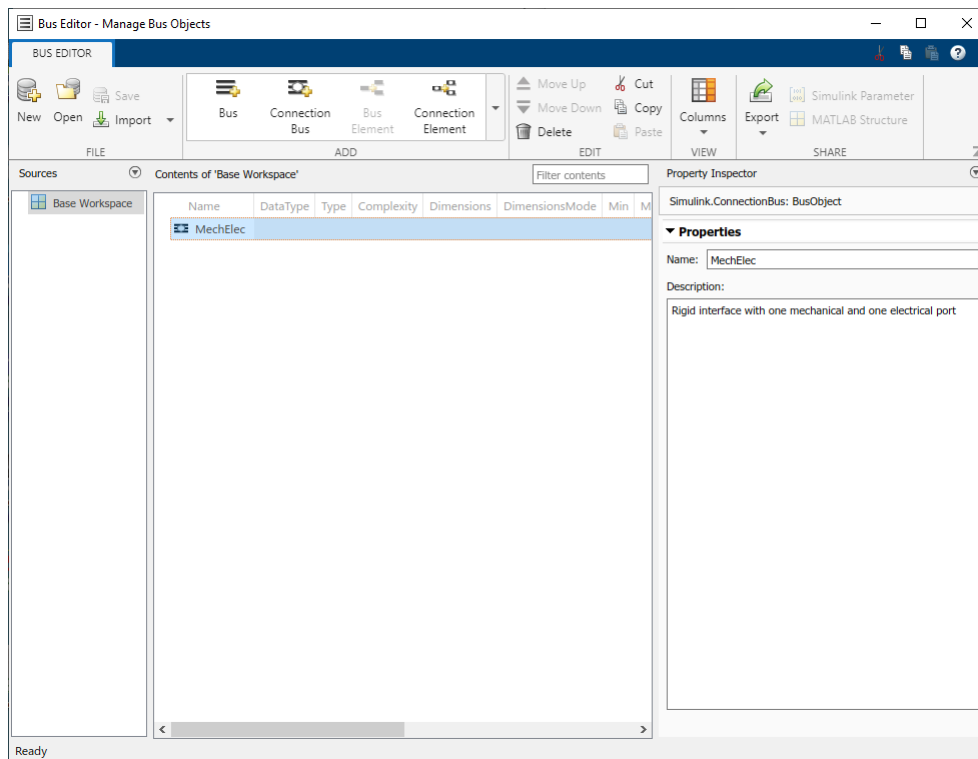
**Tip** By default, when you create objects or edit properties, all changes are applied automatically.

**4** Click the **Connection Element** button.

A connection element with a default name and default properties is created in the `MechElec ConnectionBus` object. The connection element appears in the middle pane nested under the `MechElec ConnectionBus` object.
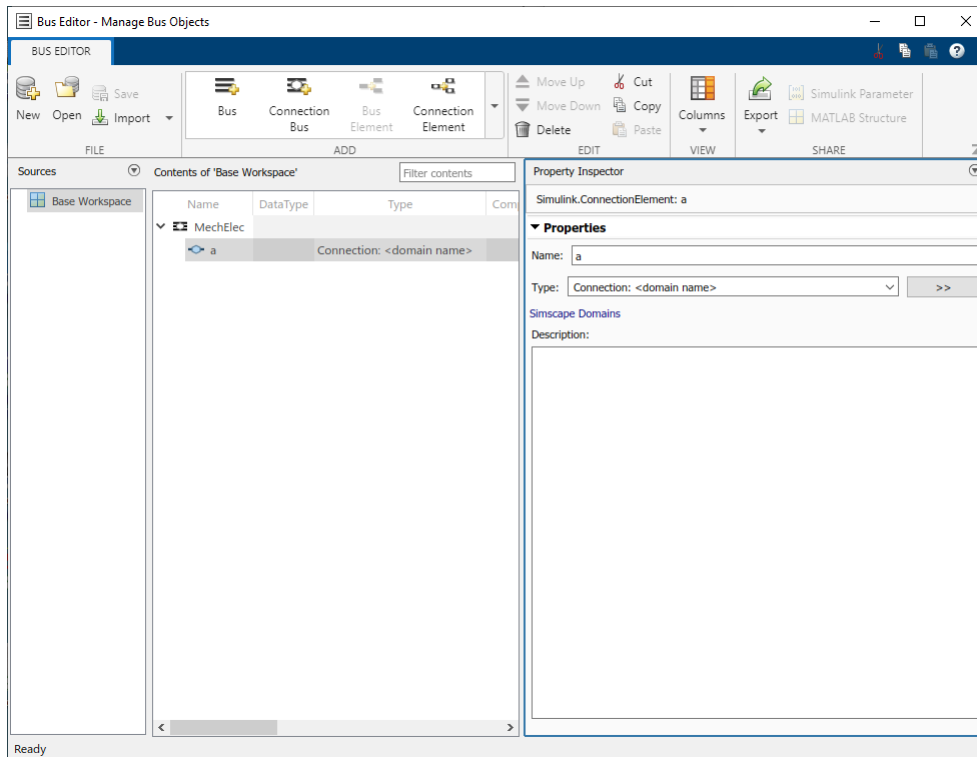
5   In the **Simulink.ConnectionElement** dialog pane, set **Name** to `mech`.

6   From the **Type** drop-down list, select `Connection:`
    `foundation.mechanical.translational.translational`, which corresponds to the
    Foundation mechanical translational domain.

---

**Tip**   For a list of Foundation domain types, see "Domain-Specific Line Styles" on page 1-34.

---

Optionally, add a description for the `ConnectionElement` object using the **Description**
property.
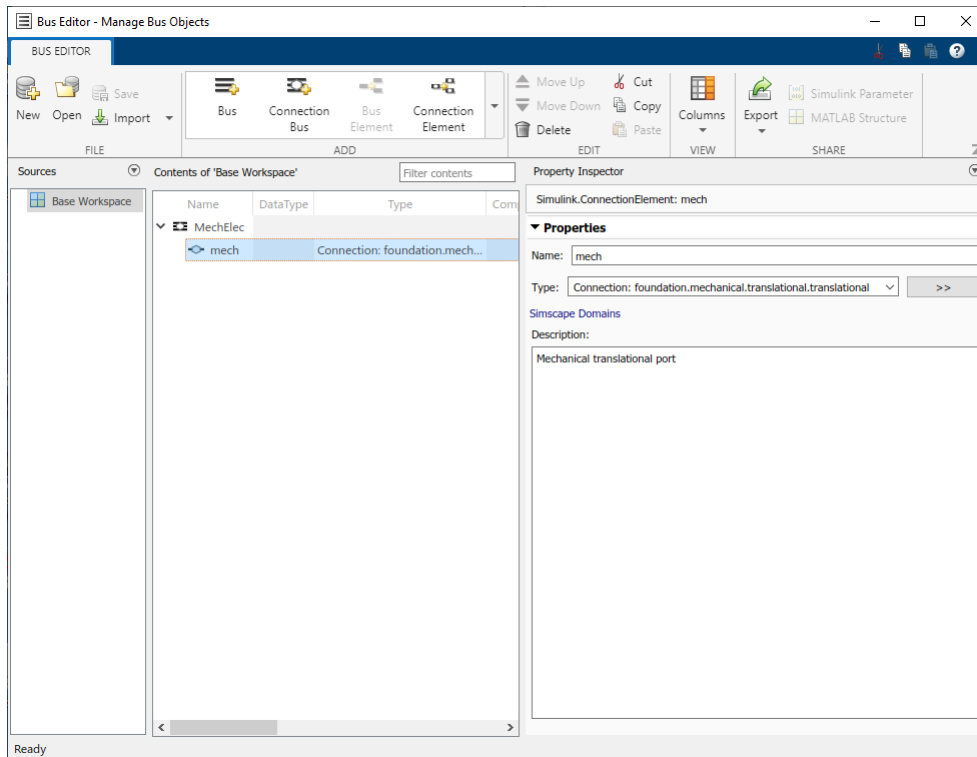
**7**   Similarly, create a `ConnectionElement` object named `elec` that corresponds to the electrical port.

You now have a `ConnectionBus` object with one translational mechanical and one electrical connection port.
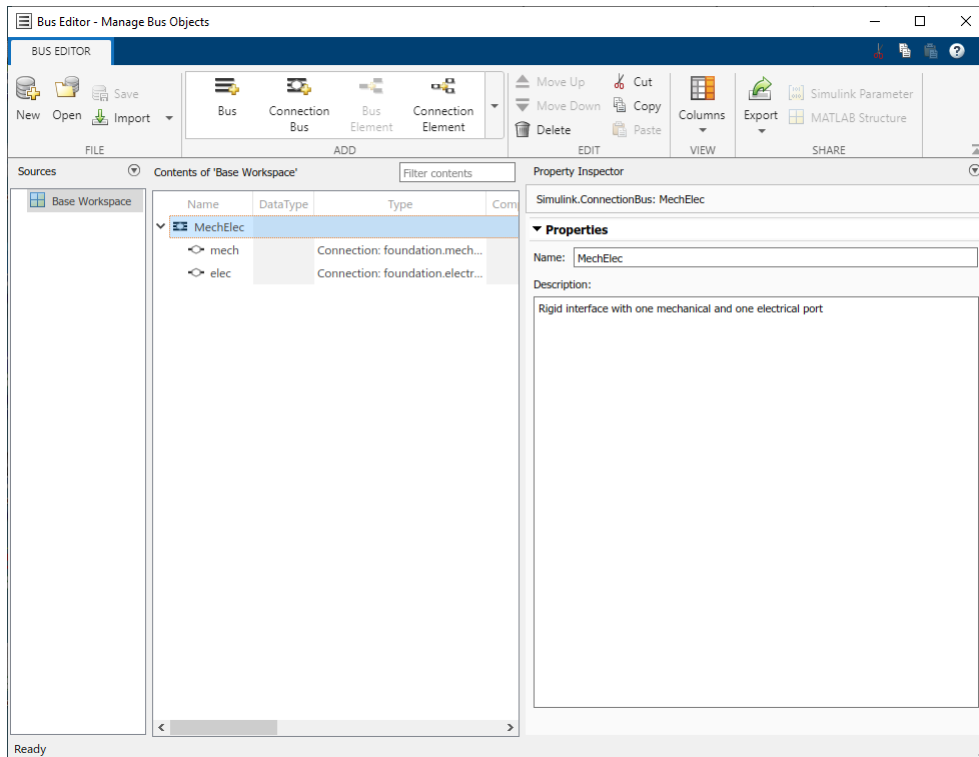
**8** Save the `ConnectionBus` object to a MAT-file called `MechElec`.

If you do not save a `ConnectionBus` object, then when you reopen a model that uses that `ConnectionBus` object, you need to recreate the object. You save and load `ConnectionBus` objects similar to `Simulink.Bus` objects. For more information, see "Determine How to Manage Simulink.Bus Objects".

## Apply a Rigid Interface Specification to a Simscape Bus Block

In this example, you apply the rigid interface specification, which you created in "Create Connection Bus Objects Using the Bus Editor" on page 1-36, to a Simscape Bus block. Before starting the exercise, make sure that you have the `MechElec` object in your base workspace or a data dictionary.

**1** Create a new model or open an existing model containing blocks with both electrical and mechanical translational ports.

**2** Open the **Simscape** > **Utilities** block library and add a Simscape Bus block to the model diagram.



**3** Double-click the Simscape Bus block.
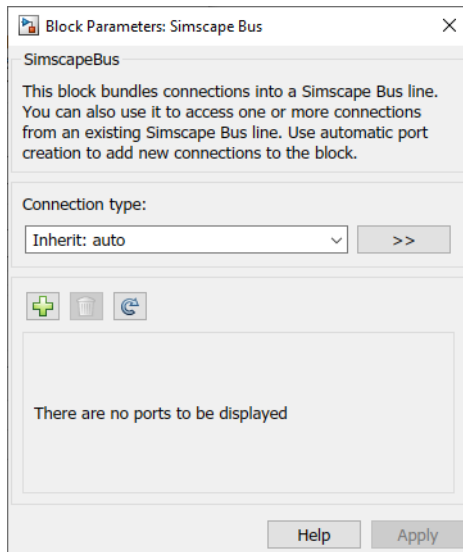
**4** From the drop-down list under **Connection type**, select `Bus: MechElec` and click **Apply**. Note that the block now has two children ports, `mech` and `elec`, according to the interface definition.
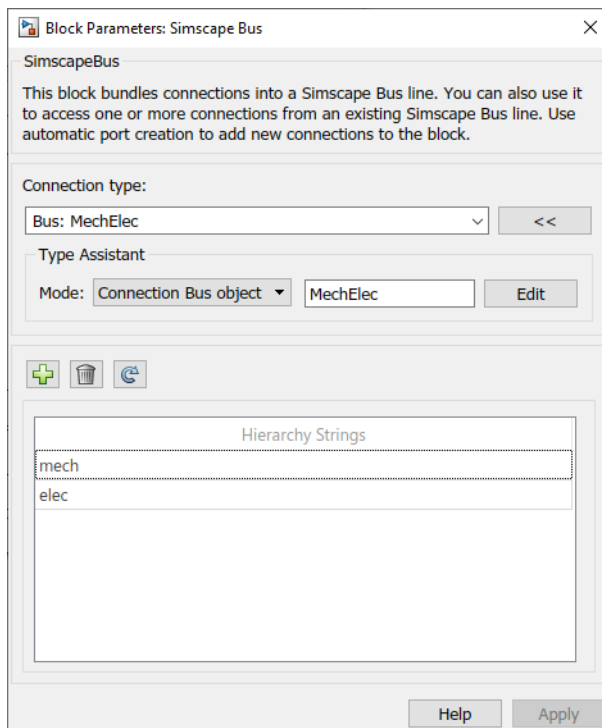


These ports appear under **Hierarchy Strings** in the block dialog, and the block icon changes as well to indicate that the block is a rigid bus.
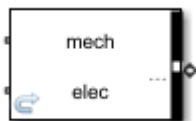
**5**    Add a second Simscape Bus block and apply the same interface definition to it. You can do this either by repeating the previous steps or by copying the first block.

**6**    Flip the second block to face the first one and connect the bus ports of the two blocks. You can connect these bus ports because the Simscape Bus blocks use the same interface definition, and therefore have identical children ports. The connection line style (double line) indicates a rigid bus connection.



**7**    You can now connect electrical and mechanical translational ports on other blocks in the model to the children ports of the Simscape Bus blocks, for example:



## See Also

**Blocks**
Connection Port | Simscape Bus

**Objects**
`Simulink.ConnectionBus` | `Simulink.ConnectionElement`

**Tools**
**Bus Editor**

## More About

- "Domain-Specific Line Styles" on page 1-34

# Plot Lookup Tables

You can plot lookup table data specified for the PS Lookup Table (1D) and PS Lookup Table (2D) blocks in your model. Plotting the tables lets you visualize the data before simulating the model, to make sure that the table is correct. The plots reflect tabulated data specified for the block, as well as the selected interpolation and extrapolation options.

If you change the underlying table data, plotting it again opens a new window. This way, you can compare the plots side by side and see how the block parameter values affect the resulting lookup function.

**1**    Create a new model and add a PS Lookup Table (1D) block. Specify the block parameters as shown.
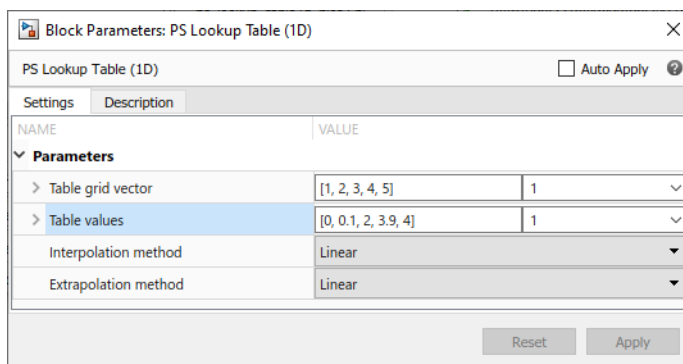


**2**    Right-click the block in your model. From the context menu, select **Foundation Library > Plot Table**.

A figure window containing the plot of the data opens.
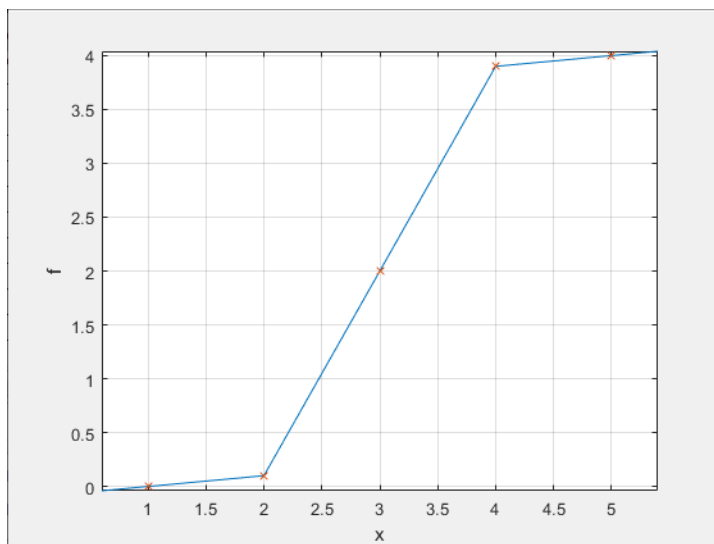


**3**    In the block dialog box, change the **Interpolation method** parameter value to `Smooth`.

**4**    Plot the data again by right-clicking the block and selecting **Foundation Library > Plot Table**.

A new figure window opens.
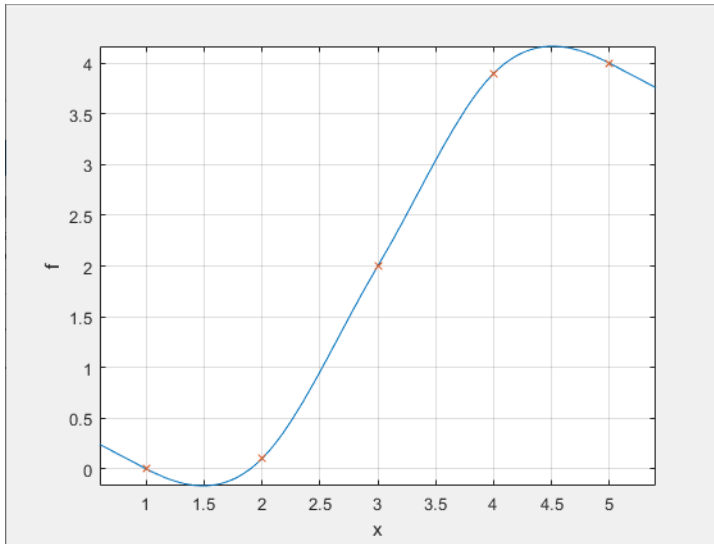
The curve shape has changed because of the new interpolation method.

**5**    In the block dialog box, change the **Extrapolation method** parameter value to `Nearest`.

**6**    Plot the data again by right-clicking the block and selecting **Foundation Library > Plot Table**.

A new figure window opens.
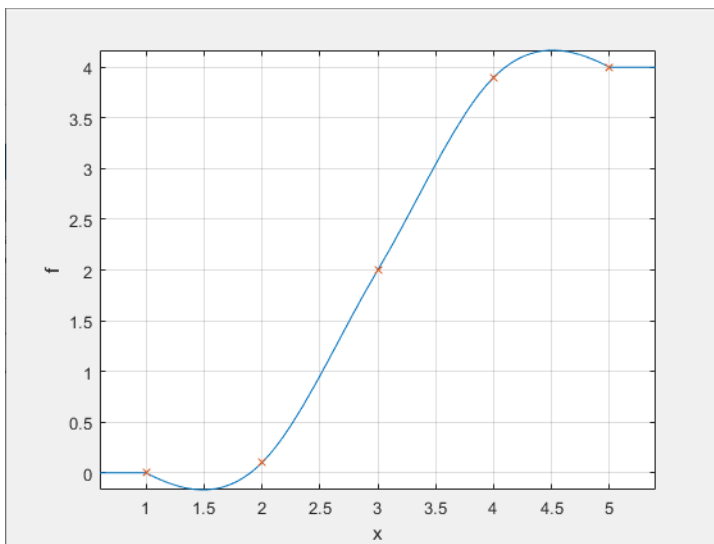


The curve shape within the table grid (between 1 and 5 along the *x*-axis) has not changed, but the new extrapolation method affects how the curve continues outside the specified range.

**Note** If you change the **Extrapolation method** parameter value to `Error`, there is no extrapolation region and the plot gets cut off at the first and last grid points.

## See Also

PS Lookup Table (1D) | PS Lookup Table (2D)

# Physical Signal Unit Propagation

Physical signals have units associated with the signal value. You specify the units along with the parameter values in the block dialogs and Property Inspector, and Simscape software performs the necessary unit conversion operations when solving a physical network. If the signal is a vector or a matrix, all its elements have the same unit. Unitless signals have their unit designated as 1.

Simscape blocks in the Physical Signals block library (PS blocks) allow you to graphically implement equations inside the physical network by performing math operations and other functions on physical signals. These blocks have untyped input and output ports, to facilitate unit propagation:

- The physical unit associated with the input signal of a PS block is propagated from the connected output signal.
- The physical unit associated with the output signal of a PS block is determined by the unit of the input signal and the equations inside the block. If a block performs a math operation, that operation is performed both on the value and the unit of the input physical signal.

For example, consider a PS Gain block connected to a Current Sensor output port. Then, the physical unit at the PS Gain input port is A. The physical unit at the PS Gain output port is the input signal unit multiplied by the unit of the **Gain** parameter:

- If the **Gain** parameter unit is 1 (unitless), then the output signal has the same unit as the input signal, that is, A.
- If the **Gain** parameter unit is V, then the output physical signal has the unit of W.

Similarly, the PS Product block multiplies both the values and units of the two input signals.
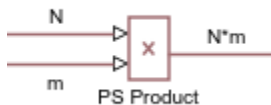


The PS Add and PS Subtract blocks perform addition and subtraction on the two input signals, and therefore, the physical signal units at the two input ports of these blocks must be commensurate. If the two input signals have the same unit, then the output signal unit has that unit as well. If the input signal units are commensurate, then the output signal unit is the fundamental unit for that dimension.
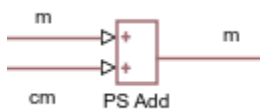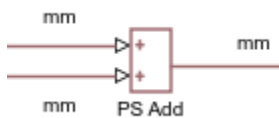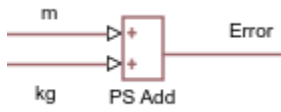
The PS Signal Specification block lets you explicitly specify the size and unit of a physical signal. Use this block when the signal size and unit cannot be determined implicitly, based on model connections.

## See Also

PS Signal Specification

## More About

*   "Upgrading Models with Legacy Physical Signal Blocks" on page 1-47

# Upgrading Models with Legacy Physical Signal Blocks

| In this section... |
| --- |
| "Example of an Automatic Upgrade" on page 1-49 |
| "Example of a Nonautomatic Upgrade" on page 1-50 |

In R2019a, all blocks in the Physical Signals library have been reimplemented with untyped inputs and outputs, to facilitate signal size and unit propagation. These new blocks do not automatically replace the respective legacy blocks in your model. To upgrade your blocks to the latest version, use the Upgrade Advisor.

After running the **Check and update outdated Simscape Physical Signal blocks** check, you get a list of links to the outdated blocks in the right pane of the Upgrade Advisor window. Clicking a link highlights the corresponding block in the model.

**Check and update outdated Simscape Physical Signal blocks**

Analysis (^Triggers Update Diagram)

Identify outdated Simscape Physical Signal blocks in a model and offer options to upgrade the block instances to the latest version. The latest version of Physical Signal blocks will propagate port unit and sizes.

Run This Check

Result:    ⚠ Warning

**Warning**
The model contains outdated Simscape Physical Signal blocks. The following blocks can be automatically upgraded:

- PSUpgradeExample/Thermal Model1/PS Product1
- PSUpgradeExample/Thermal Model2/PS Product2

**Recommended Action**
Use the upgrade link below to upgrade the above blocks to the latest version.
Upgrade

---

**Warning**

The following Simscape Physical Signal blocks, when switched to propagate signal units, will result in a compilation error or different answer. To update:

1. Click each action link 'Switch to new version' below
2. Visit affected blocks in the model and address the indicated upgrade issue

Typically changing block parameter units, or adding a PS Gain block to convert units as needed, will fix the issue. For more information see Upgrading Models with Legacy Physical Signal Blocks.

| Blocks | Upgrade Issue | Action |
|---|---|---|
| • PSUpgradeExample/Loss Model/PS Gain1<br>• PSUpgradeExample/Loss Model/PS Gain2<br>• .../Loss Model/PS Lookup Table (1D) | Error using **foundation.signal.lookup_tables.one_dimensional>** (line 35) ['PSUpgradeExample/Loss Model/PS Lookup Table (1D)']: Function, tablelookup, is wrong. Please check 1) whether input data points have correct sizes; 2) query values are scalar; 3) query values and table data have the commensurate units; and 4) constants or compile time parameters are passed to interpolation and extrapolation argument.<br><br>Caused by:<br>Argument 1 = [1x26 double]<br>Argument 2 = [1x26 double]<br>Argument 3 = {[1x1 double], 'rad/s'}<br>x = [1x26 double]<br>f = [1x26 double]<br>I = {[1x1 double], 'rad/s'}<br>extrap_method = int32(int32(1))<br>interp_method = int32(int32(1)) | Switch to new version |

Depending on your model, the links can be divided into multiple groups:

| Report Section | Upgrade Action |
|---|---|
| If the model contains outdated blocks that can be updated automatically, they are listed in the first section of the report, followed by the **Upgrade** link. | Click the **Upgrade** link under the list of block links. The software automatically replaces each of the listed blocks with its latest library version, while keeping all the parameter values and setting the parameter units, if appropriate. For more information, see "Example of an Automatic Upgrade" on page 1-49. |
| Sometimes, legacy blocks cannot be converted automatically because direct conversion would result in a compilation error or a different answer. These blocks are listed in a table, grouped by the underlying issue. Each row of the table contains:<br><br>1   A list of links to blocks affected by the issue<br><br>2   Issue description<br><br>3   A **Switch to new version** link | Review the table that groups the blocks based on the underlying issue. For each table row, click the **Switch to new version** link to convert all blocks listed in the first cell of this row, and then visit the affected blocks individually to resolve the issue. For more information, see "Example of a Nonautomatic Upgrade" on page 1-50. |

## Example of an Automatic Upgrade

If the Upgrade Advisor finds legacy Physical Signal blocks that can be updated automatically, it lists all these blocks in the first group of links, followed by the **Upgrade** link.

**Warning**

The model contains outdated Simscape Physical Signal blocks. The following blocks can be automatically upgraded:

- PSUpgradeExample/Thermal Model1/PS Product1
- PSUpgradeExample/Thermal Model2/PS Product2

**Recommended Action**

Use the upgrade link below to upgrade the above blocks to the latest version.

Upgrade

This is an example of a model that can be upgraded automatically.
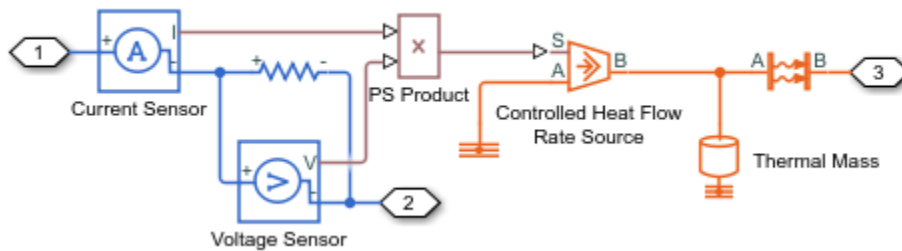


The legacy PS Product block does not propagate units. However, if you replace this block with the current version of the PS Product block, there is no issue with unit propagation. The first input signal, from the Current Sensor, is in A. The second input signal, from the Voltage Sensor, is in V. Their

product, the output signal, is in W, which is the expected unit at the input port **S** of the Controlled Heat Flow Rate Source block.

When you click the **Upgrade** link, the software automatically replaces the legacy PS Product block with its latest library version.

## Example of a Nonautomatic Upgrade

Sometimes, legacy blocks cannot be converted automatically because direct conversion would result in a compilation error or a different answer. In this case, you must inspect the affected blocks individually to resolve the issue and ensure that the model works as intended.

This is an example of a model that cannot be upgraded automatically.
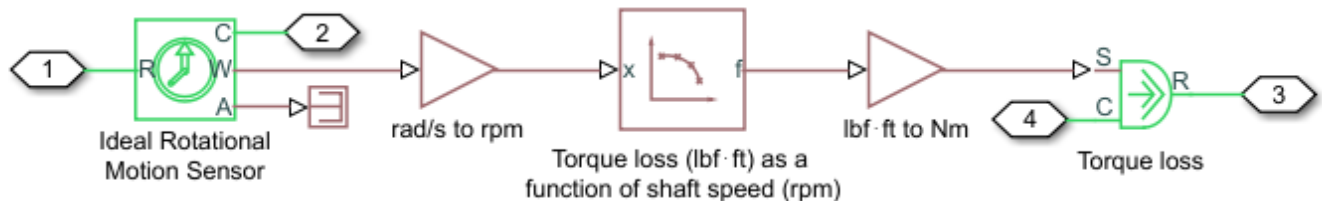


The PS Lookup Table (1D) block in this diagram contains tabulated data of torque loss, in lbf*ft, as a function of shaft speed, in rpm.
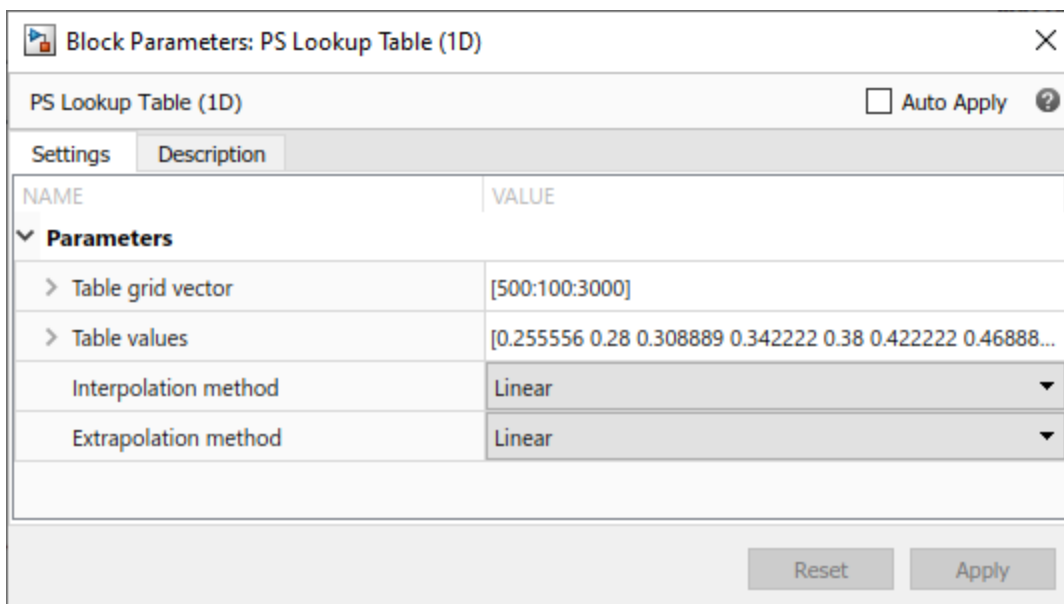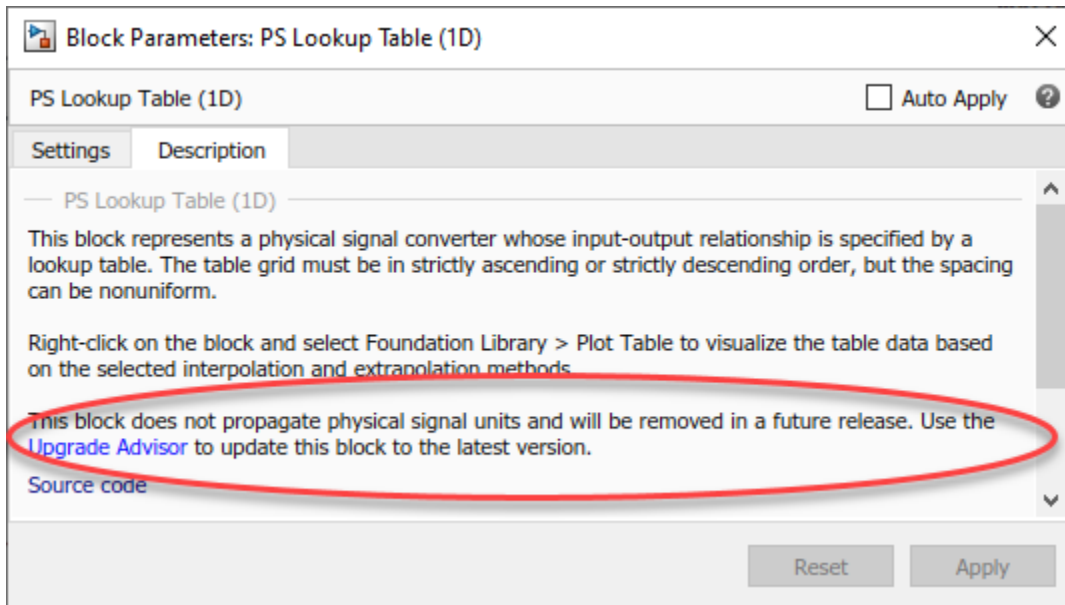
The output signal coming from the Ideal Rotational Motion Sensor block is in rad/s, and the input port **S** of the Ideal Torque Source block, which models the torque loss, expects the unit of N/m. Legacy Physical Signal blocks did not propagate units, therefore the model contains two PS Gain blocks, on each side of the PS Lookup Table (1D) block, to account for unit conversion. For example, the first PS Gain block provides the coefficient to convert rad/s to rpm.

When you run the **Check and update outdated Simscape Physical Signal blocks** check on this model, the Upgrade Advisor detects the unit mismatch but cannot determine whether the PS Gain blocks address the issue. Therefore, it lists all three blocks in the same row of the table, followed by the error description and the **Switch to new version** link.

When you click **Switch to new version**, the three legacy blocks are replaced with their respective latest versions. However, this does not resolve the unit mismatch issue. You have to address it manually.

The new Physical Signal blocks propagate units, and therefore you no longer need the two PS Gain blocks. The correct way to resolve the issue in this model is to delete these two blocks and set the proper units for the PS Lookup Table (1D) block parameters.



Ideal Rotational
Motion Sensor

Torque loss (lbf·ft) as a
function of shaft speed (rpm)

Torque loss



**1-53**
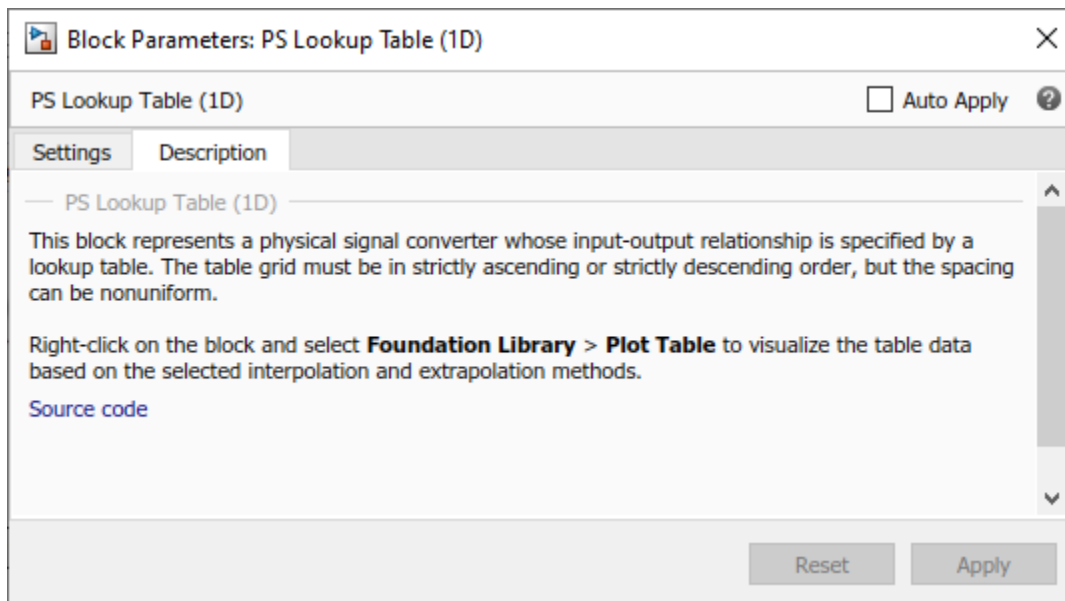
## See Also

## More About

- "Physical Signal Unit Propagation" on page 1-45
- "Model Upgrades"

# Connecting Simscape Networks to Simscape Multibody Joints

| In this section... |
|---|
| "How to Use Interface Blocks" on page 1-55 |
| "How to Pass Position Information" on page 1-57 |
| "How to Model Masses and Inertias" on page 1-59 |

You can model 3-D mechanical systems by using Simscape Multibody™ blocks representing bodies, joints, constraints, force elements, and sensors. These models can also include Simscape networks that represent hydraulic, electrical, pneumatic, and other physical systems. Examples of multidomain models that require connections between Simscape networks and Simscape Multibody blocks are:

- A mechanical system consisting of Simscape Multibody blocks that is powered by a hydraulic system and therefore needs to interface with hydraulic actuators from Simscape Fluids libraries
- A Simscape electrical circuit with a motor, where the motor is used to actuate a Simscape Multibody joint
- Modeling nonlinear springs, dampers, friction for Simscape Multibody joints

Simscape networks are one-dimensional, while Simscape Multibody provides 3-D modeling capabilities. Therefore, you cannot directly connect Simscape mechanical rotational and translational ports to Simscape Multibody blocks. You need to establish bidirectional connections between the Simscape portions of a block diagram and Simscape Multibody joints that involve both sensing and actuation, and in some cases must pass position information as well:

- Force and relative velocity must be equal across the translational interface. To make the connection more robust, sense the velocity of the joint and provide force actuation to the joint.
- Torque and relative angular velocity must be equal across the rotational interface. To make the connection more robust, sense the angular velocity of the joint and provide torque actuation to the joint.
- Simscape Multibody connections contain position information, while Simscape mechanical connections account only for relative velocity. Some Simscape blocks depend only on velocity, but certain Simscape blocks depend on velocity and position. If the behavior of a Simscape block depends on position or angle, its position or angle must be equal to the position or angle of the connected Simscape Multibody joint, both at initialization time and during simulation.

Anytime you need to connect a Simscape network to a Simscape Multibody joint, use the Translational Multibody Interface or Rotational Multibody Interface block and follow the steps in "How to Use Interface Blocks" on page 1-55. If the Simscape network contains blocks that depend on position or angle, additionally follow the steps in "How to Pass Position Information" on page 1-57.

## How to Use Interface Blocks

The Translational Multibody Interface and Rotational Multibody Interface blocks help you connect Simscape portions of a block diagram to Simscape Multibody joints:

- The Translational Multibody Interface block matches the force and relative velocity across the interface. You can connect it to any Simscape Multibody joint that has a prismatic primitive. The Translational Multibody Interface block has two mechanical translational ports, **C** and **R**, for

connections to the Simscape network, and two physical signal ports, **f** and **v**, that you connect to the respective actuation and sensing ports of the Simscape Multibody joint.

- The Rotational Multibody Interface block matches the torque and relative angular velocity across the interface. You can connect it to any Simscape Multibody joint that has a revolute primitive. The Rotational Multibody Interface block has two mechanical rotational ports, **C** and **R**, for connections to the Simscape network, and two physical signal ports, **t** and **w**, that you connect to the respective actuation and sensing ports of the Simscape Multibody joint.

This block diagram shows a Simscape mechanical translational network, containing Translational Friction and Translational Damper blocks. This network is connected to a Simscape Multibody Prismatic Joint block by using a Translational Multibody Interface block.
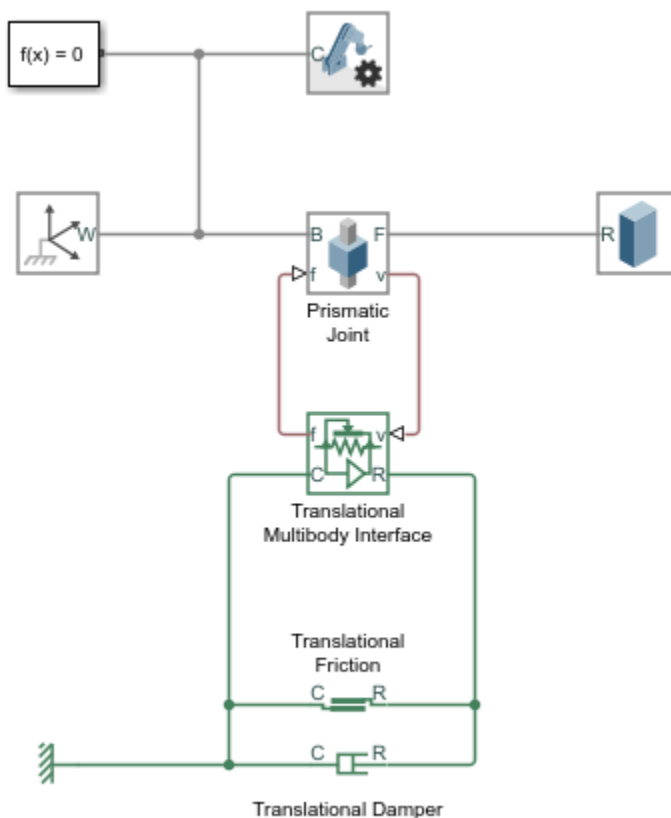


To use the Translational Multibody Interface block:

1. In the Prismatic Joint block, specify the actuation and sensing options for the prismatic joint primitive:

   - Under **Actuation**, set **Force** to `Provided by Input`, to enable the force actuation port **f**.
   - Under **Sensing**, select **Velocity**, to enable the velocity sensing port **v**.

   If the joint has multiple degrees of freedom, make sure that the selected actuation and sensing options correspond to the same degree of freedom.

2. Connect physical signal ports **f** and **v** of the Translational Multibody Interface block to ports **f** and **v** of the Prismatic Joint block.

**3**  Connect ports **C** and **R** of the Translational Multibody Interface block to the Simscape mechanical translational network. In this example, connect ports **C** and **R** of the Translational Friction and Translational Damper blocks to the respective ports of the Translational Multibody Interface block.

## How to Pass Position Information

Blocks like Translational Friction and Translational Damper do not require position information, and for these blocks the interface based on force and relative velocity is sufficient. Other blocks, like hydraulic actuators, require information on relative position, or relative angle, between their ports.

To connect these blocks to a Simscape Multibody joint:

**1**  Use the Translational Multibody Interface or Rotational Multibody Interface block to connect the Simscape network to the joint. Enable the actuation and velocity sensing ports on the joint, and connect the ports as described in "How to Use Interface Blocks" on page 1-55.

**2**  Additionally, enable the position sensing port **p** (or, for rotational degrees of freedom, port **q**) on the joint. If the joint has multiple degrees of freedom, make sure that the position sensing corresponds to the same degree of freedom as the actuation and velocity sensing options selected in Step 1.

**3**  On the actuator block, enable the position input port **p** (or, for rotational actuators, port **q**), by setting the **Interface displacement** (or the **Interface rotation**) parameter to `Provide input signal from Multibody joint`. Connect the input port on the actuator block to the respective sensing port of the Simscape Multibody joint.
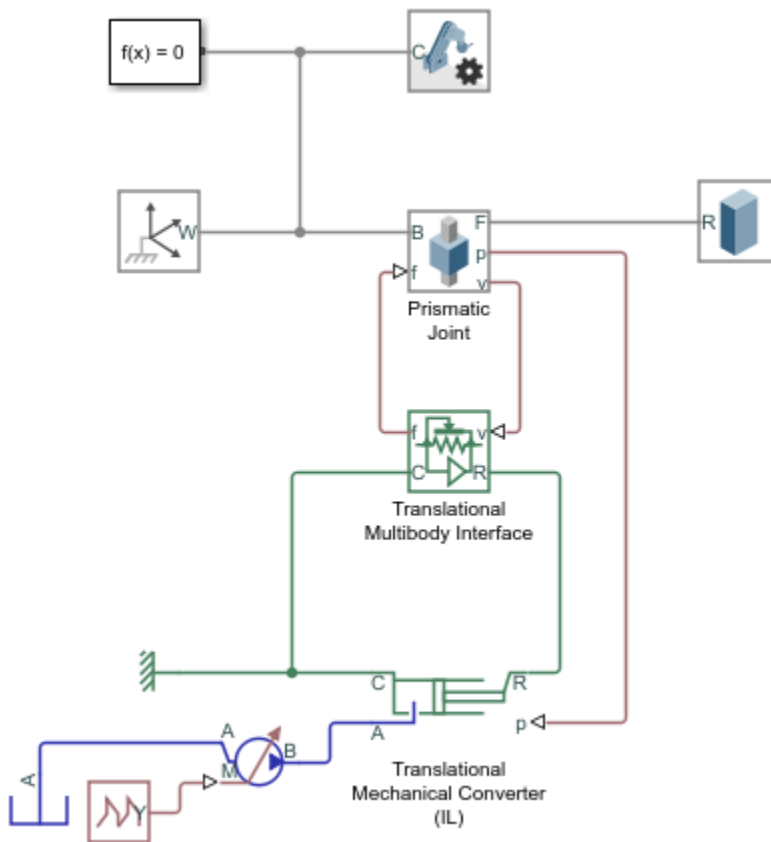
For example, to connect a Translational Mechanical Converter (IL) block to a Prismatic Joint block, as shown in the block diagram:

**1** In the Prismatic Joint block, specify the actuation and sensing options for the prismatic joint primitive:

- Under **Actuation**, set **Force** to `Provided by Input`, to enable the force actuation port **f**.
- Under **Sensing**, select **Velocity**, to enable the velocity sensing port **v**.
- Under **Sensing**, select **Position**, to enable the position sensing port **p**.

If the joint has multiple degrees of freedom, make sure that the selected actuation and sensing options correspond to the same degree of freedom.

**2** Connect physical signal ports **f** and **v** of the Translational Multibody Interface block to ports **f** and **v** of the Prismatic Joint block.

**3** Connect ports **C** and **R** of the Translational Multibody Interface block to ports **C** and **R** of the Translational Mechanical Converter (IL) block.

**4** In the Translational Mechanical Converter (IL) block, set the **Interface displacement** parameter to `Provide input signal from Multibody joint`, to enable the physical signal input port **p**.

**5** Connect the position sensing port **p** of the joint to the physical signal input port **p** of the Translational Mechanical Converter (IL) block.

For a detailed example of passing position information from a Multibody joint to hydraulic actuators, while also accounting for different mechanical orientation of the actuators, see "Modeling a Double-Acting Actuator" on page 1-60.

## How to Model Masses and Inertias

For models with Translational Multibody Interface or Rotational Multibody Interface blocks, it is recommended that you use Simscape Multibody blocks to model masses and inertias. The reason is that Simscape networks need to have a ground (reference) node, with all the masses and inertias in the network accelerating with respect to this node. In a Simscape Multibody joint, both the base and follower frames may be accelerating. Therefore, a mass or inertia in the Simscape network connected to a joint may not have the correct inertial reference.

## See Also

Rotational Multibody Interface | Translational Multibody Interface

## Related Examples

- "Modeling a Double-Acting Actuator" on page 1-60
- "Hydraulic Interface - Dump Trailer with Hydraulic Cylinder" (Simscape Multibody)

# Modeling a Double-Acting Actuator

This example shows how to model a double-acting actuator with Simscape Multibody and Simscape. Simscape Multibody models the mechanical system of the cylinder, and Simscape models the hydraulic system. You can use Translational Multibody Interface block to connect the two systems.

## Cylinder Model

The schematics show the cylinder in three configurations: half-retracted, fully retracted, and fully extended.
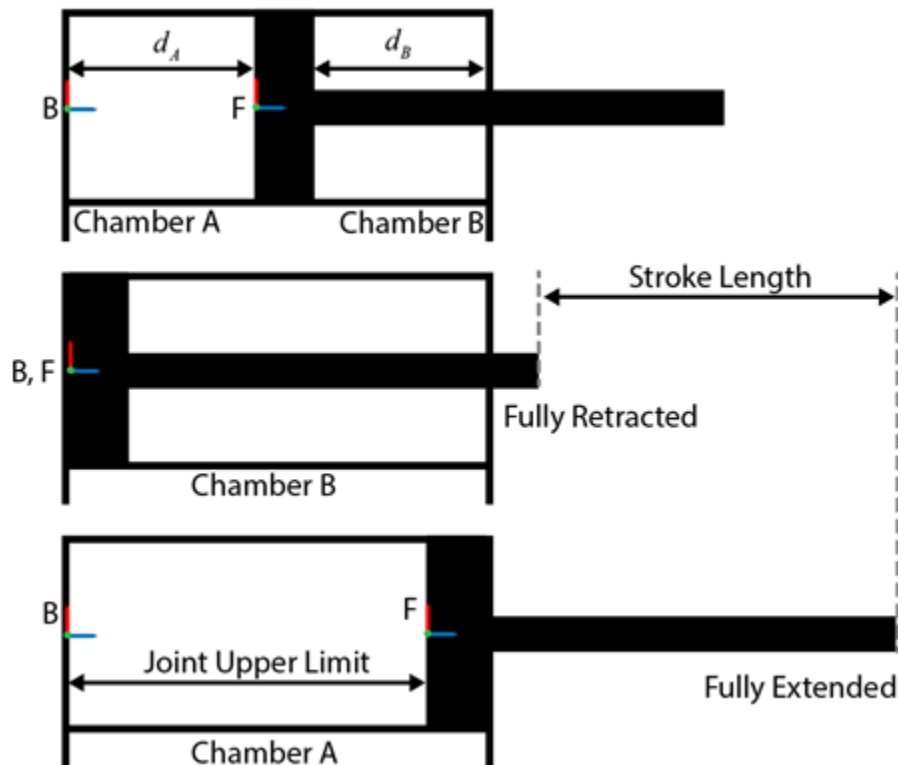


You can model the mechanical part of the system in Simscape Multibody. For example, to model the cylinder and piston, you can use the Revolved Solid and Cylindrical Solid blocks. To provide the translational degree of freedom for the piston, use a Prismatic Joint block. To easily calculate the lengths of the chambers A and B, you can attach the base frame (**B**) and follower frame (**F**) of the joint to the centers of the left inner surface of the chamber A and the left surface of the piston, respectively. Make sure that the *Z*-axes of the **B** and **F** frames are aligned and both point toward chamber B.

Using this convention, the length of the chamber A, $d_A$, is the position output of the joint block, and the length of the chamber B, $d_B$, equals the difference of the stroke length and $d_A$. To ensure that the piston does not travel beyond the ends of the cylinder, you can specify the upper and lower position limits of the Prismatic Joint block.

**Note** You must create custom frames before attaching the joint frames at desired locations. See "Custom Solid Frames" (Simscape Multibody) for more information.

This figure shows the block diagram of the double-acting cylinder model described above. The example uses two Translational Mechanical Converter (IL) blocks to model the chambers A and B of the cylinder. A Translational Multibody Interface block connects the Translational Mechanical Converter (IL) blocks to the Prismatic Joint block. See "Connecting Simscape Networks to Simscape Multibody Joints" on page 1-55 for details of how to use the Translational Multibody Interface block.
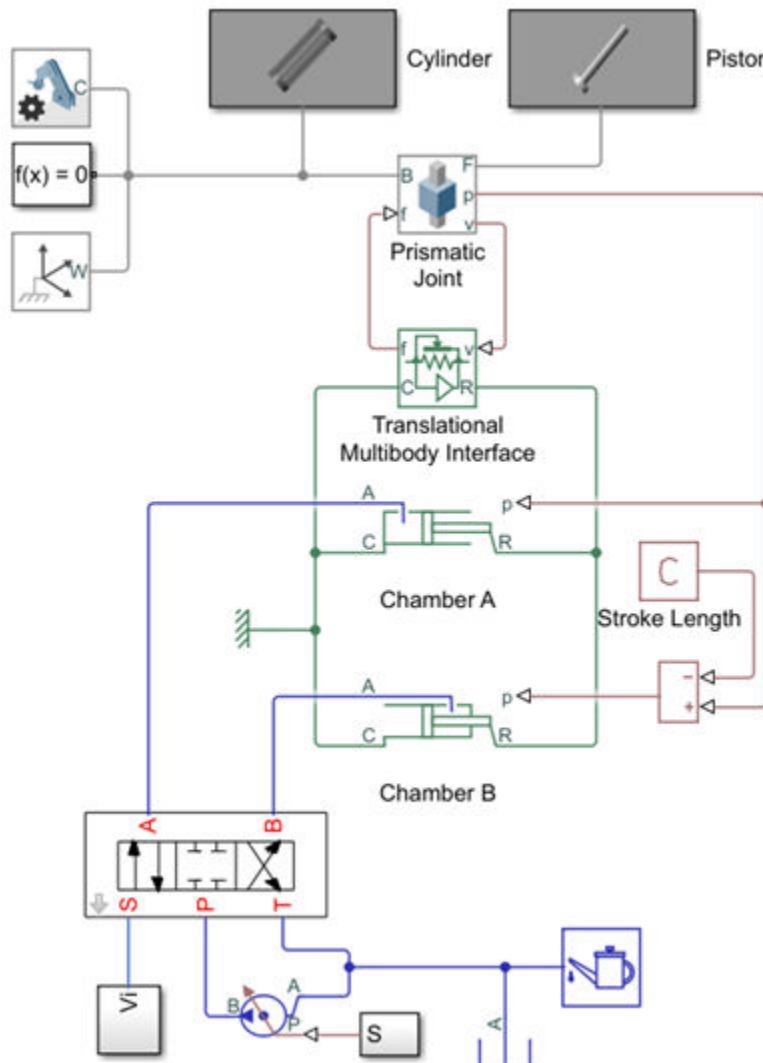


In chamber A, a pressure at port **A** moves the piston from 0 to stroke length. The displacement of the piston equals to the position output of the Prismatic Joint block. To model chamber A:

- Connect the ports **C** and **R** of the Translational Multibody Interface block to the ports **C** and **R** of the upper Translational Mechanical Converter (IL) block, respectively.
- In the Translational Mechanical Converter (IL) block, set **Interface displacement** to `Provide input signal from Multibody joint` to enable port **p**.
- Connect ports **p** of the Prismatic Joint block and Translational Mechanical Converter (IL) block.

In chamber B, the pressure at port **A** moves the piston from 0 to negative stroke length. Therefore, you need to change the default orientation and add an offset to the input position signal of the lower Translational Mechanical Converter (IL) block. The offset equals $-d_B$.

- In the lower Translational Mechanical Converter (IL) block, set **Mechanical orientation** to `Pressure at A causes negative displacement of R relative to C`.
- Set **Interface displacement** to `Provide input signal from Multibody joint` to enable the port **p**.
- Add a PS Constant block and specify the constant as the stroke length of the cylinder.
- Add a PS Subtract block, then connect it to the PS Constant and Prismatic Joint blocks as shown in the diagram. Note that the output of the PS Subtract block equals $-d_B$.

To limit the travel of the piston:

- In the Prismatic Joint block, under the **Limits** section, select the **Specify Lower Limit** and **Specify Upper Limit** parameters.
- Under the **Specify Lower Limit** section, the **Bound** parameter indicates the minimum distance between the frames **B** and **F** of the joint block. In this example, specify the **Bound** parameter as 0 m.
- Under **Specify Upper Limit**, the **Bound** parameter indicates the maximum distance between frames **B** and **F** of the Prismatic Joint block. In this example, specify the **Bound** parameter as the stroke length of the cylinder.

Double-acting cylinder systems can be used in many applications. For example, a dump trailer can use a double-acting cylinder to actuate a scissor hoist mechanism that raises and lowers the dump bed. See "Hydraulic Interface - Dump Trailer with Hydraulic Cylinder" (Simscape Multibody) for more details. Under the mask of the Double-Acting Hydraulic Cylinder block in the scissor hoist are the mechanical and hydraulic subsystems, which are connected by a Translational Multibody Interface block. The Prismatic Joint block in the mechanical subsystem provides the position information to the two actuators in the hydraulic subsystem, Head Chamber and Bottom Chamber, that have the opposite settings of the **Mechanical orientation** parameter. The Constant block C provides a position signal offset to the Head Chamber actuator, because when the Bottom Chamber is at dead volume (where position $p = 0$), the Head Chamber is at maximum stroke.

## See Also
Rotational Multibody Interface | Translational Multibody Interface

## More About
- "Connecting Simscape Networks to Simscape Multibody Joints" on page 1-55
- "Hydraulic Interface - Dump Trailer with Hydraulic Cylinder" (Simscape Multibody)

# Isothermal Liquid Models

- "Modeling Isothermal Liquid Systems" on page 2-2
- "Isothermal Liquid Modeling Options" on page 2-4
- "Upgrading Hydraulic Models To Use Isothermal Liquid Blocks" on page 2-9

# Modeling Isothermal Liquid Systems

| In this section... |
| --- |
| "Intended Applications" on page 2-2 |
| "Network Variables" on page 2-2 |
| "Blocks with Fluid Volume" on page 2-2 |
| "Reference Node and Grounding Rules" on page 2-3 |

## Intended Applications

The Isothermal Liquid library contains basic elements, such as orifices, chambers, and hydromechanical converters, as well as sensors and sources. Use these blocks to model hydraulic systems where the liquid temperature is assumed constant, for applications such as:

- Hydraulic actuation of mechanical systems
- Isothermal liquid transport through pipe networks
- Hydraulic turbines for power generation

In the isothermal liquid domain, the working fluid is assumed to be at a constant temperature during simulation. To model thermodynamic effects, use the blocks from the Thermal Liquid and Two-Phase Fluid libraries. For more information on fluid domains available for Simscape multidomain system modeling, see "Simscape Fluid Domains" on page 1-14.

You specify the properties of the working fluid in a connected loop by using the Isothermal Liquid Properties (IL) block. This block lets you choose between several modeling options (see "Isothermal Liquid Modeling Options" on page 2-4).

## Network Variables

In the isothermal liquid domain, the Across variable is pressure and the Through variable is mass flow rate. Note that these choices result in a pseudo-bond graph, because the product of pressure and mass flow rate is not power.

## Blocks with Fluid Volume

Components in the isothermal liquid domain are modeled using control volumes. The control volume encompasses the fluid inside the component and separates it from the surrounding environment and other components. Mass flow rates across the control surface are represented by ports. The fluid volume inside the component is represented using an internal node, which provides the pressure inside the component. This internal node is not visible, but you can access its parameters and variables using Simscape data logging. For more information, see About Simulation Data Logging on page 13-2.

The following blocks in the Isothermal Liquid library are modeled as components with a fluid volume. In the case of Reservoir (IL), the volume is assumed to be infinitely large.

| Block | Gas Volume |
| --- | --- |
| Constant Volume Chamber (IL) | Finite |

| Block | Gas Volume |
|---|---|
| Pipe (IL) | Finite |
| Rotational Mechanical Converter (IL) | Finite |
| Translational Mechanical Converter (IL) | Finite |
| Reservoir (IL) | Infinite |

Other components have relatively small volumes, so that fluid entering the component spends negligible time inside the component before exiting. These components are considered quasi-steady-state and they do not have an internal node.

## Reference Node and Grounding Rules

Unlike mechanical and electrical domains, where each topologically distinct circuit within a domain must contain at least one reference block, isothermal liquid networks have different grounding rules.

Blocks with a fluid volume contain an internal node, which provides the pressure inside the component and therefore serves as a reference node for the isothermal liquid network. Each connected isothermal liquid network must have at least one reference node. This means that each connected isothermal liquid network must have at least one of the blocks listed in "Blocks with Fluid Volume" on page 2-2. In other words, an isothermal liquid network that contains no fluid volume is an invalid network.

## See Also

## More About

# Isothermal Liquid Modeling Options

| **In this section...** |
|---|
| "Common Equation Symbols" on page 2-4 |
| "Ideal Fluid" on page 2-5 |
| "Constant Amount of Entrained Air" on page 2-6 |
| "Air Dissolution Is On" on page 2-7 |

In the isothermal liquid domain, the working fluid is a mixture of liquid and a small amount of entrained air. Entrained air is the relative amount of nondissolved gas trapped in the fluid. You can control the liquid and air properties separately:

- You can specify zero amount of entrained air. Fluid with zero entrained air is ideal, that is, it represents pure liquid.

- Mixture bulk modulus can be either constant or a linear function of pressure.

- If the mixture contains nonzero amount of entrained air, then you can select the air dissolution model. If air dissolution is off, the amount of entrained air is constant. If air dissolution is on, entrained air can dissolve into liquid.

Equations used to compute various fluid properties depend on the selected model.

| **Zero Entrained Air** | **Constant Entrained Air** | **Air Dissolution Is On** |
|---|---|---|
| "Constant Bulk Modulus" on page 2-5 | "Constant Bulk Modulus" on page 2-6 | "Constant Bulk Modulus" on page 2-7 |
| "Bulk Modulus Is a Linear Function of Pressure" on page 2-5 | "Bulk Modulus Is a Linear Function of Pressure" on page 2-6 | "Bulk Modulus Is a Linear Function of Pressure" on page 2-7 |

Use the Isothermal Liquid Properties (IL) block to select the appropriate modeling options.

## Common Equation Symbols

The equations use these symbols:

| | |
|---|---|
| $p$ | Liquid pressure |
| $p_0$ | Reference pressure |
| $p_{min}$ | Minimum valid pressure |
| $p_c$ | Critical pressure |
| $\beta_{mix}$ | Mixture isothermal bulk modulus |
| $\beta_L$ | Pure liquid bulk modulus |
| $\beta_{L0}$ | Pure liquid bulk modulus at reference pressure $p_0$ |
| $K_{\beta p}$ | Proportionality coefficient when bulk modulus is a linear function of pressure |
| $\rho_{mix}$ | Mixture density |
| $\rho_L$ | Pure liquid density |

| | |
|---|---|
| $\rho_{L0}$ | Pure liquid density at reference pressure $p_0$ |
| $\rho_g$ | Air density |
| $\rho_{g0}$ | Air density at reference pressure $p_0$ |
| $\theta(p)$ | Fraction of entrained air as a function of pressure |
| $\alpha$ | Volumetric fraction of entrained air in the fluid mixture |
| $\alpha_0$ | Volumetric fraction of entrained air in the fluid mixture at reference pressure $p_0$ |
| $V$ | Total mixture volume |
| $V_L$ | Pure liquid volume |
| $V_{L0}$ | Pure liquid volume at reference pressure $p_0$ |
| $V_g$ | Air volume |
| $V_{g0}$ | Air volume at reference pressure $p_0$ |
| $M$ | Total mixture mass |
| $M_L$ | Pure liquid mass |
| $M_{L0}$ | Pure liquid mass at reference pressure $p_0$ |
| $M_g$ | Air mass |
| $M_{g0}$ | Air mass at reference pressure $p_0$ |
| $n$ | Air polytropic index |

## Ideal Fluid

Fluid with zero entrained air is ideal, that is, it represents pure liquid.

### Constant Bulk Modulus

For this model, the defining equations are:

- Mixture density

$$\rho_{mix} = \rho_{L0} \cdot e^{(p - p_0)/\beta_L}$$

- Mixture density partial derivative

$$\frac{\partial \rho_{mix}}{\partial p} = \frac{\rho_{L0}}{\beta_L} e^{(p - p_0)/\beta_L}$$

- Mixture bulk modulus

$$\beta_{mix} = \beta_L$$

### Bulk Modulus Is a Linear Function of Pressure

For this model, the defining equations are:

- Mixture density

$$\rho_{mix} = \rho_{L0} \left( 1 + \frac{K_{\beta p}}{\beta_{L0}} (p - p_0) \right)^{1/K_{\beta p}}$$

- Mixture density partial derivative

$$\frac{\partial \rho_{mix}}{\partial p} = \frac{\rho_{L0}}{\beta_{L0}} \left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p - p_0)\right)^{1/K_{\beta p} - 1}$$

- Mixture bulk modulus

$$\beta_{mix} = \beta_{L0} + K_{\beta p}(p - p_0)$$

## Constant Amount of Entrained Air

In practice, the working fluid contains a small amount of entrained air. This set of models assumes that the amount of entrained air remains constant during simulation.

### Constant Bulk Modulus

For this model, the defining equations are:

- Mixture density

$$\rho_{mix} = \frac{\rho_{L0} + \left(\frac{\alpha_0}{1 - \alpha_0}\right)\rho_{g0}}{e^{-(p - p_0)/\beta_L} + \left(\frac{\alpha_0}{1 - \alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}}$$

- Mixture density partial derivative

$$\frac{\partial \rho_{mix}}{\partial p} = \frac{\left(\rho_{L0} + \left(\frac{\alpha_0}{1 - \alpha_0}\right)\rho_{g0}\right)\left(\frac{1}{\beta_L}e^{-(p - p_0)/\beta_L} + \frac{1}{n}\left(\frac{\alpha_0}{1 - \alpha_0}\right)\left(\frac{p_0^{1/n}}{p^{1/n + 1}}\right)\right)}{\left(e^{-(p - p_0)/\beta_L} + \left(\frac{\alpha_0}{1 - \alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\right)^2}$$

- Mixture bulk modulus

$$\beta_{mix} = \beta_L \frac{e^{-(p - p_0)/\beta_L} + \left(\frac{\alpha_0}{1 - \alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}}{e^{-(p - p_0)/\beta_L} + \beta_L\frac{1}{n}\left(\frac{\alpha_0}{1 - \alpha_0}\right)\left(\frac{p_0^{1/n}}{p^{1/n + 1}}\right)}$$

### Bulk Modulus Is a Linear Function of Pressure

For this model, the defining equations are:

- Mixture density

$$\rho_{mix} = \frac{\rho_{L0} + \left(\frac{\alpha_0}{1 - \alpha_0}\right)\rho_{g0}}{\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p - p_0)\right)^{-1/K_{\beta p}} + \left(\frac{\alpha_0}{1 - \alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}}$$

- Mixture density partial derivative

$$\frac{\partial \rho_{mix}}{\partial p} = \frac{\left(\rho_{L0} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\rho_{g0}\right)\left[\frac{1}{\beta_L}\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}-1} + \frac{1}{n}\left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0^{1/n}}{p^{1/n+1}}\right)\right]}{\left(\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\right)^2}$$

- Mixture bulk modulus

$$\beta_{mix} = \beta_L \frac{\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}}{\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}-1} + \beta_L\frac{1}{n}\left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0^{1/n}}{p^{1/n+1}}\right)}$$

## Air Dissolution Is On

This set of models lets you account for the air dissolution effects during simulation:

- At pressures less than or equal to the reference pressure, $p_0$ (which is assumed to be equal to atmospheric pressure), all the air is assumed to be entrained.
- At pressures equal or higher than pressure $p_c$, all the entrained air has been dissolved into the liquid.
- At pressures between $p_0$ and $p_c$, the volumetric fraction of entrained air that is not lost to dissolution, $\theta(p)$, is a function of the pressure.

### Constant Bulk Modulus

For this model, the defining equations are:

- Mixture density

$$\rho_{mix} = \frac{\rho_{L0} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\rho_{g0}}{e^{-(p-p_0)/\beta_L} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\theta(p)}$$

- Mixture density partial derivative

$$\frac{\partial \rho_{mix}}{\partial p} = \frac{\left(\rho_{L0} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\rho_{g0}\right)\left[\frac{1}{\beta_L}e^{-(p-p_0)/\beta_L} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\left(\frac{\theta(p)}{np} - \frac{d\theta(p)}{dp}\right)\right]}{\left(e^{-(p-p_0)/\beta_L} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\theta(p)\right)^2}$$

- Mixture bulk modulus

$$\beta_{mix} = \beta_L \frac{e^{-(p-p_0)/\beta_L} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\theta(p)}{e^{-(p-p_0)/\beta_L} + \beta_L\left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\left(\frac{\theta(p)}{np} - \frac{d\theta(p)}{dp}\right)}$$

### Bulk Modulus Is a Linear Function of Pressure

For this model, the defining equations are:

- Mixture density

$$\rho_{mix} = \frac{\rho_{L0} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\rho_{g0}}{\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\theta(p)}$$

- Mixture density partial derivative

$$\frac{\partial\rho_{mix}}{\partial p} = \frac{\left(\rho_{L0} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\rho_{g0}\right)\left(\frac{1}{\beta_L}\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}-1} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\left(\frac{\theta(p)}{np} - \frac{d\theta(p)}{dp}\right)\right)}{\left(\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\theta(p)\right)^2}$$

- Mixture bulk modulus

$$\beta_{mix} = \beta_L \frac{\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}} + \left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\theta(p)}{\left(1 + \frac{K_{\beta p}}{\beta_{L0}}(p-p_0)\right)^{-1/K_{\beta p}-1} + \beta_L\left(\frac{\alpha_0}{1-\alpha_0}\right)\left(\frac{p_0}{p}\right)^{1/n}\left(\frac{\theta(p)}{np} - \frac{d\theta(p)}{dp}\right)}$$

## References

[1] Gholizadeh, Hossein, Richard Burton, and Greg Schoenau. "Fluid Bulk Modulus: Comparison of Low Pressure Models." *International Journal of Fluid Power* 13, no. 1 (January 2012): 7–16. https://doi.org/10.1080/14399776.2012.10781042.

## See Also
Isothermal Liquid Properties (IL)

## More About
- "Modeling Isothermal Liquid Systems" on page 2-2

# Upgrading Hydraulic Models To Use Isothermal Liquid Blocks

| In this section... |
| --- |
| "Advantages of Using Isothermal Liquid Blocks" on page 2-9 |
| "Upgrade Considerations" on page 2-10 |
| "Upgrading Your Models" on page 2-12 |

In R2020a, Isothermal Liquid blocks were introduced to model hydraulic systems where the working fluid temperature remains constant during simulation. These blocks are intended for the same types of applications as the Hydraulic blocks in the Foundation library, or the Hydraulics (Isothermal) blocks in the Fluids library (which are available with a Simscape Fluids license). All of the hydraulic blocks in your models continue to work as before. However, Isothermal Liquid library blocks provide improved usability, increased accuracy, and enhanced simulation robustness.

## Advantages of Using Isothermal Liquid Blocks

Using Isothermal Liquid library blocks to model isothermal hydraulic systems provides improved usability, increased accuracy, and enhanced simulation robustness:

- The streamlined library is easier to use because it combines blocks that perform the same function and adds blocks that have improved parameterization options. For example, the Laminar Leakage (IL) block provides a choice of different cross-sectional geometries. Each of the sources can be configured as either controlled or constant, with the Flow Rate Source (IL) block providing the options for generating either mass flow rate or volumetric flow rate. Similarly, the Flow Rate Sensor (IL) block lets you measure mass flow rate, volumetric flow rate, or both. The Pressure Sensor (IL) block provides a choice of built-in measurements (including pressure difference, absolute pressure, or gauge pressure) and does not require additional reference blocks to account for absolute zero.

- In the Hydraulic library, only certain blocks (the ones with compressibility effect) model density as a function of pressure, while the other blocks use constant density (which is specified as a global parameter of the working fluid). In contrast, all of the blocks in the Isothermal Liquid library account for fluid density being a function of pressure. The Isothermal Liquid library makes it easy to specify your working fluid by selecting the bulk modulus model and the desired option for modeling the amount of entrained air. These selections are then reflected in all of the block equations, which increases the accuracy of the simulation.

- In the isothermal liquid domain, the Across and Through variables are absolute pressure and mass flow rate (in the hydraulic domain, the variables are gauge pressure and volumetric flow rate). Using mass flow rate as the Through variable (instead of volumetric flow rate) reduces the potential for small errors in mass conservation to accumulate over time due to the conversion between mass and volumetric quantities, and thus results in increased accuracy.

- Unlike the Hydraulic library blocks, the pipe and actuator blocks in the Isothermal Liquid library account for fluid compressibility by default, which reduces the likelihood of dry nodes and makes the simulation more robust. This also results in improved usability because you do not need to add extra Constant Volume Chamber blocks to your model to mitigate the effect of dry nodes.

- All block equations in the Isothermal Liquid library have been optimized to provide smooth transitions, reduce zero-crossings, and improve handling of zero flow and flow reversal. These improvements increase simulation robustness.

# Upgrade Considerations

The Isothermal Liquid block library is structured similar to other fluid domains, such as the Thermal Liquid library. These libraries have been streamlined to take advantage of the latest Simscape language features, such as conditional port visibility, and to improve block usability. As a result, there is often no one-to-one correspondence between the Isothermal Liquid and Hydraulic library blocks and you cannot easily substitute one for another.

The `hydraulicToIsothermalLiquid` conversion tool facilitates model upgrades by automatically replacing Hydraulic blocks with the equivalent Isothermal Liquid blocks, while trying to preserve the parameter values and connections between the blocks, where possible. After the conversion process, the tool saves the converted model under a new name and generates an HTML report that lists any issues encountered during the conversion process. You can review the HTML report and manually fix the remaining issues.

When the Isothermal Liquid block port locations are different from the original Hydraulic block, the tool places the new block in a subsystem to minimize rerouting the connection lines in the block diagram. In most cases, the subsystem port locations match the original ports of the Hydraulic block, and therefore the original layout of the block diagram is preserved. After the conversion process, you can move the new block out of the subsystem and manually reroute the connection lines, if desired.

After the upgrade, the simulation results for the original and the converted model might be slightly different. This difference can be more pronounced at higher pressures. You can use the Simulation Data Inspector to compare the results.

You can use the `hydraulicToIsothermalLiquid` tool to convert any type of a block diagram system, such as a model, subsystem, or library. For more information, see "Upgrading Systems with Enabled Library Links, Model References, and Subsystem References" on page 2-19.

### Fluid Properties

The default fluid properties in the isothermal liquid domain differ from the hydraulic domain. If your original model contains a Custom Hydraulic Fluid block, the conversion tool automatically replaces it with an Isothermal Liquid Properties (IL) block and sets its parameter values to match the original fluid properties.

Similarly, if your model contains a Hydraulic Fluid block, which is available with Simscape Fluids libraries, the tool automatically replaces this block either with a Simscape Fluids Isothermal Liquid Predefined Properties (IL) block (if that block contains a matching predefined fluid) or with an Isothermal Liquid Properties (IL) block, setting its parameter values to match the original fluid properties. For more information, see "Upgrading Simscape Fluids Models Containing Hydraulics (Isothermal) Blocks" (Simscape Fluids).

If there is no fluid properties block attached to a circuit, the blocks in this circuit use the default fluid. Because the default fluid properties in the isothermal liquid domain differ from the hydraulic domain, the conversion tool issues a warning if there is no Custom Hydraulic Fluid block or Hydraulic Fluid block anywhere in the model. However, the conversion tool does not detect situations where the model has multiple hydraulic circuits, some of which do not contain a block specifying fluid properties.
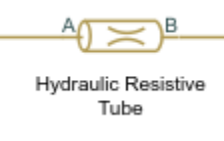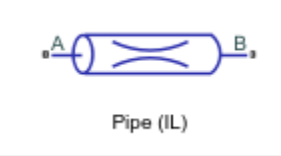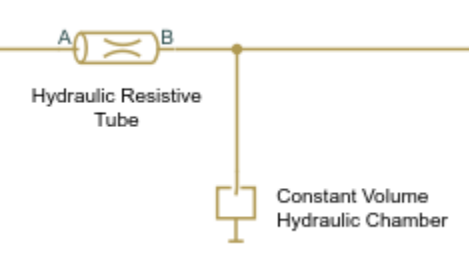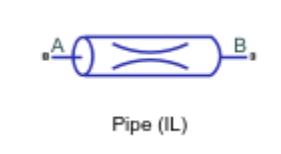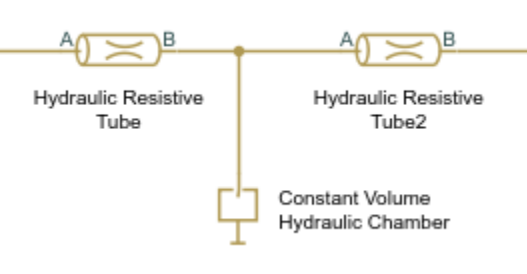
After running the conversion tool, inspect your model for circuits that do not contain an Isothermal Liquid Properties (IL) or Isothermal Liquid Predefined Properties (IL) block. Add the Isothermal Liquid Properties (IL) block manually and adjust its parameters, as needed. To match the default hydraulic fluid properties, set the following block parameter values:
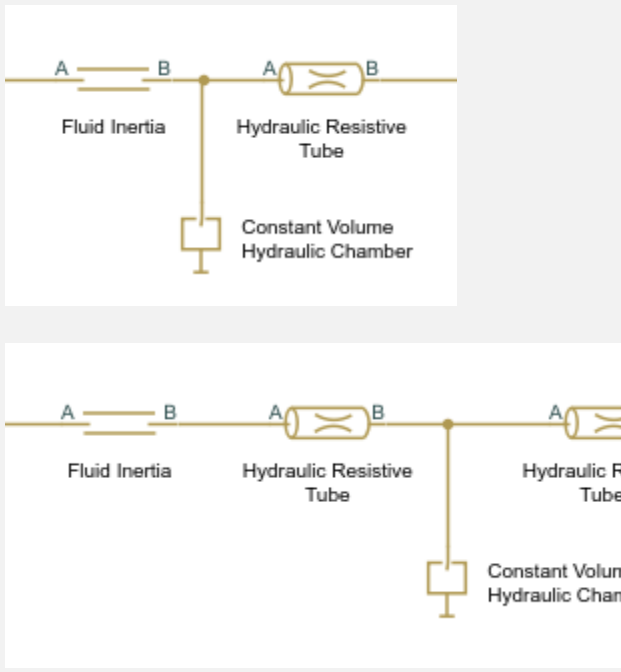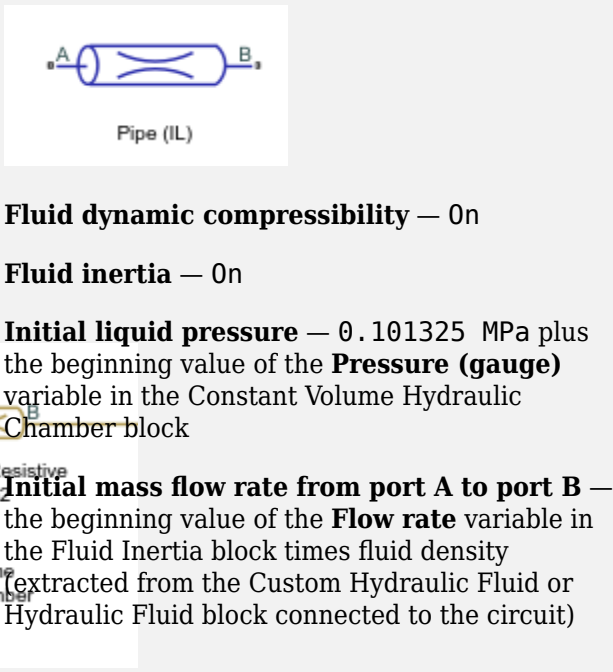
- **Liquid density at atmospheric pressure (no entrained air)**: `850 kg/m^3`
- **Liquid isothermal bulk modulus at atmospheric pressure (no entrained air)**: `0.8e9 Pa`
- **Kinematic viscosity at atmospheric pressure**: `18e-6 m^2/s`
- Use the default values for other block parameters.

**Pipes**

The Hydraulic Resistive Tube block accounts only for the pressure drop due to the viscous friction force and ignores the inertia and dynamic compressibility effects on the fluid pressure. In hydraulic models, this block is often combined with Constant Volume Hydraulic Chamber and Fluid Inertia blocks to model fluid compressibility and inertia. The conversion tool handles each block individually. However, after running the conversion tool, you can replace the whole group of blocks with a single Pipe (IL) block and configure it to model dynamic compressibility and fluid inertia effects, as needed.

The table shows some examples of Hydraulic block configurations and matching parameterizations of the replacement Pipe (IL) block.

| Old Model | New Model |
|---|---|
|  | <br><br>**Fluid dynamic compressibility** — `Off` |
| <br><br> | <br><br>**Fluid dynamic compressibility** — `On`<br><br>**Fluid inertia** — `Off`<br><br>**Initial liquid pressure** — `0.101325` MPa plus the beginning value of the **Pressure (gauge)** variable in the Constant Volume Hydraulic Chamber block |

| Old Model | New Model |
|---|---|
|  |  |
| | **Fluid dynamic compressibility** — On |
| | **Fluid inertia** — On |
| | **Initial liquid pressure** — 0.101325 MPa plus the beginning value of the **Pressure (gauge)** variable in the Constant Volume Hydraulic Chamber block |
|  | **Initial mass flow rate from port A to port B** — the beginning value of the **Flow rate** variable in the Fluid Inertia block times fluid density (extracted from the Custom Hydraulic Fluid or Hydraulic Fluid block connected to the circuit) |

**Translational Mechanical Converter**

The direction of the interface force in the Translational Hydro-Mechanical Converter block is the opposite of the force in the Translational Mechanical Converter (IL) block. In the Hydraulic block, the positive direction of the force is from port **C** to port **R**, while in the Isothermal Liquid and other fluids libraries, the positive interface force is from port **R** to port **C**.

If you are logging this variable, modify your code to change the sign. The name of the variable has also changed, from f in Hydraulic block to `interface_force` in the Isothermal Liquid block.

## Upgrading Your Models

The `hydraulicToIsothermalLiquid` conversion tool replaces Hydraulic blocks in your model with equivalent Isothermal Liquid blocks.

The table lists the Hydraulic blocks in the Foundation library and the replacement Isothermal Liquid blocks. For information on blocks from the Fluids > Hydraulics (Isothermal) library, see "Upgrading Simscape Fluids Models Containing Hydraulics (Isothermal) Blocks" (Simscape Fluids).

**Block Substitution**

| Old Model | New Model |
|---|---|
| Constant Area Hydraulic Orifice | Local Restriction (IL)<br><br>**Restriction type** — Fixed |
| Constant Volume Hydraulic Chamber | Constant Volume Chamber (IL) |
| Fluid Inertia | This block does not map directly to Isothermal Liquid libraries. The conversion tool comments the block out. To preserve the diagram layout, the conversion tool places the commented out block in a subsystem and connects the subsystem ports to bypass the block.<br><br>The conversion tool also creates a corresponding entry in the **Removed Blocks** section of the HTML report.<br><br>For information on how you can model fluid inertia using the Pipe (IL) block, see "Pipes" on page 2-11. |
| Hydraulic Cap | This block does not map to Isothermal Liquid libraries because you no longer need to terminate unconnected conserving ports in your model.<br><br>The conversion tool comments the block out, removes the connection line, and sets the Through variable at the port formerly connected to this block to 0.<br><br>The conversion tool also creates a corresponding entry in the **Removed Blocks** section of the HTML report.<br><br>If you used the Hydraulic Cap block to set the initial value for the **Pressure** variable, consider modifying the initial pressure value in the block formerly connected to the Hydraulic Cap block. |
| Hydraulic Piston Chamber | This block does not map to Isothermal Liquid libraries because you can model fluid compressibility directly in the mechanical converter blocks.<br><br>The conversion tool comments the block out and creates a corresponding entry in the **Removed Blocks** section of the HTML report. |
| Hydraulic Reference | Reservoir (IL) |
| Hydraulic Resistive Tube | Pipe (IL) |
| Infinite Hydraulic Resistance | Infinite Flow Resistance (IL) |
| Linear Hydraulic Resistance | Laminar Leakage (IL) |
| Rotational Hydro-Mechanical Converter | Rotational Mechanical Converter (IL) |
| Translational Hydro-Mechanical Converter | Translational Mechanical Converter (IL) |

| Old Model | New Model |
|---|---|
| Variable Area Hydraulic Orifice | Local Restriction (IL)<br><br>**Restriction type** — `Variable` |
| Variable Hydraulic Chamber | This block does not map to Isothermal Liquid libraries because you can model fluid compressibility directly in the mechanical converter blocks.<br><br>The conversion tool comments the block out and creates a corresponding entry in the **Removed Blocks** section of the HTML report. |
| Hydraulic Flow Rate Sensor | Flow Rate Sensor (IL) |
| Hydraulic Pressure Sensor | Pressure Sensor (IL) |
| Hydraulic Constant Flow Rate Source | Flow Rate Source (IL)<br><br>**Source type** — `Constant`<br><br>**Flow rate type** — `Volumetric flow rate` |
| Hydraulic Constant Mass Flow Rate Source | Flow Rate Source (IL)<br><br>**Source type** — `Constant`<br><br>**Flow rate type** — `Mass flow rate` |
| Hydraulic Constant Pressure Source | Pressure Source (IL)<br><br>**Source type** — `Constant` |
| Hydraulic Flow Rate Source | Flow Rate Source (IL)<br><br>**Source type** — `Controlled`<br><br>**Flow rate type** — `Volumetric flow rate` |
| Hydraulic Mass Flow Rate Source | Flow Rate Source (IL)<br><br>**Source type** — `Controlled`<br><br>**Flow rate type** — `Mass flow rate` |
| Hydraulic Pressure Source | Pressure Source (IL)<br><br>**Source type** — `Controlled` |
| Custom Hydraulic Fluid | Isothermal Liquid Properties (IL) |

**Broken Connections and Unconverted Blocks**

The tool converts the blocks from the Foundation > Hydraulic library and from the Fluids > Hydraulics (Isothermal) library, which is available with Simscape Fluids. However, the tool does not convert custom blocks with hydraulic ports. The tool also does not convert commented out blocks and referenced subsystems.

The **Broken Connections** section of the HTML report is an indicator of unconverted blocks in your model. Broken connections result if the tool fails to convert a hydraulic block and then tries to

connect it to the successfully converted Isothermal Liquid blocks. In this case, the conversion tool highlights the broken connections in the block diagram. The **Broken Connections** section of the HTML report contains hyperlinks to the Isothermal Liquid blocks with broken connections and a hyperlink to the Interface (H-IL) block. You can use this block to restore the broken connections.

**Conversion Messages**

The `hydraulicToIsothermalLiquid` conversion tool tries to preserve the block parameter values, where possible. Sometimes seamless automatic conversion is not possible, and you might have to adjust the parameter values manually or consider using a different modeling option. The table lists examples of conversion messages in the HTML report, explanations, and recommended actions. The actual messages you get might be slightly different depending on the parameter values in your model.

| Message | Reason | Suggested Actions |
|---|---|---|
| Add an Isothermal Liquid Properties (IL) block to specify the liquid properties. Default properties in the Isothermal Liquid domain differ from the Hydraulic domain. | The model does not contain a Custom Hydraulic Fluid block or a Hydraulic Fluid block. | For details and suggested actions, see "Fluid Properties" on page 2-10. |
| Beginning value of Flow rate removed. Adjustment of model initial conditions may be required. | The Constant Volume Hydraulic Chamber block in the original model lets you specify initialization priority and target for the **Volumetric flow rate into chamber** variable. The replacement Constant Volume Chamber (IL) block does not expose this variable for initialization.<br><br>If the variable in the original Hydraulic block had the initialization priority of **None**, the behavior of the new block is the same and no action is necessary.<br><br>The conversion tool issues the warning only if the initialization priority is either **High** or **Low**. | To see the initialization priority and beginning value for the variable in question, open the dialog box for the original Hydraulic block and look at the **Initial Targets** section.<br><br>Use the Variable Viewer to compare the initialization results. |

| Message | Reason | Suggested Actions |
|---|---|---|
| Beginning values of Diameter increase and Flow rate removed. Adjustment of model initial conditions may be required. | The Constant Volume Hydraulic Chamber block in the original model has the **Chamber wall type** parameter set to `Compliant`. In this configuration, the block lets you specify initialization priority and targets for the **Diameter increase** and **Volumetric flow rate into chamber** variables. The replacement Constant Volume Chamber (IL) block does not expose these variables for initialization.<br><br>If the variables in the original Hydraulic block had the initialization priority of **None**, the behavior of the new block is the same and no action is necessary.<br><br>The conversion tool issues the warning only for variables with initialization priority of either **High** or **Low**. If only one of the two variables had its initialization priority set to **High** or **Low**, then the message mentions only that variable. | To see the initialization priority and beginning value for the variable in question, open the dialog box for the original Hydraulic block and look at the **Initial Targets** section.<br><br>Use the Variable Viewer to compare the initialization results. |

| Message | Reason | Suggested Actions |
|---|---|---|
| Beginning values of Flow rate and Pressure differential removed. Adjustment of model initial conditions may be required. | Several Hydraulic blocks, such as orifices or hydro-mechanical converters, have the **Initial Targets** section, where you can specify initialization priority and targets for the **Flow rate** and **Pressure differential** variables. The equivalent Isothermal Liquid blocks do not expose these variables for initialization.<br><br>If the variables in the original Hydraulic block had the initialization priority of **None**, the behavior of the new block is the same and no action is necessary.<br><br>The conversion tool issues the warning only for variables with initialization priority of either **High** or **Low**. If only one of the two variables had its initialization priority set to **High** or **Low**, then the message mentions only that variable. | To see the initialization priority and beginning value for the variable in question, open the dialog box for the original Hydraulic block and look at the **Initial Targets** section.<br><br>Use the Variable Viewer to compare the initialization results. |
| Block uses Dead volume of 1e-4 m^3. Adjustment of Dead volume may be required. | The Rotational Hydro-Mechanical Converter or Translational Hydro-Mechanical Converter block in the original model has the **Compressibility** parameter set to `Off`. When compressibility is off, hydro-mechanical converters do not account for liquid volume and do not expose the **Dead volume** parameter. The equivalent Isothermal Liquid converter blocks log liquid volume even when compressibility is off. | The conversion tool sets the **Dead volume** parameter in the replacement Isothermal Liquid block to the same value as in the original Hydraulic block. (The original Hydraulic block used this parameter only with compressibility on.)<br><br>The tool also prints the parameter value, for your convenience. Adjust this value, if desired. |

| Message | Reason | Suggested Actions |
|---|---|---|
| Block uses Interface initial displacement of 0 m. Adjustment of Interface initial displacement may be required. | The Translational Hydro-Mechanical Converter block in the original model has the **Compressibility** parameter set to `Off`. Hydraulic translational converters expose the **Piston initial position** parameter only when compressibility is on. The equivalent Isothermal Liquid converter blocks let you specify initial displacement of the interface even when compressibility is off. | The conversion tool sets the **Interface initial displacement** parameter in the replacement Isothermal Liquid block to the value of the **Piston initial position** parameter in the original Hydraulic block (even though this parameter was not used with compressibility off.)<br><br>The tool also prints the parameter value for your convenience. Adjust this value, if desired. |
| Block uses Interface initial rotation of 0 rad. Adjustment of Interface initial rotation may be required. | The Rotational Hydro-Mechanical Converter block in the original model has the **Compressibility** parameter set to `Off`. Hydraulic rotational converters expose the **Shaft initial angle** parameter only when compressibility is on. The equivalent Isothermal Liquid converter blocks let you specify initial rotation of the interface even when compressibility is off. | The conversion tool sets the **Interface initial rotation** parameter in the replacement Isothermal Liquid block to the value of the **Shaft initial angle** parameter in the original Hydraulic block (even though this parameter was not used with compressibility off.)<br><br>The tool also prints the parameter value for your convenience. Adjust this value, if desired. |
| Chamber specification set to rigid. To model flexible volume, consider using the Simscape Fluids Pipe (IL) block. | The Constant Volume Hydraulic Chamber block in the original model has the **Chamber wall type** parameter set to `Compliant`. The replacement Constant Volume Chamber (IL) block does not have this option. | If you have a Simscape Fluids license, you can use the Pipe (IL) block from the Fluids > Isothermal Liquid > Pipes & Fittings library as a replacement. |
| Consider adjusting the initial pressure in a previously connected block. | The Hydraulic Cap block in the original model has been removed. | For details and suggested actions, see Block Substitution. |
| Consider modeling fluid compressibility with a Translational Mechanical Converter (IL) block. | The Variable Hydraulic Chamber or Hydraulic Piston Chamber block in the original model has been removed. | For details and suggested actions, see Block Substitution. |
| Consider modeling fluid inertia with a Pipe (IL) block. | The Fluid Inertia block in the original model has been removed. | For details and suggested actions, see Block Substitution. |

| Message | Reason | Suggested Actions |
|---|---|---|
| Critical Reynolds number set to 150. | The Constant Area Hydraulic Orifice or Variable Area Hydraulic Orifice block in the original model has the **Laminar transition specification** parameter set to `Pressure ratio`. These Hydraulic blocks have two options for specifying the transition between the laminar and turbulent regime: by pressure ratio or by critical Reynolds number. The replacement Local Restriction (IL) block does not have the pressure ratio option. | The Local Restriction (IL) block specifies the transition between the laminar and turbulent regime by the critical Reynolds number, and the conversion tool sets the default value of 150 for the **Critical Reynolds number** parameter.<br><br>If the original Hydraulic block used the default **Laminar flow pressure ratio** parameter value of `0.999`, no action is necessary.<br><br>If the **Laminar flow pressure ratio** parameter value in the original block was significantly different, you might need to adjust the **Critical Reynolds number** parameter value in the replacement block. |
| Maximum restriction area set to 1e10 m^2. | The Variable Area Hydraulic Orifice block does not have a parameter that specifies the maximum area, the block assumes that the maximum area is `inf`. The replacement Local Restriction (IL) block, with **Restriction type** set to `Variable`, requires a **Maximum restriction area** parameter value less than `inf`. | The conversion tool sets the **Maximum restriction area** parameter in the replacement Local Restriction (IL) block to an arbitrary large value, `1e10 m^2`.<br><br>Adjust this value, if desired. |
| Original block had Specific heat ratio of 1.4. Set Air polytropic index to this value in an Isothermal Liquid Properties (IL) block. | Several Hydraulic blocks, such as chambers or hydro-mechanical converters, have a **Specific heat ratio** parameter. In the isothermal liquid domain, all of the fluid properties are defined in the Isothermal Liquid Properties (IL) block. | The conversion tool prints the value of the **Specific heat ratio** parameter in the original block for your convenience. Open the Isothermal Liquid Properties (IL) block connected to the circuit and set its **Air polytropic index** parameter to this value. |

**Upgrading Systems with Enabled Library Links, Model References, and Subsystem References**

You can use the `hydraulicToIsothermalLiquid` tool to convert any type of a block diagram system, such as a model, subsystem, or library. However, the tool appends `_converted` to the name of the original file, to avoid overriding original files and allow comparison between the original and converted files. When you convert models that contain hydraulic blocks from custom Simulink libraries, or referenced models or subsystems with hydraulic blocks, it is important to preserve the

links between the files, so that the converted models point to the converted libraries, referenced models, and subsystems. To preserve the links during the conversion, convert these files together, by supplying either a list of files or a top folder containing these files as an input argument to the `hydraulicToIsothermalLiquid` conversion tool.

If your model contains references or links to other libraries, models, or subsystems that also need to be converted, then the following approach is recommended:

**1** Use one of these syntaxes for the `hydraulicToIsothermalLiquid` conversion tool to convert the files together:

- `convertedFiles = hydraulicToIsothermalLiquid(oldfiles)` — Convert all files in the `oldfiles` list and return the names of converted files in `convertedFiles`. `oldfiles` is a cell array of file names, such as `{'file1';'file2';'file3'}`. If a file in the list does not contain hydraulic blocks and does not refer to a file listed in `oldfiles` that contains hydraulic blocks, it is not converted.

- `convertedFiles = hydraulicToIsothermalLiquid(topfolder)` — Convert all files in `topfolder` and its subfolders that are on the MATLAB path, if the files contain hydraulic blocks. If a file does not contain hydraulic blocks and does not refer to a file under `topfolder` that contains hydraulic blocks, it is not converted. `topfolder` is the path name of the top folder containing block diagram systems to convert, specified as a character vector or string scalar, such as `'MyLibraries'`. `convertedFiles` returns the names of converted files in a cell array of character vectors.

**2** In the common HTML report generated during the conversion, use the **Broken Connections** section to find and investigate broken links, if any. When you convert the files together, the links between the files and the connections within files are preserved. Broken connections might indicate that you forgot to include a reference model or library in the conversion. In this case, either repair the link manually or rerun the conversion process with the expanded list of files.

**3** Compare the simulation results of the converted models to the original models to ensure that the results are as expected. In the HTML report, investigate the messages in the **Removed Blocks** and **Parameter Warnings** sections. The messages in these sections indicate whether behavior changes are expected and suggest appropriate actions.

**4** Once satisfied that the converted systems behave as expected, change the system names back to the original ones:

`finalFiles = hydraulicToIsothermalLiquidPostProcess(convertedFiles)`

This function overwrites the original files by removing the `_converted` suffix from the file names and from the links between the files. For more information, see `hydraulicToIsothermalLiquidPostProcess`.

An alternative approach is to convert block diagram systems piece-by-piece, using the Interface (H-IL) block to connect the converted parts of the block diagram to any unconverted libraries or subsystems. Then, once each of the libraries, referenced models, and referenced subsystems is also converted and verified, remove `_converted` from its name, remove the Interface (H-IL) blocks and restore the broken connections.

## See Also
`hydraulicToIsothermalLiquid` | `hydraulicToIsothermalLiquidPostProcess` | Interface (H-IL) | **Simulation Data Inspector**

## More About

- "Modeling Isothermal Liquid Systems" on page 2-2

# Thermal Liquid Models

# Modeling Thermal Liquid Systems

| In this section... |
| --- |
| "When to Use Thermal Liquid Blocks" on page 3-2 |
| "Modeling Workflow" on page 3-2 |
| "Establish Model Requirements" on page 3-3 |
| "Model Physical Components" on page 3-3 |
| "Prepare Model for Analysis" on page 3-4 |
| "Run Simulation" on page 3-4 |

## When to Use Thermal Liquid Blocks

The Thermal Liquid library expands the fluid modeling capability of Simscape. With this library, you can account for thermal effects in a fluid system. For example, you can model the warming effect of viscous dissipation in a pipe. You can also account for the temperature dependence of fluid properties, e.g., density and viscosity.

To decide whether Thermal Liquid blocks fit your modeling needs, consider the fluid system you are trying to represent. Other Simscape blocks, such as Hydraulic or Two-Phase Fluid, may better suit your application. Assess the following:

- Number of phases

  Is the fluid medium single phase or multiphase?
- Relevant phases

  Is the fluid medium a gas, a liquid, or a multiphase mixture?
- Thermal effects

  Does temperature change significantly in the time scale of the simulation? Are thermal effects important for analysis? Are the temperature dependences of the liquid properties important?

As a rule, use Thermal Liquid blocks for fluid systems in which a single-phase liquid experiences significant temperature changes.

## Modeling Workflow

The suggested workflow for Thermal Liquid models includes four steps:

1. Establish model requirements — Define the purpose and scope of the model. Then, identify the relevant components and interactions in the model. Use this information as a guide when building the model.
2. Model physical components — Determine the appropriate blocks for modeling the relevant components and interactions. Then, add the blocks to the model canvas and connect them according to the Simscape connection rules. Specify the block parameters.
3. Prepare model for analysis — Add sensors to the model. Alternatively, configure the model for Simscape data logging. Check the physical units of each sensed variable.
4. Run simulation — Configure the solver settings. Then, run the simulation. If necessary, refine the model until you achieve the desired fidelity level.

## Establish Model Requirements

The foundation of a good model is a clear understanding of its purpose and requirements. What are you trying to accomplish with the model? What are the relevant components, processes, and states? Determine what is essential and what is not. Start simple, using a rough approximation of the physical system as a guide. Then, iteratively add detail to reach the appropriate model fidelity for your application.

An insulated oil pipeline buried underground provides an example. As oil flows through the pipeline, it experiences conductive heat losses due to the cooler pipeline surroundings. Heat flows across three material layers—pipe wall, insulant, and soil—causing oil temperature to drop. However, only conduction across soil and insulant layers matter. A typical pipe wall is thin and conductive, and its effect on conductive heat loss is minimal at best. Omitting this process simplifies the model and speeds up simulation.

You also must determine the dimensions and properties of each component. During modeling, you specify these parameters in the Simscape blocks for the components. Obtain the physical properties of the liquid medium. Manufacturer data sheets typically provide this data. You can also use analytical expressions to define the physical property lookup tables.

When modeling pipes, consider the impact that dynamic compressibility and flow inertia have on the transient system behavior. If the time scale of an effect exceeds the simulation run time, the impact is usually negligible. During modeling, turn off negligible effects to improve simulation speed. Characteristic time scales for dynamic compressibility and flow inertia are approximately L/c and L/v, respectively, where:

- L is the length of the pipe.
- v is the mean flow velocity through the pipe.
- c is the speed of sound in the liquid medium.

If you are unsure whether an effect is relevant to your model, simulate the model with and without that effect. Then, compare the two simulation results. If the difference is substantial, leave that effect in place. The result is greater model fidelity at small time scales, e.g., during transients associated with flow reversal in a pipe.

## Model Physical Components

Start by adding a Thermal Liquid Settings (TL) block to the model canvas. Use this block to provide the physical properties of the liquid medium. Double-click the block and enter the physical property lookup tables that you acquired during the planning stage.

Identify the appropriate blocks for representing the physical components and their interactions. Components can be simple, requiring a single block, or complex, requiring multiple blocks, typically within a Simulink Subsystem block. Add the blocks to the model canvas and connect them according to the Simscape connection rules.

The `ssc_tl_hydraulic_fluid_warming` example shows simple and complex components. The Mass Flow Rate Source (TL) represents an ideal power source. It is a simple component. The Double-Acting Cylinder subsystem block represents the mechanical part of a hydraulic actuator. It contains two Translational Mechanical Converter (TL) blocks and is a complex component.

Once you have connected the blocks, specify the relevant parameters. These include dimensions, physical states, empirical correlation coefficients, and initial conditions. In Pipe (TL), Rotational

Mechanical Converter (TL), and Translational Mechanical Converter (TL) blocks, select the appropriate setting for effects such as dynamic compressibility and flow inertia.

---

**Note** For accurate simulation results, always replace the default parameter values with data appropriate for your model.

---

## Prepare Model for Analysis

To analyze a model, you must set up that model for data collection. The simplest approach is to add sensor blocks to the model. The Thermal Liquid library provides two sensor block types: one for Through variables (mass and energy flow rates), the other for Across variables (pressure and temperature). By using the PS-Simulink Converter block, you can specify the physical units of the sensed variable.

An alternative approach is to use Simscape data logging. This approach, which uses MATLAB commands instead of blocks, provides access to a broader range of model variables and parameters. One example is the kinematic viscosity of the liquid medium inside a pipeline segment. You can analyze this parameter using Simscape data logging but not sensor blocks.

For an overview of Simscape data logging, see "About Simulation Data Logging" on page 13-2. For an example of how to plot logged data, see "Log and Plot Simulation Data" on page 13-8.

## Run Simulation

The final step in the modeling workflow is to simulate the model. Before running simulation, check that the numerical solver is appropriate for your model. To do this, use the Configuration Parameters dialog box.

For physical models, variable-step solvers such as `daessc`, `ode23t`, and `ode15s` typically perform best. Reduce step sizes and tolerances for greater simulation accuracy. Increase them instead for faster simulation.

Run the simulation. Plot simulation data from sensors and Simscape data logging, or process it for further analysis. If necessary, refine the model. For example, correct simulation issues or to improve model fidelity.

## See Also

### Related Examples
- "Heat Transfer in Insulated Oil Pipeline" on page 3-11

### More About
- "Thermal Liquid Library" on page 3-5
- "Thermal Liquid Modeling Framework" on page 3-8

# Thermal Liquid Library

| In this section... |
| --- |
| "Why Use Thermal Liquid Blocks?" on page 3-5 |
| "Representing Thermal Liquid Components" on page 3-5 |
| "Specifying Thermal Liquid Medium" on page 3-6 |
| "Modeling Multidomain Systems" on page 3-6 |

## Why Use Thermal Liquid Blocks?

The thermal behavior of liquid systems is of interest in many engineering applications. Liquids can store energy and release it back to their surroundings, often doing work in the process. Oil flow through an underground pipeline and hydraulic fluid flow in an aircraft actuator are two examples.

When temperature fluctuations are negligible, liquids behave as isothermal fluids, which simplifies the modeling process. However, when detailed thermal analysis is a goal, or when temperature fluctuations are significant, this assumption is no longer suitable.

The Thermal Liquid library provides a modeling tool that you can use to analyze the thermal behavior of *thermal* liquid systems. Three featured examples show some applications well-suited for Thermal Liquid modeling:

- `ssc_tl_oil_pipeline` — Model oil temperature along an insulated underground pipeline.
- `ssc_tl_hydraulic_fluid_warming` — Model hydraulic fluid warming due to viscous dissipation inside a hydraulic actuator.
- `ssc_tl_water_hammer` — Model the water hammer effect due to a fast-turning hydraulic valve.

## Representing Thermal Liquid Components

Thermal liquid systems can range in complexity from basic to highly specialized. To model a basic system, simple components often suffice. These are components such as chambers, pipes, pumps, and the liquid medium itself. Simple components are often industry independent and can be modeled using a single Thermal Liquid block. For example, you can model a pipeline segment using a single Pipe (TL) block.

To model a specialized system, generally you use custom components. These are components that you cannot represent by a single Thermal Liquid block. The five-way directional control valve in the `ssc_tl_hydraulic_fluid_warming` example is one such component. Custom components are often industry specific and must be modeled by grouping Thermal Liquid blocks into more complex subsystems.

The Thermal Liquid library shares the structure of other Simscape Foundation libraries. Four sublibraries supply the Thermal Liquid blocks: Elements, Sources, Sensors, and Utilities. With these sublibraries you can represent the most common components of a thermal liquid system. The table summarizes these components.

| Component Type | Description | Thermal Liquid Blocks |
|---|---|---|
| Liquid storage | Store liquid in chambers or reservoirs. | Constant Volume Chamber (TL), Reservoir (TL), Controlled Reservoir (TL) |
| Liquid transport | Transport thermal liquid through closed conduits such as pipes. | Pipe (TL) |
| Flow restriction | Restrict thermal liquid flow, e.g., due to valves or fittings. | Local Restriction (TL) |
| Mechanical interfaces | Interface thermal liquid and mechanical systems, e.g., to convert liquid mechanical energy into useful work. | Translational Mechanical Converter (TL), Rotational Mechanical Converter (TL) |
| Power sources | Provide a power source to the thermal liquid system, e.g. , pressure difference or mass flow rate. | Mass Flow Rate Source (TL), Pressure Source (TL), Controlled Mass Flow Rate Source (TL), Controlled Pressure Source (TL) |
| Sensors | Output measurement data for dynamic variables such as mass flow rate, energy flow rate, pressure, and temperature. | Pressure & Temperature Sensor (TL), Mass & Energy Flow Rate Sensor (TL), Thermodynamic Properties Sensor (TL), Volumetric Flow Rate Sensor (TL) |
| Thermal liquid | Specify thermodynamic properties and pressure-temperature validity region of thermal liquid medium. | Thermal Liquid Settings (TL) |

## Specifying Thermal Liquid Medium

The Thermal Liquid Settings (TL) block specifies the thermodynamic properties of the liquid medium. These properties are assumed functions of both pressure and temperature. This assumption boosts model fidelity, especially in models in which pressure, temperature, or both, vary widely.

The block accepts two-way lookup tables as input. These tables provide the different thermodynamic property values at discrete pressures and temperatures. You can populate these tables using empirical data from product data sheets or values calculated from analytical expressions.

## Modeling Multidomain Systems

Thermal Liquid blocks can contain different types of conserving ports. These ports include not only Thermal Liquid conserving ports but also thermal and mechanical conserving ports. By using these ports, you can interface a Thermal Liquid subsystem with thermal and mechanical subsystems.

For instance, you can use the thermal conserving port of a Pipe (TL) block to model conductive heat transfer through a pipe wall. Oil pipeline modeling is one application. The example `ssc_tl_oil_pipeline` shows this approach.

Similarly, you can use the translational mechanical conserving ports of a Translational Mechanical Converter (TL) block to convert hydraulic pressure in a thermal liquid system into a mechanical actuation force. Hydraulic actuator modeling is one application. The example `ssc_tl_hydraulic_fluid_warming` shows this approach.

The table lists the Thermal Liquid blocks that have thermal or mechanical conserving ports. You can use these blocks to create a multidomain model containing thermal liquid, thermal, and mechanical subsystems.

| Thermal Liquid Block | Thermal Conserving Port | Mechanical Conserving Port |
|---|---|---|
| Constant Volume Chamber (TL) | ✓ | ✗ |
| Pipe (TL) | ✓ | ✗ |
| Rotational Mechanical Converter (TL) | ✓ | ✓ |
| Translational Mechanical Converter (TL) | ✓ | ✓ |

## See Also

## Related Examples
- "Heat Transfer in Insulated Oil Pipeline" on page 3-11

## More About
- "Modeling Thermal Liquid Systems" on page 3-2
- "Thermal Liquid Modeling Framework" on page 3-8

# Thermal Liquid Modeling Framework

| **In this section...** |
| --- |
| "How Blocks Represent Components" on page 3-8 |
| "How Ports Represent Interfaces" on page 3-9 |
| "Full Flux Scheme" on page 3-9 |

## How Blocks Represent Components

Thermal Liquid models are based on the finite volume method. This method discretizes a thermal liquid system into multiple control volumes that interact via shared interfaces. An oil pipeline system is one example: you can model this system as a set of pipeline segments that connect serially along the pipeline length.
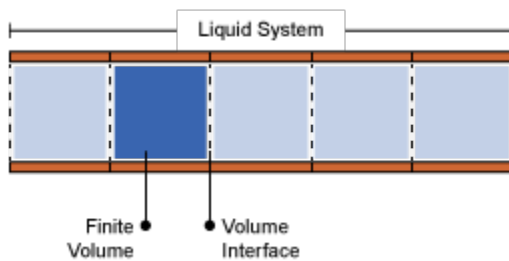


**Discretization of Pipeline System**

A control volume can represent a thermal liquid component, such as an oil pipeline, or a part of a component, such as a pipeline segment. You can discretize a thermal liquid system and its components as finely as you need, for example to increase simulation accuracy. However, the finer the discretization, the greater the model complexity—and the slower the simulation.

Thermal Liquid blocks represent the control volume of a component using an internal node. This node provides the liquid pressure and temperature inside the component. The node is not visible, but you can access its parameters and variables using Simscape data logging. For more information, see "About Simulation Data Logging" on page 13-2.
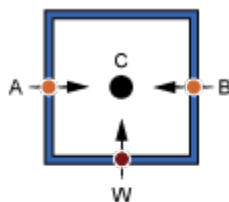


A, B — Thermal Liquid Conserving Port
W — Thermal Conserving Port
C — Internal Node

**Simscape Nodes in Pipe (TL) Block**

Two physical principles govern the dynamic evolution of liquid pressure and temperature at the internal node of a control volume: mass conservation and energy conservation. Pressure and temperature computation is carried out for the control volume surrounding the internal node. This control volume is the total volume of the thermal liquid component the block represents.

A second set of nodes represents the interfaces through which a finite volume can interact with its neighbors. These nodes are visible as Simscape conserving ports, of which Thermal Liquid conserving ports are the most important. By allowing the exchange of mass, momentum, and energy between adjacent liquid volumes, Thermal Liquid conserving ports govern the dynamic evolution of the finite volume as it tends to a steady state.

## How Ports Represent Interfaces

Thermal Liquid conserving ports provide the liquid pressure and temperature at the interfaces they represent. They also provide the flow rates of mass and heat, which govern the interactions between thermal liquid components. Pressure and temperature are the Across variables of the Thermal Liquid domain, while the flow rates are the Through variables.

Two physical principles govern the mass and heat flow rates through a Thermal Liquid conserving port: momentum conservation and energy conservation. The mass flow rate at a port is computed from the momentum conservation principle. The heat flow rate at a port is computed from the thermal energy conservation principle.

The flow rate computations are carried out for half the control volume of a thermal liquid component. The half control volume is bounded on one end by the interface the port represents, and on another end by a parallel surface passing through the control volume centroid.

The figure shows the half control volume for flow rate computations at interface A of a pipeline segment. Interface A corresponds to Thermal Liquid conserving port A of a Pipe (TL) block. Node C corresponds to the internal node of the block, which is coincident with the control volume centroid.



- A — Control Volume Interface
- C — Control Volume Centroid
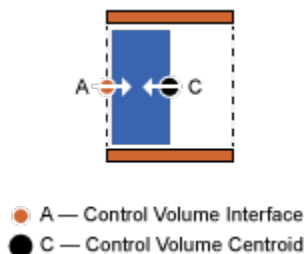
**Half Control Volume for Flow Rate Calculations**

## Full Flux Scheme

Blocks in the Thermal Liquid library implement a full flux scheme. Using this scheme, the net heat flux through a Thermal Liquid conserving port contains both convective and conductive flux contributions. By including thermal conduction in the flow direction, Thermal Liquid blocks provide more realistic simulation of the physical system they represent.

Other advantages of the full flux scheme include enhanced simulation robustness of thermal liquid models. This robustness becomes relevant in models where the conductive flux contribution can be

dominant. Examples include instances of low mass flow rates and flow reversal, during which the convective flux becomes negligible or vanishes altogether.

## See Also

## Related Examples

- "Heat Transfer in Insulated Oil Pipeline" on page 3-11

## More About

- "Modeling Thermal Liquid Systems" on page 3-2
- "Thermal Liquid Library" on page 3-5

# Heat Transfer in Insulated Oil Pipeline

| In this section... |
| --- |
| "Oil Pipelines" on page 3-11 |
| "Modeling Considerations" on page 3-11 |
| "Simscape Model" on page 3-13 |
| "Run Simulation" on page 3-14 |
| "Run Optimization Script" on page 3-19 |

## Oil Pipelines

Temperature plays an important role in oil pipeline design. Below the so-called cloud point, paraffin waxes precipitate from crude oil and start to accumulate along the pipe wall interior. The waxy deposits restrict oil flow, increasing the power requirements of the pipeline. At still-lower temperatures—below the pour point of oil—these crystals become so numerous that, if allowed to quiesce, oil becomes semisolid.
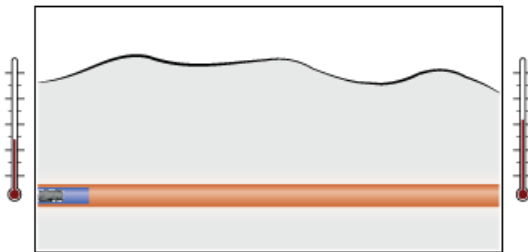


In cold climates, conductive heat losses through the pipe wall can be significant. To keep oil in its favorable temperature range, pipelines include some temperature control measures. Heating stations placed at intervals along the pipeline help to warm the oil. An insulant liner covering the pipe wall interior helps to retard the cooling rate of the oil.

Viscous dissipation provides an additional heat source. As adjacent parcels of oil flow against each other, they experience energy losses that appear in the form of heat. The warming effect is small, but sufficient to at least partially offset the conductive heat losses that occur through the insulant liner.

At a certain insulation thickness, viscous dissipation exactly balances the conductive heat loss. Oil stays at its ideal temperature throughout the pipeline length and the need for heating stations is reduced. From a design standpoint, this insulation thickness is optimal.

In this example, you simulate an insulated oil pipeline segment. You then run an optimization script to determine the optimal insulation thickness. This example is based on Simscape model `ssc_tl_oil_pipeline`.

## Modeling Considerations

The physical system in this example is an oil pipeline segment. Insulation lines the pipe wall interior, while soil covers the pipe wall exterior, retarding conductive heat loss. The simplifying assumption is made that the physical system is symmetric about the pipe center line.

**3-11**

A — Oil    B — Insulation    C — Pipe    D — Soil
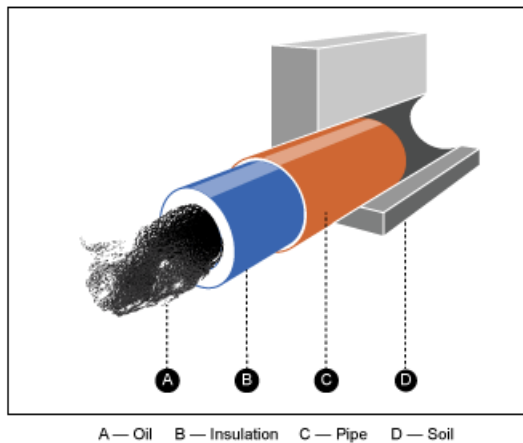
Flow through the pipeline segment is assumed fully developed: the velocity profile of the flowing oil remains constant along the pipeline length. In addition, oil is assumed Newtonian and compressible: shear stress is proportional to the shear strain, and mass density varies with both temperature and pressure.

Oil enters the pipeline segment at a fixed temperature, *TUpstream*, with a fixed mass flow rate, *Vdot * rho0*, where:

- *Vdot* is the volumetric flow rate of oil through the pipe.
- *rho0* is the mass density of oil entering the pipeline segment.

Inside the pipeline segment, viscous dissipation heats the flowing oil, while thermal conduction through the pipe wall cools it. The balance between the two processes governs the temperature of oil exiting the pipeline segment.

The amount of heat *gained* through viscous dissipation depends partly on oil viscosity and mass flow rate. The greater these quantities are, the greater the viscous heat gain is, and the warmer the oil tends to get. The amount of heat *lost* via thermal conduction depends partly on the thermal resistances of the insulation, pipe wall, and soil layer. The smaller the thermal resistances are, the greater the conductive heat loss is, and the cooler the oil tends to get.

Using an electrical circuit analogy, the combined thermal resistance of three material layers arranged in series equals the sum of the individual thermal resistances:

$$R_{combined} = R_{wall} + R_{ins.} + R_{soil}$$
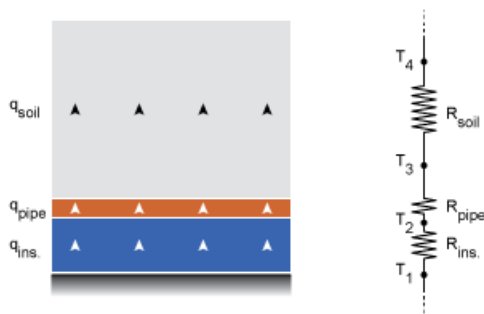
Assuming the pipe wall is thin and its material is a good thermal conductor, you can safely ignore the thermal resistance of the pipe wall. The combined thermal resistance is then simply the sum of the insulation and soil contributions, $R_{\text{ins.}}$ and $R_{\text{soil}}$.

The thermal resistance of the insulation layer is directly proportional to its thickness, (*D2-D1*)/2, and inversely proportional to its thermal conductivity, *kInsulant*. Likewise, the thermal resistance of the soil layer is directly proportional to its thickness, *z*, and inversely proportional to its thermal conductivity, *kSoil*.

The figure shows the relevant dimensions of the pipeline segment. Variable names match those specified in the model. The inner insulation diameter, *D1*, is also the hydraulic diameter of the pipeline segment.



z — Soil layer thickness    D1 — Inner insulation diameter

length — Pipeline segment length    D2 — Outer insulation diameter
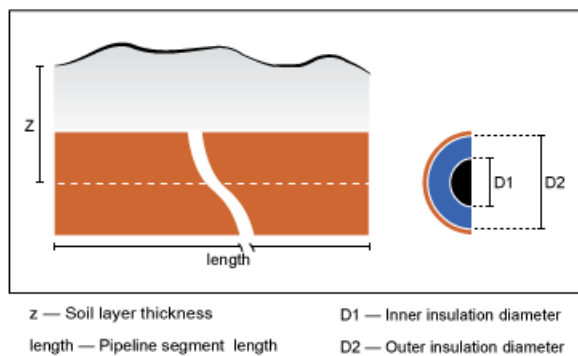
## Simscape Model

The Simscape model `ssc_tl_oil_pipeline` represents an insulated oil pipeline segment buried underground. To open this model, at the MATLAB command prompt, enter `ssc_tl_oil_pipeline`. The figure shows the model.
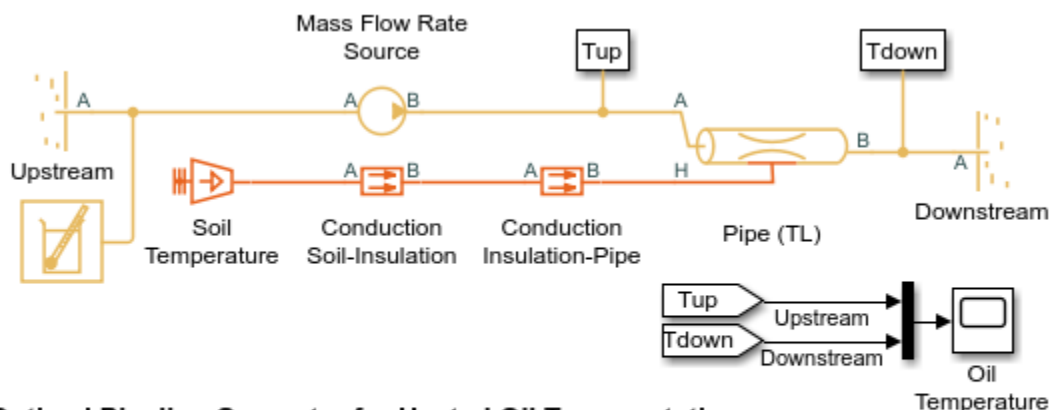


**Optimal Pipeline Geometry for Heated Oil Transportation**

1. Optimize pipe insulation inner diameter (see code)
2. Plot fluid properties (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

The Pipe (TL) block represents the physical system in this example, that is, the oil pipeline segment. Port **A** represents its inlet and port **B** its outlet. Port **H** represents thermal conduction through the pipe wall. The block accounts for viscous heating.

The Mass Flow Rate Source (TL) block provides the flow rate through the pipe. The Upstream block acts as a temperature source for the pipe inlet, while the Downstream block acts as a temperature sink at the pipe outlet.

The Conduction Insulation-Pipe and Conduction Soil-Insulation blocks represent thermal conduction through insulant and soil layers, respectively. These blocks appear in the Simscape Thermal library as Conductive Heat Transfer. The Soil Temperature (Temperature Source) block provides the temperature boundary condition at the soil surface.

The Thermal Liquid Settings (TL) block provides the physical properties of the oil, expressed as two-dimensional lookup tables containing the temperature and pressure dependence of the properties. The table summarizes these blocks.

| Block | Description |
|---|---|
| Pipe (TL) | Pipeline segment |
| Conduction Insulation-Pipe | Insulant thermal conduction |
| Conduction Soil-Insulation | Soil thermal conduction |
| Soil Temperature | Soil temperature |
| Upstream | Pipe inlet temperature sink |
| Downstream | Pipe outlet temperature sink |
| Mass Flow Rate Source (TL) | Oil mass flow rate |
| Thermal Liquid Settings (TL) | Oil thermodynamic properties |

## Run Simulation

To analyze the performance of the oil pipeline segment, simulate the model. The Oil Temperature scope plots the upstream and downstream oil temperatures. Open this scope. The insulation thickness is near its optimal value, resulting in only a small temperature change over a 1000 meter length. At a rate of ~0.020 K/km, oil temperature changes approximately 2 K over a 100 kilometer length.
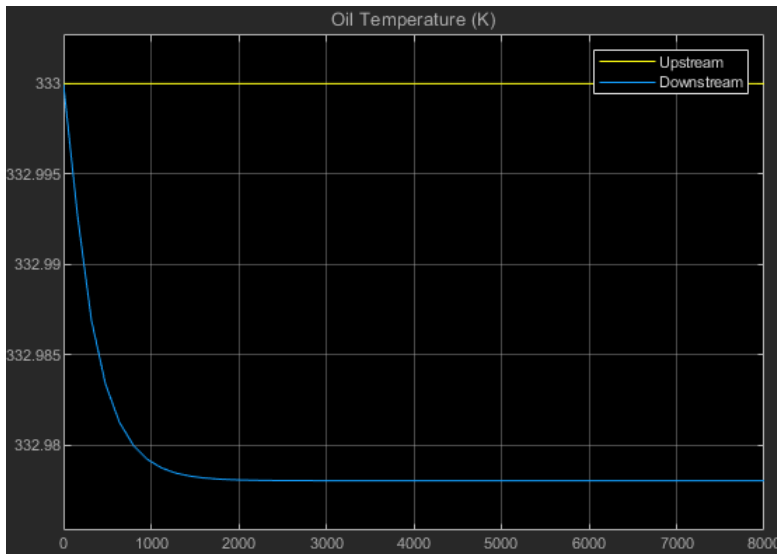
## Plot Physical Properties Using Data Logging

As an alternative to using sensors and scopes, you can use Simscape data logging to view how the physical properties of oil and other system variables change during simulation.

1. Select the Pipe (TL) block.

2. On the **Simscape Block** tab at the top of the model window, under **Review Results**, click **Results Explorer**.

3. In the left pane of the Simscape Results Explorer window, expand the `Pipe (TL)` node, which contains logged data for the Pipe (TL) block. Then expand the `A` and `B` nodes, which correspond to the **A** and **B** ports of the block.

4. Select variable `T` under node `A`, which is the upstream temperature of the pipe, to display its plot in the right pane of the Simscape Results Explorer window. To plot multiple variables at once, press the `Ctrl` key and select variable `T` under node B, which is the downstream temperature of the pipe.
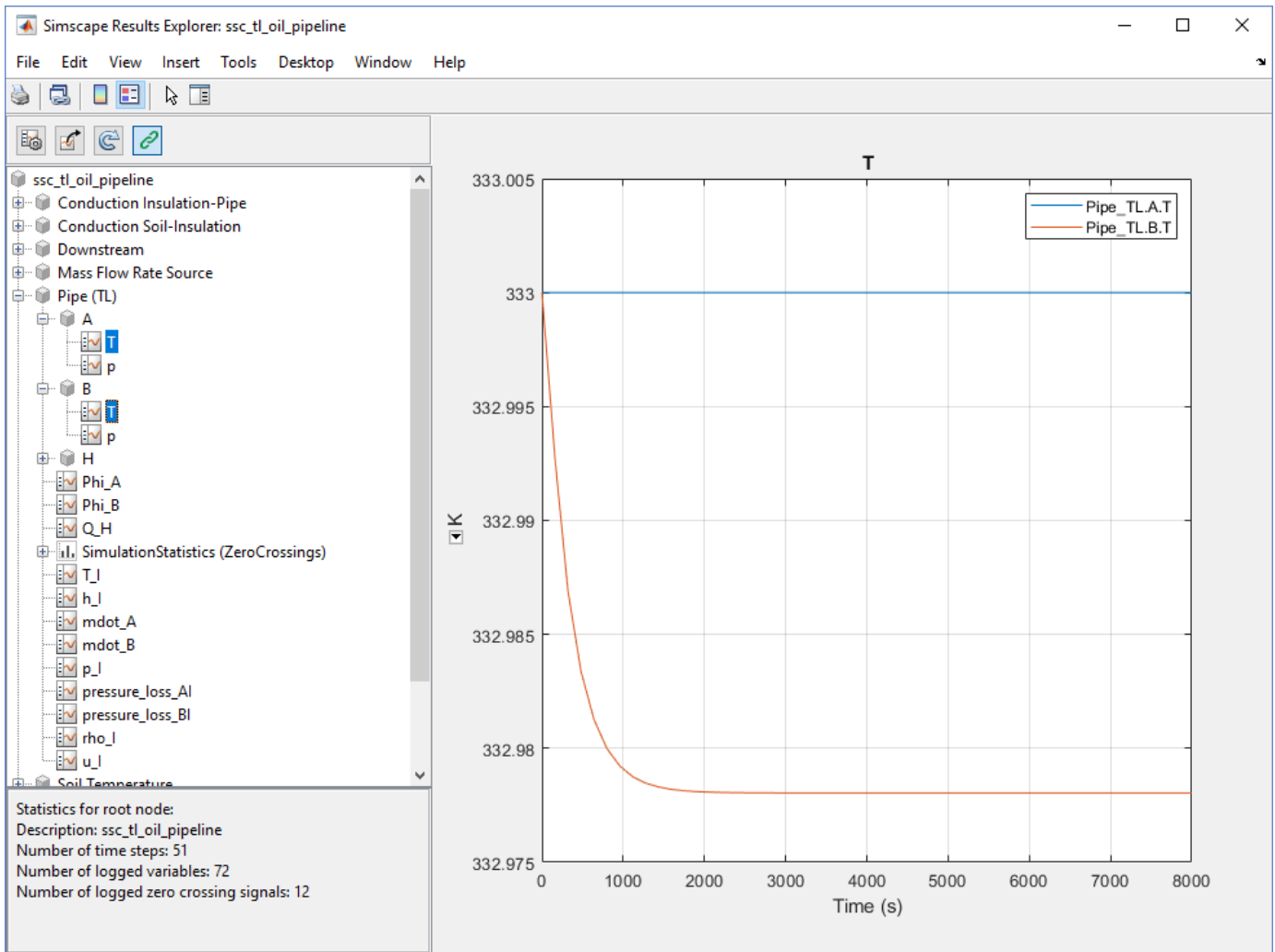
As expected, the plots in the right pane of the Simscape Results Explorer window are equivalent to the Oil Temperature scope results.

**5** You can also use the Simscape Results Explorer to plot other physical properties of the oil as a function of simulation time. For example, `rho_I` is the oil density.
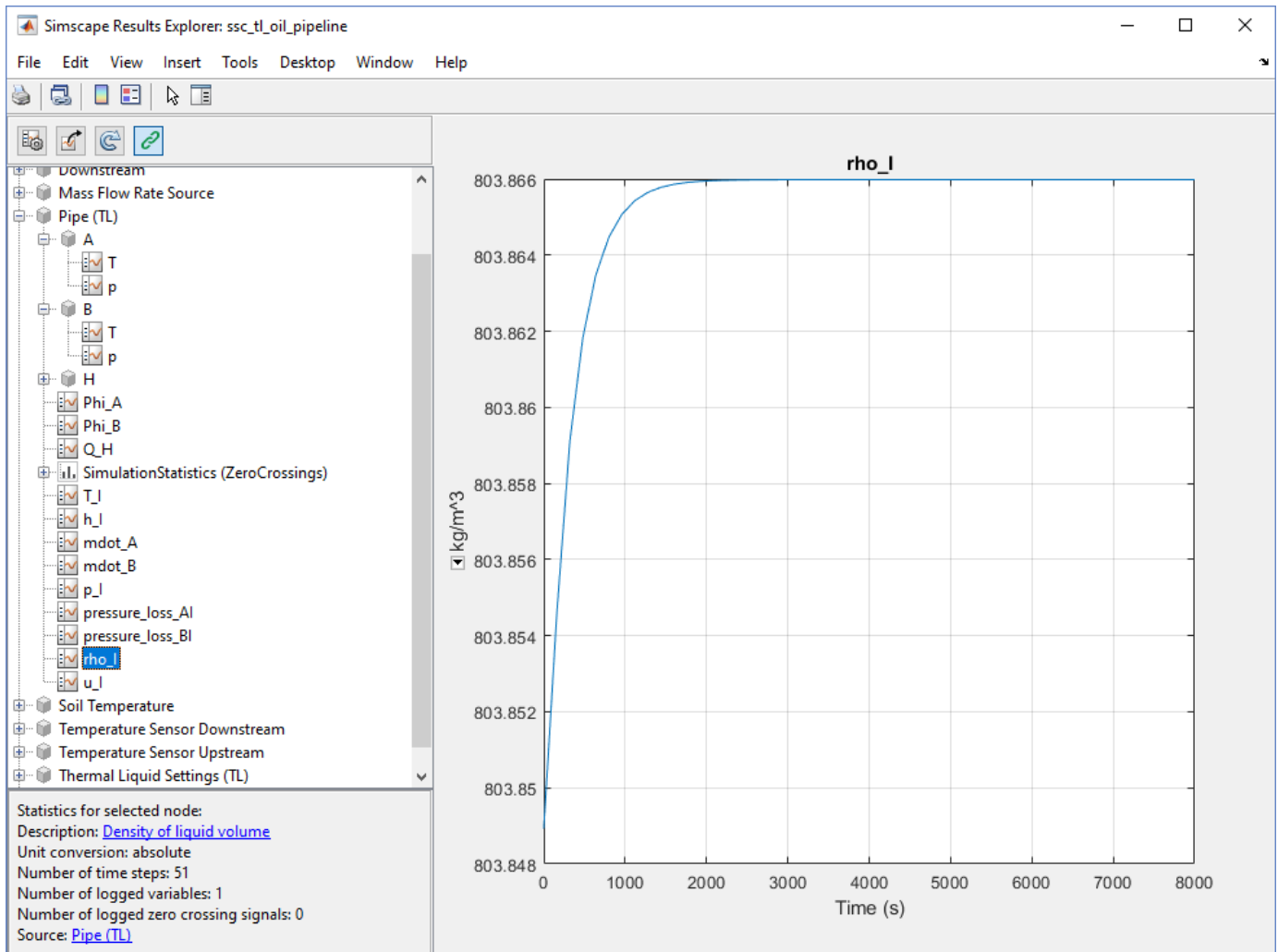
**Note** For more information about Simscape logging, see "About Simulation Data Logging" on page 13-2.

### Simulate Effects of Changing Insulation Diameter

Experiment with different values for the insulation inner diameter. By varying this parameter, you offset the balance between viscous dissipation, which heats the oil, and thermal conduction, which cools the oil.

1   Open Model Explorer.
2   In the **Model Hierarchy** pane, select **Base Workspace**.
3   In the **Contents** pane, click the value of parameter D1.
4   Enter `0.20`.

By reducing the inner diameter of the insulation layer to 0.20, you increase the insulation thickness, slowing down heat loss through the pipe wall via thermal conduction. Run the simulation. Then, open the Oil Temperature scope and autoscale to view full plot.
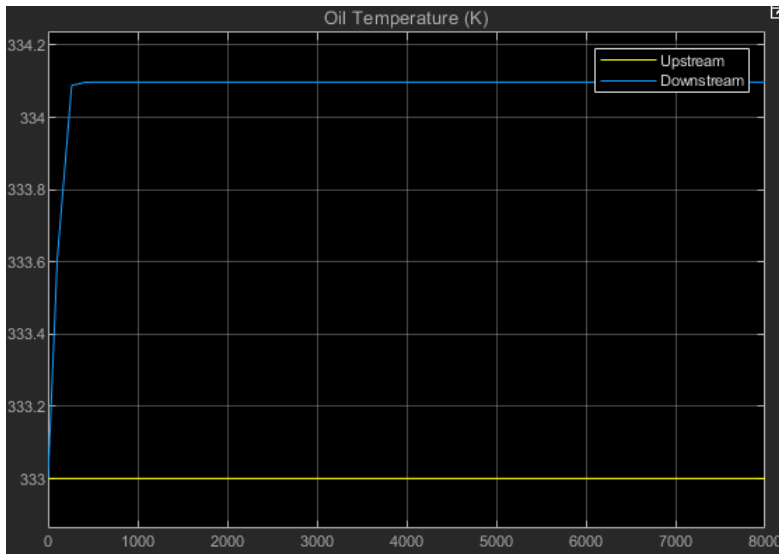
The new plot shows an oil temperature at the pipe outlet (top curve) that significantly exceeds the temperature at the pipe inlet (bottom line). Viscous dissipation now dominates the thermal energy balance in the pipeline segment. The new insulation thickness poses a design problem: in a long pipeline, a 1.1 K/km heating rate can raise the oil temperature substantially at the receiving end of the pipeline.

Try increasing the inner diameter of the insulation layer, D1, to 0.55. By increasing this value, you decrease the insulation thickness, accelerating heat loss through the pipe wall via thermal conduction. Then, run the simulation. Open the Oil Temperature scope and autoscale to view the full plot.
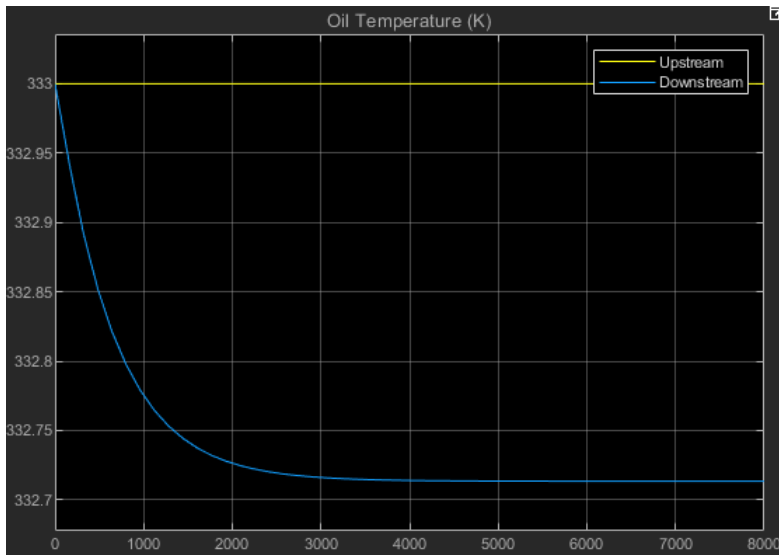


The resulting plot shows that the oil temperature at the pipe outlet is now significantly lower than that at the pipe inlet. Thermal conduction clearly dominates the thermal energy balance in the pipeline segment. This insulation thickness also poses a design issue: at a rate of 0.25K/km, oil flowing through a long pipeline will cool down substantially.

## Run Optimization Script

The model provides an optimization script that you can run to determine the optimal inner diameter of the pipe insulation, D1. The script iterates the model simulation at different D1 values, plotting the rates of viscous warming and conductive cooling against each other. The intersection point between the two curves identifies the optimal insulation thickness for the model:

**1** In the model window, click **Optimize** to run the optimization script for the pipe insulation inner diameter.
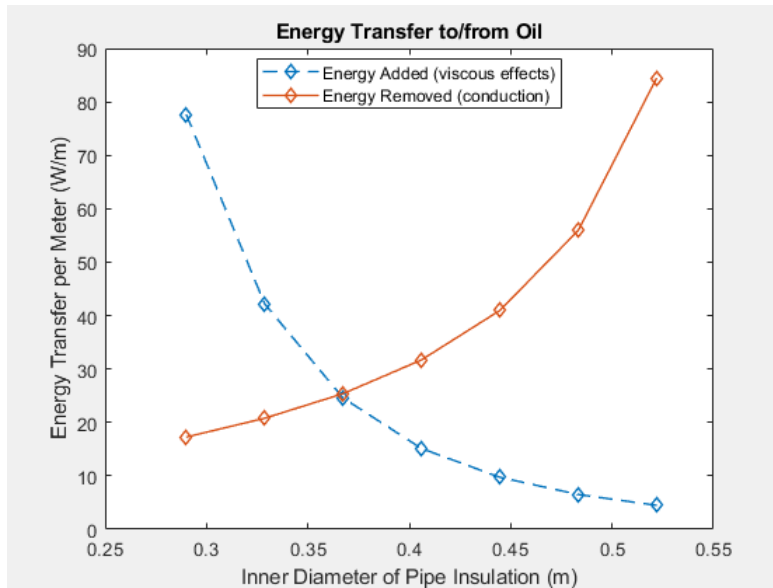


**2** In the plot that opens, visually determine the horizontal-axis value for the intersection point between the two curves.

The optimal inner diameter of the insulation layer is 0.37 m. Update parameter D1 to this value:

**1** Open Model Explorer.
**2** In the **Model Hierarchy** pane, click **Base Workspace**.
**3** In the **Contents** pane, click the value of D1.
**4** Enter 0.37.

Now, run the simulation. Open the Oil Temperature scope and autoscale to view the full plot. The temperature difference between the inlet and the outlet is negligible.
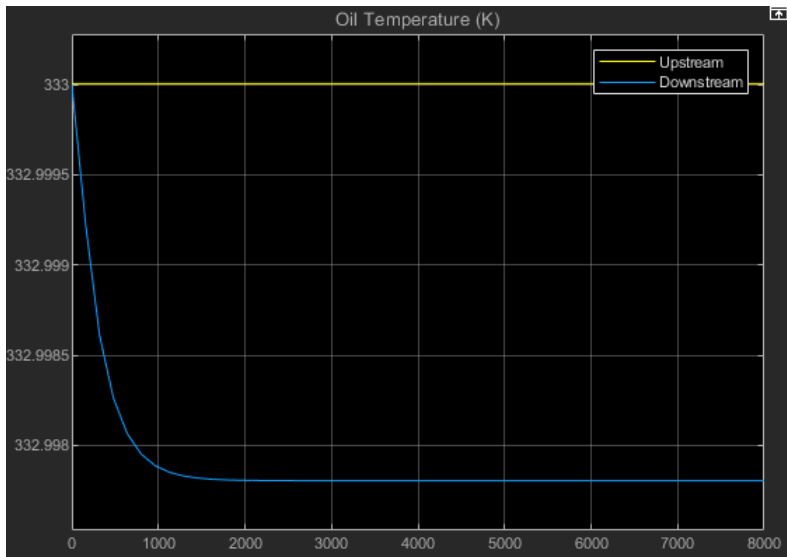
## See Also

### More About

- "Modeling Thermal Liquid Systems" on page 3-2
- "Thermal Liquid Library" on page 3-5
- "Thermal Liquid Modeling Framework" on page 3-8

# Two-Phase Fluid Models

# Manually Generate Fluid Property Tables

| In this section... |
| --- |
| |
| |
| |
| |
| |
| |
| |
| |
| |

## Fluid Property Tables

Fluid property tables provide the basic inputs to the Two-Phase Fluid Properties (2P) block. If you have REFPROP software by the National Institute of Standards and Technology or CoolProp software installed, you can automatically generate these tables using the `twoPhaseFluidTables` function. If you obtain the fluid properties from a different source, you can still generate the tables using a MATLAB script. This tutorial shows how to create a script to generate the fluid temperature tables.

The tables must provide the fluid properties at discrete pressures and normalized internal energies. The pressures must correspond to the table columns and the normalized internal energies to the table rows. Setting pressure and normalized internal energy as the independent variables enables you to specify the liquid and vapor phase property tables on separate rectangular grids using MATLAB matrices.

The figure shows two fluid property grids in pressure-specific internal energy space (left) and pressure-normalized internal energy space (right). If you obtain the fluid property tables on a pressure-specific internal energy grid, you must transform that grid into its pressure-normalized internal energy equivalent. In this tutorial, this transformation is handled by the MATLAB script that you create.
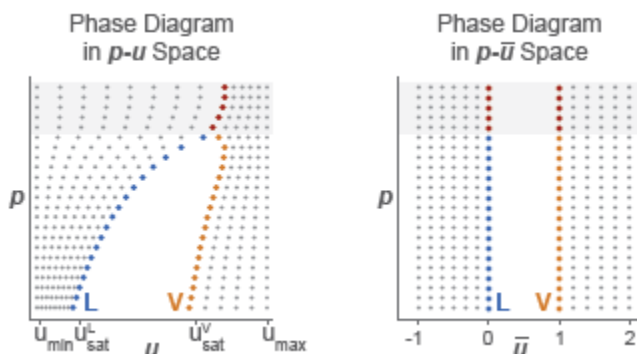
## Steps for Generating Property Tables

The MATLAB script that you create in this tutorial performs the following tasks:

- Define property table criteria, including dimensions and pressure-specific internal energy domain.
- Create rectangular grids in pressure-normalized internal energy space.
- Map the grids onto pressure-specific internal energy space.
- Obtain the fluid properties on the pressure-specific internal energy grids.

## Before Generating Property Tables

You must obtain fluid property data in pressure-specific internal energy space, e.g., through direct calculation, from a proprietary database, or from a third-party source. In this tutorial, you create four MATLAB functions to provide example property data. In a real application, you must replace these functions with equivalent functions written to access real property data.

## Create Fluid Property Functions

Create the following MATLAB functions. These functions provide the example property data you use in this tutorial. Ensure that the function files are on the MATLAB path. Use the function names and code shown:

- Name — `liquidTemperature`

```
function T = liquidTemperature(u, p)
% Returns artificial temperature data as a function
% of specific internal energy and pressure.
T = 300 + 0.2*u - 0.08*p;
```

- Name — `vaporTemperature`

```
function T = vaporTemperature(u, p)
% Returns artificial temperature data as a function
% of specific internal energy and pressure.
T = -1000 + 0.6*u + 5*p;
```

- Name — `saturatedLiquidInternalEnergy`

```
function u = saturatedLiquidInternalEnergy(p)
% Returns artificial data for saturated liquid specific
% internal energy as a function of pressure.
u = sqrt(p)*400 + 150;
```

- Name — `saturatedVaporInternalEnergy`

```
function u = saturatedVaporInternalEnergy(p)
% Returns artificial data for saturated vapor specific
% internal energy as a function of pressure.
u = -3*p.^2 + 40*p + 2500;
```

## Set Property Table Criteria

Start a new MATLAB script. Save the script in the same folder as the MATLAB functions you created to generate the example fluid property data. In the script, define the criteria for the property tables. Do this by entering the following code for the table dimensions and pressure-specific internal energy valid ranges:

```
% Number of rows in the liquid property tables
mLiquid = 25;
```

```
% Number of rows in the vapor property tables
mVapor = 25;
% Number of columns in the liquid and vapor property tables
n = 60;

% Minimum specific internal energy, kJ/kg
uMin = 30;
% Maximum specific internal energy, kJ/kg
uMax = 4000;
% Minimum pressure, MPa
pMin = 0.01;
% Maximum pressure, MPa
pMax = 15;

% Store minimum and maximum values in structure fluidTables
fluidTables.uMin = uMin;
fluidTables.uMax = uMax;
fluidTables.pMin = pMin;
fluidTables.pMax = pMax;
```

## Create Pressure-Normalized Internal Energy Grids

Define the pressure and normalized internal energy vectors for the grid. These vectors provide the discrete pressure and normalized internal energy values associated with each grid point. The pressure vector is logarithmically spaced due to the wide pressure range considered in this example. However, you can use any type of spacing that suits your data. In your MATLAB script, add this code:

```
% Pressure vector, logarithmically spaced
fluidTables.p = logspace(log10(pMin), log10(pMax), n);

% Normalized internal energy vectors, linearly spaced
fluidTables.liquid.unorm = linspace(-1, 0, mLiquid)';
fluidTables.vapor.unorm = linspace(1, 2, mVapor)';
```

## Map Grids Onto Pressure-Specific Internal Energy Space

Obtain the saturated liquid and vapor specific internal energies as functions of pressure. The saturation internal energies enable you to map the normalized internal energy vectors into equivalent vectors in specific internal energy space. In your MATLAB script, add this code:

```
% Initialize the saturation internal energies of the liquid and vapor phases
fluidTables.liquid.u_sat = zeros(1, n);
fluidTables.vapor.u_sat = zeros(1, n);

% Obtain the saturation internal energies at the pressure vector values
for j = 1 : n
    fluidTables.liquid.u_sat(j) = saturatedLiquidInternalEnergy(fluidTables.p(j));
    fluidTables.vapor.u_sat(j) = saturatedVaporInternalEnergy(fluidTables.p(j));
end
```

This code calls two functions written to generate example data. Before using this code in a real application, you must replace the functions with equivalent expressions capable of accessing real data. The functions you must replace are:

- saturatedLiquidInternalEnergy
- saturatedVaporInternalEnergy

Map the normalized internal energy vectors onto equivalent specific internal energy vectors. In your MATLAB script, add this code:

```
% Map pressure-specific internal energy grid onto
% pressure-normalized internal energy space
fluidTables.liquid.u = (fluidTables.liquid.unorm + 1)*...
(fluidTables.liquid.u_sat - uMin) + uMin;
```

```
fluidTables.vapor.u = (fluidTables.vapor.unorm - 2)*...
(uMax - fluidTables.vapor.u_sat) + uMax;
```

## Obtain Fluid Properties at Grid Points

You can now obtain the fluid properties at each grid point. The following code shows how to generate the temperature tables for the liquid and vapor phases. Use a similar approach to generate the remaining fluid property tables. In your MATLAB script, add this code:

```
% Obtain temperature tables for the liquid and vapor phases
for j = 1 : n
    for i = 1 : mLiquid
        fluidTables.liquid.T(i,j) = ...
liquidTemperature(fluidTables.liquid.u(i,j), fluidTables.p(j));
    end
    for i = 1 : mVapor
        fluidTables.vapor.T(i,j) = ...
vaporTemperature(fluidTables.vapor.u(i,j), fluidTables.p(j));
    end
end
```

This code calls two functions written to generate example data. Before using this code in a real application, you must replace the functions with equivalent expressions capable of accessing real data. The functions you must replace are:

- `liquidTemperature`
- `vaporTemperature`

To view the temperature tables generated, first run the script. Then, at the MATLAB command prompt, enter `fluidTables`. MATLAB lists the contents of the `fluidTables` structure array.

```
fluidTables =

     uMin: 30
     uMax: 4000
     pMin: 0.0100
     pMax: 15
        p: [1x20 double]
   liquid: [1x1 struct]
    vapor: [1x1 struct]
```

To list the property tables stored in the `liquid`substructure, at the MATLAB command prompt enter `fluidTables.liquid`.

```
  305.9992   305.9988   305.9983   305.9975   ...
  309.5548   309.7430   309.9711   310.2475   ...
  313.1103   313.4872   313.9440   314.4976   ...
  316.6659   317.2314   317.9169   318.747    ...
  ...
```

## Visualize Grids

To visualize the original grid in pressure-normalized internal energy space, at the MATLAB command prompt enter this code:

```
% Define p and unorm matrices with the grid
% point coordinates
pLiquid = repmat(fluidTables.p, mLiquid, 1);
pVapor = repmat(fluidTables.p, mVapor, 1);
```

```
unormLiquid = repmat(fluidTables.liquid.unorm, 1, n);
unormVapor = repmat(fluidTables.vapor.unorm, 1, n);

% Plot grid
figure;
hold on;

plot(unormLiquid, pLiquid, 'b.');
plot(unormVapor, pVapor, 'b.');

plot(zeros(1, n), fluidTables.p, 'k-');
plot(ones(1, n), fluidTables.p, 'k-');

hold off;
set(gca, 'yscale', 'log');
xlabel('Normalized Internal Energy');
ylabel('Pressure');
title('Grid in Normalized Internal Energy');
```

A figure opens with the pressure-normalized internal energy grid.
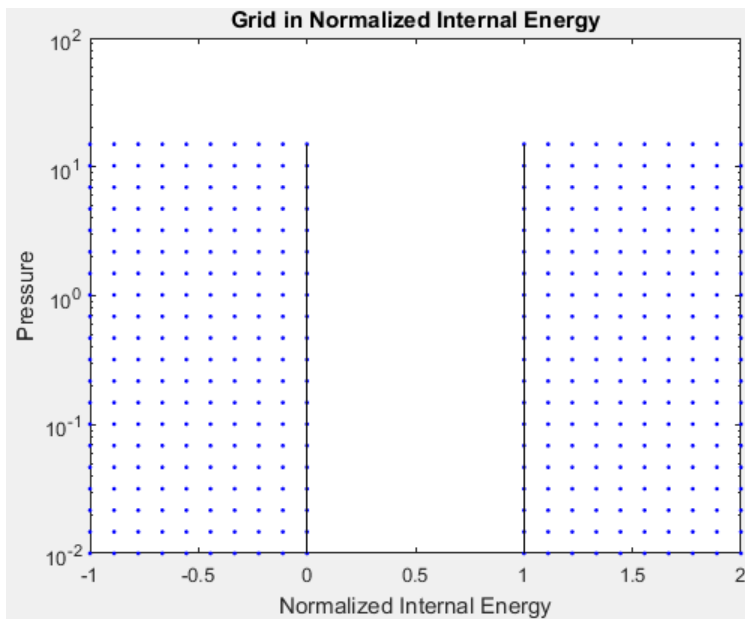


To visualize the transformed grid in pressure-specific internal energy space, at the MATLAB command prompt enter this code:

```
% Define horizontal and vertical axes

% Plot grid
figure;
hold on;

plot(fluidTables.liquid.u, pLiquid, 'b.');
plot(fluidTables.vapor.u, pVapor, 'b.');

plot(fluidtables.liquid.u_sat, fluidTables.p, 'k-');
plot(fluidtables.vapor.u_sat, fluidTables.p, 'k-');

hold off;
set(gca, 'yscale', 'log');
xlabel('Specific Internal Energy');
ylabel('Pressure');
title('Grid in Specific Internal Energy');
```

A figure opens with the pressure-specific internal energy grid.
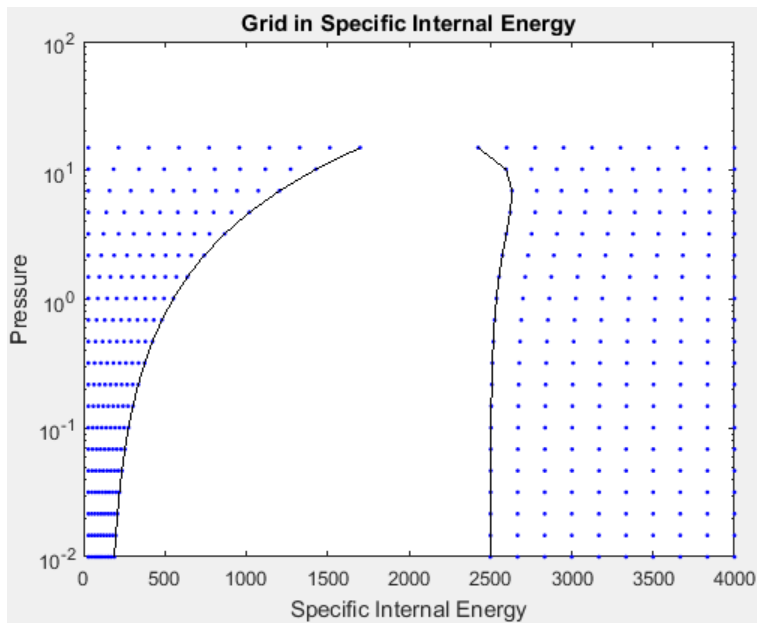


## See Also
Two-Phase Fluid Properties (2P) | `twoPhaseFluidTables`

# Gas System Models

# Modeling Gas Systems

| In this section... |
| --- |

## Intended Applications

The Gas library contains basic elements, such as orifices, chambers, and pneumatic-mechanical converters, as well as sensors and sources. Use these blocks to model gas systems, for applications such as:

- Pneumatic actuation of mechanical systems
- Natural gas transport through pipe networks
- Gas turbines for power generation
- Air cooling of thermal components

You specify the gas properties in the connected loop by using the Gas Properties (G) block. This block lets you choose between three idealization levels: perfect gas, semiperfect gas, or real gas (see "Gas Property Models" on page 5-2).

Unless explicitly stated otherwise, all pressures and temperatures used in modeling gas systems are static pressure and static temperature.

## Network Variables

The Across variables are pressure and temperature, and the Through variables are mass flow rate and energy flow rate. Note that these choices result in a pseudo-bond graph, because the product of pressure and mass flow rate is not power.

## Gas Property Models

The Gas library supports perfect gas, semiperfect gas, and real gas within the same gas domain in order to cover a wide range of modeling requirements. The three gas property models provide trade-offs between simulation speed and accuracy. They also enable the incremental workflow: you start with a simple model, which requires minimal information about the working gas, and then build upon the model when more detailed gas property data becomes available.

You select the gas property model by using the Gas Properties (G) block, which specifies the gas properties in the connected circuit.

The following table summarizes the different assumptions for each gas property model.

- Thermal equation of state indicates the relationship of density with temperature and pressure.
- Caloric equation of state indicates the relationship of specific heat capacity with temperature and pressure.
- Transport properties indicate the relationship between dynamic viscosity and thermal conductivity with temperature and pressure.

| Gas Property Model | Thermal Equation of State | Caloric Equation of State | Transport Properties |
| --- | --- | --- | --- |
| Perfect | Ideal gas law | Constant | Constant |
| Semiperfect | Ideal gas law | 1-D table lookup by temperature | 1-D table lookup by temperature |
| Real | 2-D table lookup by temperature and pressure | 2-D table lookup by temperature and pressure | 2-D table lookup by temperature and pressure |

The ideal gas law is implemented in the Simscape Foundation Gas library as

$p = Z\rho RT$

where:

- $p$ is the pressure.
- $Z$ is the compressibility factor.
- $R$ is the specific gas constant.
- $T$ is the temperature.

The compressibility factor, $Z$, is typically a function of pressure and temperature. It accounts for the deviation from ideal gas behavior. The gas is ideal when $Z = 1$. In the perfect and semiperfect gas property models, $Z$ must be constant but it does not have to be equal to 1. For example, if you are modeling a nonideal gas ($Z \neq 1$) but the temperature and pressure of the system do not vary significantly, you can use the perfect gas model and specify an appropriate value of $Z$. The following table lists the compressibility factor $Z$ for various gases at 293.15 K and 0.101325 MPa:

| Gas | Compressibility Factor |
| --- | --- |
| Dry Air | 0.99962 |
| Carbon Dioxide | 0.99467 |
| Oxygen | 0.99930 |
| Hydrogen | 1.00060 |
| Helium | 1.00049 |
| Methane | 0.99814 |
| Natural Gas | 0.99797 |
| Ammonia | 0.98871 |
| R-134a | 0.97814 |

Using the perfect gas model, with the constant value of $Z$ adjusted based on the type of gas and the operating conditions, lets you avoid the additional complexity and computational cost of moving to the semiperfect or real gas property model.

The perfect gas property model is a good starting choice when modeling a gas network because it is simple, computationally efficient, and requires limited information about the working gas. It is correct for monatomic gases and, typically, it is sufficiently accurate for gases such as dry air, carbon dioxide, oxygen, hydrogen, helium, methane, natural gas, and so on, at standard conditions.

When the gas network is operating near the saturation boundary or is operating over a very wide temperature range, the working gas can exhibit mild nonideal behavior. In this case, after successfully simulating the gas network with the perfect gas property model, consider switching to the semiperfect gas property model.

Finally, consider switching to the real gas property model if the working gas is expected to exhibit strongly nonideal behavior, such as heavy gases with large molecules. This model is the most expensive in terms of computational cost and requires detailed information about the working gas, because it uses 2-D interpolation for all properties.

## Blocks with Gas Volume

Components in the gas domain are modeled using control volumes. The control volume encompasses the gas inside the component and separates it from the surrounding environment and other components. Gas flows and heat flows across the control surface are represented by ports. The gas volume inside the component is represented using an internal node, which provides the gas pressure and temperature inside the component. This internal node is not visible, but you can access its parameters and variables using Simscape data logging. For more information, see About Simulation Data Logging on page 13-2.

The following blocks in the Gas library are modeled as components with a gas volume. In the case of Controlled Reservoir (G) and Reservoir (G), the volume is assumed to be infinitely large.

| Block | Gas Volume |
|---|---|
| Constant Volume Chamber (G) | Finite |
| Pipe (G) | Finite |
| Rotational Mechanical Converter (G) | Finite |
| Translational Mechanical Converter (G) | Finite |
| Reservoir (G) | Infinite |
| Controlled Reservoir (G) | Infinite |

Other components have relatively small gas volumes, so that gas entering the component spends negligible time inside the component before exiting. These components are considered quasi-steady-state and they do not have an internal node.

## Reference Node and Grounding Rules

Unlike mechanical and electrical domains, where each topologically distinct circuit within a domain must contain at least one reference block, gas networks have different grounding rules.

Blocks with a gas volume contain an internal node, which provides the gas pressure and temperature inside the component and therefore serves as a reference node for the gas network. Each connected gas network must have at least one reference node. This means that each connected gas network must have at least one of the blocks listed in "Blocks with Gas Volume" on page 5-4. In other words, a gas network that contains no gas volume is an invalid gas network.

The Foundation Gas library contains the Absolute Reference (G) block but, unlike other domains, you do not use it for grounding gas circuits. The purpose of the Absolute Reference (G) block is to provide a reference for the Pressure & Temperature Sensor (G). If you use the Absolute Reference (G) block elsewhere in a gas network, it will trigger a simulation assertion because gas pressure and temperature cannot be at absolute zero.

## Initial Conditions for Blocks with Finite Gas Volume

This section discusses the specific initialization requirements for blocks modeled with finite gas volume. These blocks are listed in "Blocks with Gas Volume" on page 5-4.

The state of the gas volume evolves dynamically based on interactions with connected blocks via mass and energy flows. The time constants depend on the compressibility and thermal capacity of the gas volume.

The state of the gas volume is represented by differential variables at the internal node of the block. As differential variables, they require initial conditions to be specified prior to the start of simulation. The dialog box of each block modeled with finite gas volume has an **Initial Targets** section, which lists three variables:

- **Pressure of gas volume**
- **Temperature of gas volume**
- **Density of gas volume**

By default, **Pressure of gas volume** and **Temperature of gas volume** have high priority, with target values equal to the standard condition (`0.101325 MPa` and `293.15 K`). You can adjust the target values to represent the appropriate initial state of the gas volume for the block. **Density of gas volume** has the default priority `None` because only the initial conditions of two of the three variables are needed to completely determine the initial state of the gas volume. If desired, an alternative way to specify the initial conditions is to change **Density of gas volume** to high priority with an appropriate target value, and then change either **Pressure of gas volume** or **Temperature of gas volume** to a priority of none.

It is important that only two of the three variables have their priorities set to `High` for each block with a finite gas volume. Placing high-priority constraints on all three variables results in over-specification, with the solver unable to find an initialization solution that satisfies the desired initial values. Conversely, placing high-priority constraint only on one variable makes the system under-specified, and the solver might resolve the variables with arbitrary and unexpected initial values. For more information on variable initialization and dealing with over-specification, see "Initialize Variables for a Mass-Spring-Damper System" on page 8-6.

In blocks that are modeled with an infinitely large gas volume, the state of the gas volume is assumed quasisteady and there is no need to specify an initial condition.

## Choked Flow

Gas flow through Local Restriction (G), Variable Local Restriction (G), or Pipe (G) blocks can become choked. Choking occurs when the flow velocity reaches the local speed of sound. When the flow is choked, the velocity at the point of choking cannot increase any further. However, the mass flow rate can still increase if the density of the gas increases. This can be achieved, for example, by increasing the pressure upstream of the point of choking. The effect of choking on a gas network is that the mass flow rate through the branch containing the choked block depends completely on the upstream

pressure and temperature. As long as the choking condition is maintained, this choked mass flow rate is independent of any changes occurring in the pressure downstream.

The following model illustrates the choked flow. In this model, the Ramp block has a slope of 0.005 and the start time of 10. The Simulink-PS Converter block has **Input signal unit** set to Mpa. All other blocks have default parameter values. Simulation time is 50 s. When you simulate the model, the pressure at port A of the Local Restriction (G) block increases linearly from atmospheric pressure, starting at 10 s. The pressure at port B is fixed at atmospheric pressure.
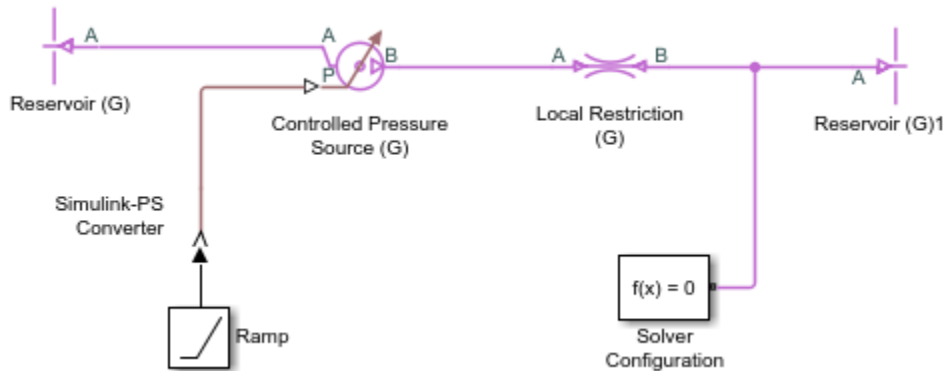


The following illustration shows the logged simulation data for the Local Restriction (G) block. The Mach number at the restriction (**Mach_R**) reaches 1 at around 20 s, indicating that the flow is choked. The mass flow rate (**mdot_A**) before the flow is choked follows the typical quadratic behavior with respect to an increasing pressure difference. However, the mass flow rate after the flow is choked becomes linear because the choked mass flow rate depends only on the upstream pressure and temperature, and the upstream pressure is increasing linearly.

The fact that the choked mass flow rate depends only on the upstream conditions can cause an incompatibility with a Mass Flow Rate Source (G) or a Controlled Mass Flow Rate Source (G) connected downstream of the choked block. Consider the model shown in the next illustration, which contains the Controlled Mass Flow Rate Source (G) block instead of the Controlled Pressure Source (G).
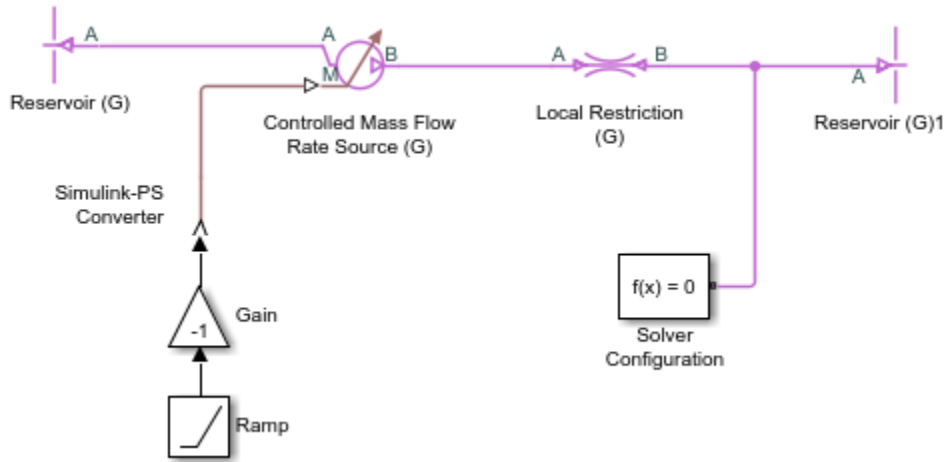


If the source commanded an increasing mass flow rate from left to right through the Local Restriction (G), the simulation would succeed even if the flow became choked because the Controlled Mass Flow Rate Source (G) would be upstream of the choked block. However, in this model the Gain block reverses the flow direction, so that the Controlled Mass Flow Rate Source (G) is downstream of the choked block. The pressure upstream of the Local Restriction (G) is fixed at atmospheric pressure. Therefore, the choked mass flow rate in this situation is constant. As the commanded mass flow rate increases, eventually it will become greater than this constant value of choked mass flow rate. At this point, the commanded mass flow rate and the choked mass flow rate cannot be reconciled and the simulation fails. Viewing the logged simulation data in the Simscape Results Explorer shows that simulation fails just at the point when the Mach number reaches 1 and the flow becomes choked.

In general, if a model is likely to choke, use pressure sources rather than mass flow rate sources. If a model contains mass flow rate source blocks and simulation fails, use the Simscape Results Explorer to inspect the Mach number variables in all Local Restriction (G), Variable Local Restriction (G), and Pipe (G) blocks connected along the same branch as the mass flow rate source. If the simulation failure occurs when the Mach number reaches 1, it is likely that there is a downstream mass flow rate source trying to command a mass flow rate greater than the possible choked mass flow rate.

The Mach number variable for the restriction blocks is called **Mach_R**. The Pipe (G) block has two Mach number variables, **Mach_A** and **Mach_B**, representing the Mach number at port A and port B, respectively.

## Flow Reversal

The flow of gas through the circuit carries energy from one gas volume to another gas volume. Therefore, the energy flow rate between two connected blocks depends on the direction of flow. If the gas flows from block A to block B, then the energy flow rate between the two blocks is based on the specific total enthalpy of block A. Conversely, if the gas flows from block B to block A, then the energy flow rate between the two blocks is based on the specific total enthalpy of block B. To smooth the transition for simulation robustness, the energy flow rate also includes a contribution based on the difference in the specific total enthalpies of the two blocks at low mass flow rates. The smoothing region is controlled by the Gas Properties (G) block parameter **Mach number threshold for flow reversal**.

A consequence of this approach is that the temperature of a node between two connected blocks represents the temperature of the gas volume upstream of that node. If there are two or more upstream flow paths merging at the node, then the temperature at the node represents the weighted average temperature based on the ideal mixing of the merging gas flows.

Simulation robustness can be challenging for models that exhibit quick flow reversals and large temperature differences between blocks. Quick flow reversals may be a result of having low flow resistances (for example, short pipes) between large gas volumes. Large temperature differences may be a result of the energy added by sources to maintain large pressure differences in a model with little heat dissipation. In these models, it may be necessary to increase the **Mach number threshold for flow reversal** parameter value to avoid simulation failure.

## See Also

## Related Examples

- "Simple Gas Model" on page 5-11
- "Change Flow Boundary Conditions" on page 5-15
- "Change Flow Direction" on page 5-20
- "Choked Flow in Gas Orifice" on page 20-174
- "Pneumatic Actuation Circuit" on page 20-160
- "Pneumatic Motor Circuit" on page 20-167

# Simple Gas Model

In this example, you create a simple open-loop gas model. The model consists of a local restriction between two reservoirs. The local restriction represents a valve or an orifice. The reservoir blocks set up the boundary conditions for the local restriction.

Reservoir blocks are useful for setting up pressure and temperature boundary conditions. If you want the pressure and temperature boundary conditions to change over time, use controlled reservoir blocks.

To open the completed model, in the MATLAB Command Window, type `ssc_gas_tutorial_step1`.

To create this model:

1   In the MATLAB Command Window, type:

    `ssc_new`
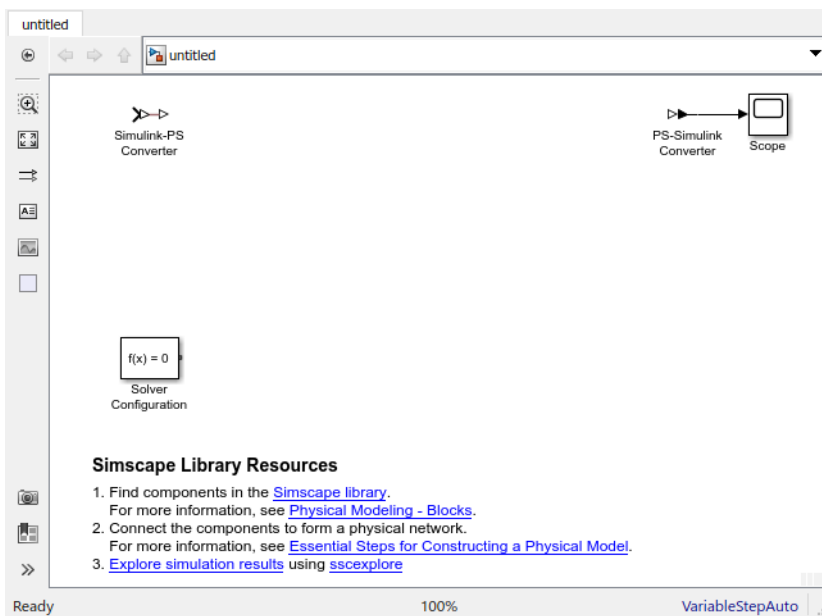


> **Note** By default, Simulink Editor hides the automatic block names in model diagrams. To display hidden block names for training purposes, clear the **Hide Automatic Block Names** check box. For more information, see "Manage Block Names and Ports".

2   Delete the Simulink-PS Converter block.

3   To reduce diagram clutter, right-click the PS-Simulink Converter block and, from the context menu, select **Format** > **Show Block Name** > **Off**.

4   Add the following blocks.

| Block Name | Library | Quantity |
|---|---|---|
| Local Restriction (G) | Gas/Elements | 1 |
| Reservoir (G) | Gas/Elements | 2 |

| Block Name | Library | Quantity |
|---|---|---|
| Gas Properties (G) | Gas/Utilities | 1 |
| Mass & Energy Flow Rate Sensor (G) | Gas/Sensors | 1 |

**5** Change the reservoir block names and connect the blocks as shown in the diagram.
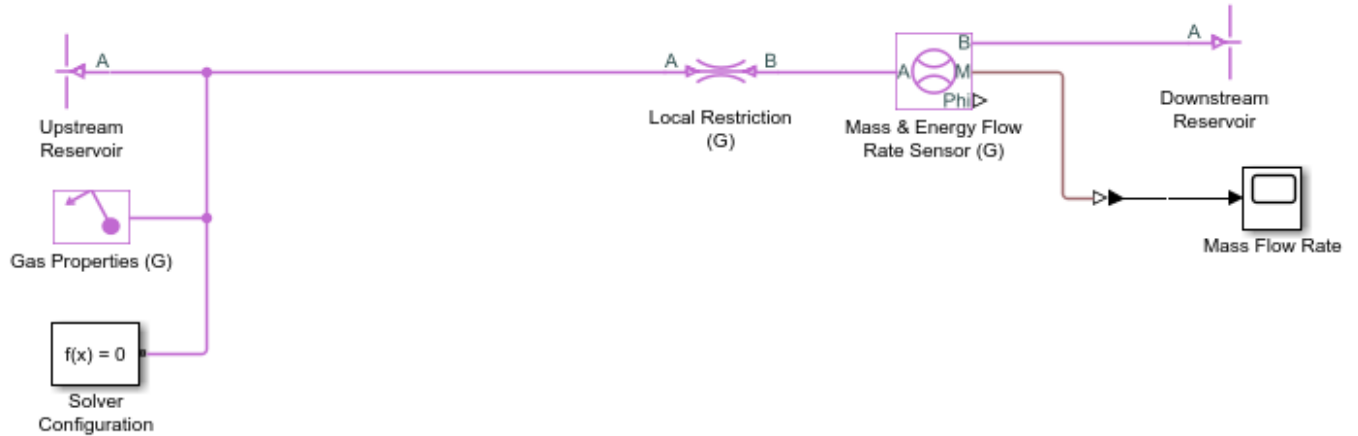


**6** Leave the Downstream Reservoir block at standard atmospheric conditions.
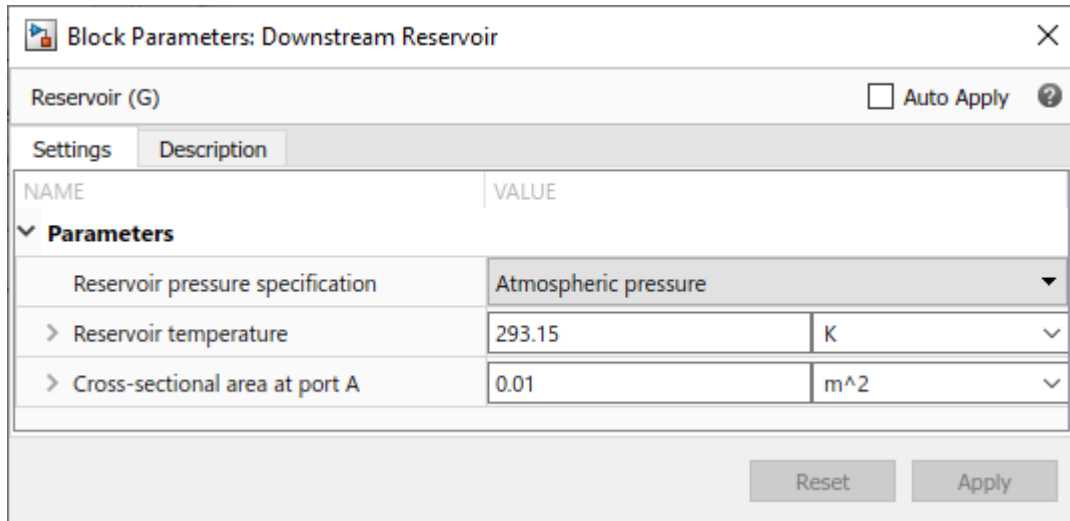


**7** Change the Upstream Reservoir block to have a specified pressure of 0.12 MPa and temperature of 400 K.

Block Parameters: Upstream Reservoir ✕

Reservoir (G)                                ☐ Auto Apply  ?

**Settings**  Description

| NAME | VALUE | | |
|---|---|---|---|
| **∨ Parameters** | | | |
|    Reservoir pressure specification | Specified pressure | | ▼ |
|   > Reservoir pressure | 0.12 | MPa | ∨ |
|   > Reservoir temperature | 400 | K | ∨ |
|   > Cross-sectional area at port A | 0.01 | m^2 | ∨ |

Reset    Apply

**8**    Simulate the model. The mass flow rate through the restriction is approximately 0.13 kg/s.
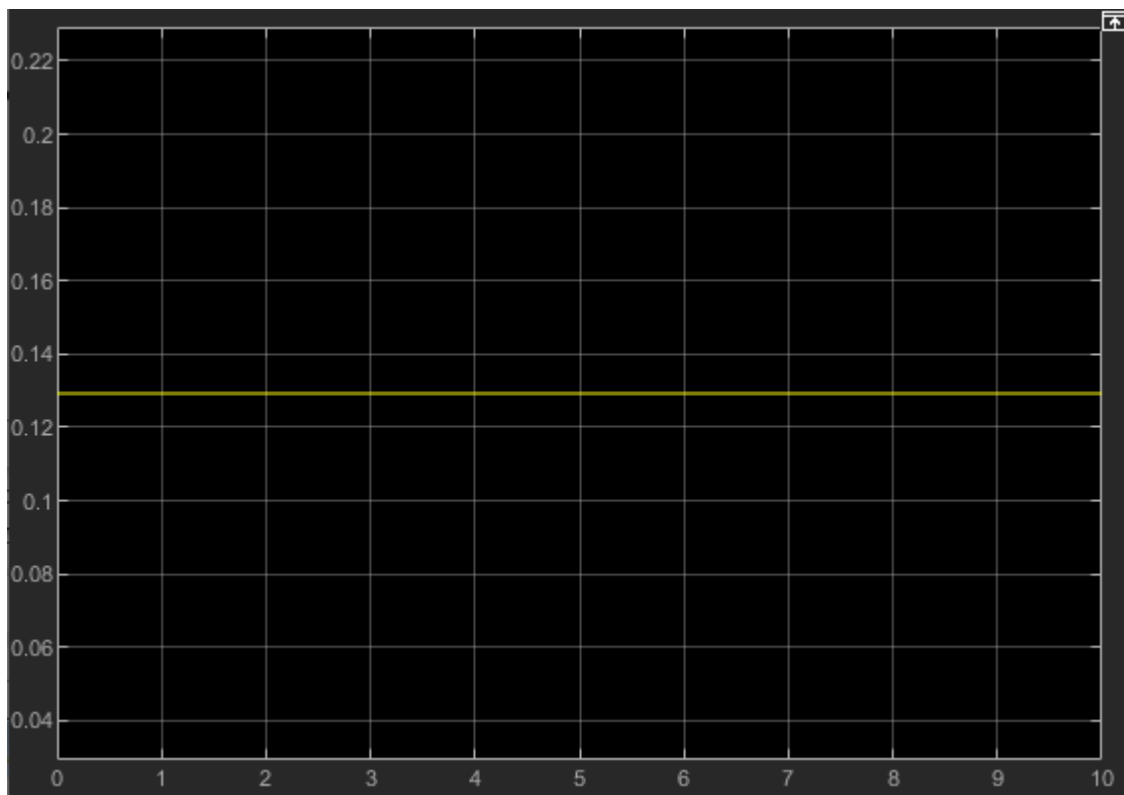
**See Also**

**Related Examples**
- "Change Flow Boundary Conditions" on page 5-15
- "Change Flow Direction" on page 5-20

**More About**
- "Modeling Gas Systems" on page 5-2

# Change Flow Boundary Conditions

In the "Simple Gas Model" on page 5-11 tutorial, you created a simple open-loop gas model. This example shows how to modify this model by changing the gas flow boundary conditions without affecting temperature. To open the completed model, in the MATLAB Command Window, type `ssc_gas_tutorial_step2`.

To change the upstream boundary conditions from specified pressure and temperature to specified mass flow rate and temperature:

1  Open the model created in the "Simple Gas Model" on page 5-11 tutorial, by typing `ssc_gas_tutorial_step1`.
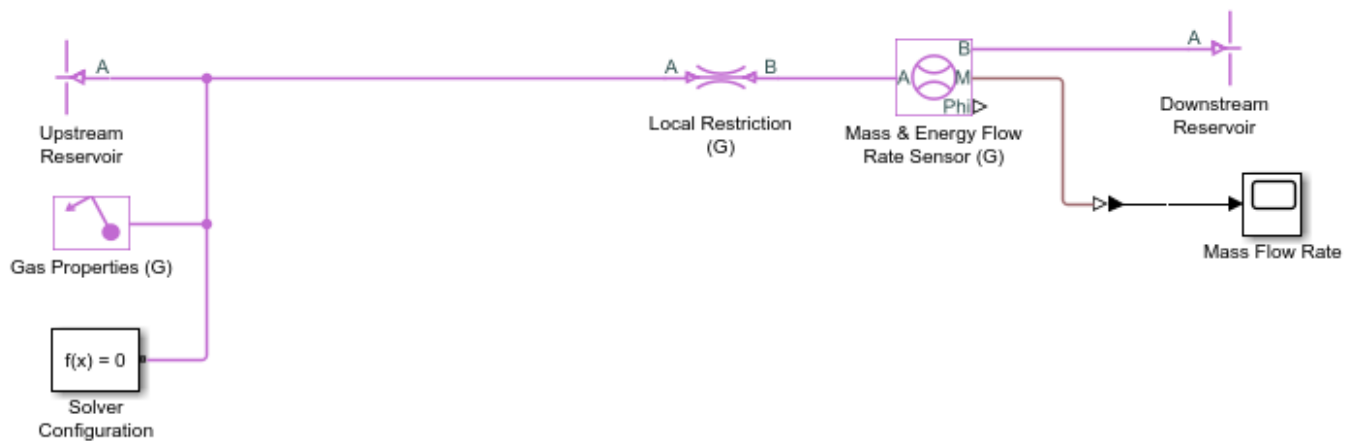


2  Change the Upstream Reservoir block back to `Atmospheric pressure`, but keep the temperature of 400 K.

3  Add a Mass Flow Rate Source (G) block upstream from the local restriction. Set the **Mass flow rate** parameter `0.15` kg/s.

**4** Simulate the model. The mass flow rate through the restriction is now 0.15 kg/s.
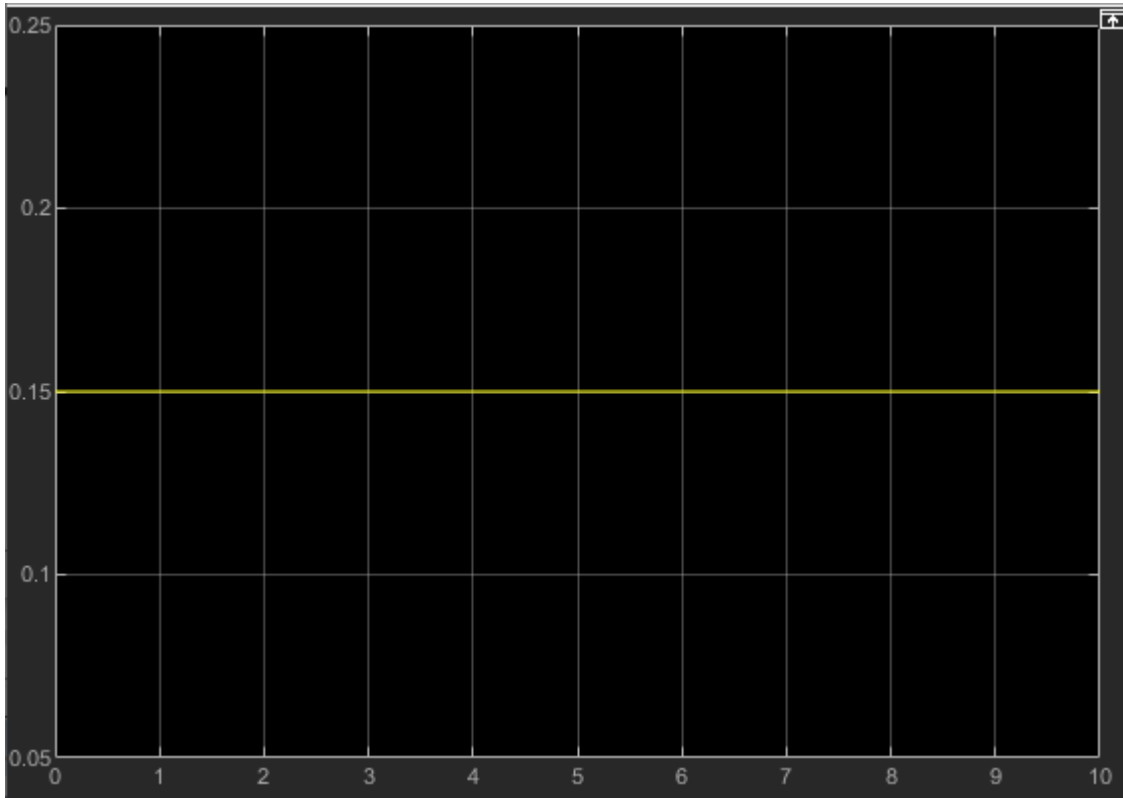


**5** To measure the absolute pressure and temperature upstream of the local restriction, add a Pressure & Temperature Sensor (G) block and connect an Absolute Reference (G) block to the B node of the sensor. Duplicate the converter-scope block pair to add the Pressure and Temperature scopes to the model, as shown in the diagram.
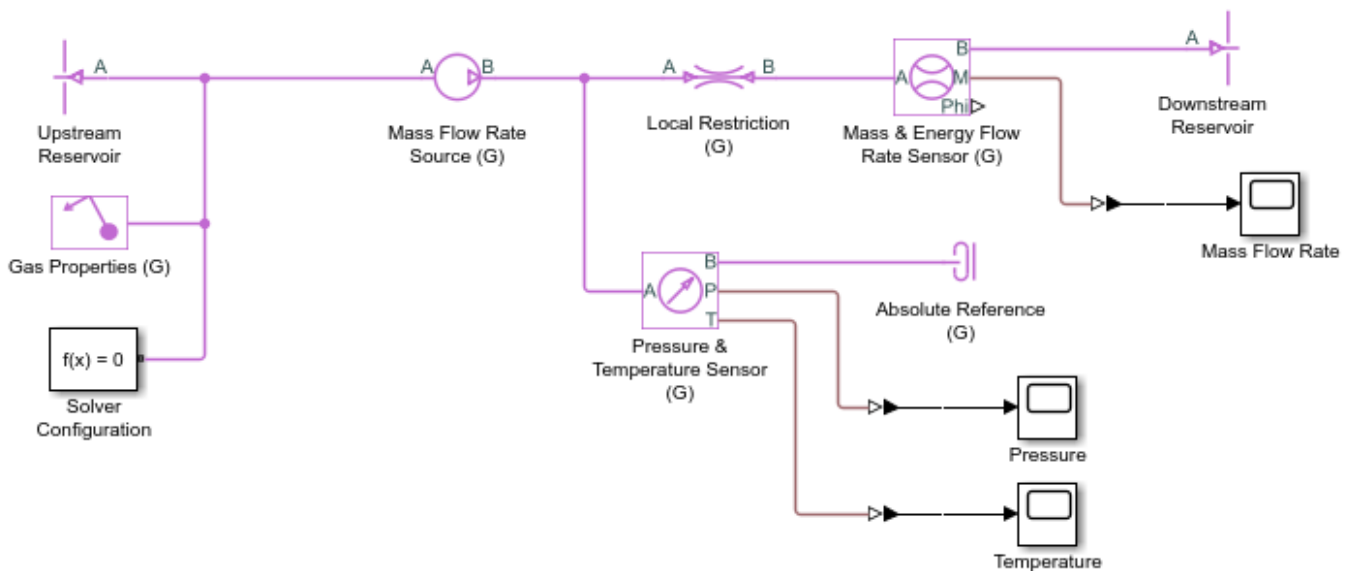
**6** Simulate the model. To drive 0.15 kg/s of gas through the restriction, the Mass Flow Rate Source (G) block increased the pressure from atmospheric (as specified by the Upstream Reservoir block) to almost 0.13 MPa.
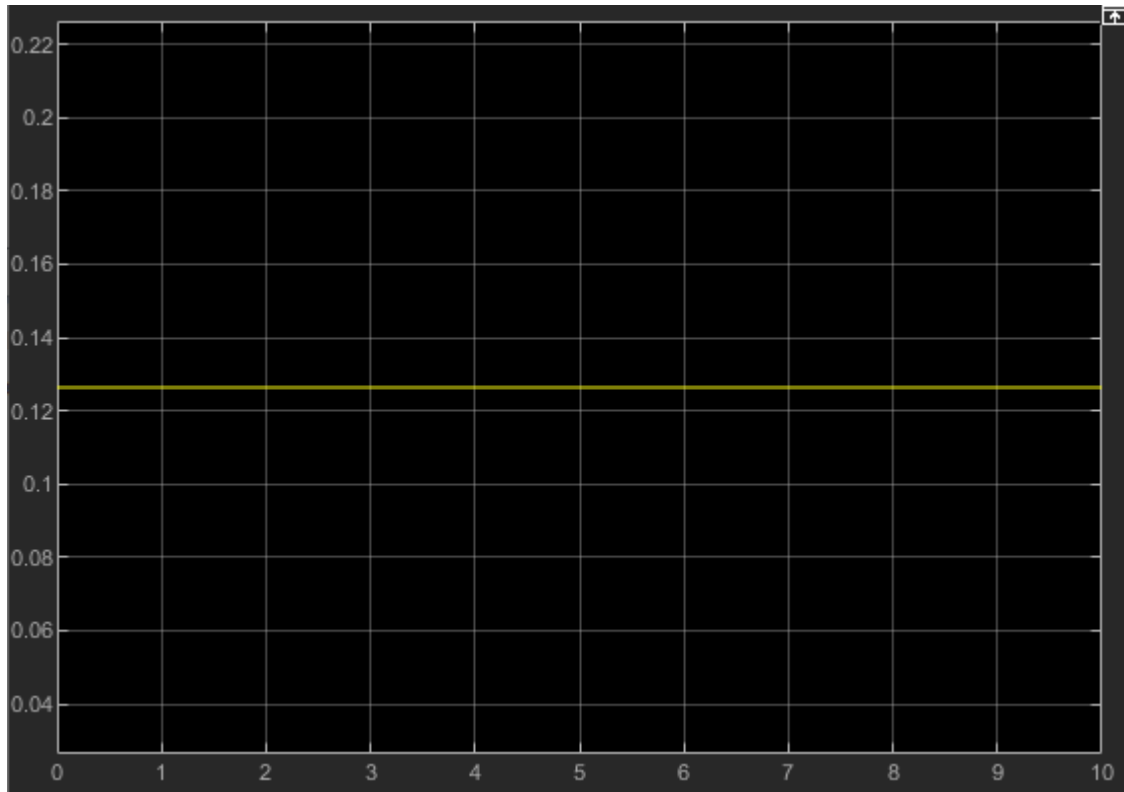


The temperature upstream of the restriction is approximately 427 K, not 400 K (as specified by the Upstream Reservoir block).
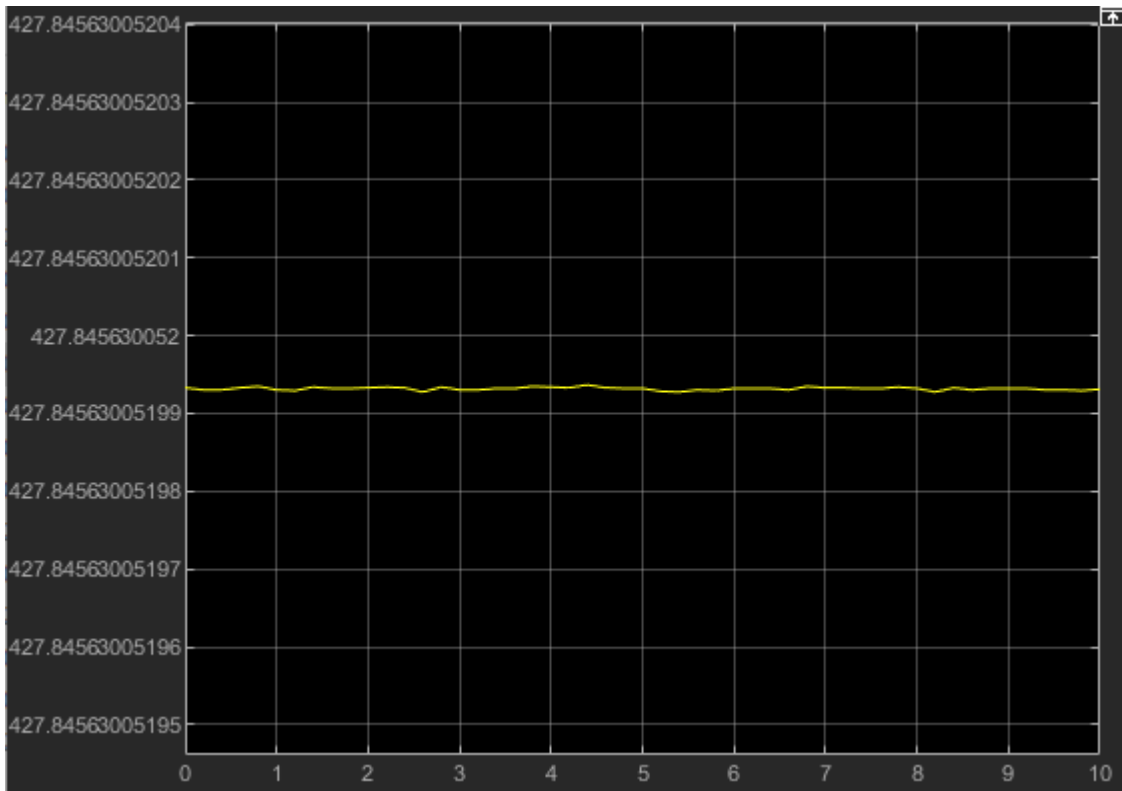
**7** The reason for the temperature increase is that the source needs to do work, to bring the pressure up and drive the desired flow rate through the system, which adds energy to the gas. This way, the source can be treated as an idealized compressor or pump. However, our intent is just to specify an upstream boundary condition of 400 K and 0.15 kg/s, regardless of whether there is actually a compressor upstream or not. Therefore, in the Mass Flow Rate Source (G) block dialog, switch the **Power added** parameter to None.



**8** Simulate the model. The temperature upstream of the restriction is now 400 K.

## See Also

### Related Examples

- "Simple Gas Model" on page 5-11
- "Change Flow Direction" on page 5-20

### More About

- "Modeling Gas Systems" on page 5-2

# Change Flow Direction

In the "Simple Gas Model" on page 5-11 tutorial, you created a simple open-loop gas model, which you then modified in the "Change Flow Boundary Conditions" on page 5-15 tutorial. This example shows how to further modify this model, to investigate how the change in the gas flow direction affects the upstream temperature. To open the completed model, in the MATLAB Command Window, type `ssc_gas_tutorial_step3`.

To investigate the change in the flow direction on the upstream temperature:

**1**    Open the model used in the "Change Flow Boundary Conditions" on page 5-15 tutorial, by typing `ssc_gas_tutorial_step2`.
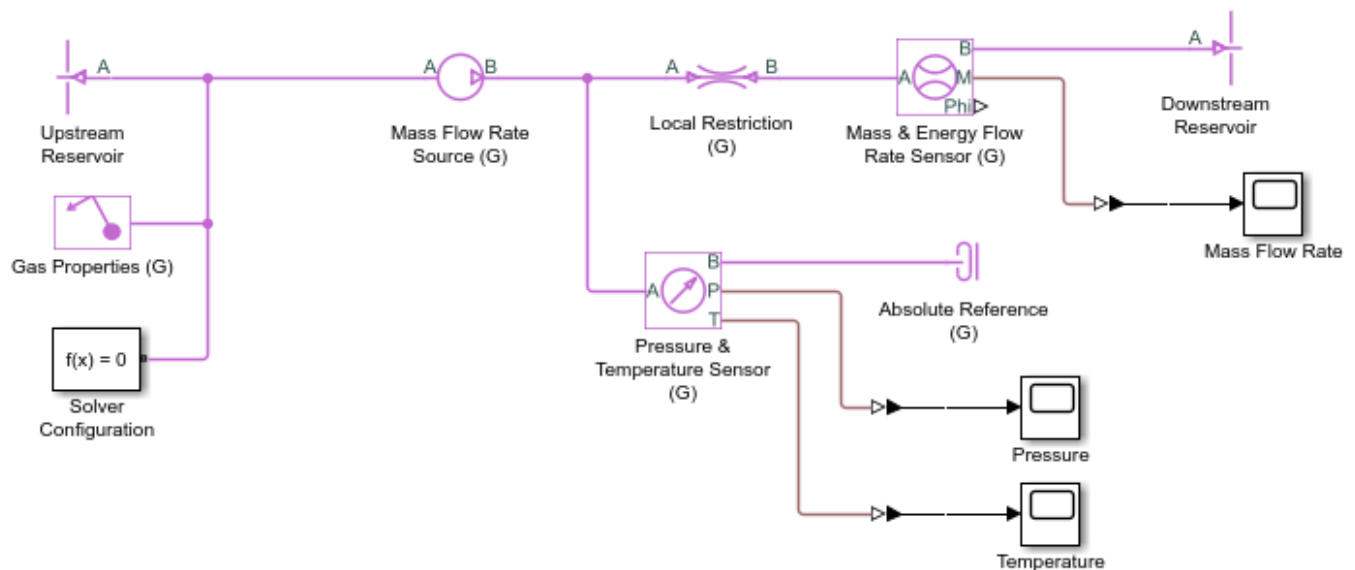


**2**    Replace the Mass Flow Rate Source (G) block with a Controlled Mass Flow Rate Source (G) block, with the **Power added** parameter also set to `None`.

**3**    Add a PS Sine Wave block and connect it to the input port of the Controlled Mass Flow Rate Source (G) block.
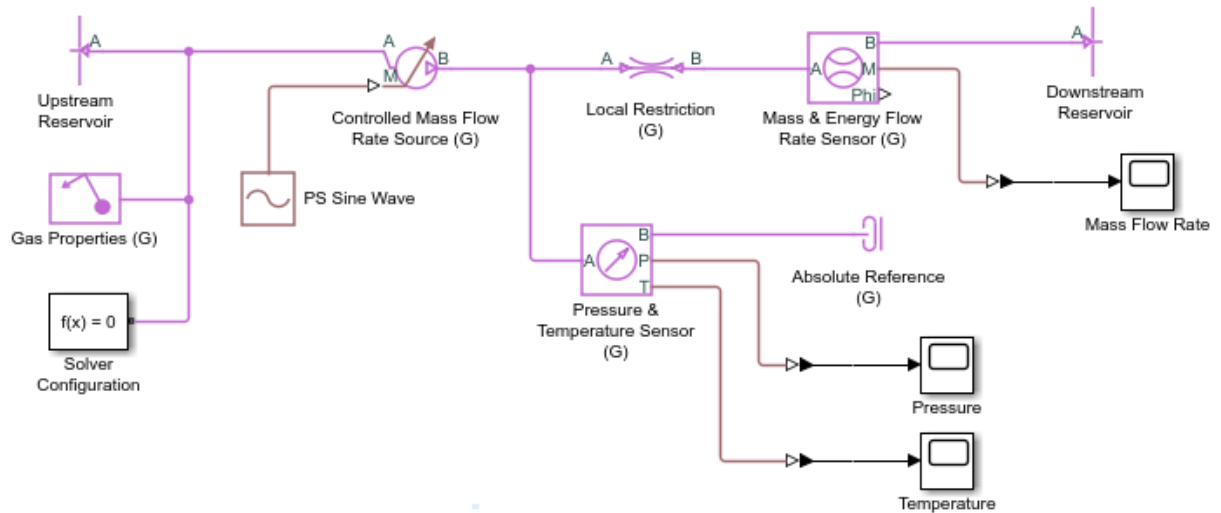
**4**    Set the **Amplitude** parameter of the PS Sine Wave block to `0.15 kg/s` and **Frequency specification** to `Angular`.



**5**    Simulate the model. The mass flow rate through the restriction is now a sinusoidal wave with an amplitude of 0.15 kg/s.
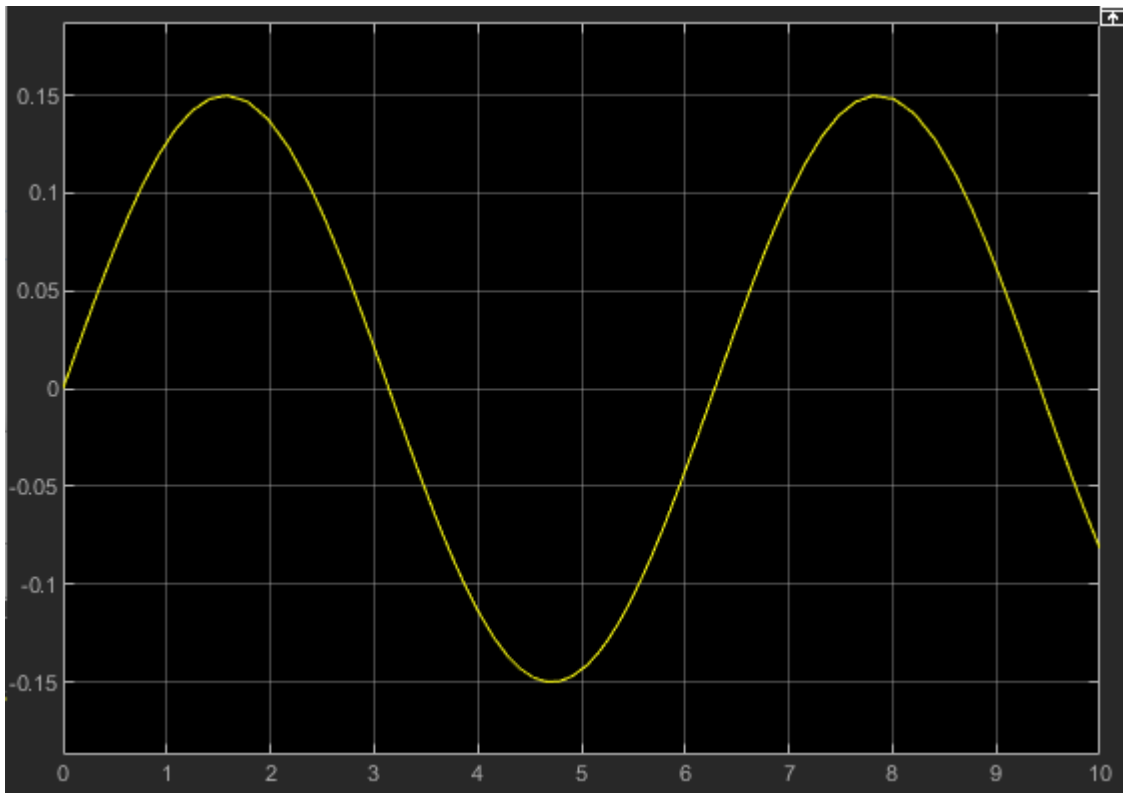
Because the pressure on the right side of the restriction is fixed at atmospheric (as specified by the Downstream Reservoir block), the left side pressure (measured by the sensor) goes up and down, in order to produce the sinusoidal mass flow rate.
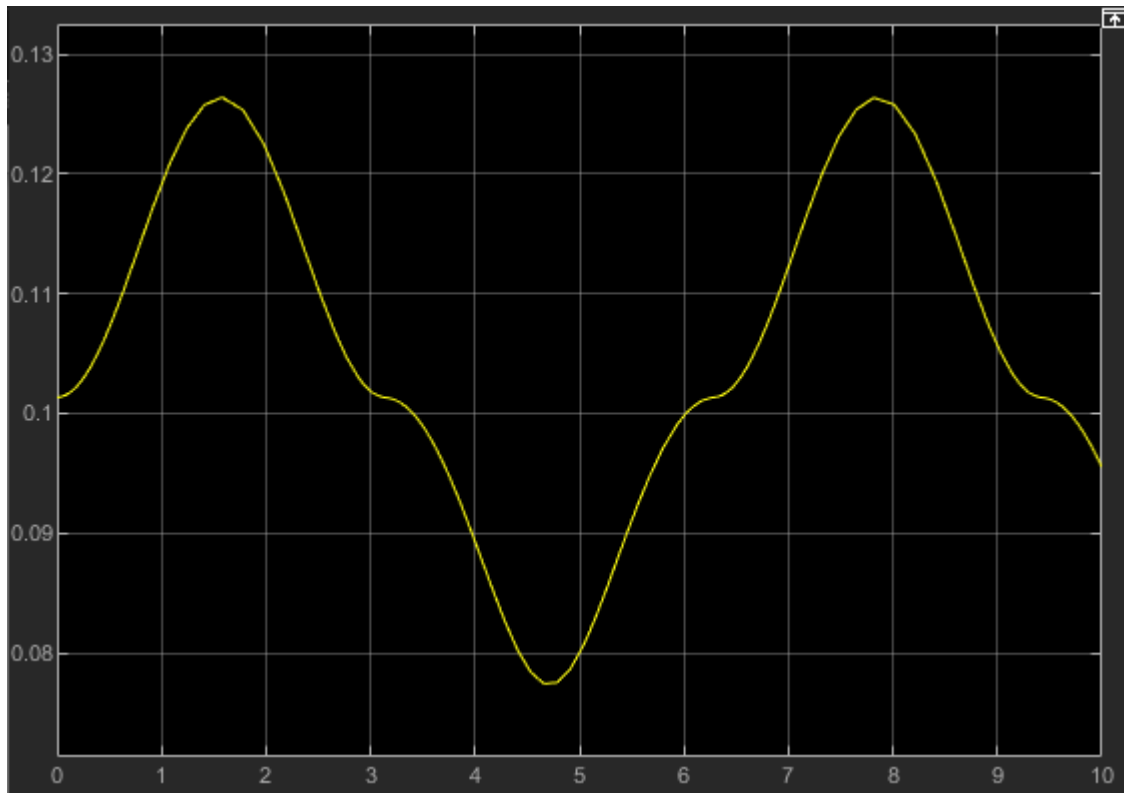
The temperature measured by the sensor switches between 400 K and 293 K. This is because the sensor measures the temperature of the gas just upstream of the measured node. Therefore, when the mass flow rate is positive, the upstream temperature is 400 K (as specified by the Upstream Reservoir block). When the mass flow rate is negative, the upstream temperature is 293 K (as specified by the Downstream Reservoir block), because even though the local restriction does change the temperature of the gas flowing through it, in this model the change is minimal.
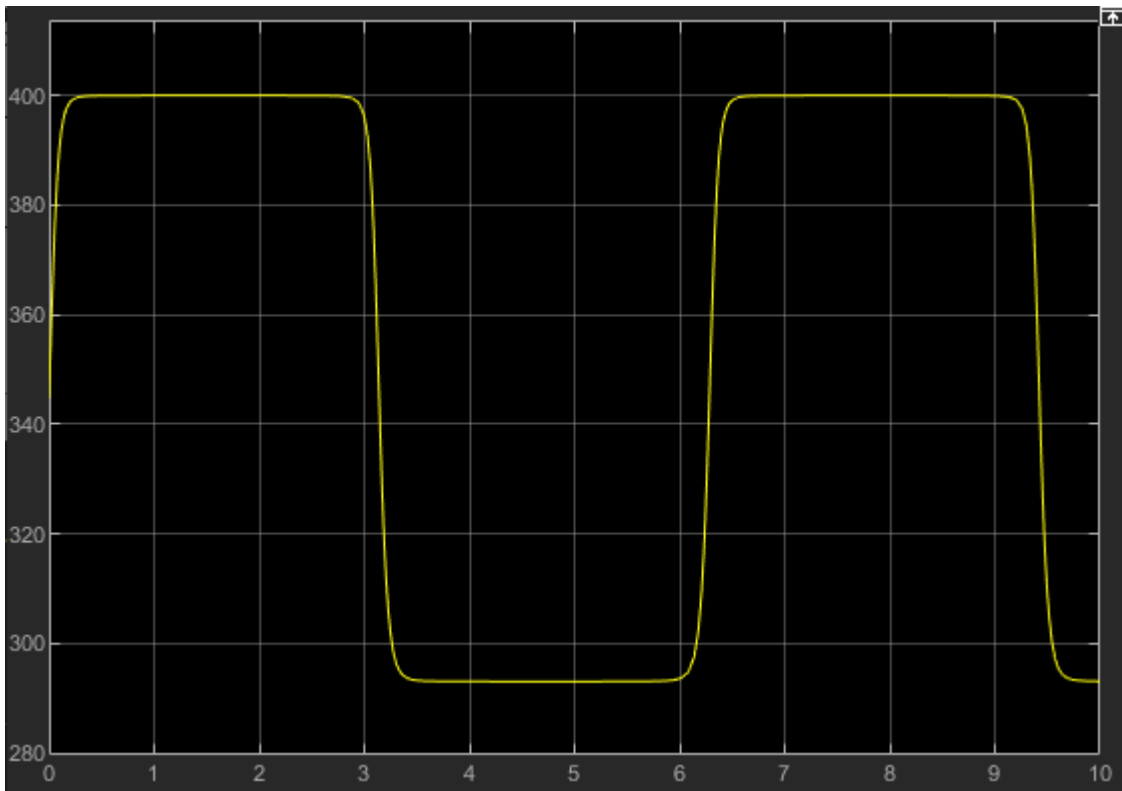
**6**   As you can see, the switching between 400 K and 293 K is smoothed. This improves the
simulation speed and robustness for flow reversal, that is, when the flow changes direction. The
smoothing is controlled by the **Mach number threshold for flow reversal** parameter in the
Gas Properties (G) block. Change this parameter value from `0.01` to `0.001` and rerun the
simulation.
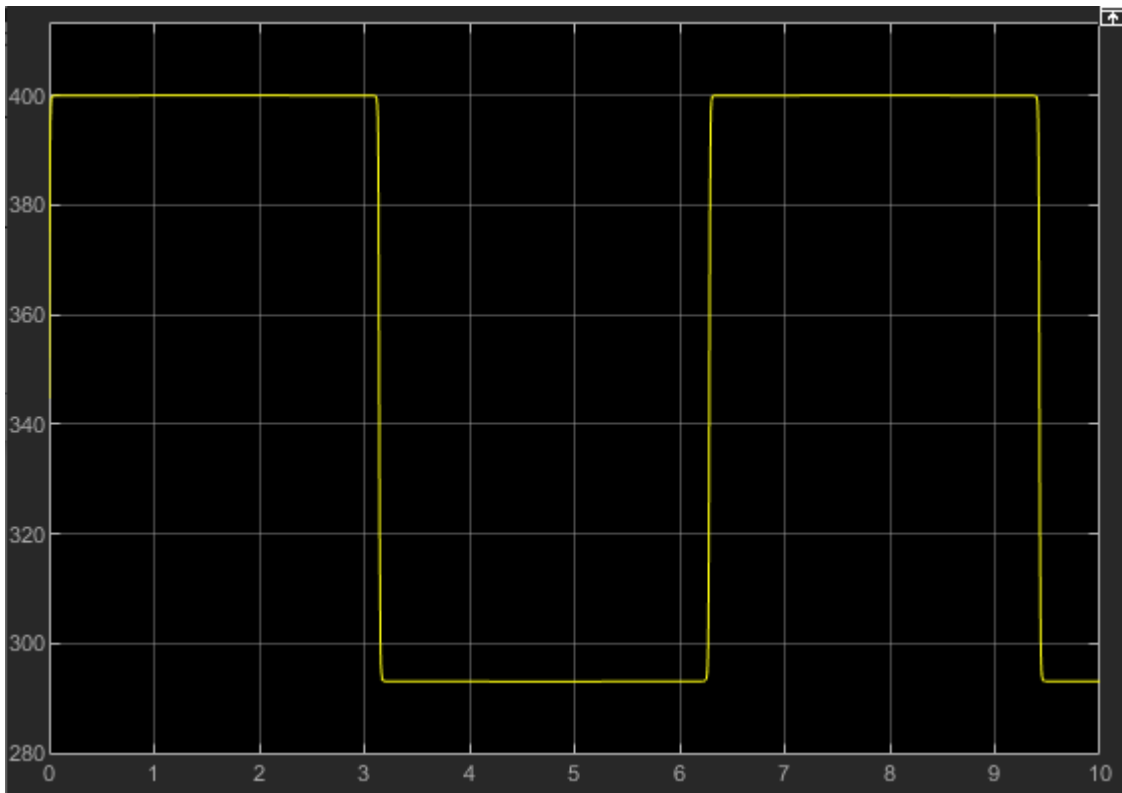
The temperature now switches between 400 K and 293 K more sharply. Reducing the value of the **Mach number threshold for flow reversal** parameter can improve accuracy of the results, but may cause difficulties for the solver in a larger model, with large temperature variations and fast flow reversals.

**7** Change the **Mach number threshold for flow reversal** parameter back to the default value of `0.01`, which provides sufficient accuracy for this model.

Because the temperature at a node represents the temperature of the gas just upstream of that node, other thermodynamic properties that depend on temperature also represent the gas just upstream of the node. For example, if you use the Thermodynamic Properties Sensor (G) block to measure gas density, the density value also goes up and down as the flow reverses, similar to the temperature plot.

## See Also

## Related Examples
- "Simple Gas Model" on page 5-11
- "Change Flow Boundary Conditions" on page 5-15

## More About
- "Modeling Gas Systems" on page 5-2

# Moist Air System Models

# Modeling Moist Air Systems

## Intended Applications

The Moist Air library contains basic elements, such as reservoirs, chambers, and pneumatic-mechanical converters, as well as sensors and sources. Use these blocks to model HVAC systems, environmental control systems, and other similar applications.

Relevant industries include automotive, aerospace, building. The key aspect of these applications is the need to keep track of humidity levels in different parts of the model over time. The moist air domain is a two-species gas domain, where the species are air and water vapor. Furthermore, the water vapor can condense out of the system. This effect is important to HVAC applications because latent heat of water condensation affects the thermodynamics of the fluid flow.

The moist air mixture is composed of dry air and water vapor. Trace gas is an optional third species in the moist air mixture. Example of the trace gas usage is to track carbon dioxide and pollutants such as nitrogen oxides (NOx). You specify the moist air properties in the connected loop by using the Moist Air Properties (MA) block. This block also gives you several options for modeling trace gas properties. You increase and decrease levels of moisture and trace gas in the air mixture by using the blocks in the Moisture & Trace Gas Sources library (see "Modeling Moisture and Trace Gas Levels" on page 6-13).

All gas species in the mixture are assumed to be semiperfect gas. This means that pressure, temperature, and density obey the ideal gas law. Other properties—specific enthalpy, specific heat, dynamic viscosity, and thermal conductivity—are functions of temperature only.

Unless explicitly stated otherwise, all pressures and temperatures used in modeling moist air systems are static pressure and static temperature.

Use the Moist Air domain and library to perform the following tasks:

- Develop requirements of an HVAC system for an environment, such as a building, automobile, or aircraft
- Ensure acceptable temperature, pressure, humidity, and condensation within the environment
- Determine capacity of an HVAC system to match heating, cooling, and dehumidification requirements
- Analyze HVAC system performance, efficiency, and cost

- Validate HVAC system model against test data
- Design and simulate HVAC components and tune component models to test rig data
- Simulate models including an HVAC system, environment model, and controller
- Design controllers for valves, fans, and compressors to ensure safe and optimal operation
- Perform HIL testing

## Network Variables for Moist Air Domain

The Across variables are pressure, temperature, specific humidity (water vapor mass fraction), and trace gas mass fraction. The Through variables are mixture mass flow rate, mixture energy flow rate, water vapor mass flow rate, and trace gas mass flow rate. Note that these choices result in a pseudo-bond graph, because the product of Across and Through variables is not power.

There is a separate domain for modeling moisture and trace gas levels in moist air systems. For more information, see "Moist Air Source Domain" on page 6-13.

## Moist Air Properties

The default fluid properties for the moist air library correspond to dry air, water vapor, and carbon dioxide (the optional trace gas). However, you can modify the fluid properties in the Moist Air Properties (MA) block to model mixtures of other gases and vapors. You can replace dry air and carbon dioxide with other gas species. You can also change water vapor to another condensing vapor (or even to another noncondensing gas species, by supplying large enough values for the saturation pressure, so that it would never reach saturation during simulation). In this way, you can model any three-species gas mixture.

All gas species in the mixture are assumed to be semiperfect gas. This means that pressure $p$, temperature $T$, and density $\rho$ of the constituents obey the ideal gas law:

$$p_a = \rho_a R_a T,$$
$$p_w = \rho_w R_w T,$$
$$p_g = \rho_g R_g T,$$

where $R$ is the specific gas constant. Subscripts $a$, $w$, and $g$ indicate dry air, water vapor, and trace gas, respectively.

Dalton's law applies to ideal gases:

$$p = p_a + p_w + p_g .$$

Therefore, the mixture also obeys the ideal gas law:

$$p = \rho R T,$$

where:

$$\rho = \rho_a + \rho_w + \rho_g,$$
$$R = x_a R_a + x_w R_w + x_g R_g .$$

$x_a$, $x_w$, and $x_g$ are mass fractions of dry air, water vapor, and trace gas, respectively.

Other properties of each constituent are assumed to be functions of temperature only:

- $h_a(T)$, $h_w(T)$, $h_g(T)$ — Specific enthalpy of dry air, water vapor, and trace gas, respectively.
- $\mu_a(T)$, $\mu_w(T)$, $\mu_g(T)$ — Dynamic viscosity of dry air, water vapor, and trace gas, respectively.
- $k_a(T)$, $k_w(T)$, $k_g(T)$ — Thermal conductivity of dry air, water vapor, and trace gas, respectively.

For ideal gases, the enthalpy of mixing is zero. Therefore, the mixture specific enthalpy is a combination of the constituent specific enthalpy based on their mass fractions:

$$h = x_a h_a(T) + x_w h_w(T) + x_g h_g(T).$$

You can compute the entropy of mixing from the mole fractions:

$$\Delta s^{mix} = -x_a R_a \ln(y_a) + x_w R_w \ln(y_w) + x_g R_g \ln(y_g),$$

where $y_a$, $y_w$, and $y_g$ are mole fractions of dry air, water vapor, and trace gas, respectively.

Therefore, the mixture specific entropy is

$$s = x_a s_a + x_w s_w + x_g s_g + \Delta s^{mix}.$$

## Humidity and Trace Gas Property Definitions

The equations describing humidity and trace gas properties use these symbols and property definitions. Subscripts a, w, and g indicate the properties of dry air, water vapor, and trace gas, respectively. Subscript ws indicates water vapor at saturation.

| Symbol | Property | Definition |
|---|---|---|
| $p$ | Pressure | Pressure of the moist air mixture (as opposed to the partial pressure of water vapor or partial pressure of trace gas). |
| $T$ | Temperature | The dry-bulb temperature, which is the temperature in the common thermodynamic sense. (The wet-bulb temperature is a different quantity, which measures the moisture level.) |
| $R$ | Specific gas constant | Universal gas constant divided by the molar mass of the species. Specific gas constant of the mixture is $$R = x_a R_a + x_w R_w + x_g R_g.$$ |

| Symbol | Property | Definition |
| --- | --- | --- |
| $\varphi_w$ | Relative humidity | Moles of water vapor as a fraction of the moles of water vapor needed to saturate at the same temperature. Water vapor saturation pressure is a property of water and is a function of temperature only, $p_{ws}(T)$. The ideal gas law (due to the assumed semiperfect gas) means that mole fraction is equivalent to partial pressure fraction. The mole fraction $y_w$ cannot be greater than 1. Therefore, at high temperature or low pressure, it may not be possible to achieve relative humidity of 1. $$\varphi_w = \frac{y_w}{y_{ws}}\bigg|_T = \frac{y_w p}{p_{ws}(T)}$$ |
| $x_w$ | Specific humidity | Mass of water vapor as a fraction of the total mass of the moist air mixture. It is another term for water vapor mass fraction. It may not be possible to achieve specific humidity of 1 due to saturation. $$x_w = \frac{M_w}{M} = \frac{\dot{m}_w}{\dot{m}}$$ |
| $y_w$ | Water vapor mole fraction | Moles of water vapor as a fraction of the total moles of the moist air mixture. It may not be possible to achieve water vapor mole fraction of 1 due to saturation. $$y_w = \frac{N_w}{N} = \frac{p_w}{p}$$ |
| $r_w$ | Humidity ratio | Ratio of mass of water vapor to mass of dry air and trace gas. For conditions in typical HVAC applications, it is close to the specific humidity. $$r_w = \frac{M_w}{M_a + M_g}$$ |
| $\rho_w$ | Absolute humidity | Mass of water vapor over the volume of the moist air mixture. It is another term for water vapor density. $$\rho_w = \frac{M_w}{V}$$ |
| $x_g$ | Trace gas mass fraction | Mass of trace gas as a fraction of the total mass of the moist air mixture. $$x_g = \frac{M_g}{M} = \frac{\dot{m}_g}{\dot{m}}$$ |
| $y_g$ | Trace gas mole fraction | Moles of trace gas as a fraction of the total moles of the moist air mixture. $$y_g = \frac{N_g}{N} = \frac{p_g}{p}$$ |

Water vapor mole fraction is related to specific humidity (that is, to mass fraction) as follows:

$$y_w = \frac{R_w}{R}x_w.$$

Trace gas mole fraction is related to trace gas mass fraction as follows:

$$y_g = \frac{R_g}{R}x_g.$$

## Blocks with Moist Air Volume

Components in the moist air domain are modeled using control volumes. A control volume encompasses the moist air inside the component and separates it from the surrounding environment and other components. Air flows and heat flows across the control surface are represented by ports. The moist air volume inside the component is represented using an internal node. This internal node is not visible, but you can access its parameters and variables using Simscape data logging. For more information, see About Simulation Data Logging on page 13-2.

For a moist air volume, you must specify the pressure, temperature, moisture level, and trace gas level. For more information, see "Initial Conditions for Blocks with Finite Moist Air Volume" on page 6-7.

The following blocks in the Moist Air library are modeled as components with a moist air volume. In the case of Controlled Reservoir (MA) and Reservoir (MA) blocks, the volume is assumed to be infinitely large.

| Block | Gas Volume |
|---|---|
| Constant Volume Chamber (MA) | Finite |
| Pipe (MA) | Finite |
| Rotational Mechanical Converter (MA) | Finite |
| Translational Mechanical Converter (MA) | Finite |
| Reservoir (MA) | Infinite |
| Controlled Reservoir (MA) | Infinite |

Other components have relatively small moist air volumes, so that the air mixture entering the component spends negligible time inside the component before exiting. These components are considered quasi-steady-state and they do not have an internal node.

## Reference Node and Grounding Rules

Unlike mechanical and electrical domains, where each topologically distinct circuit within a domain must contain at least one reference block, moist air networks have different grounding rules.

Blocks with a moist air volume contain an internal node, which provides the pressure, temperature, moisture level, and trace gas level inside the component and therefore serves as a reference node for the moist air network. Each connected moist air network must have at least one reference node. This means that each connected moist air network must have at least one of the blocks listed in "Blocks with Moist Air Volume" on page 6-6. In other words, a moist air network that contains no air volume is an invalid network.

The Foundation Moist Air library contains the Absolute Reference (MA) block but, unlike other domains, you do not use it for grounding moist air circuits. The purpose of the Absolute Reference

(MA) block is to provide a reference for the Pressure & Temperature Sensor (MA) block. If you use the Absolute Reference (MA) block elsewhere in a moist air network, it triggers a simulation assertion because air mixture pressure and temperature cannot be at absolute zero.

## Initial Conditions for Blocks with Finite Moist Air Volume

This section discusses the specific initialization requirements for blocks modeled with finite moist air volume. These blocks are listed in "Blocks with Moist Air Volume" on page 6-6.

The fluid states of a moist air volume are pressure, temperature, moisture level, and trace gas level. These fluid states evolve dynamically based on the mixture mass conservation, water vapor mass conservation, trace gas mass conservation, and mixture energy conservation. Therefore, it is necessary to specify initial conditions for these blocks, to define the initial fluid states. The dialog box of each block modeled with finite moist air volume has an **Initial Targets** section, which lets you specify the initial conditions. To ensure consistent initial conditions, specify high priority targets for four variables:

- **Pressure of moist air volume**
- **Temperature of moist air volume**
- One of the variables representing the moisture level:

    - **Relative humidity of moist air volume**
    - **Specific humidity of moist air volume**
    - **Water vapor mole fraction of moist air volume**
    - **Humidity ratio of moist air volume**
- One of the variables representing the trace gas level:

    - **Trace gas mass fraction of moist air volume**
    - **Trace gas mole fraction of moist air volume**

It is important that only four variables, as described, have their priorities set to `High` for each block with a finite moist air volume. Placing high-priority constraints on additional variables results in overspecification, with the solver being unable to find an initialization solution that satisfies the desired initial values. Set the priority of remaining variables to `None`. You can use the equations in "Humidity and Trace Gas Property Definitions" on page 6-4 and "Trace Gas Modeling Options" on page 6-13 to convert values from one moisture or trace gas measure to another. For more information on variable initialization and dealing with overspecification, see "Initialize Variables for a Mass-Spring-Damper System" on page 8-6.

The fluid states of moist air volume in these blocks are reported by the physical signal output port **F**. Connect port **F** to the Measurement Selector (MA) block to extract the measurements of pressure, temperature, moisture level, and trace gas level during simulation.

In blocks that are modeled with an infinitely large moist air volume, the state of the volume is assumed quasisteady and there is no need to specify an initial condition. Instead, these blocks represent boundary conditions for the moist air network.

## Saturation and Condensation

Blocks with finite moist air volume (listed in "Blocks with Moist Air Volume" on page 6-6) can become saturated when the relative humidity $\varphi_w$ reaches the relative humidity at saturation $\varphi_{ws}$. The

saturated state represents the maximum amount of moisture that the moist air volume can hold at that pressure and temperature. Any additional moisture condenses into liquid water.

By definition, the relative humidity at saturation is 1. However, you can specify a different value for $\varphi_{ws}$ to model some empirical effect or other phenomenon. When $\varphi_{ws} > 1$, water vapor partial pressure can become greater than the water vapor saturation pressure. When $\varphi_{ws} < 1$, moisture can condense before the water vapor partial pressure reaches the water vapor saturation pressure.

Condensation does not occur instantaneously. Consequently, it is possible for $\varphi_w$ to be slightly greater than $\varphi_{ws}$. The condensation time constant represents the characteristic time it takes to condense out enough moisture to bring $\varphi_w$ back to $\varphi_{ws}$. A larger value of the time constant causes $\varphi_w$ to exceed $\varphi_{ws}$ to a greater degree, but is more numerically robust.

Moisture that condenses is considered to have left the moist air network, therefore, the mass and energy of condensed liquid water is subtracted from the moist air volume. The rate of condensation is reported by the physical signal output port **W**. If you want to model the flow of the condensed liquid water, you can use the rate of condensation as an input for another fluid network (hydraulic, thermal liquid, two-phase fluid, or another moist air network). The following example shows how to use the thermal liquid network to model the condensate that drains from a Constant Volume Chamber (MA) through a pipe.
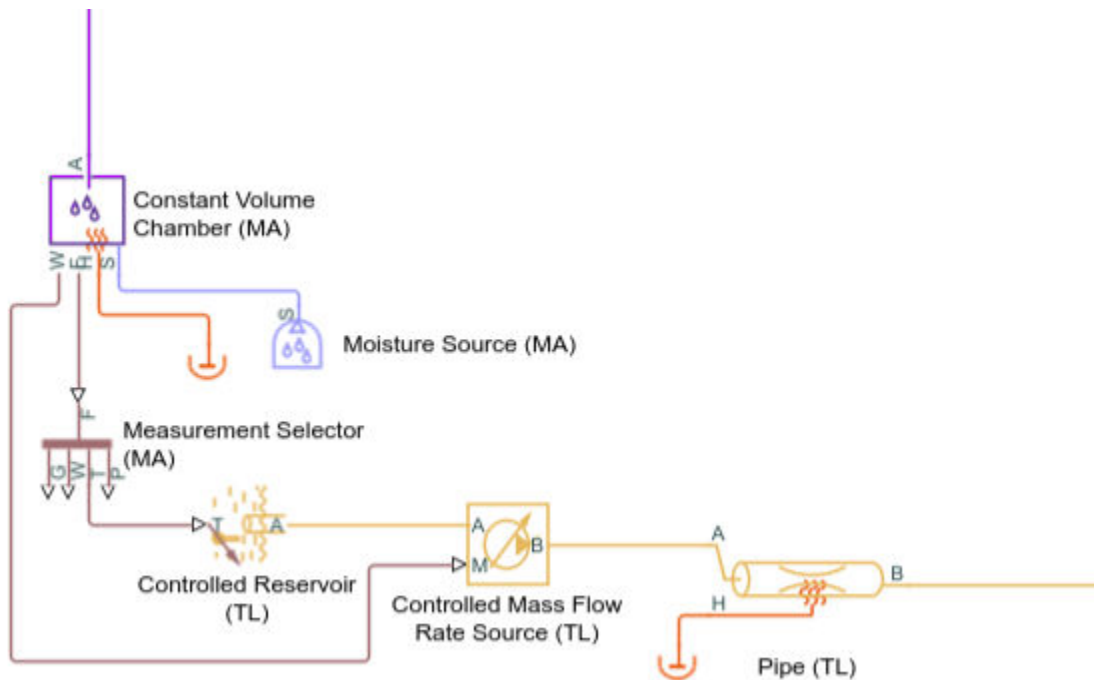


If you have a Simscape Fluids license, you can also use the Tank (TL) block to model a condensation collection tray. The liquid level in the tank represents the amount of condensation collected but not yet drained from the tank.
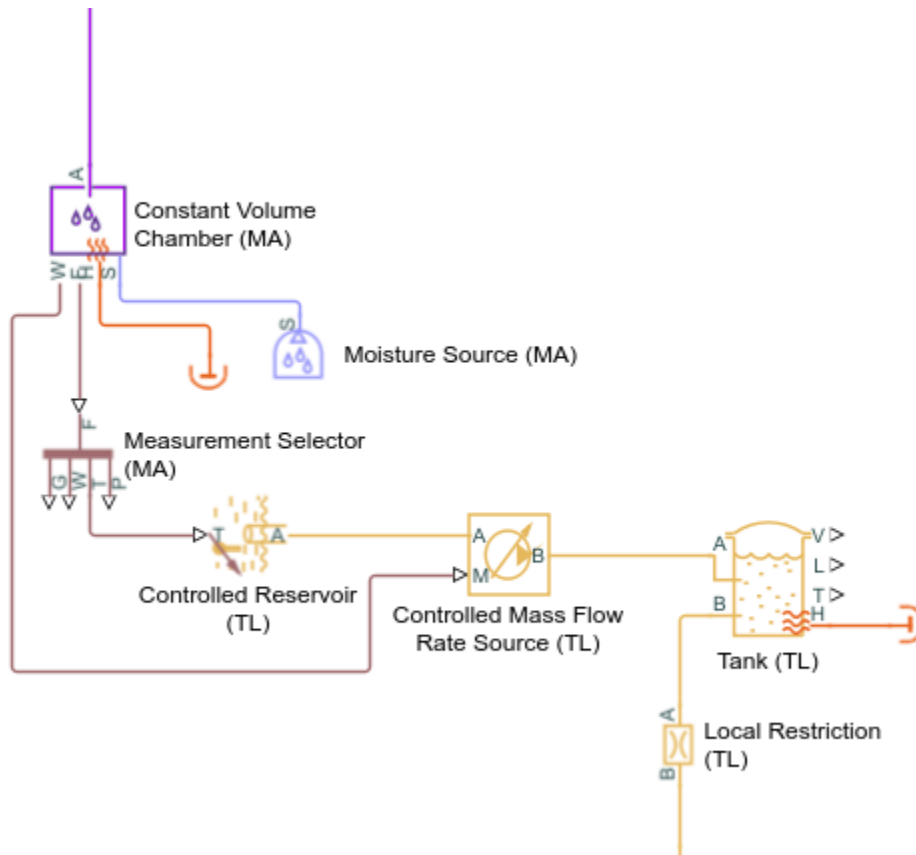
## Choked Flow

Moist air flow through Local Restriction (MA), Variable Local Restriction (MA), or Pipe (MA) blocks can become choked. Choking occurs when the flow velocity reaches the local speed of sound. When the flow is choked, the velocity at the point of choking cannot increase any further. However, the mass flow rate can still increase if the density of the air mixture increases. This can be achieved, for example, by increasing the pressure upstream of the point of choking. The effect of choking on a moist air network is that the mass flow rate through the branch containing the choked block depends completely on the upstream pressure and temperature. As long as the choking condition is maintained, this choked mass flow rate is independent of any changes occurring in the pressure downstream.

The following model illustrates the choked flow. In this model, the Ramp block has a slope of 0.005 and the start time of 10. The Simulink-PS Converter block has **Input signal unit** set to MPa. All other blocks have default parameter values. Simulation time is 50 s. When you simulate the model, the pressure at port A of the Local Restriction (MA) block increases linearly from atmospheric pressure, starting at 10 s. The pressure at port B is fixed at atmospheric pressure.
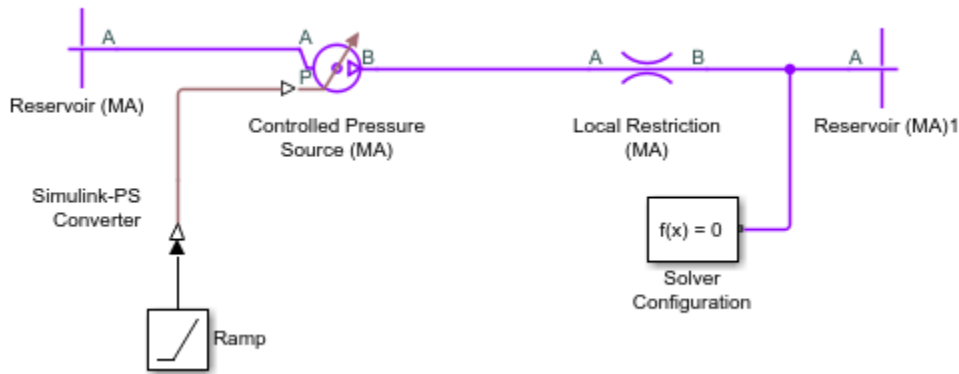
The following two plots show the logged simulation data for the Local Restriction (MA) block. The Mach number at the restriction (**Mach**) reaches 1 at around 20 s, indicating that the flow is choked. The mass flow rate (**mdot_A**) before the flow is choked follows the typical quadratic behavior with respect to an increasing pressure difference. However, the mass flow rate after the flow is choked becomes linear because the choked mass flow rate depends only on the upstream pressure and temperature, and the upstream pressure is increasing linearly.

The fact that the choked mass flow rate depends only on the upstream conditions can cause an incompatibility with a Mass Flow Rate Source (MA) or a Controlled Mass Flow Rate Source (MA) block connected downstream of the choked block. Consider this model, which contains the Controlled Mass Flow Rate Source (MA) block instead of the Controlled Pressure Source (MA) block.
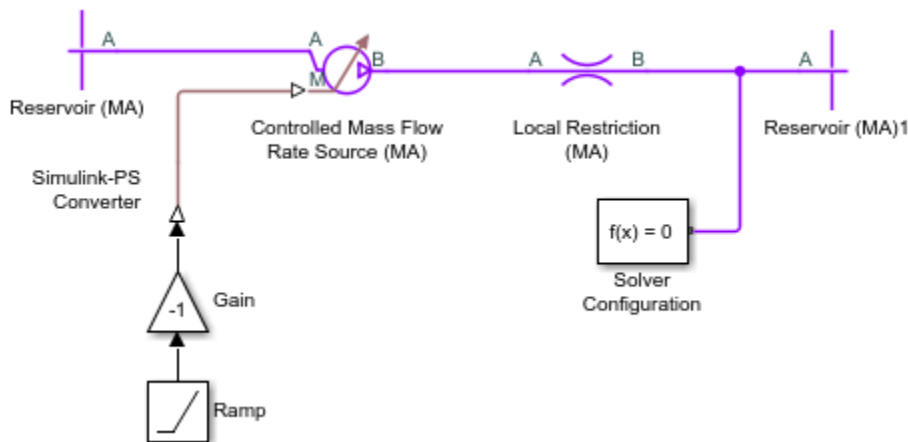


If the source commanded an increasing mass flow rate from left to right through the Local Restriction (MA) block, the simulation would succeed even if the flow became choked because the Controlled Mass Flow Rate Source (MA) block would be upstream of the choked block. However, in this model, the Gain block reverses the flow direction, so that the Controlled Mass Flow Rate Source (MA) block is downstream of the choked block. The pressure upstream of the Local Restriction (MA) block is

fixed at atmospheric pressure. Therefore, the choked mass flow rate in this situation is constant. As the commanded mass flow rate increases, eventually it will become greater than this constant value of choked mass flow rate. At this point, the commanded mass flow rate and the choked mass flow rate cannot be reconciled and the simulation fails. Viewing the logged simulation data in the Simscape Results Explorer shows that simulation fails just at the point when the Mach number reaches 1 and the flow becomes choked.



In general, if a model is likely to choke, use pressure sources rather than mass flow rate sources. If a model contains mass flow rate source blocks and the simulation fails, use the Simscape Results Explorer to inspect the Mach number variables in all Local Restriction (MA), Variable Local Restriction (MA), and Pipe (MA) blocks connected along the same branch as the mass flow rate source. If the simulation failure occurs when the Mach number reaches 1, it is likely that there is a downstream mass flow rate source trying to command a mass flow rate greater than the possible choked mass flow rate.

## See Also

## More About

- "Modeling Moisture and Trace Gas Levels" on page 6-13

# Modeling Moisture and Trace Gas Levels

## Moist Air Source Domain

Unlike the moist air domain, the moist air source domain is a separate domain for modeling moisture and trace gas levels in moist air systems.

Regular connection ports (boundary nodes) of the Moist Air library blocks belong to the moist air domain. These ports are usually named **A** or **B**.

Blocks with a finite moist air volume also contain an internal node, which provides the pressure, temperature, moisture level, and trace gas level inside the component. This internal node connects to the moist air source domain. The corresponding port is named **S**.

For example, these are the ports of a Pipe (MA) block.
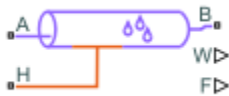


- **A** and **B** — Moist air conserving ports associated with the boundary nodes. Use these ports to connect this block to other blocks in a moist air circuit.
- **S** — Moist air source conserving port associated with the internal node. Use this port to model moisture and trace gas levels, by connecting it to a block in the Moisture & Trace Gas Sources library.
- **H** — Thermal conserving port associated with the temperature either of the air mixture inside the block or of a block element, such as pipe wall.
- **W** and **F** — Physical signal output ports.

Blocks in the Moisture & Trace Gas Sources library, which inject or extract moisture and trace gas, also have a port **S** that belongs to the moist air source domain. This naming convention helps to distinguish the two different domain types. For more information, see "Using Moisture and Trace Gas Sources" on page 6-14.

## Trace Gas Modeling Options

The Moist Air Properties (MA) block provides three ways to model the amount of trace gas in the air mixture of the connected circuit:

- None — No trace gas is present. The moist air mixture consists only of dry air and water vapor.

- `Track mass fraction only` — The trace gas level can be nonzero and vary during simulation. However, the amount of trace gas is assumed to be small enough to have a negligible impact on the fluid properties of the moist air mixture.
- `Track mass fraction and gas properties` — The trace gas level can be nonzero and vary during simulation. The fluid properties of the moist air mixture depend on the amount of trace gas in the mixture.

**Trace Gas — None**

If you set the **Trace gas model** parameter of a Moist Air Properties (MA) block to `None`, the moist air mixture in the connected circuit consists only of dry air and water vapor. Any nonzero values of trace gas level in parameters, variable targets, and inputs of the blocks connected to the circuit are ignored. These include, for example, the amount of trace gas in a reservoir, the initial amount of trace gas in a moist air volume, and all trace gas sources. The block equations listed on the block reference pages are simplified by replacing the trace gas mass or mole fraction terms in mixture properties calculations with 0.

You do not have to enter any fluid properties associated with trace gas, and there is no run-time trace gas properties table lookup. The underlying system equations are also simplified for increased run-time efficiency.

**Trace Gas — Track mass fraction only**

If you select the `Track mass fraction only` option, this means that you want to keep track of varying amounts of trace gas in the model, but consider its impact on the mixture properties negligible. Use this option if you expect the amount of trace gas in the mixture to be very small.

You must specify the trace gas diffusivity in air (for the smoothed upwind scheme) and the gas constant, but do not need to enter any other fluid properties associated with trace gas, and there is no run-time trace gas properties table lookup. Therefore, this option is also useful if you do not have property data for the trace gas.

Trace gas properties in block equations are replaced with the corresponding properties of dry air. There are no other equation simplifications.

**Trace Gas — Track mass fraction and gas properties**

If you select the `Track mass fraction and gas properties` option, this means that you want to keep track of varying amounts of trace gas, including its impact on the mixture properties. This is the most general and complete option. You must provide the properties of trace gas. All equations listed on the block reference pages assume this option.

## Using Moisture and Trace Gas Sources

Moisture and trace gas can be injected into blocks with finite moist air volume. (For a complete list of these blocks, see "Blocks with Moist Air Volume" on page 6-6.) For example, adding moisture to a Constant Volume Chamber (MA) block can represent respiration of occupants in a room. Moisture and trace gas can also be extracted from these blocks. For example, removing trace gas from a Pipe (MA) block can represent a filter that extracts CO2 from moist air flow.

You can use the blocks in the Moisture & Trace Gas Sources library to inject or extract moisture and trace gas. The moisture and trace gas sources can be connected only to the conserving port **S** of the blocks with moist air volume, which is the port associated with the moist air source domain. For more information, see "Moist Air Source Domain" on page 6-13.

All Moist Air library blocks with a finite moist air volume have the **Moisture and trace gas source** parameter, which controls the visibility of port **S** and provides options for modeling moisture and trace gas levels inside the component:

- `None` — No moisture or trace gas is injected into or extracted from the block. Port **S** is hidden. This is the default.

- `Constant` — Moisture and trace gas are injected into or extracted from the block at a constant rate. The same parameters as in the Moisture Source (MA) and Trace Gas Source (MA) blocks become available in the **Moisture and Trace Gas** section of the block interface. Port **S** is hidden. This option provides an easy way to model a constant rate of change for moisture and trace gas levels directly inside the component, without connecting additional blocks. It is equivalent to connecting an external constant source.

- `Controlled` — Moisture and trace gas are injected into or extracted from the block at a time-varying rate. Port **S** is exposed. Connect the Controlled Moisture Source (MA) and Controlled Trace Gas Source (MA) blocks to this port. You can also use this option to connect multiple moisture and trace gas sources to the same block with moist air volume, to represent different effects. For example, a Constant Volume Chamber (MA) block can have all of these sources connected to its port **S**:

  - A Moisture Source (MA) block, representing respiration of water vapor
  - A second Moisture Source (MA) block, representing a spray of liquid water
  - A Trace Gas Source (MA) block, representing respiration of CO2

If no moisture or trace gas is injected to or extracted from a block with moist air volume, set its **Moisture and trace gas source** parameter to `None` to hide the unused port **S**.

---

**Note** Using moisture source blocks to remove moisture is a different effect than condensation. Condensation occurs independently whenever $\varphi_w \geq \varphi_{ws}$. For more information, see "Saturation and Condensation" on page 6-7.

---

## Measuring Moisture and Trace Gas Levels

The Sensors sublibrary of the Moist Air library contains the regular sensor blocks, similar to the ones found in other domains. The Humidity & Trace Gas Sensor (MA) block lets you measure the moisture level and trace gas level in a moist air network, upstream of the measured node. These blocks can be connected only to boundary nodes (regular moist air conserving ports). Therefore, they cannot measure moist air properties at internal nodes representing a moist air volume.

To measure the amount of moisture and trace gas in a moist air volume, along with the pressure and temperature, each block with a finite internal moist air volume has a physical signal port **F**, which outputs a vector physical signal in base SI units:

```
% Moist air volume measurements
F == [value(p_I, 'Pa'); value(T_I, 'K'); RH_I; x_w_I; y_w_I; HR_I; x_g_I; y_g_I];
```

where:

- *p_I* is the pressure of the moist air volume, in Pa
- *T_I* is the temperature of the moist air volume, in K
- *RH_I* is the relative humidity of the moist air volume

- *x_w_I* is the specific humidity of moist air volume
- *y_w_I* is the water vapor mole fraction of the moist air volume
- *HR_I* is the humidity ratio of the moist air volume
- *x_g_I* is the trace gas mass fraction of the moist air volume
- *y_g_I* is the trace gas mole fraction of the moist air volume

Use the Measurement Selector (MA) block to unpack the vector signal and reassign units to pressure and temperature values. Connect the Measurement Selector (MA) block to the physical signal output port **F** of a block with finite internal moist air volume to access the data.

## See Also

## More About
- "Modeling Moist Air Systems" on page 6-2

# Model Simulation

# How Simscape Models Represent Physical Systems

| In this section... |
| --- |

## Representations of Physical Systems

This section describes important characteristics of the mathematical representations of physical systems, and how Simscape software implements such representations. You might find this overview helpful if you:

- Require details of such representations to improve your model fidelity or simulation performance.
- Are constructing your own, custom Simscape components using the Simscape language.
- Need to troubleshoot Simscape modeling or simulation failures.

Mathematical representations are the foundation for physical simulation. For more information about simulation, see "How Simscape Simulation Works" on page 7-6.

## Differential, Differential-Algebraic, and Algebraic Systems

The mathematical representation of a physical system contains *ordinary differential equations* (ODEs), *algebraic equations*, or both.

- ODEs govern the rates of change of *system variables* and contain some or all of the time derivatives of the system variables.
- Algebraic equations specify functional constraints among system variables, but contain no time derivatives of system variables.

- Without algebraic constraints, the system is differential (ODEs).
- Without ODEs, the system is algebraic.
- With ODEs and algebraic constraints, the system is mixed *differential-algebraic* (DAEs).

A system variable is differential or algebraic, depending on whether or not its time derivative appears in the system equations.

## Stiffness

A mathematical problem is *stiff* if the solution you are seeking varies slowly, but there are other solutions within the error tolerances that vary rapidly. A stiff system has several intrinsic time scales of very different magnitude [1].

A stiff physical system has one or more components that behave "stiffly" in the ordinary sense, such as a spring with a large spring constant. Mathematical equivalents include quasi-incompressible fluids and low electrical inductance. Such systems often exhibit high frequency oscillations in some of their components or modes.

## Events and Zero Crossings

Events are discontinuous changes in system state or dynamics as the system evolves in time; for example, a valve opening, or a hard stop. For more information on how events are represented in the Simscape language, see "Discrete Event Modeling".

A *zero crossing* is a specific event type, represented by the value of a mathematical function changing sign. Variable-step solvers take smaller steps when they detect a zero-crossing event. Smaller steps help to capture the dynamics that cause the zero crossing, but they also significantly slow down the simulation. Various methods of zero crossing detection and analysis help you strike the right balance between the simulation speed and accuracy. For more information, see "Managing Zero Crossings in Simscape Models" on page 7-3.

## Working with Simscape Representation

A Simscape model is equivalent to a set of equations representing one or more physical systems as physical networks.

* Start by assuming that your physical network is a DAE system: a mix of differential and algebraic equations and variables on page 7-2.

   Remember that some physical networks are represented by ODEs only.
* Physical networks may contain stiff differential equations on page 7-2.
* Identify discrete and continuous components that might change discontinuously on page 7-3 during a simulation.

## Managing Zero Crossings in Simscape Models

Your model can contain zero-crossing conditions arising from several sources:

* Simscape and Simulink blocks copied from their respective block libraries
* Custom blocks programmed in the Simscape language

Simulink software has global methods for managing zero-crossing events. For more information, see "Zero-Crossing Detection".

You can disable zero-crossing detection on individual blocks, or globally across the entire model. Zero-crossing detection often improves simulation accuracy, but can slow simulation speed.

---

**Tip** If the exact times of zero crossings are important in your model, then keep zero-crossing detection enabled. Disabling it can lead to major simulation inaccuracies.

---

**Detecting and Minimizing Zero Crossings in Simscape Models**

In addition to generic Simulink methods, Simscape software has specific tools that let you detect and manage zero-crossings in your models:

- Prior to simulation, you can use the Statistics Viewer to identify the potential zero-crossing signals in the model. These signals are typically generated from operators and functions that contain discontinuities, such as comparison operators, `abs`, `sqrt` functions, and so on. During simulation it is possible for none of these signals to produce a zero-crossing event or for one or more of these signals to have multiple zero-crossing events. For more information, see "View Model Statistics" on page 14-10.

- When logging simulation data for a model, you can select the **Log simulation statistics** option. The data log then includes the actual zero-crossing data during simulation. For more information, see "Log Simulation Statistics" on page 13-13.

  You can access and analyze zero-crossing data logged during simulation by using the Simscape Results Explorer. For more information, see "About the Simscape Results Explorer" on page 13-24.

- The `sscprintzcs` function prints information about zero crossings detected during simulation, based on logged simulation data. Before you call this function, you must have the simulation log variable, which includes simulation statistics data, in your current workspace. For more information and examples, see `sscprintzcs`.

Managing zero crossing is especially important when you prepare your models for real-time simulation. See "Reduce Zero Crossings" on page 11-55 for a detailed example of this workflow.

**Enabling and Disabling Zero-Crossing Conditions in Simscape Language**

When writing code for your own custom blocks using the Simscape language, you can create or avoid zero-crossing conditions in your model by switching between different implementations of discontinuous conditional expressions. You can:

- Use relational operators, which create zero-crossing conditions. For example, programming the operator relation: `a < b` creates a zero-crossing condition.

- Use relational functions, which do not create zero-crossing conditions. For example, programming the functional relation: `lt(a,b)` does not create a zero-crossing condition. For more information on whether a particular function creates discontinuities when used in Simscape language, see `equations`.

---

**Note** Using relational functions, like `lt(a,b)`, in event predicates always creates a zero-crossing condition. For more information about event predicates, see "Discrete Event Modeling".

---

# References

[1] Moler, C. B., *Numerical Computing with MATLAB,* Philadelphia, Society for Industrial and Applied Mathematics, 2004, chapter 7

[2] Horowitz, P., and Hill, W., *The Art of Electronics*, 2nd Ed., Cambridge, Cambridge University Press, 1989, chapter 2

[3] Brogan, W. L., *Modern Control Theory*, 2nd Ed., Englewood Cliffs, New Jersey, Prentice-Hall, 1985

# How Simscape Simulation Works

| **In this section...** |
| --- |
| |
| |
| |
| |
| |
| |
| |

## Simscape Simulation Phases

You might find this brief overview helpful for constructing models and understanding errors. For more information, see "How Simscape Models Represent Physical Systems" on page 7-2.

Simscape software gives you multiple ways to simulate and analyze physical systems in the Simulink environment. Running a physical model simulation is similar to simulating any Simulink model. It entails setting various simulation options, starting the simulation, and viewing the simulation results. This topic describes various aspects of simulation specific to Simscape models. For specifics of simulating and analyzing with individual Simscape add-on products, refer to the documentation for those individual add-on products.

This flow chart presents the Simscape simulation sequence.
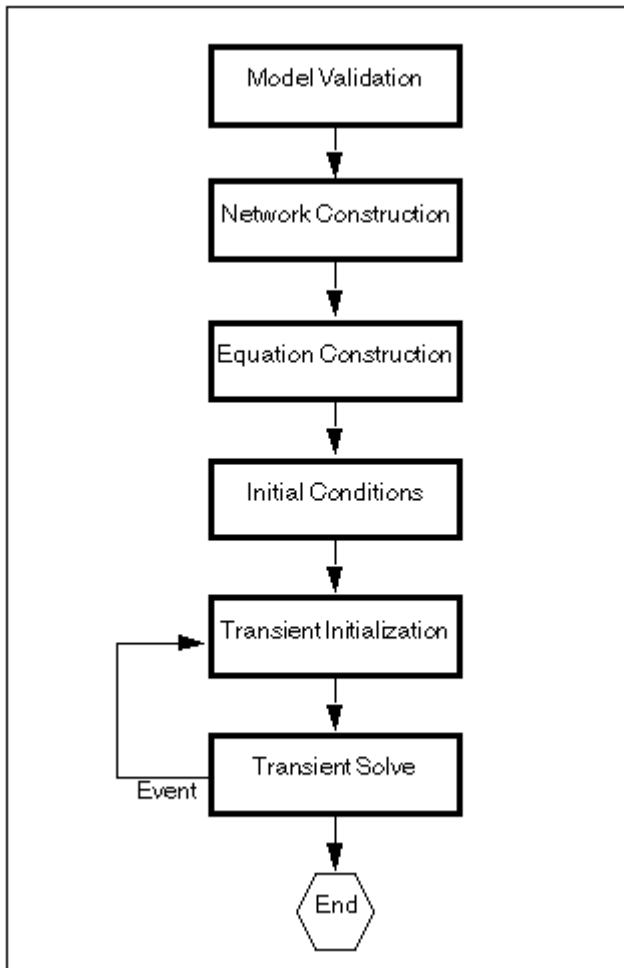
The flow chart consists of the following major phases:

**1**  "Model Validation" on page 7-7
**2**  "Network Construction" on page 7-8
**3**  "Equation Construction" on page 7-8
**4**  "Initial Conditions Computation" on page 7-8
**5**  "Transient Initialization" on page 7-9
**6**  "Transient Solve" on page 7-10

## Model Validation

The Simscape solver first validates the model configuration and checks your data entries from the block dialog boxes.

- All Simscape blocks in a diagram must be connected into one or more physical networks.
- Each topologically distinct physical network in a diagram requires exactly one Solver Configuration block.
- If your model contains fluid elements (such as two-phase fluids, gas, moist air, isothermal or thermal liquid), each topologically distinct circuit in a diagram can contain a block that defines the

fluid properties for all the blocks that connect to the circuit. If no fluid block is attached to a loop, the blocks in this loop use the default fluid. However, more than one fluid block in a loop generates an error.

- Signal units specified in a Simulink-PS Converter block must match the input type expected by the Simscape block connected to it. For example, when you provide the input signal for an Ideal Angular Velocity Source block, specify angular velocity units, such as `rad/s` or `rpm`, in the Simulink-PS Converter block, or leave it unitless. Similarly, units specified in a PS-Simulink Converter block must match the type of physical signal provided by the Simscape block outport.

## Network Construction

After validating the model, the Simscape solver constructs the physical network based on the following principles:

- Two directly connected Conserving ports have the same values for all their Across variables (such as voltage or angular velocity).

- Any Through variable (such as current or torque) transferred along the Physical connection line is divided among the multiple components connected by the branches. For each Through variable, the sum of all its values flowing into a branch point equals the sum of all its values flowing out.

## Equation Construction

Based on the network configuration, the parameter values in the block dialog boxes, and the global parameters defined by the fluid properties, if applicable, the Simscape solver constructs the system of equations for the model.

These equations contain system variables of the following types:

- Dynamic — Time derivatives of these variables appear in equations. Dynamic, or differential, variables add dynamics to the system and require the solver to use numerical integration to compute their values. Dynamic variables can produce either independent or dependent states for simulation.

- Algebraic — Time derivatives of these variables do not appear in equations. These variables appear in algebraic equations but add no dynamics, and this typically occurs in physical systems due to conservation laws, such as conservation of mass and energy. The states of algebraic variables are always dependent on dynamic variables, other algebraic variables, or inputs.

The solver then performs the analysis and eliminates variables that are not needed to solve the system of equations. After variable elimination, the remaining variables (algebraic, dynamic dependent, and dynamic independent) get mapped to Simulink state vector of the model.

For information on how to view and analyze model variables, see "Model Statistics".

## Initial Conditions Computation

The Simscape solver computes the initial conditions only once, at the beginning of simulation ($t = 0$). In the Solver Configuration block, the default is that the **Start simulation from steady state** check box is not selected. If it is selected in your model, see "Finding an Initial Steady State" on page 7-9.

The solver computes the initial conditions by finding initial values for all the system variables that exactly satisfy all the model equations. You can affect the initial conditions computation by block-level

variable initialization, that is, by specifying the priority and target initial values in the **Initial Targets** section of the block dialog box. You can also initialize variables for a whole model from a saved operating point.

The values you specify during variable initialization are not the actual values of the respective variables, but rather their target values at the beginning of simulation ($t = 0$). Depending on the results of the solve, some of these targets may or may not be satisfied. The solver tries to satisfy the high-priority targets first, then the low-priority ones:

- At first, the solver tries to find a solution where all the high-priority variable targets are met exactly, and the low-priority targets are approximated as closely as possible. If the solution is found during this stage, it satisfies all the high-priority targets. Some of the low-priority targets might also be met exactly, the others are approximated.

- If the solver cannot find a solution that exactly satisfies all the high-priority targets, it issues a warning and enters the second stage, where `High` priority is relaxed to `Low`. That is, the solver tries to find a solution by approximating both the high-priority and the low-priority targets as closely as possible.

After you initialize the variables and prior to simulating the model, you can open the Variable Viewer to see which of the variable targets have been satisfied. For more information on block-level variable initialization, see "Variable Initialization".

### Finding an Initial Steady State

When you select the **Start simulation from steady state** check box in the Solver Configuration block:

- For models compatible with frequency-and-time equation formulation, the solver attempts to perform sinusoidal steady-state initialization. In other words, initialization is performed using frequency-time equations, and then the simulation proceeds using the actual equation formulation and other options selected in the Solver Configuration block. For more information, see "Frequency and Time Simulation Mode" on page 7-55.

- If the model is not frequency-and-time compatible, the solver attempts to find the steady state that would result if the inputs to the system were held constant for a long enough time, starting from the initial state obtained from the initial conditions computation described in the previous section. Steady state means that the system variables are no longer changing with time.

If the steady-state solve succeeds, the state found is some steady state (within tolerance), but not necessarily the state expected from the given initial conditions. Simulation then starts from this steady state.

A model can have more than one steady state. In this case, the solver selects the steady-state solution that is consistent with the variable targets specified during block-level variable initialization. For more information, see "Variable Initialization".

## Transient Initialization

After computing the initial conditions, or after a subsequent event (such as a discontinuity resulting, for example, from a valve opening, or from a hard stop), the Simscape solver performs transient initialization. Transient initialization fixes all dynamic variables and solves for algebraic variables and derivatives of dynamic variables. The goal of transient initialization is to provide a consistent set of initial conditions for the next phase, transient solve.

## Transient Solve

Finally, the Simscape solver performs transient solve of the system of equations. In transient solve, continuous differential equations are integrated in time to compute all the variables as a function of time.

The solver continues to perform the simulation according to the results of the transient solve until the solver encounters an event, such as a zero crossing or discontinuity. The event may be within the physical network or elsewhere in the Simulink model. If the solver encounters an event, the solver returns to the phase of transient initialization, and then back to transient solve. This cycle continues until the end of simulation.

## See Also
Solver Configuration

# Setting Up Solvers for Physical Models

| In this section... |
| --- |
| "About Simulink and Simscape Solvers" on page 7-11 |
| "Choosing Simulink and Simscape Solvers" on page 7-11 |
| "Harmonizing Simulink and Simscape Solvers" on page 7-12 |

## About Simulink and Simscape Solvers

This section explains how to select solvers for physical simulation. Proper simulation of Simscape models requires certain changes to Simulink defaults and consideration of physical simulation trade-offs. For recommended choices, see "Making Optimal Solver Choices for Physical Simulation" on page 7-18.

## Choosing Simulink and Simscape Solvers

Simulink and Simscape solver technologies provide a range of tools to simulate physical systems, including the powerful Simscape technique of local solvers. You choose global, or model-wide, solvers through Simulink. After making these choices, check that they are consistent; see "Harmonizing Simulink and Simscape Solvers" on page 7-12.

- "Working with Global Simulink Solvers" on page 7-11
- "Working with Local Simscape Solvers" on page 7-11

### Working with Global Simulink Solvers

In the Configuration Parameters dialog box of your model, on the **Solver** pane, the solver and related settings that you select are global choices. For more information, see "Solver Selection Criteria".

When you first create a model, the default Simulink solver is `VariableStepAuto`. For more information, see "Select Solver Using Auto Solver". To select a different solver, follow a procedure similar to the procedure in "Modifying Initial Settings" on page 1-21.

- You can choose one from a suite of both variable-step and fixed-step solvers.
- You can also select from among explicit and implicit solvers. For physical models, it is recommended that you use implicit solvers, such as daessc, ode23t, and ode15s. Implicit solvers require fewer time steps than explicit solvers, such as ode45, ode113, and ode1.

  See "Switching from the Default Explicit Solver to Other Simulink Solvers" on page 7-13.
- If all the Simulink and Simscape states in your model are discrete, Simulink automatically switches to a discrete solver and issues a warning. Otherwise, a continuous solver is the default.
- By default, Simulink variable-step solvers attempt to locate events in time by zero-crossing detection. See "Managing Zero Crossings in Simscape Models" on page 7-3.

### Working with Local Simscape Solvers

You can switch one or more physical networks to a local implicit, fixed-step Simscape solver by selecting **Use local solver** in the network Solver Configuration block. The solver and related settings you make in each Solver Configuration block are specific to the connected physical network and can differ from network to network.

A physical network using a local solver appears to the global Simulink solver as if it has discrete states. You can still use any continuous global solver.

**Choosing Local Solvers and Sample Times**

To use a local solver, choose a solver type (Backward Euler, Trapezoidal Rule, or Partitioning) and a sample time. Backward Euler is the default.

**Choosing Fixed-Cost Simulation**

You can select a fixed-cost simulation for one or more physical networks by selecting **Use fixed-cost runtime consistency iterations**, as well as **Use local solver**, and fixing the number of nonlinear and mode iterations. For more information, see "Fixed-Cost Simulation".

**Choosing Multirate Simulation**

With the local solver option, you can perform multirate simulations, with:

- Different sample times in different physical networks, through their respective Solver Configuration blocks
- A sample-based Simulink block in the model with a sample time different from the Solver Configuration block or blocks

## Harmonizing Simulink and Simscape Solvers

Your Simulink and Simscape solver choices must work together consistently. To ensure consistency of your Simulink and Simscape solver choices for a particular model, open the model Configuration Parameters dialog box. In the model window, open the **Modeling** tab and click **Model Settings**. Review and adjust the following settings.

- "Switching from the Default Explicit Solver to Other Simulink Solvers" on page 7-13
- "Enabling or Disabling Simulink Zero-Crossing Detection" on page 7-14
- "Making Multirate Simulation Consistent" on page 7-14
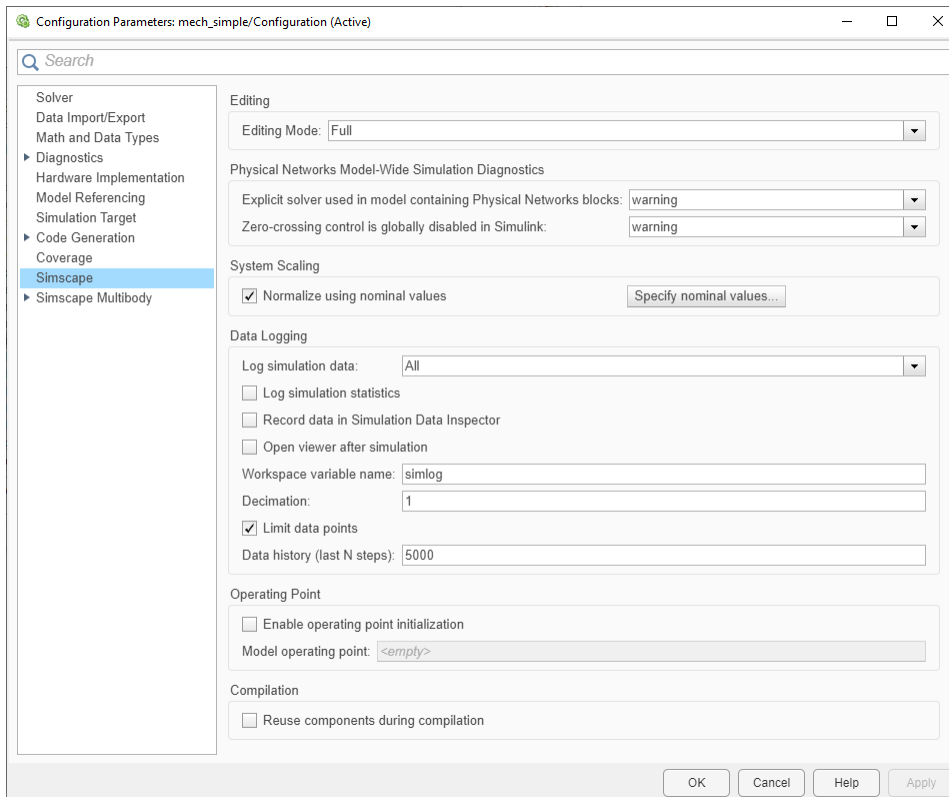
**Simscape Pane of the Configuration Parameters Dialog Box**

**Switching from the Default Explicit Solver to Other Simulink Solvers**

When you first create a model, the default Simulink solver is `VariableStepAuto`. Auto solver chooses a suitable solver as described in "Select Solver Using Auto Solver", and for some types of models it can choose an explicit solver, `ode45`. If you do not modify the default (explicit) solver, your performance may not be optimal. Implicit solvers are better for most physical simulations. For more information about implicit solvers and physical systems, see "Important Concepts and Choices in Physical Simulation" on page 7-15.

**Diagnostic Messages About Explicit Solvers**

When you use an explicit solver in a model containing Simscape blocks, the system issues a warning to alert you to a potential problem.

To turn off this default warning or to change it to an error message, go to the **Simscape** pane of the Configuration Parameters dialog box:

**1**   From the **Explicit solver used in model containing Physical Networks blocks** drop-down list, select the option that you want:

- `warning` — If the model uses an explicit solver, the system issues a warning upon simulation. This is the default option that alerts you to a potential problem if you use the default solver.

- `error` — If the model uses an explicit solver, the system issues an error message upon simulation. If your model is stiff, and you do not want to use explicit solvers, select this option to avoid future errors.

- none — If the model uses an explicit solver, the system issues no warning or error message upon simulation. If you want to work with explicit solvers, in particular for models that are not stiff, select this option.

**2** Click **OK**.

### Enabling or Disabling Simulink Zero-Crossing Detection

By default, Simulink tracks an important class of simulation events by detecting zero crossings. With a global variable-step solver and without a local solver, Simulink attempts to locate the simulated times of zero crossings, if present. See "Managing Zero Crossings in Simscape Models" on page 7-3.

#### Diagnostic Messages About Globally Disabling Zero-Crossing Detection

You can globally disable zero-crossing detection in the **Solver** pane of the Configuration Parameters dialog box, under **Zero-crossing options**. If you do, and if you are using a global variable-step solver without a local solver, the system issues a warning or error when you simulate with Simscape blocks.

You can choose between warning and error messages in the **Simscape** pane of the Configuration Parameters dialog box.

**1** From the **Zero-crossing control is globally disabled in Simulink** drop-down list, select the option that you want, if you globally disable zero-crossing detection:

- warning — The system issues a warning message upon simulation. This option is the default.
- error — The system issues an error message upon simulation, which stops.

**2** Click **OK**.

### Making Multirate Simulation Consistent

The sample time or step size of the global Simulink solver must be the smallest time step of all the solvers in a multirate Simscape simulation.

To avoid simulation errors in sample time propagation, go to the **Solver** pane in the Configuration Parameters dialog box and select the **Automatically handle rate transition for data transfer** check box.

# Important Concepts and Choices in Physical Simulation

| In this section... |
| --- |
| "Variable-Step and Fixed-Step Solvers" on page 7-15 |
| "Explicit and Implicit Solvers" on page 7-15 |
| "Full and Sparse Linear Algebra" on page 7-16 |
| "Event Detection and Location" on page 7-16 |
| "Unbounded, Bounded, and Fixed-Cost Simulation" on page 7-16 |
| "Global and Local Solvers" on page 7-17 |

This section describes advanced concepts and trade-offs you might want to consider as you configure and test solvers and other simulation settings for your Simscape model. For a summary of recommended settings, see "Making Optimal Solver Choices for Physical Simulation" on page 7-18. For background information, consult "How Simscape Models Represent Physical Systems" on page 7-2 and "How Simscape Simulation Works" on page 7-6.

## Variable-Step and Fixed-Step Solvers

*Variable-step solvers* are the usual choice for design, prototyping, and exploratory simulation, and to precisely locate events during simulation. They are not useful for real-time simulation and can be costly if there are many events.

A variable-step solver automatically adjusts its step size as it moves forward in time to adapt to how well it controls solution error. You control the accuracy and speed of the variable-step solution by adjusting the solver tolerance. With many variable-step solvers, you can also limit the minimum and maximum time step size.

*Fixed-step solvers* are recommended or required if you want to make performance comparisons across platforms and operating systems, to generate a code version of your model, and to bound or fix simulation cost. A typical application is real-time simulation. For more information, see "Real-Time Simulation".

With a fixed-step solver, you specify the time step size to control the accuracy and speed of your simulation. Fixed-step solvers do not adapt to improve accuracy or to locate events. These limitations can lead to significant simulation inaccuracies.

## Explicit and Implicit Solvers

The degree of stiffness and the presence of algebraic constraints in your model influence the choice between an *explicit* or *implicit* solver. Explicit and implicit solvers use different numerical methods to simulate a system.

- If the system is a nonstiff ODE system, choose an explicit solver. Explicit solvers require less computational effort than implicit solvers, if other simulation characteristics are fixed.

  To find a solution for each time step, an explicit solver uses a formula based on the local gradient of the ODE system.

- If the system is stiff, use an implicit solver. Though an explicit solver may require less computational effort, for stiff problems an implicit solver is more accurate and often essential to

**7-15**

obtain a solution. Implicit solvers require per-step iterations within the simulated time steps. With some implicit solvers, you can limit or fix these iterations.

An implicit solver starts with the solution at the current step and iteratively solves for the solution at the next time step with an algebraic solver. An implicit algorithm does more work per simulation step, but can take fewer, larger steps.

- If the system contains DAEs, even if it is not stiff, use an implicit solver. Such solvers are designed to simultaneously solve algebraic constraints and integrate differential equations.

## Full and Sparse Linear Algebra

When you simulate a system with more than one state, the solver manipulates the mathematical system with matrices. For a large number of states, sparse linear algebra methods applied to large matrices can make the simulation more efficient.

## Event Detection and Location

Events, in most cases, occur between simulated time steps.

- Fixed-step solvers detect events after "stepping over" them, but cannot adaptively locate events in time. This can lead to large inaccuracies or failure to converge on a solution.
- Variable-step solvers can both detect events and estimate the instants when they occur by adapting the timing and length of the time steps.

**Tip** To estimate the timing of events or rapid changes in your simulation, use a variable-step solver.

If your simulation has to frequently adapt to events or rapid changes by changing its step size, much or all of the advantage of implicit solvers over explicit solvers is lost.

## Unbounded, Bounded, and Fixed-Cost Simulation

In certain cases, such as real-time simulation, you need to simulate with an execution time that is not only bounded, but practically fixed to a predictable value. Fixing execution time can also improve performance when simulating frequent events.

The real-time cost of a variable-step simulation is potentially unlimited. The solver can take an indefinite amount of real time to solve a system over a finite simulated time, because the number and size of the time steps are adapted to the system. You can configure a fixed-step solver to take a bounded amount of real time to complete a simulation, although the exact amount of real time might still be difficult to predict before simulation. Even a fixed-step solver can take multiple iterations to find a solution at each time step. Such iterations are variable and not generally limited in number; the solver iterates as much as it needs to.

Fixing execution time implies *fixed-cost* simulation, which both fixes the time step and limits the number of per-step iterations. Fixed-cost simulation prevents *execution overruns*, when the execution time is longer than the simulation sample time. A bounded execution time without a known fixed cost might still cause some steps to overrun the sample time.

The actual amount of computational effort required by a solver is based on a number of other factors as well, including model complexity and computer processor. For more information, see "Real-Time Simulation".

## Global and Local Solvers

You can use different solvers on different parts of the system. For example, you might want to use implicit solvers on stiff parts of a system and explicit solvers everywhere else. Such *local solvers* make the simulation more efficient and reduce computational cost.

Such multisolver simulations must coordinate the separate sequences of time steps of each solver and each subsystem so that the various solvers can pass simulation updates to one another on some or all of the shared time steps.

# Making Optimal Solver Choices for Physical Simulation

| **In this section...** |
| --- |
| "Simulating with Variable Time Step" on page 7-18 |
| "Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers" on page 7-18 |
| "Simulating with Fixed Cost" on page 7-19 |
| "Troubleshooting and Improving Solver Performance" on page 7-20 |
| "Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System" on page 7-21 |

For the key simulation concepts to consider before making these choices, see "Important Concepts and Choices in Physical Simulation" on page 7-15.

## Simulating with Variable Time Step

When you first create a model, the default Simulink solver is `VariableStepAuto`. Auto solver chooses a suitable solver as described in "Select Solver Using Auto Solver". For Simscape models, the auto solver selection depends on the type of the model:

- For new models created in R2021a and beyond, if your model contains Simscape blocks and Differential Algebraic Equations (DAEs), auto solver selects `daessc`. For such models created prior to R2021a, auto solver uses `ode23t`. For information on how to upgrade your existing models to use `daessc`, see "Upgrading Your Models to Use the daessc Solver" on page 7-24.
- If the system can be reduced to an ordinary differential equation (ODE) and the model is stiff, auto solver selects `ode15s`.
- If the system can be reduced to an ordinary differential equation (ODE) and the model is nonstiff, auto solver selects an explicit solver, `ode45`.

Rather than relying on the auto solver selection, you can explicitly choose a solver for your model. To select a solver, follow a procedure similar to the procedure in "Modifying Initial Settings" on page 1-21.

The `daessc` variable-step Simulink solver is designed specifically for physical modeling. For more information, see "Using the daessc Solver" on page 7-22.

Other variable-step solvers recommended for a typical Simscape model are `ode15s` and `ode23t`. Of these two solvers:

- The `ode15s` solver is more stable, but tends to damp out oscillations.
- The `ode23t` solver captures oscillations better but is less stable.

With Simscape models, these solvers solve the differential and algebraic parts of the physical model simultaneously, making the simulation more efficient.

## Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers

In a Simscape model, it is recommended that you implement fixed-step solvers by continuing to use a global variable-step solver and switching the physical networks within your model to local fixed-step solvers through each network Solver Configuration block. The local solver choices are:

- The Backward Euler tends to damp out oscillations, but is more stable, especially if you increase the time step.
- The Trapezoidal Rule solver captures oscillations better but is less stable.
- The Partitioning solver lets you increase real-time simulation speed by partitioning the entire system of equations corresponding to a Simscape network into a cascade of smaller equation systems. Not all networks can be partitioned. However, when a system can be partitioned, this solver provides a significant increase in real-time simulation speed. For more information, see "Understanding How the Partitioning Solver Works" on page 7-25 and "Increase Simulation Speed Using the Partitioning Solver" on page 11-19.

Regardless of which local solver you choose, the Backward Euler method is always applied:

- Right at the start of simulation.
- Right after an instantaneous change, when the corresponding block undergoes an internal discrete change. Such changes include clutches locking and unlocking, valve actuators opening and closing, and the switching of the PS Asynchronous Sample & Hold block.

### Switching to Discrete States and Solvers

- If you switch a physical network to a local solver, the global solver treats that network as having discrete states.
- If other physical networks in your model are not using local solvers, or if the non-Simscape parts of your model have continuous states, then you must use a continuous global solver.
- If all physical networks in your model use local solvers, and all other parts of your model have only discrete states, then the global solver effectively sees only discrete states. In that case, a discrete, fixed-step global solver is recommended. If you are attempting a fixed-cost simulation with discrete states, you must use a discrete, fixed-step global solver.

**Note** Input filtering may introduce continuous states. If you are using a combination of discrete and local solvers and get an error message about the model containing continuous states, check the Simulink-PS Converter blocks in the model and turn off input filtering, if needed. For more information, see "Filtering Input Signals and Providing Time Derivatives" on page 7-29.

### For Maximum Accuracy with Fixed-Step Simulation

If solution accuracy is your single overriding requirement, use the global Simulink fixed-step solver ode14x, without local solvers. This implicit solver is the best global fixed-step choice for physical systems. While it is more accurate than the Simscape local solvers for most models, ode14x can be computationally more intensive and slower when you use it by itself than it is when you use it in combination with local solvers.

In this solver, you must limit the number of global implicit iterations per time step. Control these iterations with the **Number Newton's iterations** parameter in the **Solver** pane of the Configuration Parameters dialog box.

## Simulating with Fixed Cost

Many Simscape models need to iterate multiple times within one time step to find a solution. If you want to fix the cost of simulation per time step, you must limit the number of these iterations, regardless of whether you are using a local solver, or a global solver like ode14x. For more

information, see "Unbounded, Bounded, and Fixed-Cost Simulation" on page 7-16 and "Fixed-Cost Simulation for Real-Time Viability" on page 11-71.

To limit the iterations, open the Solver Configuration block of each physical network. Select **Use fixed-cost runtime consistency iterations** and set limits for the number of nonlinear and mode iterations per time step. You can determine the optimum number of nonlinear iterations using the `simscape.getLocalSolverFixedCostInfo` function.

---

**Tip** Fixed-cost simulation with variable-step solvers is not possible in most simulations. Attempt fixed-cost simulation with a fixed-step solver only and avoid using fixed-cost iterations with variable-step solvers.

---

## Troubleshooting and Improving Solver Performance

Consider the basic trade-off of speed versus accuracy and stability. A larger time step or tolerance results in faster simulation, but also less accurate and less stable simulation. If a system undergoes sudden or rapid changes, larger tolerance or step size can cause major errors. Consider tightening the tolerance or step size if your simulation:

- Is not accurate enough or looks unphysical.
- Exhibits discontinuities in state values.
- Reaches the minimum step size allowed without converging, usually a sign that one or more events or rapid changes occur within a time step.

Any one or all of these steps increase accuracy, but make the simulation run more slowly.

### For Local Solvers

Models with friction or hard stops are particularly difficult for local solvers, and may not work or may require a very small time step.

With the Trapezoidal Rule solver, oscillatory "ringing" can become more of a problem as the time step is increased. For a larger time step in a local solver, consider switching to Backward Euler.

### For ODE Systems

In certain cases, your model reduces to an ODE system, with no dependent algebraic variables. (See "How Simscape Models Represent Physical Systems" on page 7-2.) If so, you can use any global Simulink solver, with no special physical modeling considerations. An explicit solver is often the best choice in such situations.

- Through careful analysis, you can sometimes determine if your model is represented by an ODE system.
- If you create a Simscape model from a mathematical representation using the Simscape language, you can determine directly if the resulting system is ODE.

### For Large Systems

Depending on the number of system states, you can simulate more efficiently if you switch the value of the **Linear Algebra** setting in the Solver Configuration block.

For smaller systems, `Full` provides faster results. For larger systems, `Sparse` is typically faster.

## Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System

In this example, a Simscape model contains three physical networks.

- Two networks (numbers 1 and 3) use local solvers, making these two networks appear to the global solver as if they had discrete states. Internally, these networks still have continuous states. These networks are moderately and highly stiff, respectively.

  One of these networks (number 1) uses the Backward Euler (BE) local solver. The other (number 3) uses the Trapezoidal Rule (TR) local solver.

- The remaining network (number 2) uses the global Simulink solver. Its states appear to the model as continuous. This network is not stiff and is pure ODE. Use an explicit global solver.

- Because at least one network appears to the model as continuous, you must use a continuous solver. However, if you remove network 2, and if the model contains no continuous Simulink states, Simulink automatically switches to a discrete global solver.
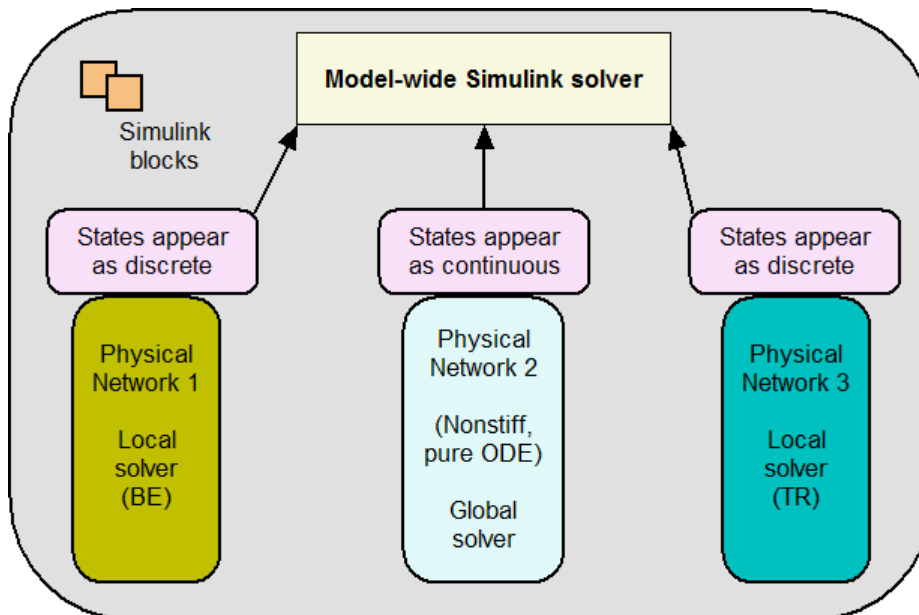
# Best Practices for Simulating with the daessc Solver

| **In this section...** |
| --- |
| "Using the daessc Solver" on page 7-22 |
| "Troubleshooting Your Models" on page 7-23 |
| "Upgrading Your Models to Use the daessc Solver" on page 7-24 |

The `daessc` variable-step Simulink solver provides algorithms specifically designed to simulate differential algebraic equations (DAEs) arising from modeling physical systems.

The `daessc` solver is available with a Simscape license only.

## Using the daessc Solver

For new models, if your model contains Simscape blocks and DAEs, `VariableStepAuto` solver defaults to `daessc`. You can also select the `daessc` solver explicitly:

1  In the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens and shows the **Solver** pane.

2  Under **Solver selection**, set **Type** to `Variable-step`, and then, from the **Solver** drop-down list, select `daessc (DAE solver for Simscape)`.

3  Expand **Solver details** and set **Daessc mode** to one of these options, to fine-tune the solver performance:

- `auto` — Automatically selects the optimal `daessc` solver mode. This is the default setting.
- `Fast` — The most efficient mode in terms of computation cost, but less robust.
- `Balanced` — Provides a balance between computational costs and robustness.
- `Robust` — More robust, but also more costly.
- `Quick debug` — Updates the solver Jacobian at every integration step, and is therefore even more costly than `Robust`. Recommended only for interactive model development, to quickly find issues with equations.
- `Full debug` — Updates the solver Jacobian at every integration step and every Newton iteration. This mode is the most expensive in terms of computational cost. Recommended only for interactive model development, to thoroughly check equations and find possible issues.

The `daessc` solver uses the following tolerance settings for Simscape states in the model:

- Sets `AutoScaleAbsTol` to `off`
- If **Absolute tolerance** is specified as `auto`, sets `AbsTol` to the same value as `RelTol`

These changes are not reflected in the Configuration Parameters dialog box and do not affect other (nonphysical) states in the model. In other words, if your model contains a Simulink controller and a Simscape plant, when you simulate it with the `daessc` solver, the controller uses the tolerance settings specified in the Configuration Parameters dialog box, but the plant uses `AutoScaleAbsTol = off` and `AbsTol = RelTol`.

The `daessc` solver assumes that the model is well-scaled. For more information, see "System Scaling by Nominal Values" on page 7-31 and "Use Scaling by Nominal Values to Improve Performance" on page 7-37.

The `daessc` solver does not support `Rsim`. You can use this solver with other real-time simulation targets.

For more information on recommended settings and best practices for using `daessc`, see "Troubleshooting Your Models" on page 7-23.

## Troubleshooting Your Models

To take advantage of the `daessc` solver, ensure that the model is well-scaled and that the tolerances specified for the solvers make engineering sense. These settings are recommended:

- **Relative tolerance** — `1e-3`
- **Absolute tolerance** — `1e-3`
- **Auto scale absolute tolerance** off

`ssc_new` and the Simscape templates use these settings. If you use another method to create your model, you can set all three with this command:

```
set_param(bdroot,'AbsTol','1e-3','RelTol','1e-3','AutoScaleAbsTol','off')
```

To improve model scaling:

**1**   In the Configuration Parameters dialog box, on the **Simscape** pane, ensure that the **Normalize using nominal values** check box is selected.

**2**   Simulate your model using Solver Profiler and check the **States Below Absolute Tolerance** tab.
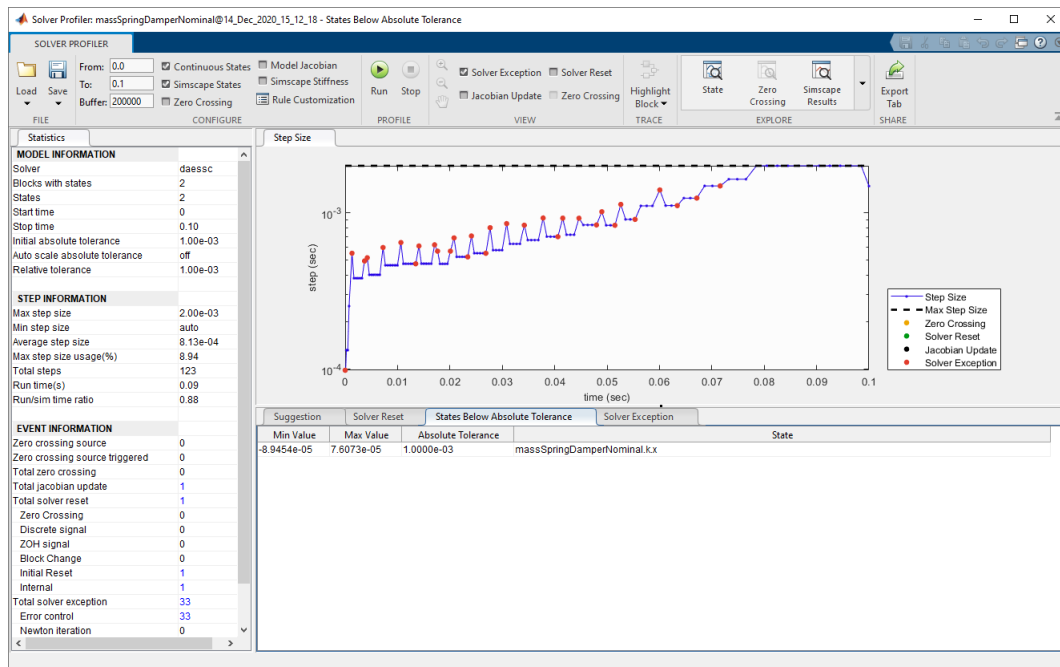


**3**   For each variable listed on this tab, determine if this variable is important for simulation. If it is, open the Property Inspector for the block that contains this variable and specify the nominal value for the variable.

**4**   You can also add or edit the value-unit pairs for the model, if needed, by using the **Specify nominal values** button on the **Simscape** pane of the Configuration Parameters dialog box.

For more information, see "Use Scaling by Nominal Values to Improve Performance" on page 7-37.

## Upgrading Your Models to Use the daessc Solver

For Simscape models with DAEs created prior to R2021a, the default `VariableStepAuto` solver is `ode23t`, and this solver selection does not change automatically when you open such an existing model in the current software version. Use the **Check integration method used by 'auto' solver for Simscape DAEs** check in the Upgrade Advisor to identify models that still use `ode23t` as the variable-step auto solver and update them to use `daessc`, which is designed specifically for physical modeling.

Before updating your models, ensure they are well-scaled and follow the other best practices described in "Troubleshooting Your Models" on page 7-23.

The `daessc` solver tends to be more robust for most Simscape models, but some models may experience adverse effects due to this change. After updating the model, simulate it and validate the results and performance. If you decide that you want to restore the previous simulation behavior, change the **Solver** model configuration parameter from `auto` (`Automatic solver selection`) to `ode23t` (`mod.stiff/Trapezoidal`).

### See Also
Solver Profiler

### More About
- "Simulating with Variable Time Step" on page 7-18
- "Important Concepts and Choices in Physical Simulation" on page 7-15
- "System Scaling by Nominal Values" on page 7-31
- "Use Scaling by Nominal Values to Improve Performance" on page 7-37

# Understanding How the Partitioning Solver Works

This topic uses the "Nonlinear Electromechanical Circuit with Partitioning Solver" on page 20-338 example to provide an in-depth look into the Partitioning solver functionality. It examines the different types of partitions and their equations and explains how the Partitioning solver solves them to yield faster simulation.

**1**   To open the Nonlinear Electromechanical Circuit with Partitioning Solver example model, type `ssc_nonlinear_electromechanical_circuit` in the MATLAB Command Window.
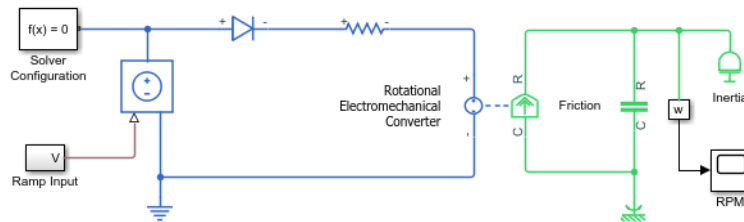


**2**   To view model statistics, in the model window, on the **Debug** tab, click **Simscape > Statistics Viewer**. Click the **Refresh** button in the toolbar of the viewer window, if necessary, to populate the viewer with data.

**3**   Expand the **Number of partitions** node.



It shows that the solver divides the system into three partitions. The first partition is solved using the Forward Euler method, and the other two partitions are solved using the Backward Euler method.

**4** By selecting **Number of variables** under each of the partition nodes, you can see the names of the variables that belong to that partition.



Partition 1 owns the `Inertia.w` variable, which represents the rotational velocity of the Inertia block.

Partition 2 owns `Diode.v` and `Sensing.Ideal Rotational Motion Sensor.phi`, and Partition 3 owns `Inertia.t`. Each partition is responsible for updating the values of the state variables it owns.

**5** The Statistics Viewer also contains links to equations in each partition.

For example, if you select **Number of equations** under Partition 1, and then click the `ssc_nonlinear_electromechanical_circuit/Inertia` link under **Source**, the source code for the Inertia blocks opens in the MATLAB editor, pointing to this equation:

```
t == inertia * w.der;
```

Similarly, you can see the equations for other partitions.

The Partitioning solver assembles all these equations into the system of equations required to simulate the model:

Here, *Sensor.phi* is the abbreviation of the `Sensing.Ideal Rotational Motion Sensor.phi` variable (used to make the presentation of the system of equations more compact). `m0` is the boolean originating from the equation in the Diode block, where `Diode.v` is compared with the **Forward voltage**:

```
if v > Vf
  i == (v - Vf*(1-Ron*Goff))/Ron;
else
  i == v*Goff;
end
```

Comparing this system of equations with the Statistics Viewer data, you can see that the first row of the system is in Partition 3, because Partition 3 owns the `Inertia.t` state variable. Similarly, the second and third rows are in Partition 2, and the fourth row is in Partition 1.

The equation type of a partition depends only on the terms involving owned states and is not affected by the connection function. For example, Partition 3 lists its **Equation Type** as **Linear time-invariant**, despite having nonlinearities in the connection function term, because it is linear time-invariant with respect to its owned states.

During simulation, the Partitioning solver solves the partitions in the same order in which they are listed in the Statistics Viewer (that is, bottom-up in the system of equations), using the specified method (Forward Euler or Backward Euler). The solver uses the updated state values, obtained after solving each partition, to perform state update for upstream partitions.

## See Also

## More About

- "Increase Simulation Speed Using the Partitioning Solver" on page 11-19
- "Model Statistics Available when Using the Partitioning Solver" on page 14-19

# Filtering Input Signals and Providing Time Derivatives

You may need to provide time derivatives of some of the input signals, especially if you use an explicit solver. One way of providing the necessary input derivatives is by filtering the input through a low-pass filter. Input filtering makes the input signal smoother and generally improves model performance. The additional benefit is that the Simscape engine computes the time derivatives of the filtered input. The first-order filter provides one derivative, while the second-order filter provides the first and second derivatives. If you use input filtering, it is very important to select the appropriate value for the filter time constant.

The filter time constant controls the filtering of the input signal. The filtered input follows the true input but is smoothed, with a lag on the order of the time constant that you choose. Set the time constant to a value no larger than the smallest time interval in the system that interests you. If you choose a very small time constant, the filtered input signal is closer to the true input signal. However, this filtered input signal increases the stiffness of the system and slows the simulation.

Instead of using input filtering, you can provide time derivatives for the input signal directly, as additional Simulink signals.

For piecewise-constant signals, you can also explicitly set the input derivatives to zero.

You can control the way you provide time derivatives for each input signal by configuring the Simulink-PS Converter block connected to that input signal:

1   Open the Simulink-PS Converter block dialog box.

2   Expand the **Input Handling** section.
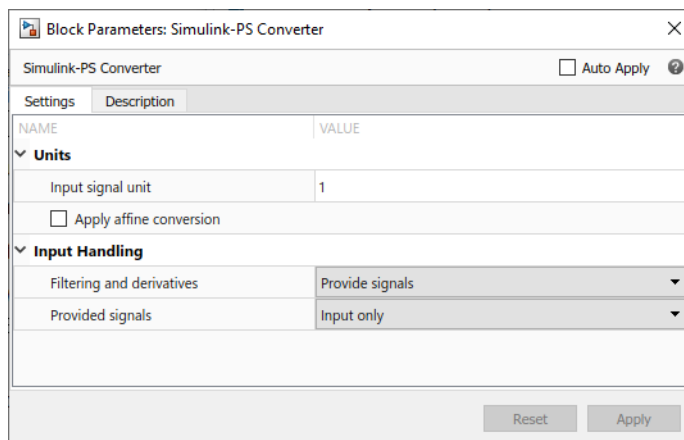


3   When you add a new Simulink-PS Converter block to your model, the default input handling options are `Provide signals` and `Input only`, and the block has one Simulink input port and one physical signal output port.

4   To turn on input filtering, set the **Filtering and derivatives** parameter to `Filter input, derivatives calculated`. Select the first-order or second-order filter, by using the **Input filtering order** parameter, and set the appropriate **Input filtering time constant (in seconds)** parameter value for your model.

5   To avoid filtering the input signal, keep the **Filtering and derivatives** parameter as `Provide signals`. Then set the **Provided signals** parameter value:

- `Input and first derivative` — If you select this option, an additional Simulink input port appears on the Simulink-PS Converter block, to let you connect the signal providing input derivative.
- `Input and first two derivatives` — If you select this option, two additional Simulink input ports appear on the Simulink-PS Converter block, to let you connect the signals providing input derivatives.

**6** Finally, if your input signal is piecewise constant (such as step), you can also explicitly set the input derivatives to zero by selecting the `Zero derivatives (piecewise constant)` value for the **Filtering and derivatives** parameter.

# System Scaling by Nominal Values

| In this section... |
| --- |
| "Enable or Disable System Scaling by Nominal Values" on page 7-31 |
| "Possible Sources of Nominal Values and Their Evaluation Order" on page 7-31 |
| "Specify Nominal Value-Unit Pairs for a Model" on page 7-32 |
| "Modify Nominal Values for a Block Variable" on page 7-33 |

Nominal values provide a way to specify the expected magnitude of a variable in a model, similar to specifying a transformer rating, or setting a range on a voltmeter. Using system scaling based on nominal values increases the simulation robustness. This functionality provides a way to fine-tune scaling of individual variables in a model. It is especially helpful for initial conditions convergence and maintaining a minimum step size.

## Enable or Disable System Scaling by Nominal Values

Using system scaling based on nominal values is a best practice for Simscape models because it improves simulation robustness. Therefore, when you create a new model, scaling by nominal values is enabled by default.

System scaling by nominal values is controlled by the **Normalize using nominal values** configuration parameter.

1 In the Simulink Toolstrip at the top of the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

2 In the Configuration Parameters dialog box, in the left pane, select **Simscape**. The right pane displays the **Normalize using nominal values** check box:

- If the check box is selected, the model provides the scaling information to the solver based on the specified nominal values. To view, add, and edit the value-unit pairs for the model, click the **Specify nominal values** button next to the **Normalize using nominal values** check box.

- If the check box is cleared, the scaling by nominal values is disabled.

## Possible Sources of Nominal Values and Their Evaluation Order

The scaling of each variable is determined by its nominal value and physical units. Nominal values can come from different sources:

- **Block** — You can specify nominal value and unit as variable declaration attributes in a Simscape component file underlying the block. These attributes translate into block parameters $x\_nominal\_value$ and $x\_nominal\_unit$ (where $x$ is the variable name). You can also override these values on individual blocks in the model by setting the corresponding block parameter $x\_nominal\_specify$ to 'on' and supplying different values for $x\_nominal\_value$ and $x\_nominal\_unit$. You can use the block dialog box, the Property Inspector, or set_param and get_param functions to view and change the values of these parameters. For more information, see "Modify Nominal Values for a Block Variable" on page 7-33.

- **Model** — In absence of a nominal value specified for the block, a variable uses the nominal value for the commensurate physical unit specified in the model table. All models have a default table of

nominal values and units (factory default). To view, add, and edit the value-unit pairs for the model, click the **Specify nominal values** button next to the **Normalize using nominal values** check box. For more information, see "Specify Nominal Value-Unit Pairs for a Model" on page 7-32.

- **Derived** — If the model table of nominal values does not contain a row for a unit commensurate with the physical unit of a variable, then the nominal value for this variable is derived from fundamental dimensions. For example, if the variable's initial value is in `lbf`, and there is no entry in the table for force, but the table contains `{10,'lbm'}`, `{12,'ft'}`, and `{2,'min'}`, then the nominal value for that variable is `{10*12/2^2,'lbm*ft/min^2'}`.

- **Fixed** — Event variables, top-level model inputs, and Simscape Multibody variables cannot be scaled according to nominal values.

The Variable Viewer in advanced configuration shows the nominal value and unit for each variable, along with the source. For more information, see "Variable Viewer" on page 8-18.

## Specify Nominal Value-Unit Pairs for a Model

All models have a default table of nominal values and units (factory default). To view, add, and edit the value-unit pairs for a model:

1. In the Simulink Toolstrip at the top of the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

2. In the Configuration Parameters dialog box, in the left pane, select **Simscape**.

3. Make sure the **Normalize using nominal values** check box is selected.

4. Click the **Specify nominal values** button next to the **Normalize using nominal values** check box.

   The model table of nominal values opens in a new window. It contains all the value-unit pairs currently defined for the model.
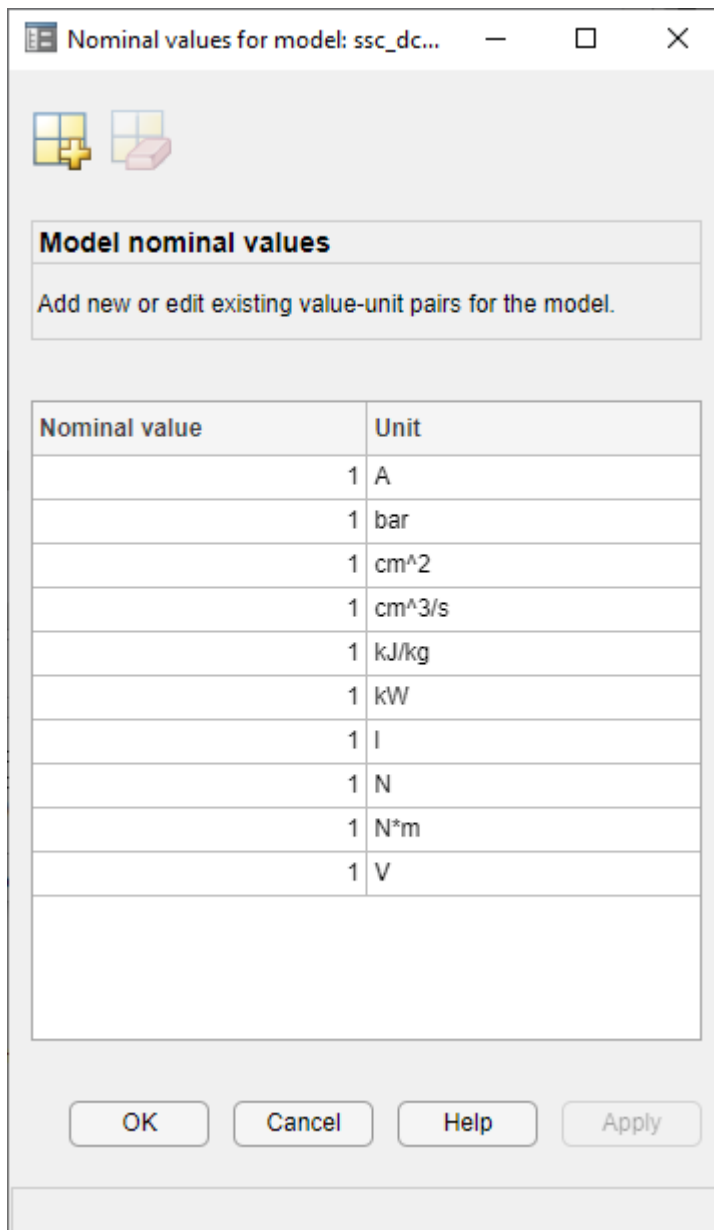
**5** To edit a value-unit pair, select the corresponding cell and enter the new value or unit.

**6**

To add a new value-unit pair, in the top toolbar of the window containing the table, click . This action adds a new empty row at the bottom of the table. Select the cells in this row and enter the nominal value and unit for the new value-unit pair.

**7**

To delete a value-unit pair, select the corresponding row and click .

**8** When finished editing the table, click **OK**. Table data is saved when you save the model.

## Modify Nominal Values for a Block Variable

Each variable in a block has three associated block parameters (where *x* is the variable name):

- *x_nominal_specify* — Lets you override the system default nominal value for variable *x* in this particular block. The default parameter value is `'off'`, in which case the variable nominal value is determined according to the evaluation order described in "Possible Sources of Nominal Values and Their Evaluation Order" on page 7-31. Set this parameter to `'on'` to use the *x_nominal_value* and *x_nominal_unit* parameter values for scaling.

- *x_nominal_value* — If the *x_nominal_specify* parameter is set to `'on'`, then this value, in conjunction with the nominal unit parameter, determines the scaling of variable *x* in this particular block. The parameter value must be a numeric value, specified as a character vector. The default parameter value is `'1'`.

- *x_nominal_unit* — If the *x_nominal_specify* parameter is set to `'on'`, then this unit, in conjunction with the nominal value parameter, determines the scaling of variable *x* in this particular block. The parameter value must be a valid physical unit name, specified as a character vector. The unit must be commensurate with the unit specified for the initial value of the variable. The default unit is the same as for the initial value.

---

**Note** Nominal value and unit can be specified as variable declaration attributes in a Simscape component file underlying the block. For more information, see "Nominal Value and Unit for a Variable". In this case, the nominal value and unit parameters for that variable get their default values from the variable declaration attributes.

---

You can use `set_param` and `get_param` functions to view and change the values of these parameters.

For example, to change the nominal value and unit for variable `i` (current) for an individual block, select this block in the model and type:

```
set_param(gcb,'i_nominal_specify','on')
set_param(gcb,'i_nominal_value','10')
set_param(gcb,'i_nominal_unit','mA')
```

This sequence of commands overrides the default nominal value for the block variable and sets it to 10 mA.

To perform the same actions using the block dialog box:

1    Double-click the block in the model.
2    In the block dialog box, expand the **Nominal Values** section.

**3** Select the check box next to **Current**. This action is equivalent to setting the `i_nominal_specify` parameter to `'on'`.

**4** Once the **Current** check box is selected, its **Value** field becomes editable. Enter `10` and select `mA` from the unit drop-down list.

## See Also

variables

## More About

*   "Variable Viewer" on page 8-18
*   "Use Scaling by Nominal Values to Improve Performance" on page 7-37

# Use Scaling by Nominal Values to Improve Performance

Nominal values provide a way to specify the expected magnitude of a variable in a model, similar to specifying a transformer rating, or setting a range on a voltmeter. This example shows how you can fine-tune scaling of individual variables in a model to improve the solver performance and increase the simulation robustness.

When the solver performs numerical simulation and analysis, it uses variables without units. Nominal values is a way to convert engineering variables with units into variables without units and to scale them for optimal solver performance. The nominal value has a value with a unit and this unit is used to strip away the unit for numerical calculations. The nominal value then determines the magnitude of the variable as seen by the numerical algorithms. It is generally beneficial to have magnitudes of similar scale.

Consider a simple mass-spring-damper model with a large spring constant.
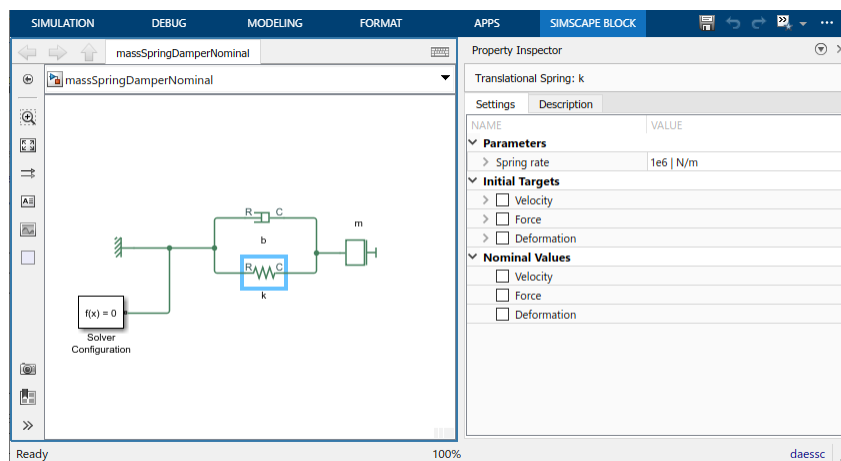


The model uses the default block parameter settings, with the exception of spring rate:

- Spring rate, $k$ = 1e6 N/m
- Damping coefficient, $b$ = 100 N/(m/s)
- Mass, $m$ = 1 kg

The initial mass velocity variable, $v$, has `High` priority and the initial target value of 0.1 m/s. The model uses the default nominal values, with `m` as the length unit.

When you simulate this model, the spring **Deformation** (position) variable $x$ is small, in the 10^-5 m range.
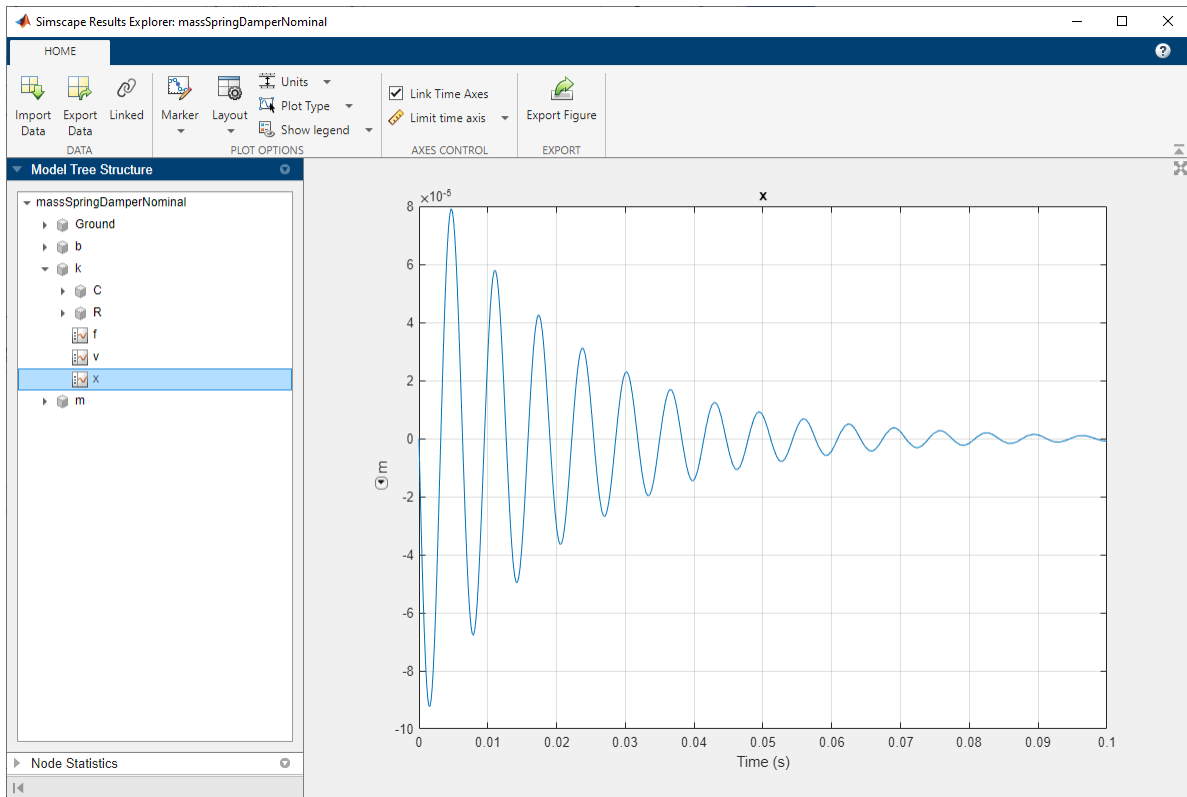
Open the Solver Profiler by clicking the hyperlink in the lower-right corner of the model window.
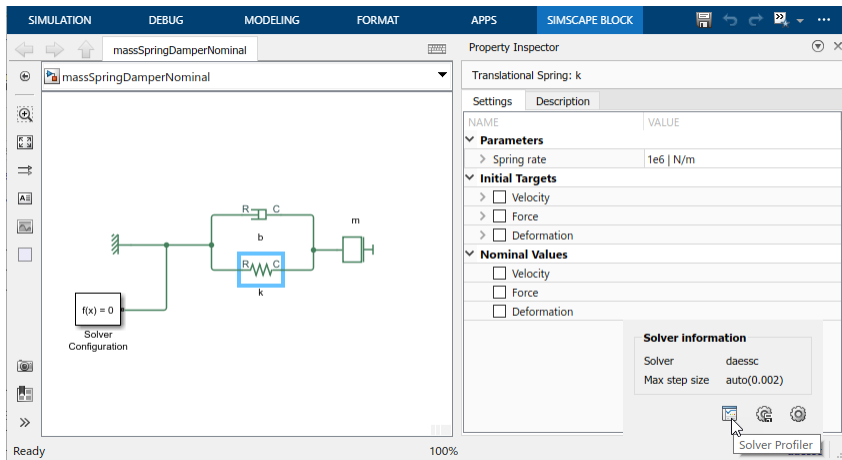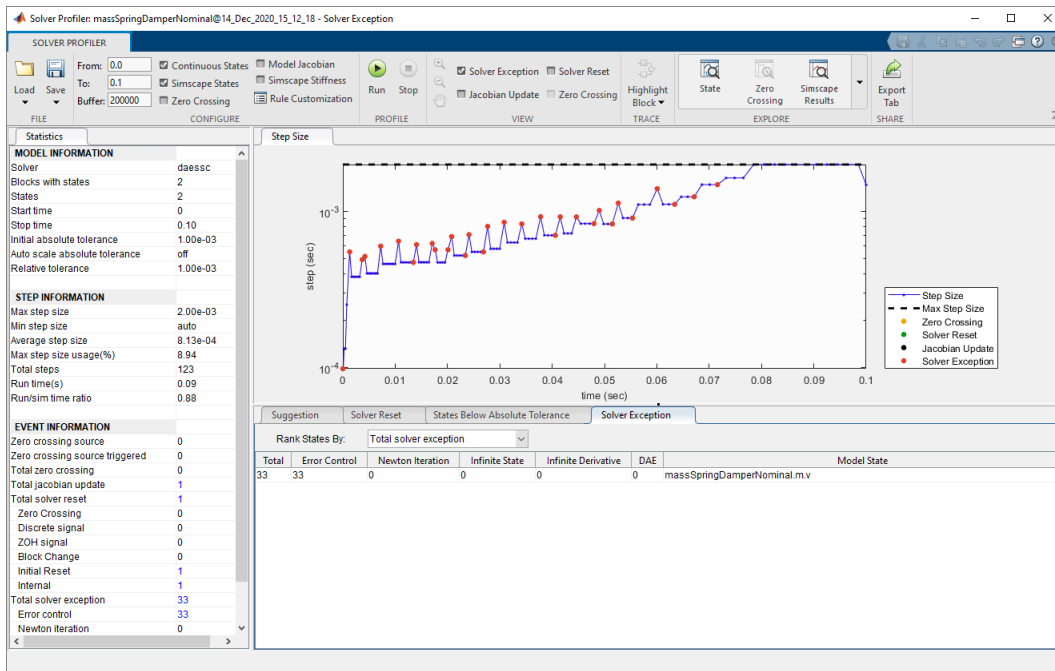
The solver completely ignores the position variable and just looks at the velocity variable. The numerical magnitude of position is below the tolerance for that variable.

The State Viewer, accessed from the Solver Profiler toolstrip, shows the magnitude of each variable as seen by the solvers.
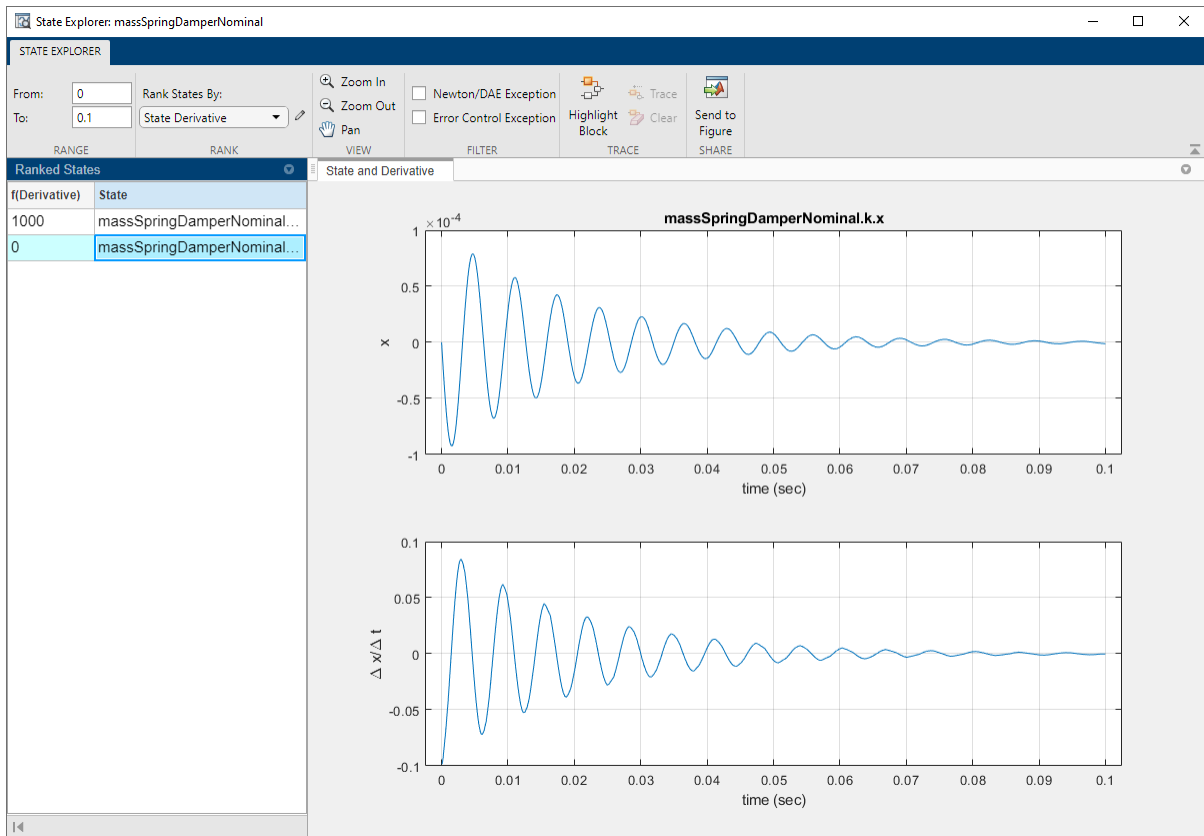
The situation is clear if we look at the model equations:

$$m \cdot \frac{dv}{dt} + b \cdot v + k \cdot x = 0$$

$$\frac{dx}{dt} = v$$

When $k$ is very large, compared to $m$ and $b$, then $x$ is small, so that the product $k*x$ is of reasonable size, compared to the other terms in the equation. To remedy the situation, we need to scale $x$. In other words, we need to choose a nominal value for $x$ that is small, so that the scaled variable, $x_s$, becomes more reasonably sized:

$$x_s = x \cdot c,$$

where $c$ is large. For example, if the nominal value is 1 μm and the original $x$ is in m, then $c$ is 1e6. The equations then become:

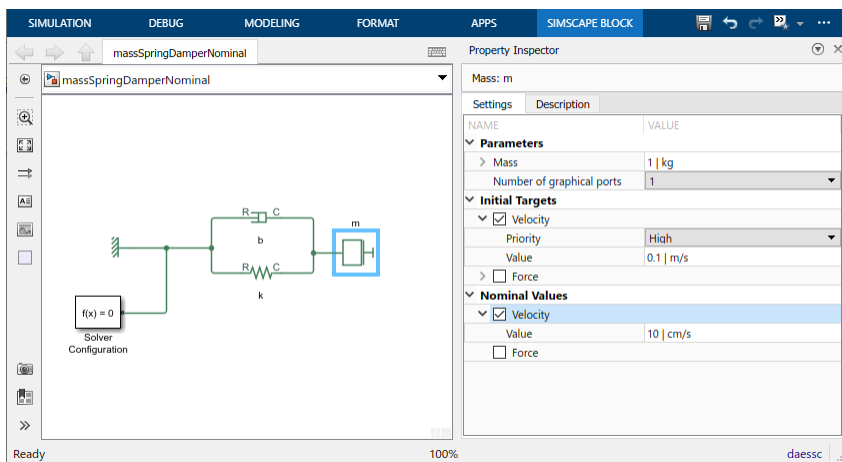$$m \cdot \frac{dv}{dt} + b \cdot v + (k/c) \cdot x_s = 0$$

$$\frac{dx_s}{dt} = c \cdot v$$

and the terms have magnitudes of similar scale.

Let us apply this scaling principle to the model variables. First, use the Property Inspector to change the nominal value of the spring **Deformation** variable, $x$, to `100 um` (1e-4 m).
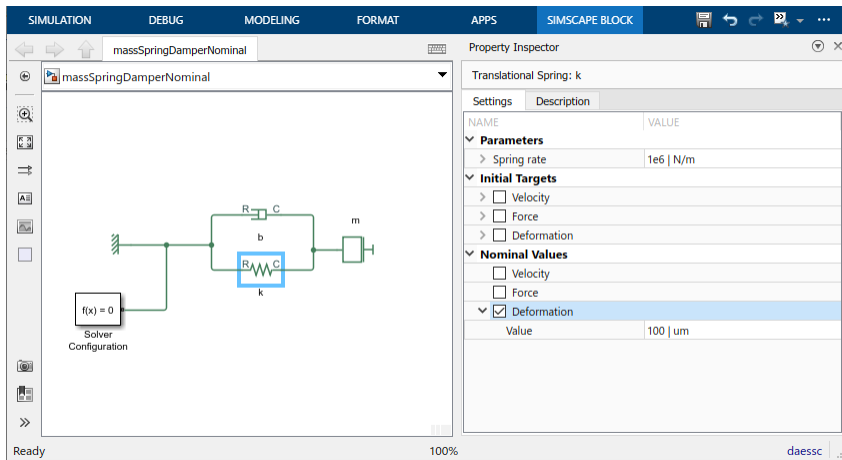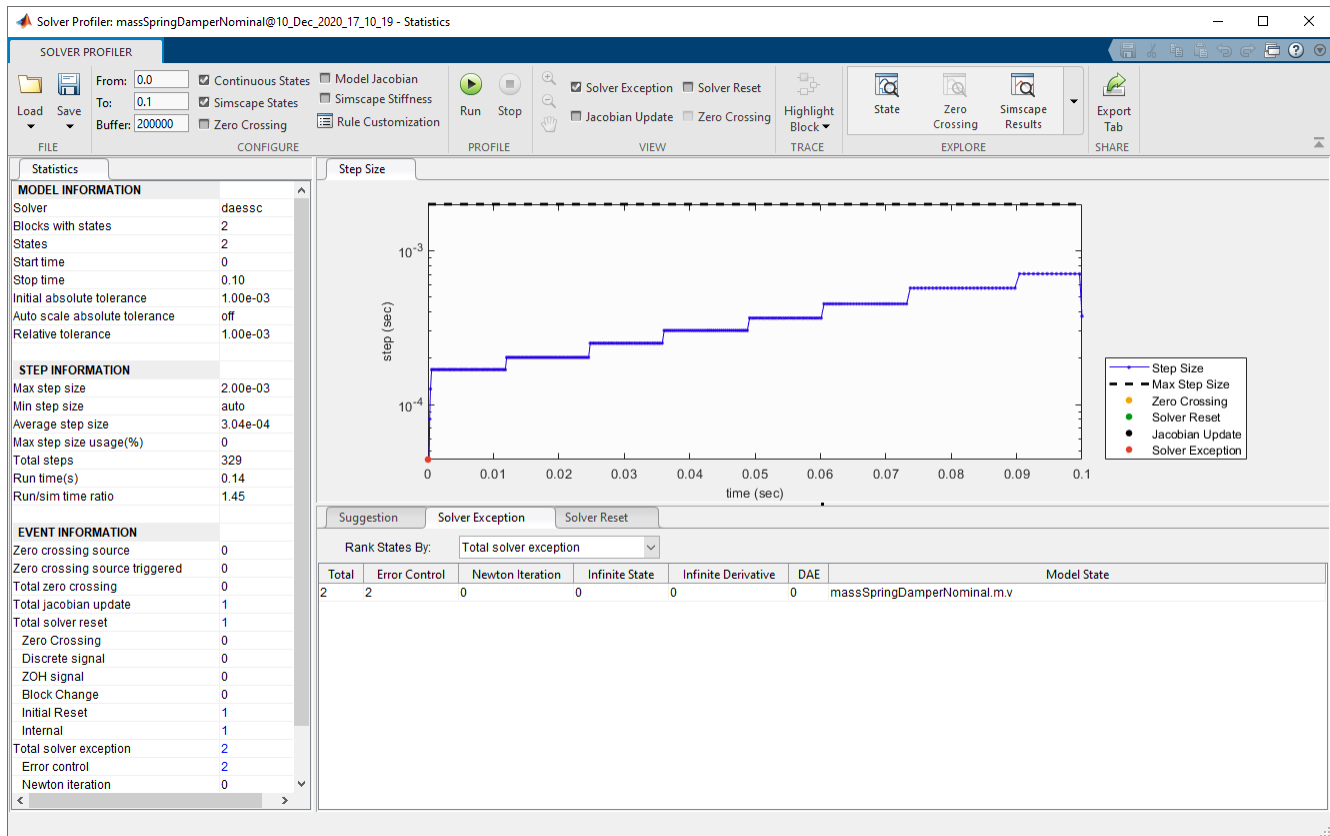
Similarly, change the nominal value of the mass **Velocity** variable to 10 cm/s (0.1 m/s).



This brings the scale of the magnitude of both variables seen by the solver to approximately 1.
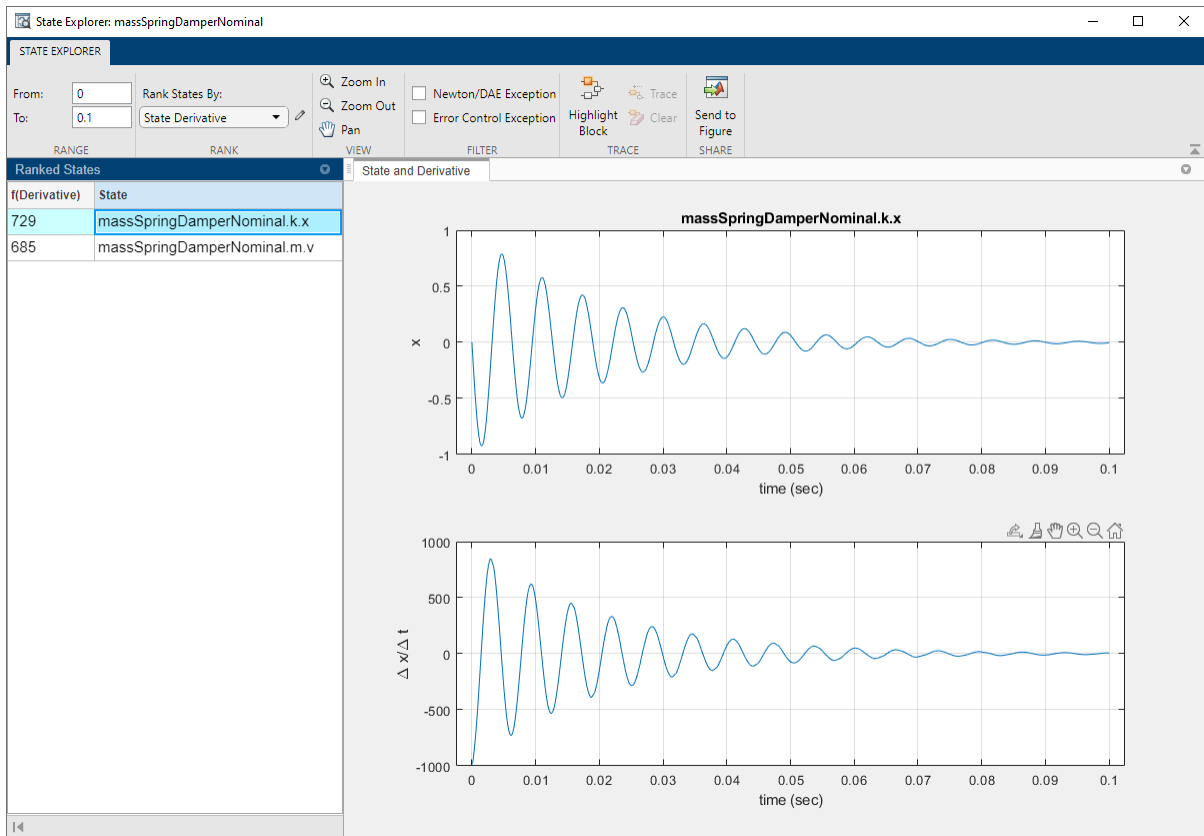
Rerun the simulation.

There are now only two solver exceptions, at the beginning of the simulation.

Note that the effective absolute tolerance is now tighter for the mass velocity. The effective absolute tolerance for Simscape models has a unit, and is computed as (nominal value * global `AbsTol` without units). In the first simulation run the effective absolute tolerance was 1e-3 m/s, and now it is 1e-4 m/s because the nominal value changed magnitude and the global `AbsTol` is still `1e-3`. However, the speed of the simulation is similar, even with the increased `AbsTol` on the mass velocity and increased time steps.

The values, as seen by the solver, are similar in magnitude for both the velocity and position variables.

Now, change the global `AbsTol` to `1e-2`, to more closely match the accuracy of the velocity variable during the first simulation run.

Rerun the simulation.
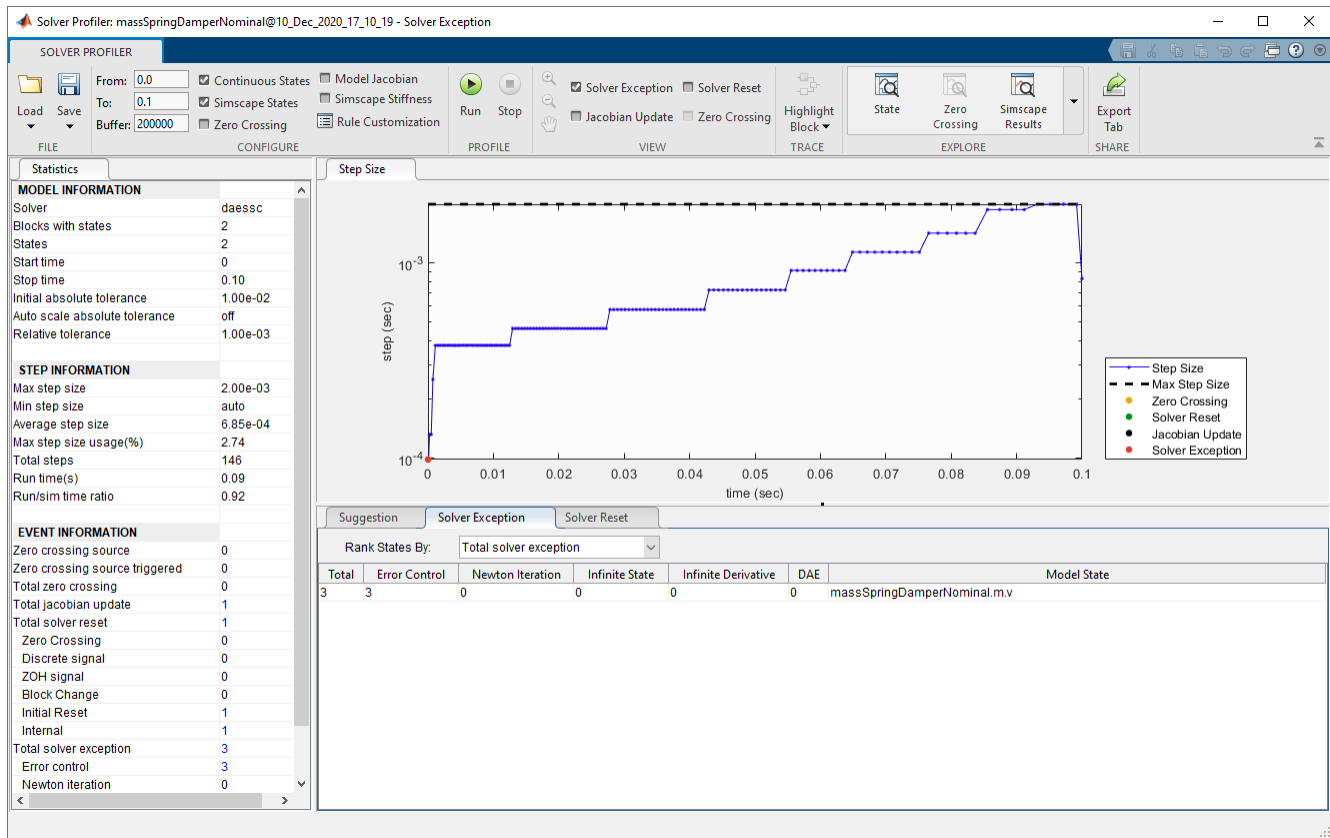
The timesteps are similar and the number of exceptions is 3, also at the beginning of simulation.

## See Also
Solver Profiler

## More About
- "System Scaling by Nominal Values" on page 7-31

# Select Nominal Values Using the Variable Scaling Analyzer

| In this section... |
|---|
| |
| |
| |
| |
| |

Simscape can perform nominal value analysis to help you select nominal values. Nominal values are a means to specify the expected magnitude of a model variable to improve simulation conditions. Choosing the right nominal values can improve the conditions to make convergence possible or improve speed. To learn more, see "System Scaling by Nominal Values" on page 7-31.

## When to Perform Variable Scaling Analysis

Solvers may encounter speed issues due to variable scaling. For example, when a state is too small relative to the **Absolute tolerance** parameter in the Simulink **Solver** pane, the solver may ignore it during some or all of the simulation. The **Simscape Variable Scaling Analyzer** app identifies states that

- remain below the **Absolute tolerance** parameter for the duration of the simulation.
- trend near the **Absolute tolerance** but do not remain below it.
- are orders of magnitude larger than other variables.

The **Solver Profiler** tool also identifies some of these issues in the **States Below Absolute Tolerance** tab. All of these cases can lead to solver performance issues.

**Tip** Your model will perform best when all of your states are approximately O(1).
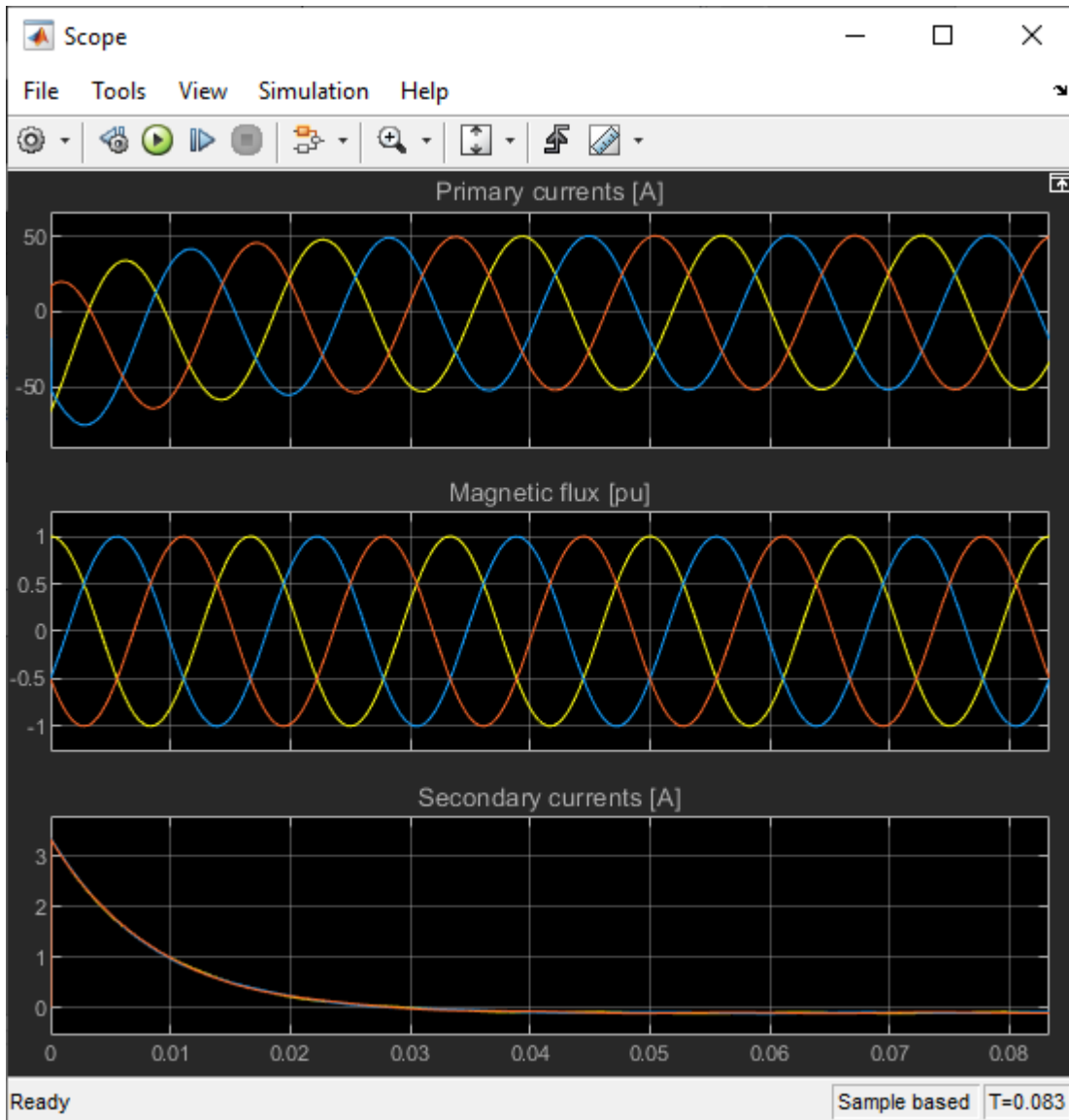
Consider the simple three-phase circuit model in the figure.
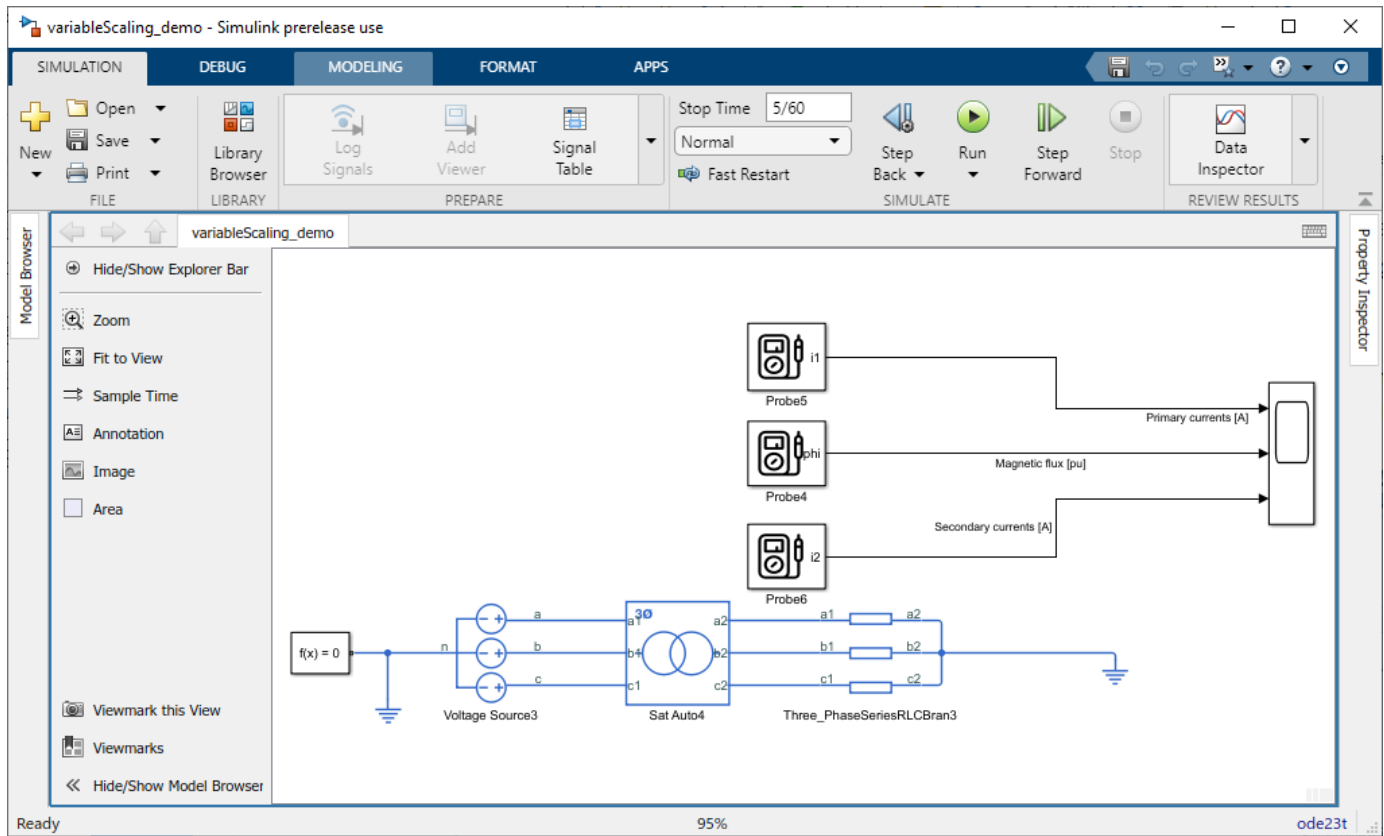
Running the model with the default nominal values generates this warning:

```
First solve for initial conditions failed to converge. Trying again with all
high priorities relaxed to low.
```

In this case, the solver ignores one variable with **Priority** set to High. This produces the result in the figure.

The Primary currents [A] subplot and the Magnetic flux [pu] subplot exhibit the expected three-phase behavior; however, the Secondary currents [A] subplot does not. The solver failed to properly initialize the secondary currents. Running the simulation within the Solver Profiler reports dense groups of solver exceptions and zero crossings but does not indicate states below the absolute tolerance. Note the Off setting for **Auto scale absolute tolerance**.
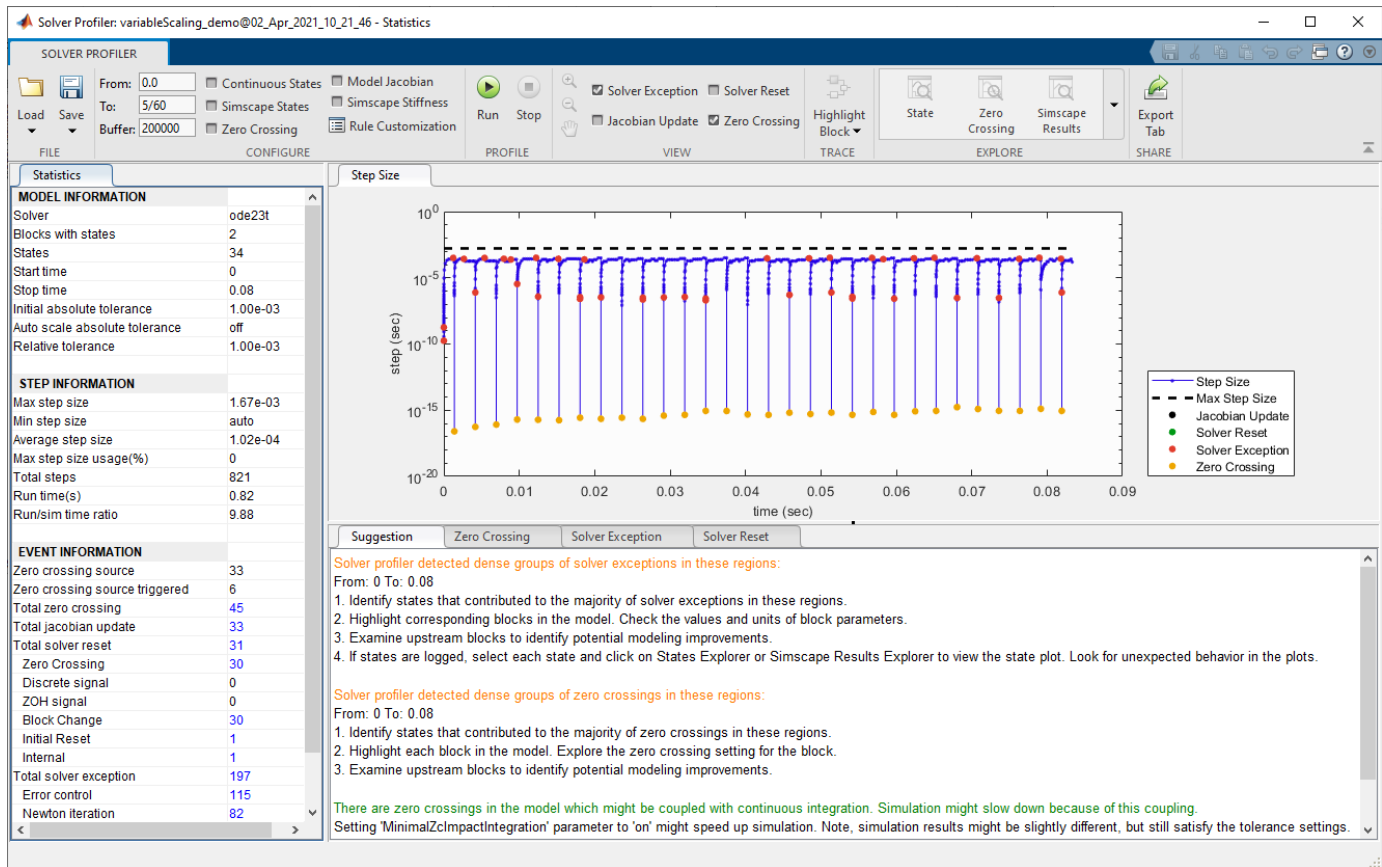
## Opening the Simscape Variable Scaling Analyzer

To open the **Simscape Variable Scaling Analyzer**, enter this command in the MATLAB command window:

```
simscapeVariableScalingAnalyzer
```

You can optionally specify the model that you want to analyze as an argument for the command.

Otherwise, you can choose which model to attach using the app interface.

## Analyzing the Model

Running the simple circuit model in the app generates the results in the figure.

The app highlights potentially problematic states and provides suggestions based on the type of issues it finds. There is a cluster of highlighted states that share the unit Wb. The solver assigns this unit a nominal value of 1. However, the **LogMaxAbsData** column reports that the log of the states with this unit trend around -2.4, which is in proximity to -3, the log of the **Absolute tolerance** parameter. You can view a visual representation of the data by selecting the highlighted states and clicking **Plot State**.
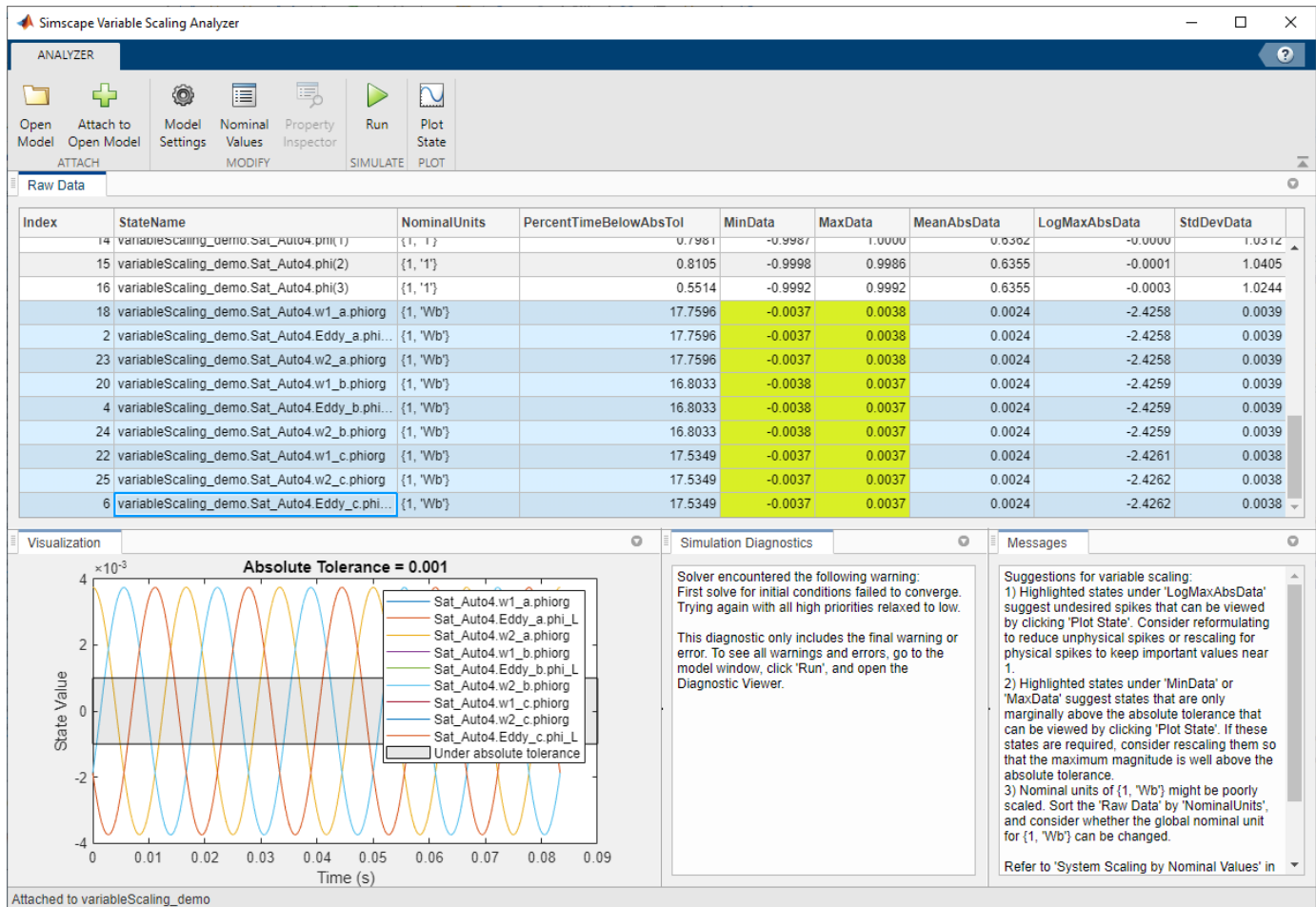
Note the desired three-phase behavior, where each phase has three superimposed variables. The proximity of the states to the absolute tolerance suggests improper scaling of the flux.

## Selecting and Rescaling Nominal Values

You can rescale nominal values locally using the Property Inspector for a block that creates a problematic state or you can globally rescale nominal values using the Nominal values for model window.

### Locally rescaling nominal values

The problematic states deal with the units Wb, which do not explicitly tie to a certain block in the model. The figure shows the Property Inspector window for the Two-Winding Transformer (Three-Phase) block. In the **Nominal Values** section, when you select the check box next to each variable name, you can view the default local block settings for each value it generates.

Adjusting these values does not improve the performance of this model. Consider globally rescaling nominal values if local rescaling is insufficient.

**Globally rescaling nominal values**

You can launch the nominal values window for the model from the **Simscape Variable Scaling Analyzer** app by clicking **Nominal Values**.

To add a nominal value-unit pair, click **Add nominal value-unit pair** . In this example, rescaling the nominal value to `1e-3` makes the nominal value closer to where the state values trend. To learn more, see "System Scaling by Nominal Values" on page 7-31.

| Nominal value | Unit |
|---:|:---|
| 1 | A |
| 1 | bar |
| 1 | cm^2 |
| 1 | cm^3/s |
| 1 | kJ/kg |
| 1 | kW |
| 1 | l |
| 1 | N |
| 1 | N*m |
| 1 | V |
| 1e-3 | Wb |

**Nominal values for model: Initializat...**

Model nominal values

Add new or edit existing value-unit pairs for the model.

OK    Cancel    Help    Apply

Changes to nominal value-unit pairs are saved.
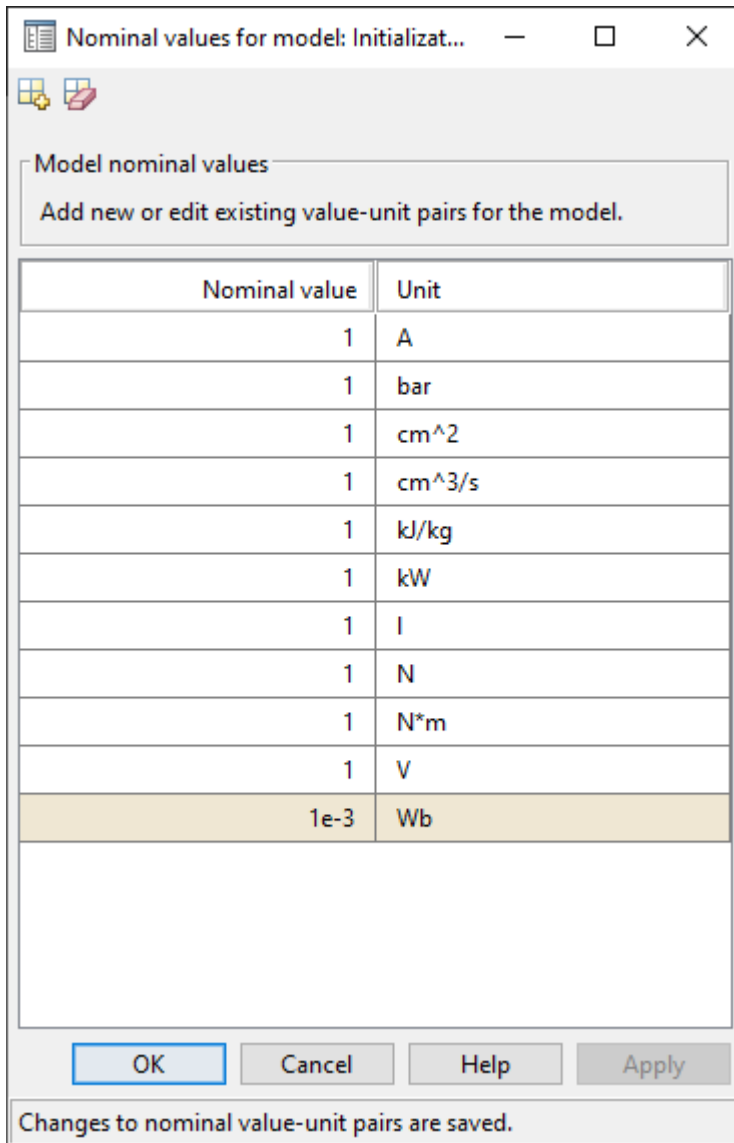
## Checking Results

Rerunning the model from the analyzer after specifying a new value-unit pair generates these results.

The states from before are no longer highlighted, but one new state appears to behave differently than before. This may need to be rescaled, depending on your requirements. However, the model Scope block displays properly initialized secondary currents.
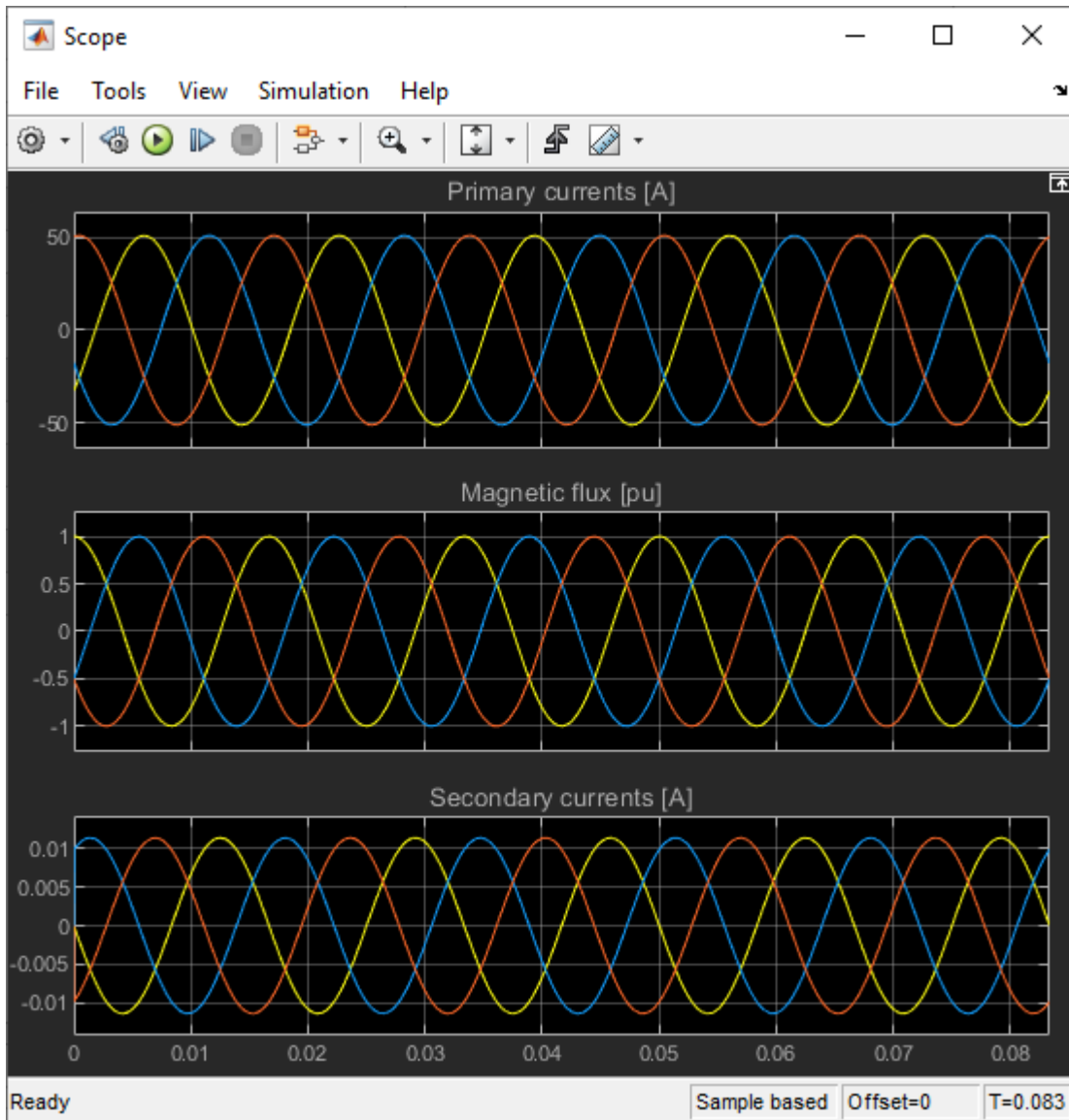
## See Also

**Apps**
Simscape Variable Scaling Analyzer | Solver Profiler

## More About

- "Absolute tolerance"
- "System Scaling by Nominal Values" on page 7-31
- "Use Scaling by Nominal Values to Improve Performance" on page 7-37

# Frequency and Time Simulation Mode

| In this section... |
| --- |
| |
| |
| |
| |

Frequency and time simulation mode speeds up simulation of systems with a single nominal frequency by letting you increase the maximum step size for variable solvers. This mode also lets you perform phasor analysis of such systems by using the blocks in the Periodic Operators sublibrary of the Physical Signals library.

Depending on your task, you can switch between time and frequency-and-time simulation modes without modifying the model. For example, use the time simulation mode to study transient effects, and then switch to the frequency-and-time mode to perform the phasor analysis of a model.

## Speeding Up Model Simulation

Frequency-and-time equation formulation is intended for linear and linear parameter-varying (LPV) systems. It speeds up the simulation using a variable-step solver because the solver step size is no longer limited by the period of the nominal frequency.

The frequency-and-time simulation mode is based on changing the equation formulation for a physical network with a nominal frequency $\omega_0$ by dividing its variables into two categories:

- Time variables, which vary slowly relative to the nominal period $2\pi/\omega_0$
- Frequency variables, which are sinusoidal and represent forced response at the nominal frequency, $x = d_x + a_x\cos(\omega_0 t) + b_x\sin(\omega_0 t)$

In time simulation mode, the solver step size is typically limited to a small fraction of a period of the nominal frequency. In frequency-and-time simulation mode, the representation of frequency, or fast, variables as sinusoids allows the variable solver to take much larger steps. The speeding-up effect is especially pronounced in complex machine systems that use three-phase Simscape Electrical™ blocks.

When you run a model in frequency-and-time simulation mode, the software automatically detects the nominal frequency and determines which of the variables are fast (frequency) and which are slow (time).

To benefit from improved performance, the time variables in the system should have slow dynamics. If time variables have time constants comparable to, or smaller than, the nominal frequency period, frequency-and-time simulation of such a system will be slow (due to the large number of timesteps required to resolve these dynamics) and possibly inaccurate. In such cases, use the time simulation mode instead.

## Variable Initialization for Frequency-and-Time Simulation

Variable initialization for frequency-and-time equation formulation follows these rules:

- For time variables and algebraic frequency variables, initialization targets and priorities are preserved.
- For dynamic frequency variables, initialization priority is switched to `None` because the solver is using the sinusoidal steady-state approximation for these variables.

## Limitations

Frequency-and-time equation formulation is intended for systems with a single nominal frequency. In other words:

- The model must have at least one sinusoidal source in its physical network.
- In case of multiple sinusoidal sources, they must all operate at the same frequency.
- Blocks outside the physical network, such as a Sine Wave block, are not considered valid sinusoidal sources.

If you try to run a frequency-and-time simulation on a model that does not meet the above criteria, you get an error message.

## Perform Sinusoidal Steady-State Analysis of a Model

This example shows how you can deploy different simulation modes on the same model, depending on the type of analysis you want to perform.

The transmission line model used in this example is built from 50 identical blocks, each block representing a single T-section segment. For more information, see "Transmission Line" on page 20-304. The model has one sinusoidal source (AC Voltage) and operates at a nominal frequency 200 MHz, which makes it a good candidate for frequency-and-time simulation.

**1**    Open the Transmission Line example model by typing `ssc_transmission_line` in the MATLAB Command Window.

Expand the Voltage Sensor subsystem, which consists of a Voltage Sensor block, a Solver Configuration block, and a PS-Simulink Converter block connected to the scope.
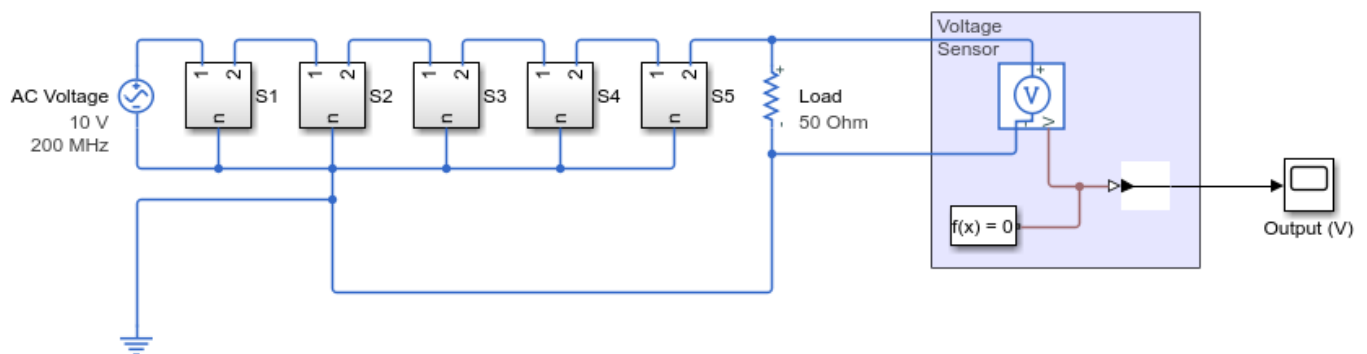


**Transmission Line**
1. Plot voltages along line (see code)
2. Explore simulation results using sscexplore
3. Open transmission line component library
4. Learn more about this example

**2**     To analyze the transient behavior of the model, run it in the time simulation mode.

Open the Solver Configuration block dialog box and verify that the **Equation formulation** parameter is set to Time. Simulate the model.
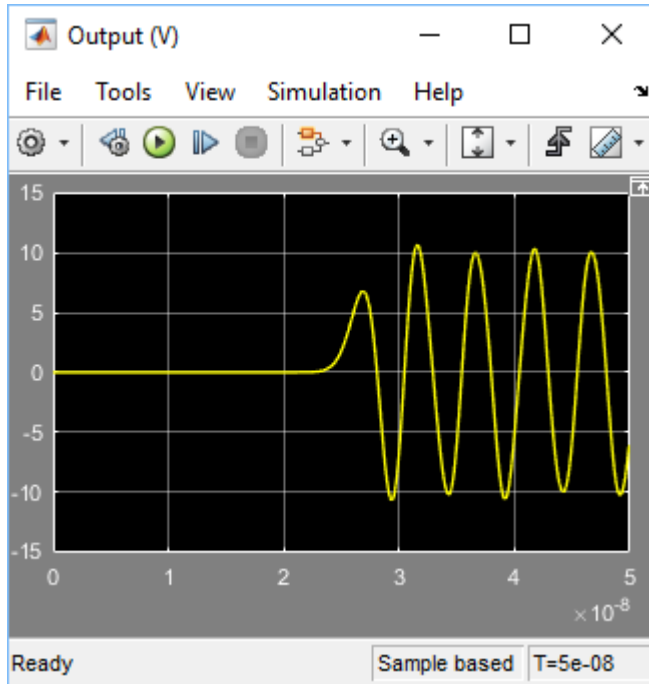


You can observe the transmission delay from the simulation results.

**3**     To perform the phasor analysis, switch to frequency-and-time simulation mode.

Open the Solver Configuration block dialog box and set the **Equation formulation** parameter to Frequency and time. Simulate the model.
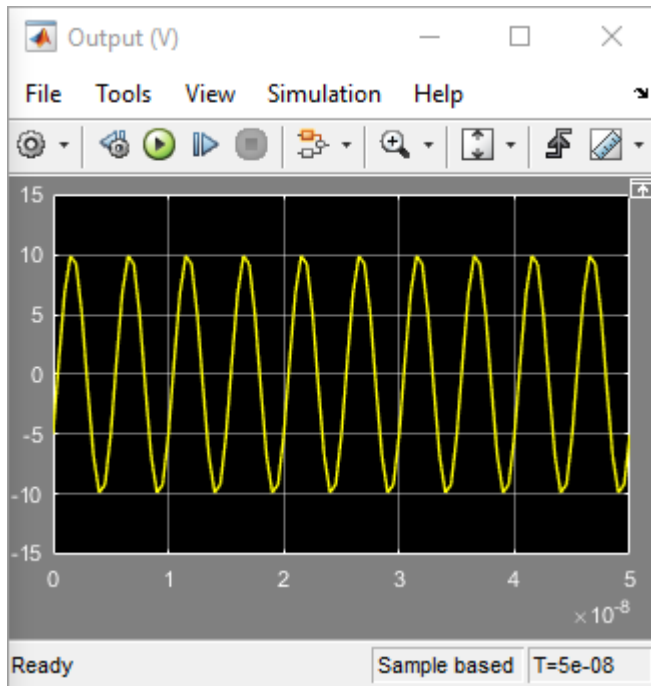
Notice that in frequency-and-time mode the simulation starts in sinusoidal steady state.

**4** To determine the amplitude and phase of the base frequency, connect the PS Harmonic Estimator (Amplitude, Phase) block to the voltage sensor output. Add the respective scopes.
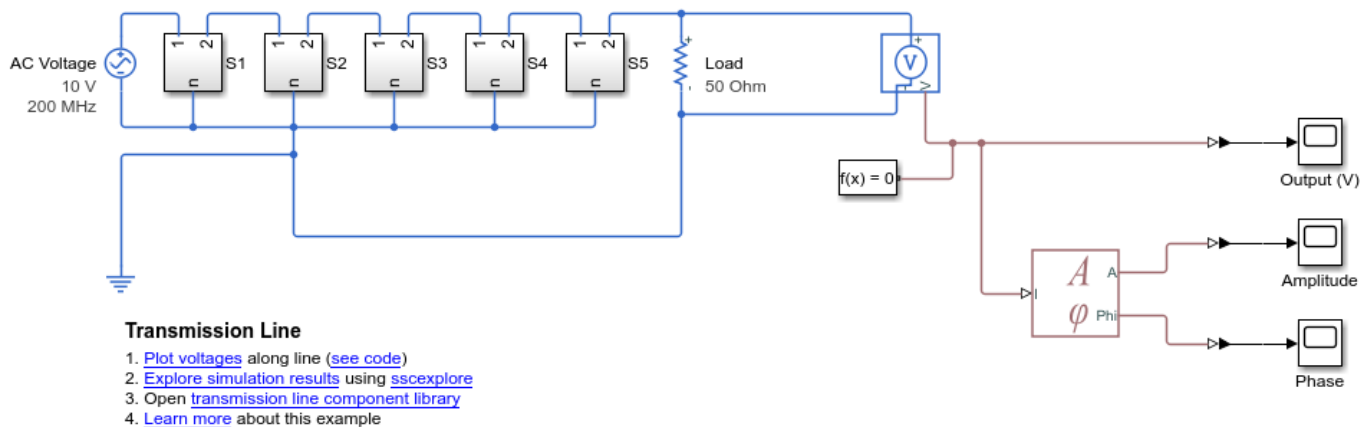


**5** Open the PS Harmonic Estimator (Amplitude, Phase) block dialog box and set the **Base frequency** parameter to 200 MHz, to match the nominal frequency of the model. Also set the **Minimum amplitude for phase detection** parameter unit to V, to match the unit of the input signal.

**6** Double-click the PS-Simulink Converter block connected to port **A** of the PS Harmonic Estimator (Amplitude, Phase) block. Set the **Output signal unit** parameter to V.

**7** Simulate the model.
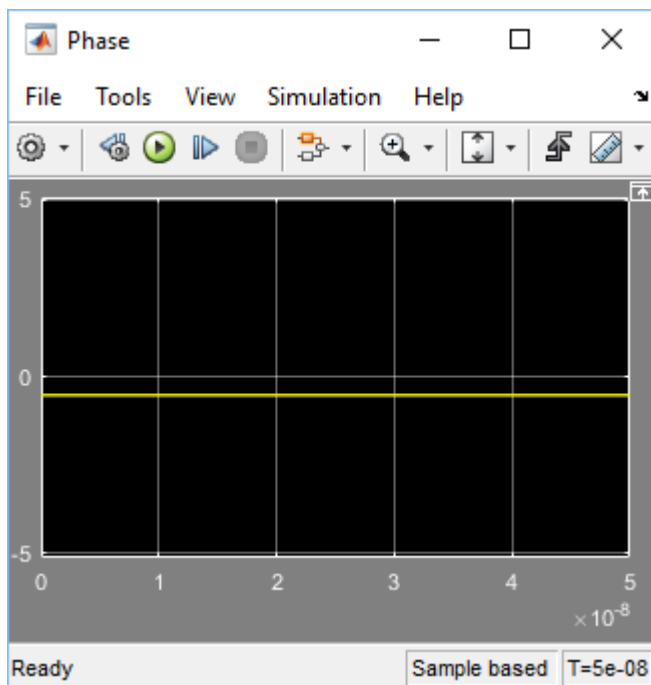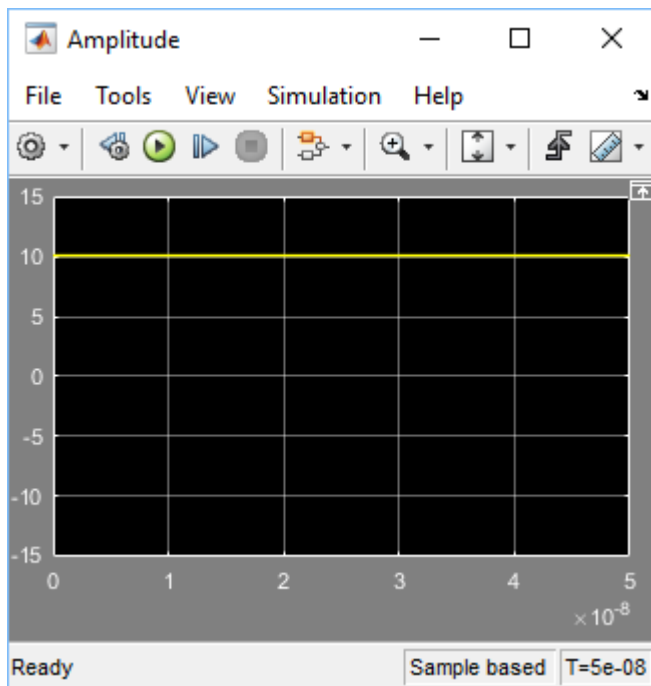
**8**  The logged simulation data for frequency variables contains subnodes that let you examine the variable instantaneous value, amplitude, phase, and offset data separately.
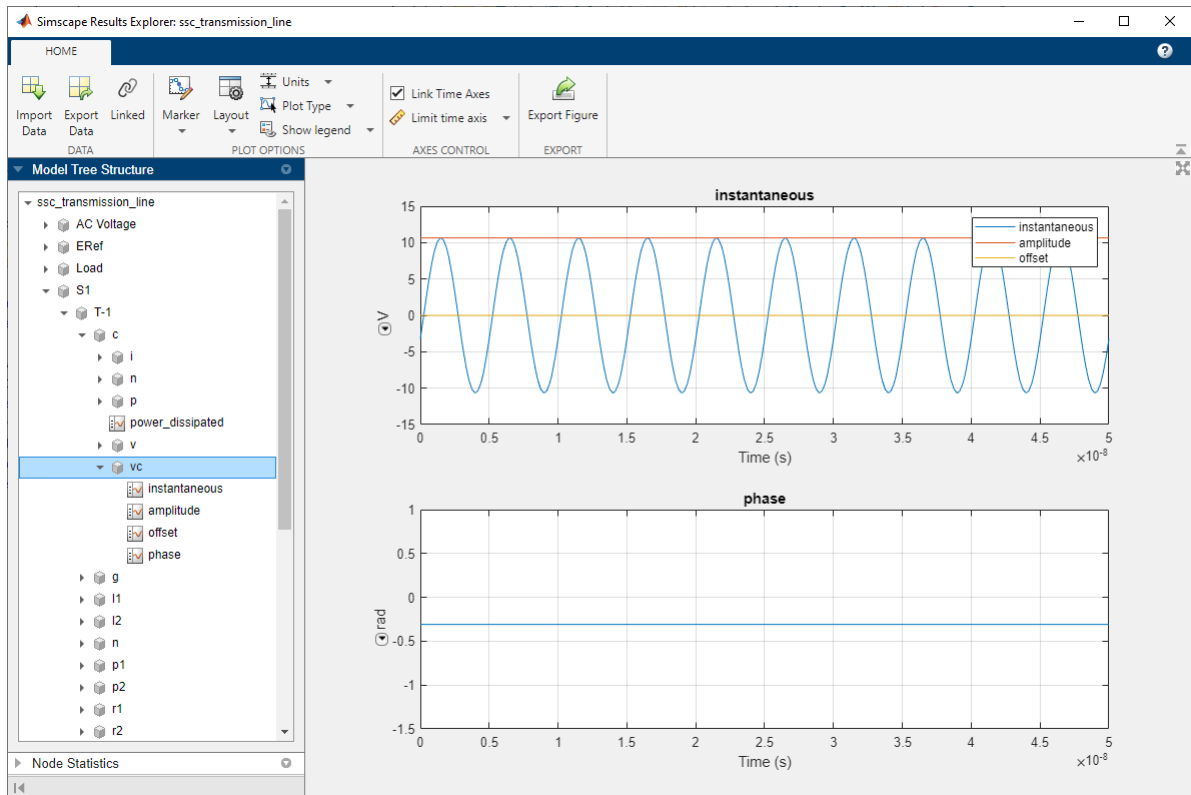
**Note** If you use the workflow of live-streaming the data to Simulation Data Inspector, the recorded simulation data does not contain these subnodes. To view the additional subnodes for frequency variables, clear the **Record data in Simulation Data Inspector** check box and rerun the simulation.

## See Also

Solver Configuration | PS Harmonic Estimator (Amplitude, Phase)

## More About

- "Phasor-Mode Simulation Using Simscape Components" (Simscape Electrical)
- "About the Simscape Results Explorer" on page 13-24

# Simscape Stiffness Impact Analysis

When you use an explicit solver, the simulation may become unstable because the system is stiff. To learn more, see "Explicit Versus Implicit Continuous Solvers". In most models, you can change the block parameter values to reduce system stiffness and remove the instability. The **Stiffness Impact Analysis** option in the **Solver Profiler** tool lets you analyze the Simscape networks in your model and determine which variables and equations have the most impact on system stiffness. You can then modify the values of parameters involved in these equations to make the system less stiff.

To perform impact analysis on your Simscape model, click the **Simscape Stiffness** drop-down button in the **Solver Profiler** interface, and enter the stiffness analysis checkpoints as a vector.

For example, consider a model where a resistor and capacitor are connected in series with a DC voltage source.
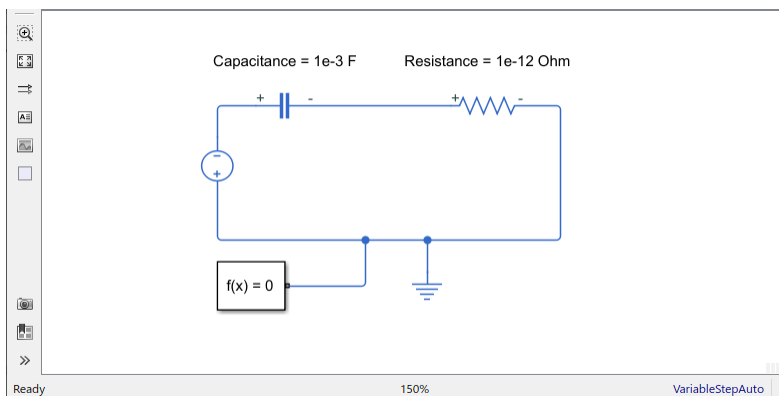


To access the Stiffness Impact Analysis tool, open the Solver Profiler by clicking the hyperlink in the lower-right corner of the model window.
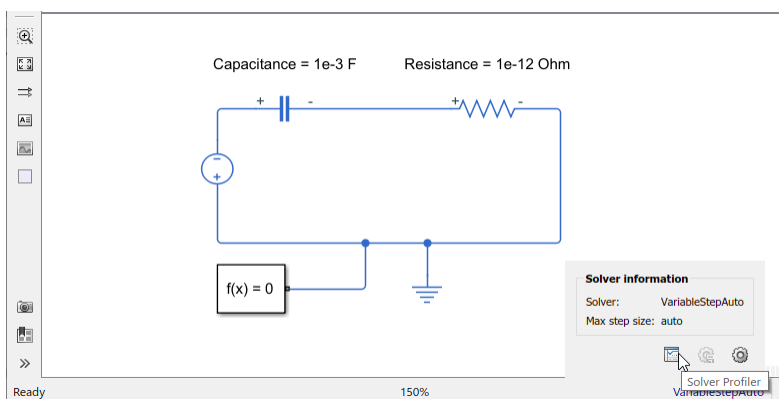


Click the **Simscape Stiffness** drop down menu in the **Solver Profiler** toolstrip. Enter a vector, and click **Run**. This example uses the vector `[0, 1, 2]`.
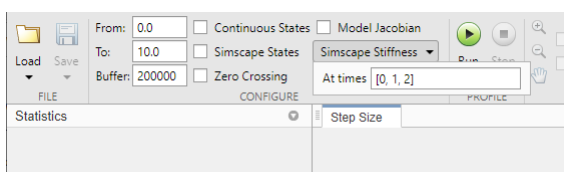
The **Simscape Stiffness** tab in the bottom pane shows that the variable with the most impact on the system stiffness is `i(Current)` in the Resistor block, and the corresponding eigenvalue is `-9.999999e+08` at each interval. Note that when the time steps do not line up with the times in your vector, the tool performs stiffness analysis at the closest time steps before and after the instances you specify, $t_-$ and $t_+$, respectively.

---

**Tip** The greater the magnitude of a negative eigenvalue, the more unstable the system becomes due to stiffness.

---
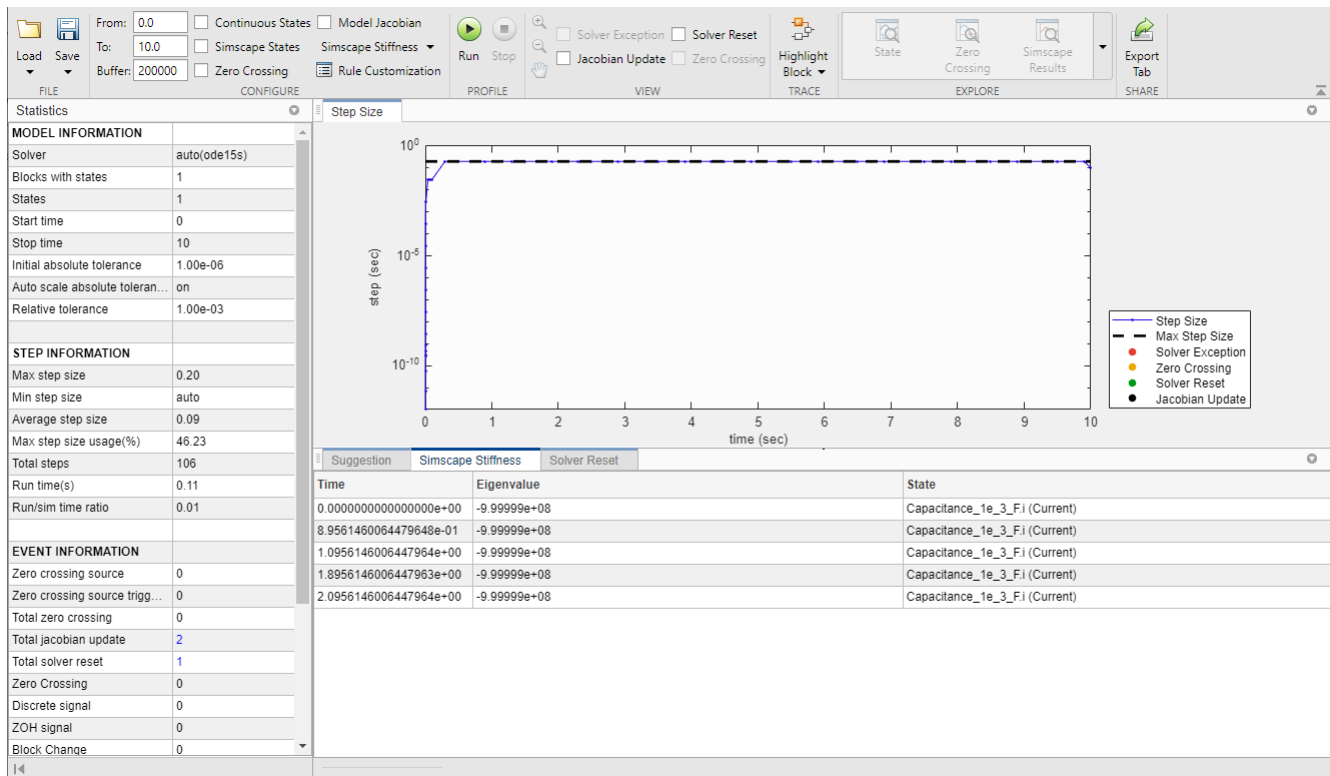


The Resistor block uses the equation

```
v == R*i;
```

Note that the block derives the current from the resistance. To reduce the stiffness due to the current, reduce the value of the resistance. For example, if you set **Resistance** to `1e-6 Ohm` and rerun the simulation using the **Stiffness Impact Analysis** tool, the corresponding eigenvalues drop to `-5.00000e+08`. A reduction of this magnitude may be enough for your model to run successfully.
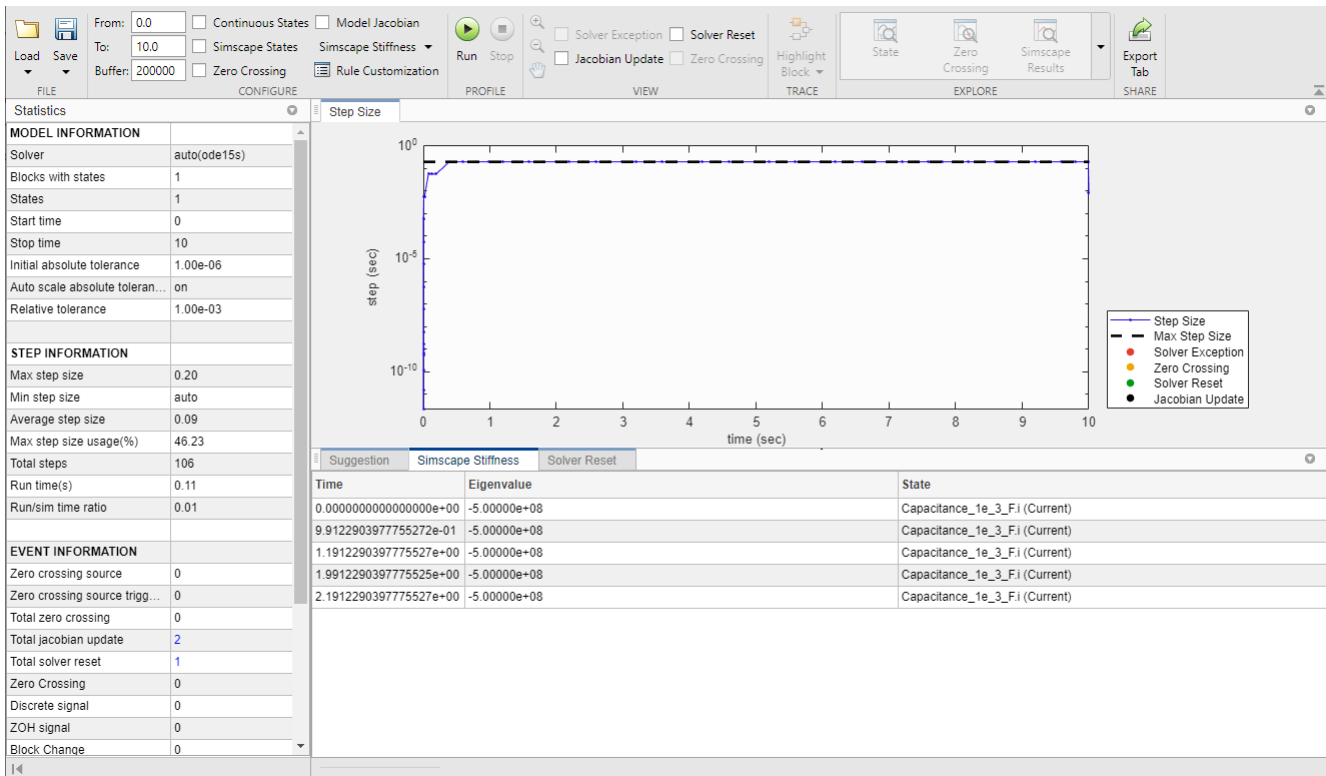
The Stiffness Impact Analysis tool has the following limitations:

- The tool performs stiffness analysis of the Simscape networks only. If a model does not contain Simscape blocks, the tool does not show any results.
- The tool does not produce results and issues a warning if the system is algebraic, high-index, uses frequency-and-time equation formulation, or contains Simscape Multibody blocks.
- The tool does not work with Simscape Electrical Specialized Power Systems.

## See Also

Solver Configuration | Solver Profiler | Simscape Variable Scaling Analyzer

## More About

- "Explicit Versus Implicit Continuous Solvers"
- "Important Concepts and Choices in Physical Simulation" on page 7-15
- "Solvers for Real-Time Simulation" on page 11-76

# Troubleshooting Simulation Errors

| **In this section...** |
| --- |
| "Troubleshooting Tips and Techniques" on page 7-64 |
| "System Configuration Errors" on page 7-64 |
| "Numerical Simulation Issues" on page 7-66 |
| "Initial Conditions Solve Failure" on page 7-67 |
| "Transient Simulation Issues" on page 7-67 |

## Troubleshooting Tips and Techniques

Simscape simulations can stop before completion with one or more error messages. This section discusses generic error types and error-fixing strategies. You might find the previous section, "How Simscape Simulation Works" on page 7-6, useful for identifying and tracing errors.

If a simulation failed:

- Review the model configuration. If your error message contains a list of blocks, look at these blocks first. Also look for:

  - Wrong connections — Verify that the model makes sense as a physical system. For example, look for actuators connected against each other, so that they try to move in opposite directions, or incorrect connections to reference nodes that prevent movement. In electrical circuits, verify polarity and connections to ground.

  - Wrong units — Simscape unit manager offers great flexibility in using physical units. However, you must exercise care in specifying the correct units, especially in the Simulink-PS Converter and PS-Simulink Converter blocks. Start analyzing the circuit by opening all the converter blocks and checking the correctness of specified units.

- Try to simplify the circuit. Unnecessary circuit complexity is the most common cause of simulation errors.

- Break the system into subsystems and test every unit until you are positive that the unit behaves as expected.

- Build the system by gradually increasing its complexity.

It is recommended that you build, simulate, and test your model incrementally. Start with an idealized, simplified model of your system, simulate it, verify that it works the way you expected. Then incrementally make your model more realistic, factoring in effects such as friction loss, motor shaft compliance, hard stops, and the other things that describe real-world phenomena. Simulate and test your model at every incremental step. Use subsystems to capture the model hierarchy, and simulate and test your subsystems separately before testing the whole model configuration. This approach helps you keep your models well organized and makes it easier to troubleshoot them.

## System Configuration Errors

- "Missing Solver Configuration Block" on page 7-65
- "Extra Fluid or Gas Properties Block" on page 7-65
- "Missing Reference Block" on page 7-65

- "Basic Errors in Physical System Representation" on page 7-65

### Missing Solver Configuration Block

Each topologically distinct Simscape block diagram requires exactly one Solver Configuration block to be connected to it. The Solver Configuration block specifies the global environment information and provides parameters for the solver that your model needs before you can begin simulation.

If you get an error message about a missing Solver Configuration block, open the Simscape Utilities library and add the Solver Configuration block anywhere on the circuit.

### Extra Fluid or Gas Properties Block

If your model contains hydraulic elements, each topologically distinct hydraulic circuit in a diagram requires a Custom Hydraulic Fluid block (or Hydraulic Fluid block, available with Simscape Fluids block libraries) to be connected to it. These blocks define the fluid properties that act as global parameters for all the blocks connected to the hydraulic circuit. If no hydraulic fluid block is attached to a loop, the hydraulic blocks in this loop use the default fluid. However, more than one hydraulic fluid block in a loop generates an error.

Similarly, more than one Thermal Liquid Settings (TL) block in a thermal liquid circuit, Two-Phase Fluid Properties (2P) block in a two-phase fluid circuit, or Gas Properties (G) block in a gas circuit generates an error.

If you get an error message about too many domain-specific global parameter blocks attached to the network, look for an extra Hydraulic Fluid block, Custom Hydraulic Fluid block, Thermal Liquid Settings (TL) block, Two-Phase Fluid Properties (2P) block, or Gas Properties (G) block and remove it.

### Missing Reference Block

Simscape libraries contain domain-specific reference blocks, which represent reference points for the conserving ports of the appropriate type. For example, each topologically distinct electrical circuit must contain at least one Electrical Reference block, which represents connection to ground. Similarly, hydraulic conserving ports of all the blocks that are referenced to atmosphere (for example, suction ports of hydraulic pumps, or return ports of valves, cylinders, pipelines, if they are considered directly connected to atmosphere) must be connected to a Hydraulic Reference block, which represents connection to atmospheric pressure. Mechanical translational ports that are rigidly clamped to the frame (ground) must be connected to a Mechanical Translational Reference block, and so on.

If you get an error message about a missing reference block, or node, check your system configuration and add the appropriate reference block based on the rules described above. The missing reference node diagnostic messages include information about the particular block and variable that needs a reference node. This is especially helpful when multiple domains are involved in the model. For more information and examples of best modeling practices, see "Grounding Rules" on page 1-28.

### Basic Errors in Physical System Representation

Physical systems are represented in the Simscape modeling environment as Physical Networks according to the Kirchhoff's generalized circuit laws. Certain model configurations violate these laws and are therefore illegal. There are two broad violations:

- Sources of domain-specific Across variable connected in parallel (for example, voltage sources, hydraulic pressure sources, or velocity sources)

- Sources of domain-specific Through variable connected in series (for example, electric current sources, hydraulic flow rate sources, force or torque sources)

These configurations are impossible in the real world and illegal theoretically. If your model contains such a configuration, upon simulation the solver issues an error followed by a list of blocks, as shown in the following example.

**Example**

The model shown in the following illustration contains two Ideal Translational Velocity Sources connected in parallel. This produces a loop of independent velocity sources, and the solver cannot construct a consistent system of equations for the circuit.
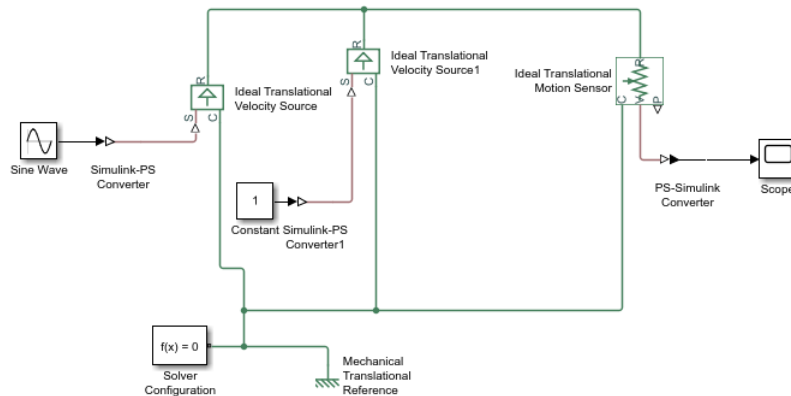


When you try to simulate the model, the solver issues an error message with links to the Ideal Translational Velocity Source and Ideal Translational Velocity Source1 blocks. To fix the circuit, you can either replace the two velocity sources by a single Ideal Translational Velocity Source block, or add a Translational Damper block between them.

## Numerical Simulation Issues

- "Dependent Dynamic States" on page 7-66
- "Parameter Discontinuities" on page 7-67

Numerical simulation issues can be either a result of certain circuit configurations or of parameter discontinuities.

### Dependent Dynamic States

Certain circuit configurations can result in dependent dynamic states, or the so-called higher-index differential algebraic equations (DAEs). Simscape solver can handle dependencies among dynamic states that are linear in the states and independent of time and inputs to the system. For example, capacitors connected in parallel or inductors connected in series will not cause any problems. Other circuit configurations with dependent dynamic states, in certain cases, may slow down the simulation or lead to an error when the solver fails to initialize.

Problems may occur when dynamic states have a nonlinear algebraic relationship. An example is two inertias connected by a nonlinear gear constraint, such as an elliptical gear. In case of simulation failure, the Simscape solver may be able to identify the components involved, and provide an error message with links to the blocks and to the equations within each block.

**Parameter Discontinuities**

Nonlinear parameters, dependent on time or other variables, may also lead to numerical simulation issues as a result of parameter discontinuity. These issues usually manifest themselves at the transient initialization stage (see "Transient Simulation Issues" on page 7-67).

# Initial Conditions Solve Failure

The initial conditions solve, which solves for all system variables (with initial conditions specified on some system variables), may fail. This has several possible causes:

- System configuration error. In this case, the Simulation Diagnostics window usually contains additional, more specific, error messages, such as a missing reference node, or a warning about the component equations, followed by a list of components involved. See "System Configuration Errors" on page 7-64 for more information.

- Dependent dynamic state. In this case, the Simulation Diagnostics window also may contain additional, more specific, error messages, such as a warning about the component equations, followed by a list of components involved. See "Dependent Dynamic States" on page 7-66 for more information.

- The residual tolerance may be too tight to produce a consistent solution to the algebraic constraints at the beginning of simulation. You can try to increase the **Consistency Tolerance** parameter value (that is, relax the tolerance) in the Solver Configuration block.

If the Simulation Diagnostics window has other, more specific, error messages, address them first and try rerunning the simulation. See also "Troubleshooting Tips and Techniques" on page 7-64.

# Transient Simulation Issues

- "Transient Initialization Not Converging" on page 7-67
- "Step-Size-Related Errors — Dependent States — High Stiffness" on page 7-67

Transient initialization happens at the beginning of simulation (after computing the initial conditions) or after a subsequent event, such as a discontinuity (for example, when a hard stop hits the stop). It is performed by fixing all dynamic variables and solving for algebraic variables and derivatives of dynamic variables. The goal of transient initialization is to provide a consistent set of initial conditions for the next transient solve step.

### Transient Initialization Not Converging

Error messages stating that transient initialization failed to converge, or that a set of consistent initial conditions could not be generated, indicate transient initialization issues. They can be a result of parameter discontinuity. Review your model to find the possible sources of discontinuity. See also "Troubleshooting Tips and Techniques" on page 7-64.

You can also try to decrease the **Consistency Tolerance** parameter value (that is, tighten the tolerance) in the Solver Configuration block.

### Step-Size-Related Errors — Dependent States — High Stiffness

A typical step-size-related error message may state that the system is unable to reduce the step size without violating the minimum step size for a certain number of consecutive times. This error message indicates numerical difficulties in solving the Differential Algebraic Equations (DAEs) for the

model. This might be caused by dependent dynamic states (higher-index DAEs) or by the high stiffness of the system. You can try the following:

- Tighten the solver tolerance (decrease the **Relative Tolerance** parameter value in the Configuration Parameters dialog box)

- Specify a value, other than `auto`, for the **Absolute Tolerance** parameter in the Configuration Parameters dialog box. Experiment with this parameter value.

- Tighten the residual tolerance (decrease the **Consistency Tolerance** parameter value in the Solver Configuration block)

- Increase the value of the **Number of consecutive min step size violations allowed** parameter in the Configuration Parameters dialog box (set it to a value greater than the number of consecutive step size violations given in the error message)

- Review the model configuration and try to simplify the circuit, or add small parasitic terms to your circuit to avoid dependent dynamic states. For more information, see "Numerical Simulation Issues" on page 7-66.

# Limitations

## Sample Time and Solver Restrictions

The default sample times of Simscape blocks are continuous. You cannot simulate Simscape blocks with discrete solvers using the default sample times.

If you switch to a local solver in the Solver Configuration block, the states of the associated physical network become discrete. If there are no continuous Simulink or Simscape states anywhere in a model, you are free to use a discrete solver to simulate the model.

You cannot override the sample time of a nonvirtual subsystem containing Simscape blocks.

## Algebraic Loops

A Simscape physical network should not exist within a Simulink algebraic loop. This means that you should not directly connect an output of a PS-Simulink Converter block to an input of a Simulink-PS Converter block of the same physical network.

For example, the following model contains a direct feedthrough between the PS-Simulink Converter block and the Simulink-PS Converter block (highlighted in magenta). To avoid the algebraic loop, you can insert a Transfer Function block anywhere along the highlighted loop.
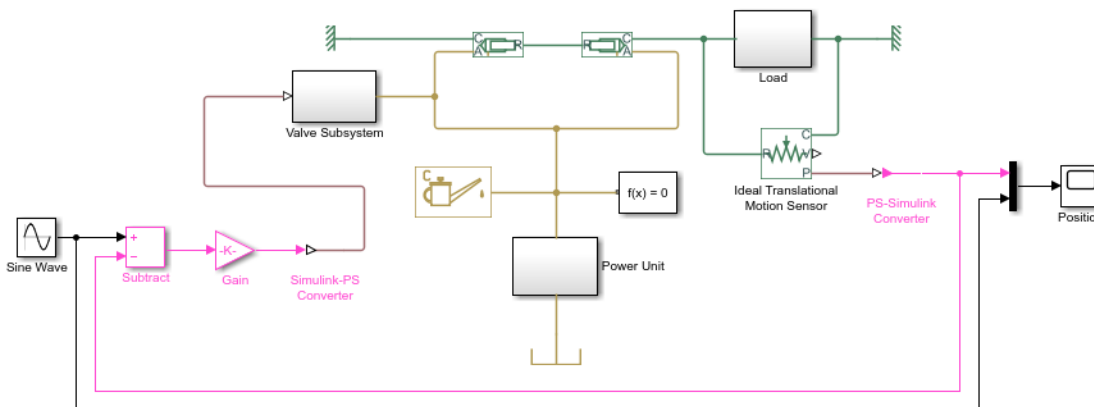


A better way to avoid an algebraic loop without introducing additional dynamics is shown in the modified model below.
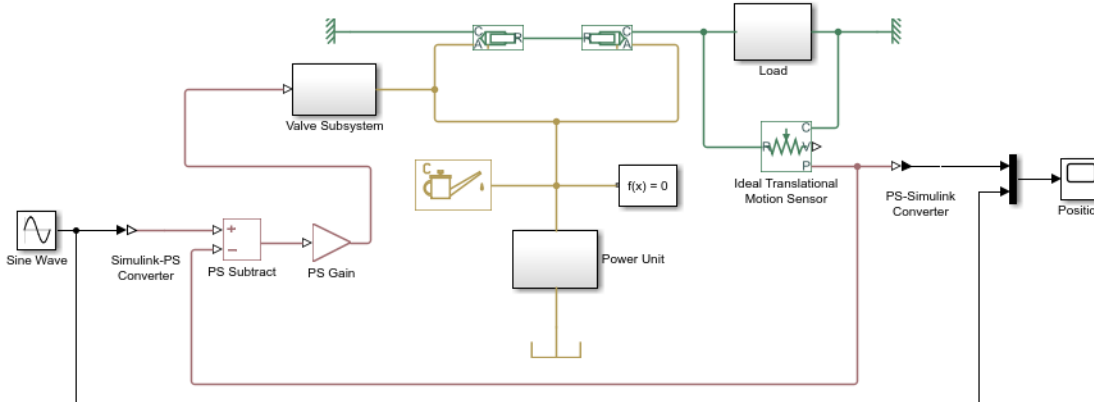
## Unsupported Simulink Tools and Features

Certain Simulink tools and features do not work with Simscape software:

- Exporting a model to a format used by an earlier version (**Simulation > Save > Previous Version**) is not supported for models containing Simscape blocks.

- The Simulink Profiler tool does not work with Simscape models.

- Physical signals and physical connection lines between conserving ports are different from Simulink signals. Therefore, the Viewers and Generators Manager tool and the signal label functionality are not supported.

## Restricted Simulink Tools

Certain Simulink tools are restricted for use with Simscape software:

- You can use the Simulink `set_param` and `get_param` commands to set or get Simscape block parameters, if the parameters correspond to fields in the block dialog box or Property Inspector. It is not recommended that you use these commands to find or change any other block parameters.

  If you make changes to block parameters at the command line, run your model first before saving it. Otherwise, you might save invalid block parameters. Any block parameter changes that you make with `set_param` are not validated unless you run the model.

- Simscape blocks accept `Simulink.Parameter` objects as parameter values in `get_param` and `set_param`, within the restrictions specified here.

- Enabled subsystems can contain Simscape blocks. Always set the **States when enabling** parameter in the Enable dialog to `held` for the subsystem's Enable port.

  Setting **States when enabling** to `reset` is not supported and can lead to fatal simulation errors.

- You can place Simscape blocks within nonvirtual subsystems that support continuous states. Nonvirtual subsystems that support continuous states include Enabled subsystems and Atomic subsystems. However, physical connections and physical signals must not cross nonvirtual boundaries. When placing Simscape blocks in a nonvirtual subsystem, make sure to place all blocks belonging to a given Physical Network in the same nonvirtual subsystem.

- Nonvirtual subsystems that do not support continuous sample time blocks (such as If Action, For Iterator, Function-Call, Triggered, While Iterator, and so on) cannot contain Simscape blocks.

- An atomic subsystem with a user-specified (noninherited) sample time cannot contain Simscape blocks.
- Simulink configurable subsystems work with Simscape blocks only if all of the block modeling options have consistent port signatures.
- When you save a Simulink operating point of a model as a `Simulink.op.ModelOperatingPoint` object, you cannot modify Simscape blocks in the model between saving the `ModelOperatingPoint` object and using it as the initial state for another simulation. For more information, see "Operating Point Behavior".

  You can use Simscape operating points to initialize models containing Simscape blocks. For more information, see "Using Operating Point Data for Model Initialization" on page 8-30.
- For models containing both Simulink and Simscape blocks, exact state restoration between releases is not guaranteed. Updates to Simscape library blocks or algorithms, such as variable elimination, can result in a different number of logged Simulink states than in a previous release, even when you do not change the model content. Therefore, restoring an operating point saved in a previous release by using Simulink operating points, datasets, and similar tools, may lead to a different result.

  For models containing Simscape blocks only, using Simscape operating points provides consistent initialization results in different releases. For more information, see "Using Operating Point Data for Model Initialization" on page 8-30.
- Linearization with the Simulink `linmod` function or with equivalent Simulink Control Design™ functions and graphical interfaces is not supported with Simscape models if you use local solvers.
- Model referencing is supported, with some restrictions:

  - All Physical connection lines must be contained within the referenced model. Such lines cannot cross the boundary of the referenced model subsystem in the referencing model.
  - The referencing model and the referenced model must use the same solver.
  - For protected model references containing Simscape blocks, you cannot run them in accelerator or rapid accelerator mode without a Simscape license.
- You cannot create Simulink signal objects directly on the PS-Simulink Converter block outputs. Insert a Signal Conversion block after the output port of a PS-Simulink Converter block and specify the signal object on the output of the Signal Conversion block instead.
- Simscape run-time parameters are run-to-run tunable. Therefore, for Dashboard blocks linked to Simscape blocks, changing the dials during simulation does not affect the simulation results.

  To use Dashboard blocks for run-to-run tuning of Simscape block parameters, designate the parameter as `Run-time` configurable, associate it with a workspace variable, and link the Dashboard block to the workspace variable. For more information, see "About Simscape Run-Time Parameters" on page 10-2 .

## Simulink Tools Not Compatible with Simscape Blocks

Some Simulink tools and features do not work with Simscape blocks:

- Execution order tags do not appear on Simscape blocks.
- Simscape blocks do not invoke user-defined callbacks.
- You cannot set breakpoints on Simscape blocks.

- Reusable subsystems cannot contain Simscape blocks.
- You cannot use the Simulink Fixed-Point Tool with Simscape blocks.
- The Report Generator reports Simscape block properties incompletely.

## Code Generation

Code generation is supported for Simscape physical modeling software and its family of add-on products. However, there are restrictions on code generated from Simscape models.

- Encapsulated C++ code generation is not supported.
- Tunable parameters are not supported.
- Run-time parameter inlining ignores global exceptions.
- `MaxStackSize` is not supported.
- Simulation of Simscape models on fixed-point processors is not supported.
- Block diagnostics in error messages are not supported. This means that if you get an error message from simulating generated code, it does not contain a list of blocks involved.
- Conversion of models or subsystems containing Simscape blocks to S-functions is not supported.

"Code Generation" describes Simscape code generation features. "Restricted Simulink Tools" on page 7-70 describes limitations on model referencing.

There are variations and exceptions as well in the code generation features of the add-on products based on Simscape platform. For details, see documentation for individual add-on products.

### Code Generation and Fixed-Step Solvers

Most code generation options for Simscape models require the use of fixed-step Simulink solvers. This table summarizes the available solver choices, depending on how you generate code.

| Code Generation Option | Solver Choices |
|---|---|
| Accelerator mode<br>Rapid Accelerator mode | Variable-step or fixed-step |
| Simulink Coder™ software: RSim Target* | Variable-step or fixed-step |
| Simulink Coder software: Targets other than RSim | Fixed-step only |

* For the RSim Target, Simscape software supports only the Simulink solver module. In the model Configuration Parameters dialog box, see the **Code Generation**: **RSim Target**: **Solver selection** menu. The default is automatic selection, which might fail to choose the Simulink solver module.

**8**

# Variable Initialization and Operating Points

# Block-Level Variable Initialization

| **In this section...** |
| --- |
| "Initializing Block Variables for Model Simulation" on page 8-2 |
| "Variable Initialization Priority" on page 8-2 |
| "Suggested Workflow" on page 8-3 |

## Initializing Block Variables for Model Simulation

At the beginning of simulation ($t = 0$), the solver computes the initial conditions to determine the simulation starting point, as described in "Initial Conditions Computation" on page 7-8. Finding a solution means finding initial values for all system variables. You can affect the initial conditions computation by block-level variable initialization, that is, by specifying the priority and target initial values for certain variables using the **Initial Targets** section of the respective block dialog boxes.

The values you specify during block-level variable initialization are not the actual values of the respective variables, but rather their target values at the beginning of simulation ($t = 0$). Depending on the results of the solve, some of these targets may or may not be satisfied.

The solver tries to find a solution that:

- Exactly satisfies all the model equations
- Exactly satisfies all the high-priority targets
- Approximates the low-priority targets as closely as possible (as a result, some of the low-priority targets might be satisfied exactly, the others are approximated)

If the solver cannot find a solution that exactly satisfies all the high-priority targets, it issues a warning and enters the second stage of the solve process, where it tries to find a solution by approximating both the high-priority and the low-priority targets as closely as possible.

If you have selected the **Start simulation from steady state** check box in the Solver block dialog box, the solver attempts to find the steady state (when the system variables are no longer changing with time). If the steady-state solve succeeds, the state found is some steady state (within tolerance), but not necessarily the state expected from the given initial conditions. In other words, if simulation starts from steady state, even the high-priority variable targets might no longer be satisfied at the start of simulation. However, if the model has more than one steady state, the variable targets you specify can affect which steady-state solution is selected by the solver.

After you initialize the block variables and prior to simulating the model, you can open the Variable Viewer to see which of the variable targets have been satisfied. The Variable Viewer displays the actual initial values of the variables obtained as a result of the solve, along with the variable target values, priority, and other information about the variable. For details, see "Variable Viewer" on page 8-18.

## Variable Initialization Priority

During block-level variable initialization, you specify the variable beginning value, unit, and the initialization priority. The priority can be one of the following:

- None — If a variable has priority of none, the initialization algorithm starts at the beginning value for this variable but does not remember this value as it finds the solution for the system of equations. The solver does not try to satisfy any specific initial value for a variable with no priority.

- Low — If a variable has low priority, the beginning value becomes a target for the algorithm and the algorithm tries to stay close to the target. The solver tries to approximate the target value of this variable as closely as possible when finding a solution. Depending on the results of the solve for high-priority variables, some of the low-priority targets might be met exactly, the others are approximated.

- High — If a variable has high priority, the beginning value becomes a target for the algorithm and the algorithm tries to meet the target exactly. The solver tries to find a solution where the actual initial values of all high-priority variables exactly satisfy their target values.

The default initialization priority, beginning value, and unit for each of the block variables come from the underlying Simscape component file. For each individual block in your model, you can override these default settings by opening the **Initial Targets** section of the block dialog box, selecting the check box next to a variable name and specifying your own values for that variable.

When you specify too many high-priority targets for system variables, it is possible to over-specify your model. In this case, the solver might not be able to find a solution that exactly satisfies all the high-priority targets, or even fail to find a solution altogether. For an example of how you can deal with over-specification by using the Variable Viewer and changing the variable priority and targets, see "Initialize Variables for a Mass-Spring-Damper System" on page 8-6.

For detailed information on how to specify variable priority and targets in block dialog boxes, see "Set Priority and Initial Target for Block Variables" on page 8-4.

## Suggested Workflow

1   Using the **Initial Targets** section of the block dialog box, specify the variable targets for initialization, by setting the priority, target values, and units for block variables as required by your model.

2   Open and refresh the Variable Viewer to see which of the initial targets have been satisfied. Although the viewer does not simulate the model, it runs the simulation for 0 seconds to initialize it, and therefore the model must be in an executable state.

3   If initialization fails, or you are not satisfied with the results, iterate by changing the block variable target values and priority, then refreshing the viewer.

4   When satisfied with initialization, run the simulation to see the results.

## See Also

## More About

- "Set Priority and Initial Target for Block Variables" on page 8-4
- "Initialize Variables for a Mass-Spring-Damper System" on page 8-6
- "Variable Viewer" on page 8-18

# Set Priority and Initial Target for Block Variables

When you open the **Initial Targets** section of the block dialog box, it lists all the public variables specified in the underlying component file, along with priority, beginning (target) value, and unit. For example, if you add a Translational Spring block to your model, double-click it to open its dialog box, and then expand the **Initial Targets** section, it looks like this:
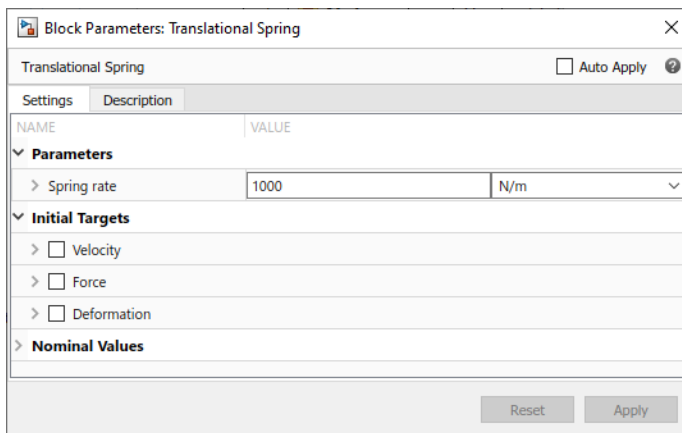


Expanding the widget for each of these variables shows the variable initialization priority, initial target value, and unit.
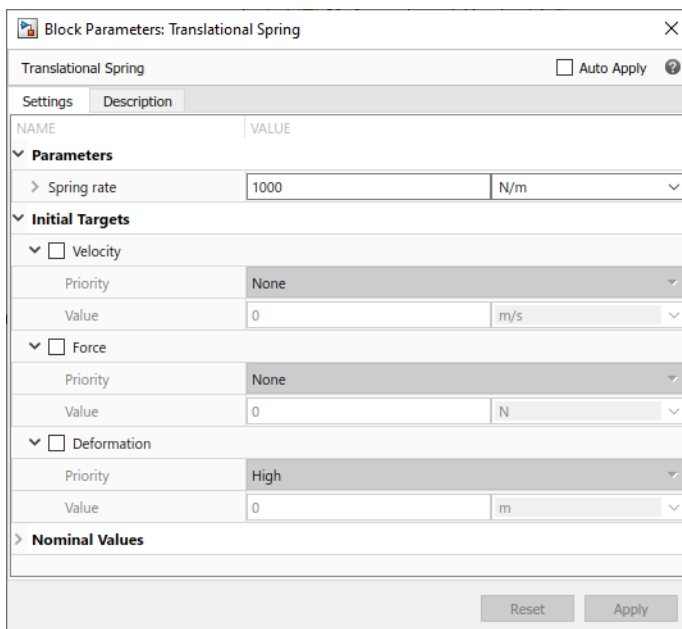


For details on these variables and their usage in the block equations, click the **Source code** link on the **Description** tab of the block dialog box to view the underlying Simscape source file.

**Note** The **Source code** link is available for all the Foundation library blocks that have **Initial Targets** settings. Blocks from the add-on products, like Simscape Electrical or Simscape Fluids, do

not have a **Source code** link in the block dialog box. See the block reference page for information on relevant equations and specific initialization considerations.

To specify the initial deformation of the spring, select the check box next to the **Deformation** variable, to indicate that you are overriding the default values. Select the initialization priority for the variable, by setting its **Priority** drop-down to `High`, `Low`, or `None`. Type a new number into the **Value** field and change the unit, if desired. The unit drop-down lists contains all the units defined in the unit registry that are commensurate with the one specified in the variable declaration. In the following dialog box, **Deformation** is specified as a high-priority variable with the initial target of 20 mm.



If you clear the check box next to a variable name, its **Priority** and **Value** fields switch back to defaults specified in the component file. However, if you select the check box again, these fields will retain their last specified value for when they were overridden.

**Note** The variables included in the **Initial Targets** settings are run-time configurable by default. You can tune a block-level variable-initialization target value between simulation runs if you specify the target value using a variable that you save to the MATLAB workspace.

For more information, see "Run-Time Configurability for Block-Level Variable Initialization Target Values" on page 10-3.

## See Also

## More About

- "Initialize Variables for a Mass-Spring-Damper System" on page 8-6
- "Block-Level Variable Initialization" on page 8-2

# Initialize Variables for a Mass-Spring-Damper System

This example shows how you can use block variable initialization, and how it affects the simulation results of a simple mechanical system.

The model is a classical unforced mass-spring-damper system, with the oscillations of the mass caused by the initial deformation of the spring.
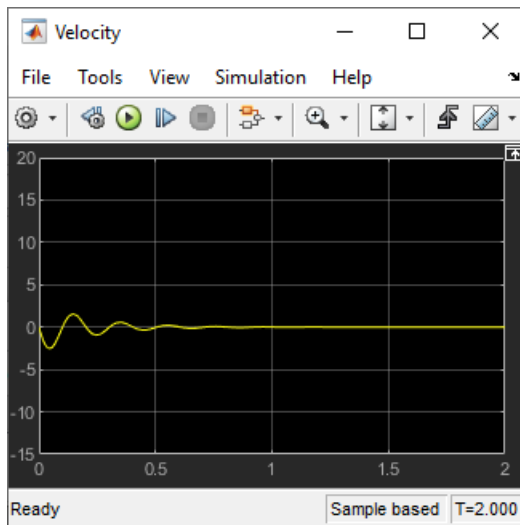
**Create and Set Up the Model**

1   Create a simple mass-spring-damper system. Use the Mass, Translational Spring, Translational Damper, Mechanical Translational Reference, Ideal Translational Motion Sensor, PS-Simulink Converter, Solver Configuration, and Scope blocks, and connect them as shown in the following illustration.
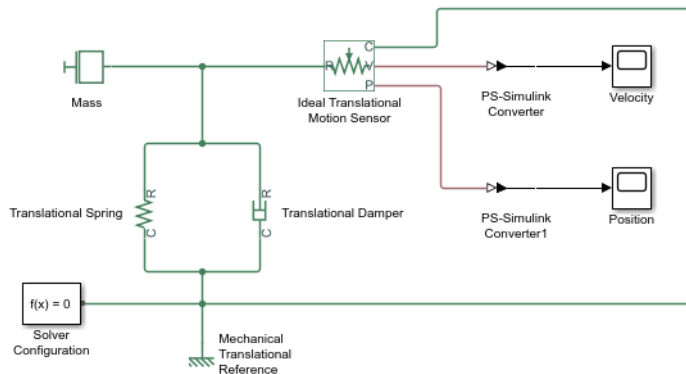


2   In the Translational Damper block dialog box, set the **Damping coefficient** parameter to `10 N/(m/s)`. Use the default parameter values for all of the other blocks.

3   Prepare the model for simulation. In the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens, showing the **Solver** pane. Set **Solver** to `ode23t (mod.stiff/Trapezoidal)` and **Max step size** to `0.2`. Also adjust the **Simulation time** to be between 0 and 2 seconds, by setting **Stop time** to `2.0`.

4   Specify the initial deformation of the spring. Double-click the Translational Spring block. In the block dialog box, expand the **Initial Targets** section, and then select the check box next to the **Deformation** variable. Keep its **Priority** as `High`. Change the **Value** to `0.1`, leaving the unit unchanged as `m`.
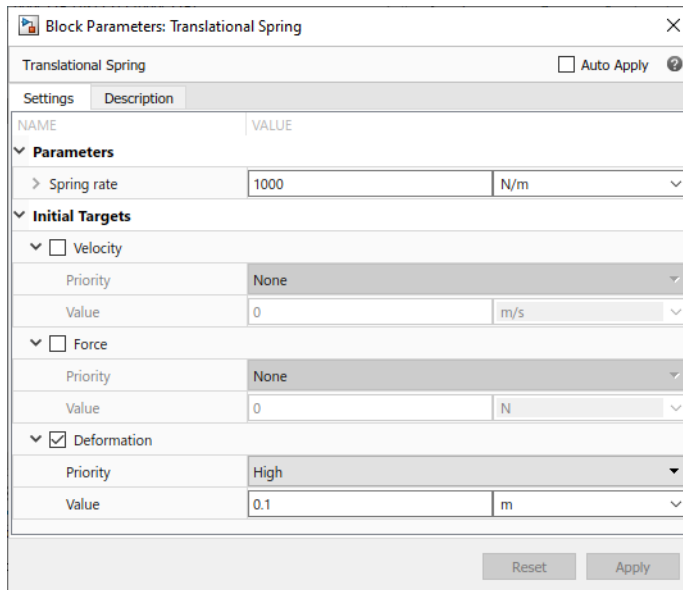
**5** Adjust the initial position of the sensor, to compensate for the spring deformation. Double-click the Ideal Translational Motion Sensor block and set its **Initial position** parameter value to `0.1` `m` as well. This way, when you simulate the model, mass oscillations center around 0.

**6** Simulate the model.

7   Open the Variable Viewer. In the model window, on the **Debug** tab, click **Simscape > Variable Viewer**.



The Translational Spring variable *x*, in the bottom row, has high priority and the target value of 0.1 m. This is the **Deformation** variable that you have just set up in the block dialog box. Its actual start value matches its target value, and therefore its **Status** column displays a green circle.

The two other high-priority variables in this model are the position, *x*, of the Ideal Translational Motion Sensor block and the velocity, *v*, of the Mass block. Both of these variables are set to high

priority inside the component file because the high priority is necessary for the correct operation of the block. For both of these variables, the actual start value also matches the target value, and the **Status** column also displays a green circle.

The rest of the variables in the model do not have initialization priority specified, therefore their **Status** column also displays green circles. The overall status at the bottom of the Variable Viewer window displays a green circle as well, and says that all the targets are satisfied.

**Change Initialization Targets**

You can now see how specifying different variable targets affects system initialization and simulation results.

1   Change the initial velocity of the mass. Double-click the Mass block, expand the **Initial Targets** section, select the check box next to the **Velocity** variable, keep its **Priority** as High, and enter a beginning value of 10, keeping the unit as m/s.
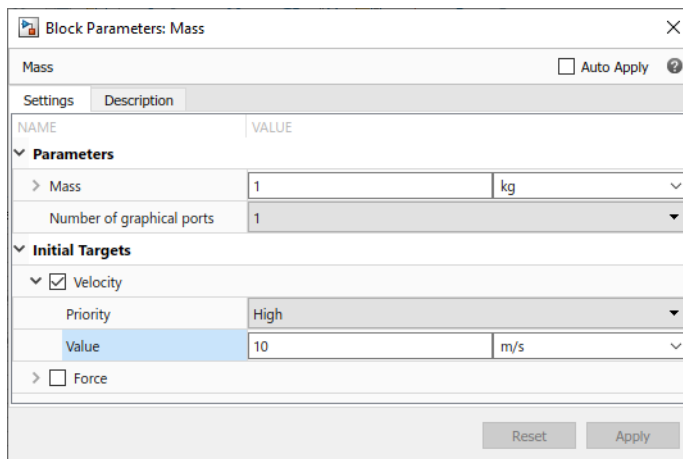


When you change variable priorities and targets or adjust the block parameters, the results in the Variable Viewer are not updated automatically. Instead, the **Update** button displays a warning symbol (exclamation point in a yellow circle), and the timestamp at the bottom of the viewer window indicates that the data in the viewer does not reflect the latest model changes.

**2** Refresh the Variable Viewer by clicking **Update**.

You can see that the solver has found a different initial solution, which satisfies your variable targets for spring deformation and mass velocity. The **Status** column displays green circles, and the overall status at the bottom of the Variable Viewer window also displays a green circle and says that all the targets are satisfied.

**3**   Notice that when you refreshed the Variable Viewer, the scopes turned blank. This happens because solver runs the simulation for 0 seconds to find the initial solution and display it in the Variable Viewer.

Rerun the simulation and examine the Velocity and Position scope windows, to see the effect of the new initial value for mass velocity on the simulation results.
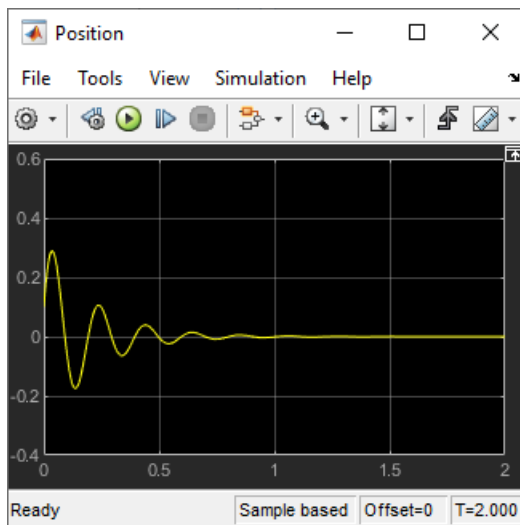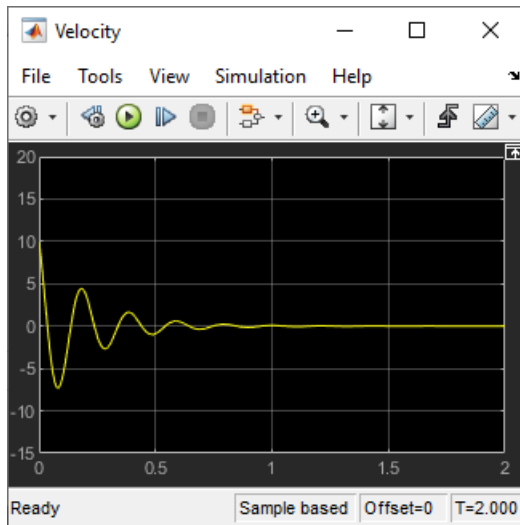




**Deal with Over-Specification**

As you specify additional variable targets, sometimes it is possible to over-specify the constraints.

**1**   Double-click the Translational Damper block, expand the **Initial Targets** section, select the check box next to the **Force** variable, change its **Priority** to High, and enter a value of 200. Keep the unit N.

**2** Refresh the Variable Viewer.



The overall status at the bottom of the Variable Viewer window now displays a red square and says that the solver is unable to satisfy all the high-priority variable targets. There are red squares in the **Status** column for the two high-priority variables with targets not satisfied.

Notice that the solver has been able to find a solution for model initialization. If you rerun the simulation, it runs without errors and you can see the new simulation results.
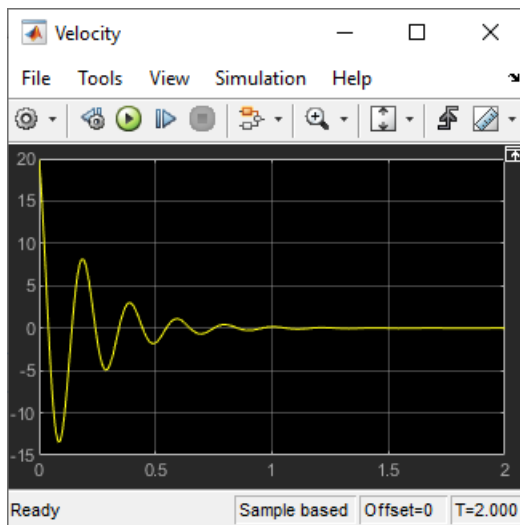
However, the Variable Viewer shows that the model initialization solution does not satisfy your target values for block variables. This happens because placing high-priority constraints on all three elements of the mass-spring-damper system results in a conflict. You can resolve the over-specification issue by relaxing the priority of some of the conflicting variable targets.

**3** Double-click the Translational Damper block again and change the priority of the **Force** variable to Low.

**4** Refresh the Variable Viewer.



The overall status at the bottom of the Variable Viewer window now displays a yellow triangle and says that all the high-priority targets are satisfied, but some of the low-priority targets are not satisfied. There are now a yellow triangle in the status column for the Translational Damper low-priority force variable $f$.

Essentially, the solution found in this case is the same as when you previously specified high-priority target for the mass velocity on page 8-0   , and the simulation results are the same.
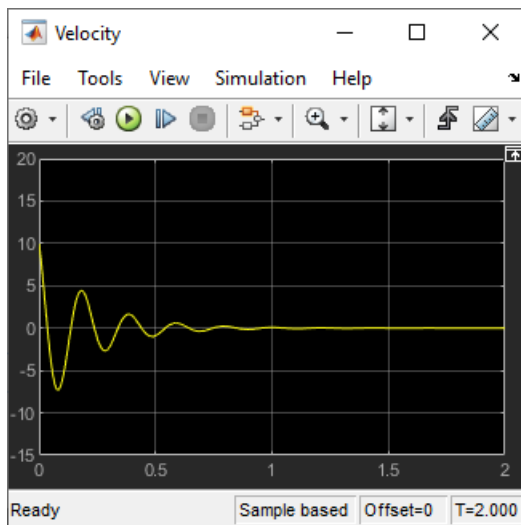
**5**  Another way to deal with over-specification is to keep the high priority on the damper force and relax the priority on mass initial velocity. Double-click the Translational Damper block again and change the priority of the **Force** variable back to `High`. Then double-click the Mass block and change the priority of the **Velocity** variable to `Low`.

**6** Refresh the Variable Viewer.



Again, the Variable Viewer status says that all the high-priority targets have been satisfied and that some of the low-priority targets are not satisfied. However, because you changed the variable priorities, the solver now tried to satisfy the initial force on the damper rather than the mass velocity, and the solution is different in this case, as are the simulation results.
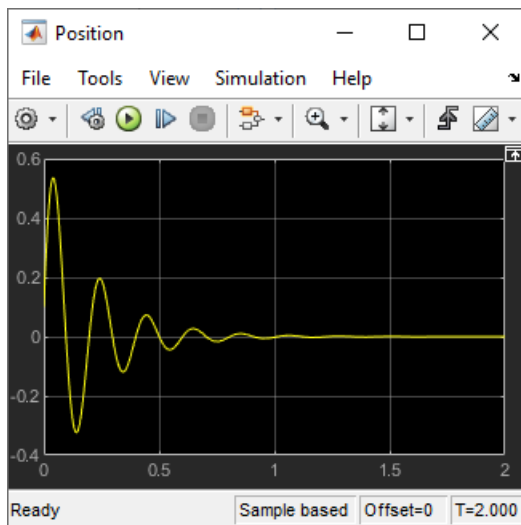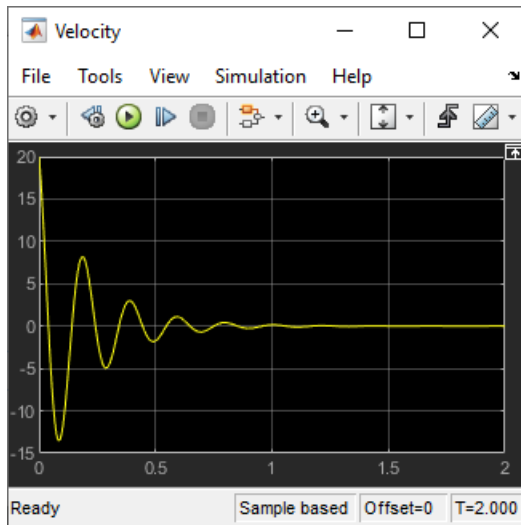
## See Also

### More About

- "Block-Level Variable Initialization" on page 8-2
- "Set Priority and Initial Target for Block Variables" on page 8-4
- "Variable Viewer" on page 8-18

# Variable Viewer

## About Variable Viewer

Prior to simulating the model, you can use the Variable Viewer to check the results of the initial conditions computation for the model and to see which of the block-level variable initialization targets have been satisfied. The Variable Viewer displays the variable priority and target values, where specified, along with the actual initial values for all the variables obtained as a result of the solve.

To open the Variable Viewer, in the model window, on the **Debug** tab, click **Simscape > Variable Viewer**.

---

**Note** If you open a model, and then open the Variable Viewer before simulating the model, then the

viewer does not contain any data. The **Update** button displays a warning symbol ( ), and a message at the top of the viewer window tells you to click the **Update** button to populate the viewer with data.

---
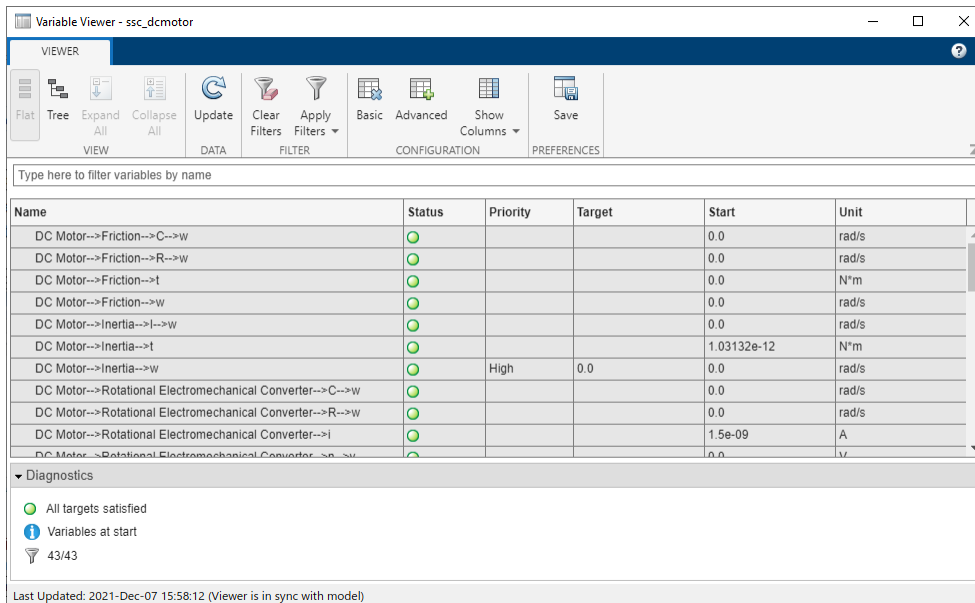
The Variable Viewer is a table, its rows listing all the blocks in the model and all the public variables under each block, and the columns providing the initialization status, priority, target and actual start values, and other information for each variable.

By default, the Variable Viewer opens in basic configuration, unless you specified another configuration as a preferred one. (For information on specifying a preferred configuration, see "Saving Viewer Configuration" on page 8-26.) In basic configuration, the Variable Viewer has the following columns:

| Name | Description |
|------|-------------|
| **Status** | Initialization status of each variable, can be one of:<br><br>• Green circle — Displayed for variables with initialization targets satisfied, and also for all variables with no initialization priority.<br><br>• Yellow triangle — Displayed for low-priority variables if the target is not satisfied.<br><br>• Red square — Displayed for high-priority variables if the target is not satisfied.<br><br>• Red cross — If initial condition solve fails, displayed for variables that could not be initialized.<br><br>• Gray rectangle — Displayed when status is not available. This can happen, for example, if model initialization failed, or if the viewer was left open during diagram update. For more information, see "Interaction with Model Updates and Simulation" on page 8-28. |
| **Priority** | Variable initialization priority, as specified in the block dialog box or in the underlying component file. For more information, see "Set Priority and Initial Target for Block Variables" on page 8-4 and "Variable Priority for Model Initialization". If the variable has no initialization priority (`None` or `priority.none`), then this field is empty. |

| Name | Description |
|------|-------------|
| **Target** | Initial target value for a high-priority or low-priority variable. If the variable has no initialization priority, then this field is empty. |
| **Start** | The actual initial value of the variable computed by the solver. |
| **Unit** | The variable base unit, common for all the values (**Target**, **Prestart**, and **Start**). Simscape unit manager automatically converts all the values as needed. For example, if you specified the target **Beginning Value** in the block dialog box as 20 and the **Unit** as mm, the Variable Viewer displays the **Target** as 0.2 and **Unit** as m. |

The Variable Viewer toolbar buttons perform the following actions:

Displays the data in the Variable Viewer in flat view, to minimize the number of rows in the table. In flat view, the rows for parent nodes are not shown, and the table contains just one row per variable, with the **Name** column including the complete path to the variable from the model root. If the Variable Viewer is in flat view, the buttons that expand and collapse nodes are disabled. This is the default view.

Displays the data in the Variable Viewer in tree view, with variable nodes grouped under the parent port, block, and subsystem nodes. By default, all nodes are collapsed. You can expand them individually or use the **Expand All** button.

Expands all nodes, showing all variables under each block name. This button is available only if the Variable Viewer is in tree view.

Collapses all variables under each block name. You can then expand the block nodes individually to see the variables under this block. This button is available only if the Variable Viewer is in tree view.

Recomputes the initial conditions for the model and refreshes the values displayed in the viewer. Use this button after adjusting the block parameter values, changing variable priorities and targets, or updating the block diagram. If the data in the Variable Viewer is out of sync with

the model, the **Update** button displays a warning symbol ( ), and the timestamp at the bottom of the viewer window turns red. For more information, see "Interaction with Model Updates and Simulation" on page 8-28.

Lets you apply filtering options. For more information, see "Useful Filtering Techniques" on page 8-26.

Clears all the filtering options. For more information, see "Useful Filtering Techniques" on page 8-26.

**Clear Filters**

Shows the Variable Viewer in its default, basic, configuration, with only the following columns displayed: **Status**, **Priority**, **Target**, **Start**, and **Unit**.

**Basic**

Shows the Variable Viewer in advanced configuration, with all the columns displayed. Use this view for troubleshooting your model, for example, if the model initialization failed.

**Advanced**

Lets you hide and show columns to create a customized viewer configuration.

**Show Columns ▼**
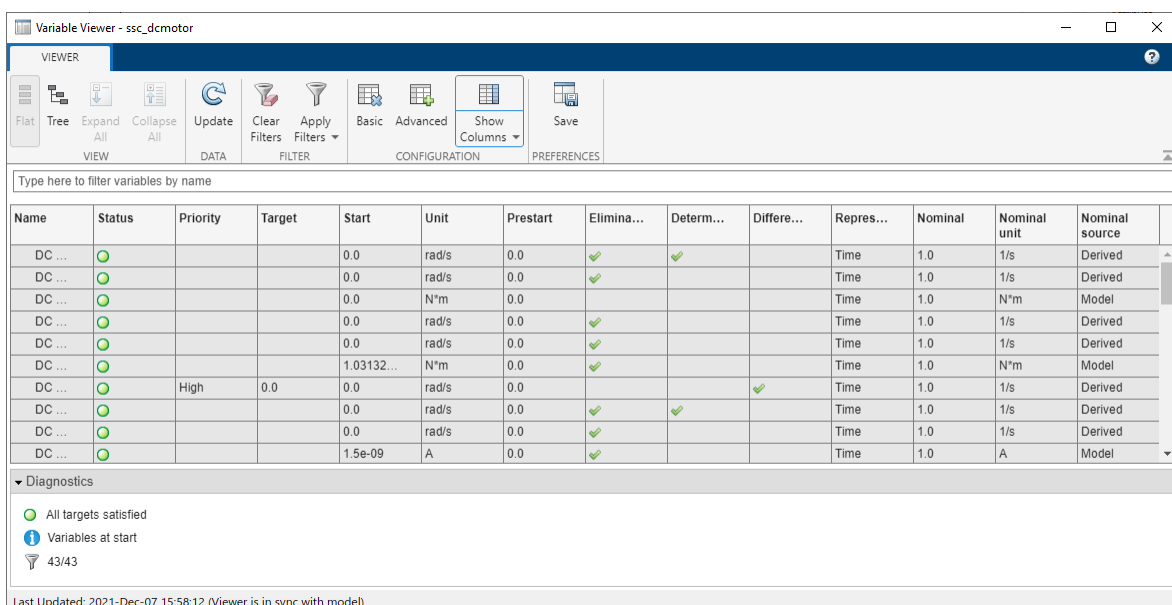
Saves the current Variable Viewer configuration. For more information, see "Saving Viewer Configuration" on page 8-26.

**Save**

## Advanced Configuration

In most cases, the default Variable Viewer configuration contains sufficient data for viewing the variable targets and verifying the model initialization results. However, if the solver is unable to satisfy all the high-priority variable targets, or if the model initialization fails, the advanced Variable Viewer configuration might provide additional data that can help you troubleshoot your model.

To switch to the advanced configuration, click **Advanced** in the Variable Viewer toolbar.

In advanced configuration, the Variable Viewer displays the following additional columns:

| Name | Description |
|---|---|
| **Prestart** | The value of the variable that the solver uses at the beginning of the initial conditions solve process. For variables with no override of initialization priority and targets, the prestart values come from the variable declaration in the underlying component file. If the initialization process fails, these values can help you determine the reason (for example, a prestart value of 0 for a variable used as a denominator in a model equation). If a variable has an undesirable prestart value, specify a better value as a low-priority (or no-priority) initialization target, to make the solver start iterations from a different point. |
| **Eliminated** | These variables are eliminated by the software prior to numerical integration and are not used in solving the system. Prestart values for these variables have no effect on the system solution. However, you can set the initialization priority and targets on these variables, in which case their targets will be represented in terms of the variables that are retained by the solver. |
| **Determined** | The values of these variables depend on the system inputs, or their values are predetermined based on the analysis of equations. Therefore, specifying initialization priority and targets for these variables has little or no impact on system solution. Also, if you specify a high-priority target for a predetermined variable, the solver most likely will not be able to satisfy this target but will spend extra time trying to find a second-stage solution. |
| **Differential** | Time derivatives of these variables appear in equations. These variables add dynamics to the system and can produce independent states. Therefore, these variables are more likely to require high initialization priority. |
| **Representation** | If frequency-and-time simulation mode is turned on, indicates how the solver marks the variables: **Frequency** ("fast") or **Time** ("slow"). For more information, see "Frequency and Time Simulation Mode" on page 7-55. In regular simulation, all variables are marked **Time**. |
| **Nominal** | Nominal value of the variable. For more information, see "System Scaling by Nominal Values" on page 7-31. |
| **Nominal unit** | Physical unit associated with the nominal value of the variable. For more information, see "System Scaling by Nominal Values" on page 7-31. |
| **Nominal source** | Source of the nominal value and unit: **Block**, **Model**, **Derived**, or **Fixed**. For more information, see "Possible Sources of Nominal Values and Their Evaluation Order" on page 7-31. |

You can customize the viewer configuration by clicking **Show Columns** in the Variable Viewer toolbar and selecting which columns to show.

Clicking **Basic** or **Advanced** in the Variable Viewer toolbar restores the default basic or advanced layout, respectively.

## Switching Between Flat View and Tree View

You can control the number of rows in the Variable Viewer by switching between the flat view (the default) and the tree view. In tree view, the variable nodes are grouped under the parent port, block, and subsystem nodes. Therefore, the Variable Viewer table contains the rows for the parent nodes (ports, blocks, and subsystems) in addition to the rows that correspond to all the public variables.

Only the rows that represent variables contain data such as targets and actual values. All rows display a status, with the status of a parent node being determined by the status of its children variables: if all the children are green, then the row for the parent node also displays a green circle in its **Status** column.

For example, in the Variable Viewer table below, the first row represents the Ideal Translational Motion Sensor block, the second row — port **C** of this block, and only the third row contains the data for the actual variable v (velocity at port **C**).



In flat view, the rows for parent nodes are not shown, and the table contains just one row per variable, with the **Name** column including the complete path to the variable from the top-level model. For example, the first row of the Variable Viewer table in flat view represents the same variable v (velocity at port **C** of the Ideal Translational Motion Sensor block), and the **Name** column includes the names of its parents and shows the path to the variable. Flat view makes the Variable Viewer table more compact.
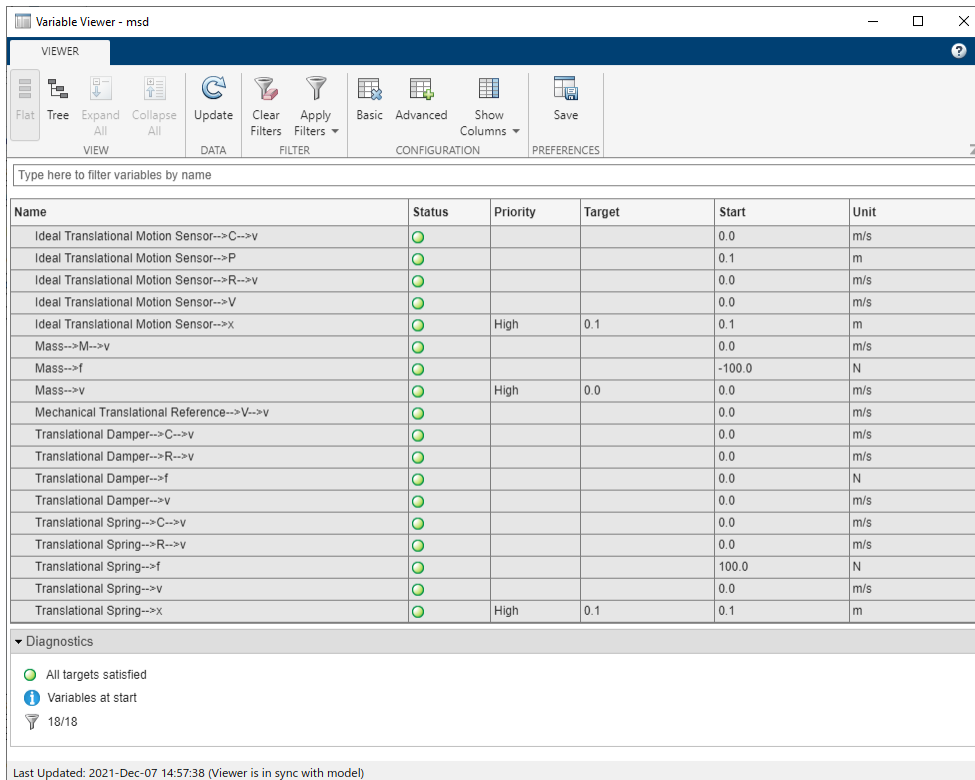
To switch to the tree view, click **Tree** in the Variable Viewer toolbar. In tree view, with variable nodes grouped under the parent port, block, and subsystem nodes.
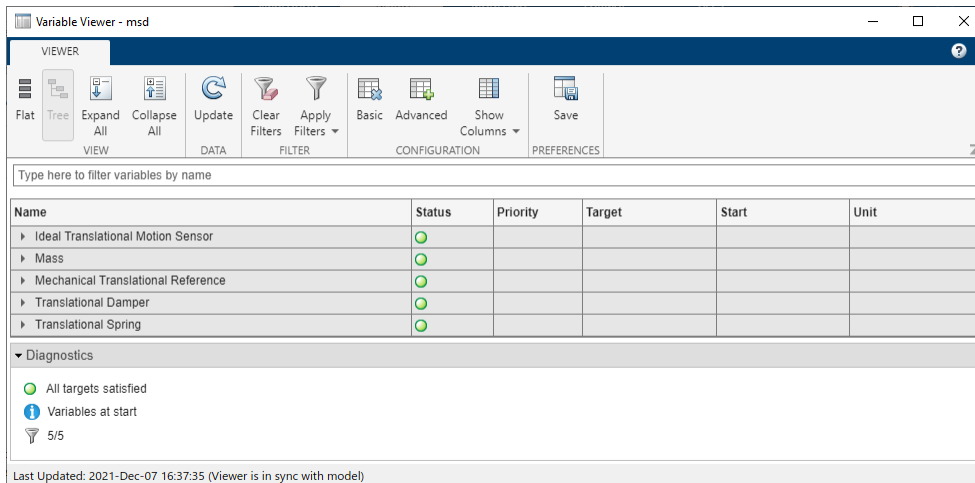


By default, all nodes are collapsed. You can expand them individually or use the **Expand All** button.

To switch back to the flat view, click **Flat** in the Variable Viewer toolbar.

## Component Array Representation in Variable Viewer

When your model contains blocks with underlying arrays of components, Variable Viewer includes variables that belong to array members.

For example, in the Variable Viewer table below, the custom Resistor Array block contains an underlying array of resistors. In tree view, the Variable Viewer table contains the nodes n and p, corresponding to the ports of the Resistor Array block itself, each with its variable v. Then the Variable Viewer table contains the tree nodes for each of the array members, numbered `resistor(1)`, `resistor(2)`, and so on. Each of these numbered nodes, in turn, contains rows that correspond to the nodes and variables of the underlying resistor component. Only the rows that represent variables contain data such as targets and actual values.



If the component array size is `1xN`, the members are numbered `comp(1)`, …, `comp(N)`. If the array size is `NxM`, the members are numbered `comp(1,1)`, `comp(1,2)`, …, `comp(NxM)`.

Flat view makes the Variable Viewer table more compact. This is how the same array of resistors looks in the flat view. The table contains just one row per variable, with the **Name** column including the complete path to the variable from the top-level model. For variables that belong to the members of the component array, the path to the variable contains the numbered component name.

## Useful Filtering Techniques

The **Apply Filters** in the Variable Viewer toolbar lets you filter the table rows based on their values.

For example, filtering on the **Priority** column values (selecting only the check boxes for `HIGH` and `LOW`) lets you view all the targets and actual values in a compact format, which can be helpful for a large model.

You might also find the following filtering techniques useful in troubleshooting your models:

- Filter the **Differential** column on `TRUE`, to display only the rows for differential variables. Time derivatives of these variables appear in equations. These variables add dynamics to the system and can produce independent states, therefore these variables are more likely to require high initialization priority.

- Filter the **Determined** column on `TRUE`, to verify that these variables have no initialization priority. The values of these variables are either predetermined by the equation analysis or depend on the system inputs, and therefore specifying initialization priority and targets for these variables has little or no effect on model initialization.

To clear all filters, click **Clear Filters** in the Variable Viewer toolbar.

## Saving Viewer Configuration

The **Save** button in the Variable Viewer toolbar lets you save the following configuration preferences:

- Variable Viewer view type (tree or flat)
- Visible columns
- Ordering of columns
- Filters applied for all columns (both visible and hidden)
- Sorting on a specific column

If you save viewer configuration, then the next time you open Variable Viewer, for this or another model, it will open with the same configuration. This behavior is consistent with saving other MATLAB preferences.

## Link to Block Diagram

The Variable Viewer tool provides direct linking to the block diagram. This link lets you highlight the appropriate block, or easily go from a variable listed in the Variable Viewer to the **Variables** tab in the corresponding block dialog box, to modify the variable priorities and targets.

When you right-click in the **Name** column of any row in the Variable Viewer table, a context menu opens with the following options:

- **Go to block** — Highlights the corresponding block in the block diagram, opening the appropriate subsystem if needed. If the row represents a variable, highlights the parent block for this variable.
- **Open block dialog** — Opens the corresponding block dialog box (for a variable, opens the parent block dialog box). In the block dialog box, expand the **Initial Targets** section to view or modify the variable priorities and targets. If the selected row represents a subsystem, this option is not available.

## Interaction with Model Updates and Simulation

Opening the Variable Viewer does not trigger an automatic update. For complex models, computing initial values for all the variables can last several minutes, and unnecessary updates could lead to loss of productivity. You have to update the data explicitly by clicking the **Update** button.

When you open the Variable Viewer, it gets populated with the data from the last simulation. The status at the bottom of the viewer window displays the timestamp of its last update. If you have modified the model since the viewer has last been updated, the **Update** button displays a warning



symbol ( ), and the timestamp at the bottom of the viewer window indicates that the data in the viewer might not reflect the latest model changes.

If you open a model, and then open the Variable Viewer before simulating the model, then the viewer does not contain any data. The **Update** button displays a warning symbol, and a message at the top of the viewer window tells you to click the **Update** button to populate the viewer with data.

The Variable Viewer computes the actual initial values of the variables by running the simulation for 0 seconds. Therefore:

- The model must be in an executable state when you refresh the viewer, otherwise you get an error message.
- If the scopes are open, they turn blank every time you refresh the viewer. Rerun the simulation to see the new results.
- If you rerun the simulation while the Variable Viewer is open, the results in the viewer are automatically refreshed when the simulation starts running.
- If you change variable priorities and targets or adjust the block parameters, the results in the viewer are not updated automatically. Refresh the viewer, by clicking **Update** in the Variable Viewer toolbar, to compute the new actual values of the variables and update the status.
- If you update block diagram (by selecting **Modeling > Update Model** in the model window) while the Variable Viewer is open, the previously computed actual values become unavailable and the **Status** column displays gray rectangles. The overall status at the bottom of the Variable Viewer window is also not available. Refresh the viewer to compute the new actual values of the variables and update the status.

## See Also

## More About

- "Block-Level Variable Initialization" on page 8-2
- "Initialize Variables for a Mass-Spring-Damper System" on page 8-6

# Using Operating Point Data for Model Initialization

| In this section... |
| --- |
| "Using Operating Points to Initialize Model Variables" on page 8-30 |
| "Suggested Workflow" on page 8-30 |
| "Extracting Variable Initialization Data into an Operating Point" on page 8-30 |
| "Manipulating Operating Point Data" on page 8-31 |
| "Applying Operating Point Data to Initialize Model" on page 8-31 |

## Using Operating Points to Initialize Model Variables

Block-level variable initialization lets you specify the priority and target for individual block variables. You can also initialize variables for a whole model from the saved operating point data.

You can use `OperatingPoint` objects to save sets of data necessary to initialize a model, manipulate this data, and then use it to initialize another model, or the same model before another simulation run. These sets of data contain a hierarchy of variable initialization targets. Each target consists of a variable value, unit, and initialization priority, as described in "Variable Initialization Priority" on page 8-2.

The `OperatingPoint` data hierarchy is a tree, with nodes corresponding to subsystems and blocks in a model. At the lowest level of the data tree, inside the block nodes, are the variable initialization targets for that block.

When you use an `OperatingPoint` to initialize a model, the solver matches the `OperatingPoint` data hierarchy to the model hierarchy and applies the initialization targets from the operating point to the respective model variables. If there is no variable matching an operating point target, this target is ignored. After applying all the data from the operating point, the solver performs model initialization as described in "Initial Conditions Computation" on page 7-8.

After you initialize the variables and prior to simulating the model, you can open the Variable Viewer to see which of the variable targets have been satisfied. For details, see "Variable Viewer" on page 8-18.

## Suggested Workflow

1   Create an `OperatingPoint` object by extracting data from the model or from the simulation log. For more information, see "Extracting Variable Initialization Data into an Operating Point" on page 8-30.

2   Modify the operating point data, if needed, by changing, adding, or removing targets and nodes. For more information, see "Manipulating Operating Point Data" on page 8-31.

3   When satisfied with the operating point data, apply it to initialize another model, or the same model for another simulation run. For more information, see "Applying Operating Point Data to Initialize Model" on page 8-31.

## Extracting Variable Initialization Data into an Operating Point

You can create an `OperatingPoint` object by extracting data from an existing model or from logged simulation data. For more information, see `simscape.op.create`.

You can extract variable initialization targets from a model in these ways:

- Start values — Initialize the model and use the variable targets corresponding to the `Start` values in the Variable Viewer.
- Prestart values — Update the model and use the variable targets corresponding to the `Prestart` values in the Variable Viewer.
- Cached data — Extract cached values of variable targets from a model that has been previously initialized or simulated. You can specify `Start` or `Prestart` values. This method lets you save time by avoiding repeated initialization of the model if the data that you want to extract has not changed.

Alternatively, you can simulate the model while logging simulation data, and then extract variable targets from the simulation log at a specified time, $t$:

- If the set of times recorded in the simulation data log contains an exact match for time $t$, then the `simscape.op.create` function extracts these variable target values into the operating point data.
- If there is no exact match, but $t$ is between the minimum and maximum times in the simulation data log, then the function uses linear interpolation to determine the target values.
- If $t$ is less than the minimum time, then the function extracts the first value for each variable in the simulation data log.
- If $t$ is greater than the maximum time, then the function extracts the last value for each variable in the simulation data log.

When you extract data from a model into an operating point, the elements in the data hierarchy of the `OperatingPoint` object match the structure of the model. The operating point data tree has nodes corresponding to subsystems and blocks in the model, with variable initialization targets for each block at the lowest level of the data tree hierarchy. Similarly, when you extract an operating point from logged simulation data, the operating point data tree matches the data tree of the simulation log. For an example, see "Find Relative Path to Block Node in Operating Point Data Tree".

## Manipulating Operating Point Data

You can create an empty `OperatingPoint` object, or populate it with the data extracted from an existing model or from logged simulation data.

Once you create an `OperatingPoint` object, you can modify it in these ways:

- Add targets one-by-one. For an example, see "Add Element to an Operating Point".
- Copy and insert elements. For an example, see "Copy Element from an Operating Point". You can then insert the copied element into another operating point using the `set` function.
- Remove elements. For an example, see "Remove an Element from Operating Point Data".
- Rename or move elements. For an example, see "Rename Element to Match New Block Name".
- Merge operating points. For an example, see "Merge Two Operating Points".

## Applying Operating Point Data to Initialize Model

To initialize a model from an operating point:

**1**   Open the Configuration Parameters dialog box.

**2**   On the **Simscape** pane, select the **Enable operating point initialization** check box.

**3**   In the **Model operating point** textbox, enter the name of the workspace variable associated with an `OperatingPoint` object.
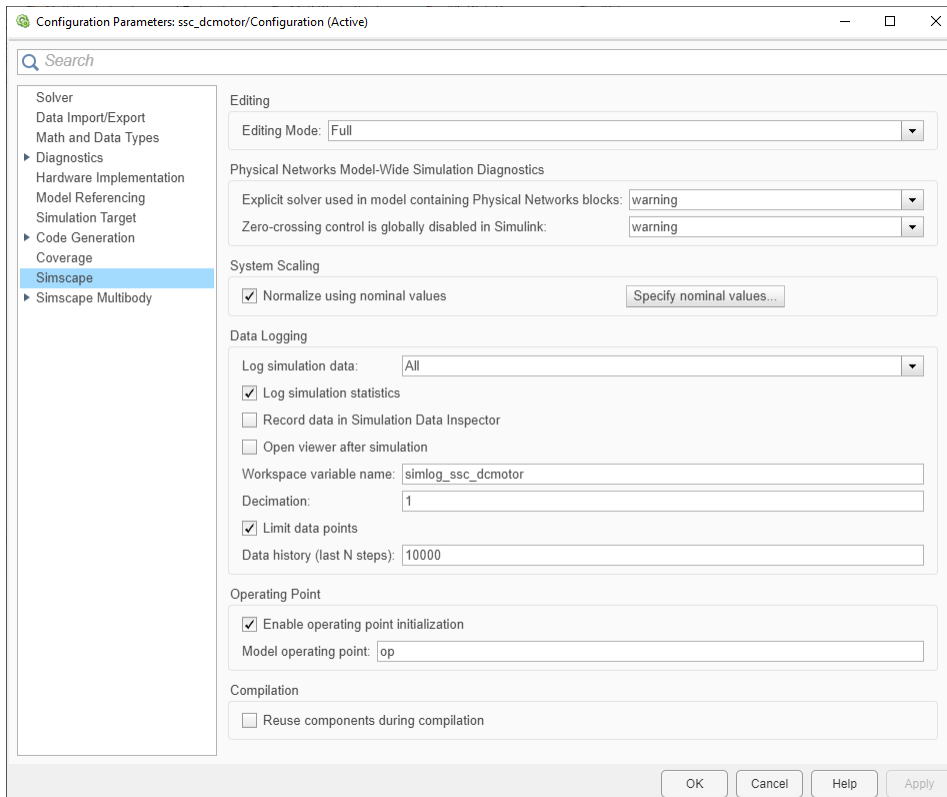


You can also use the equivalent command-line interface to set the model configuration parameters:

- `set_param('model_name','SimscapeUseOperatingPoints','on');`
- `set_param('model_name','SimscapeOperatingPoint','op_name');`

where `model_name` is the name of the model and `op_name` is the name of the `OperatingPoint` object.

## See Also

`simscape.op.OperatingPoint` | `simscape.op.Target`

## More About

- "Initialize Model Using Operating Point from Logged Simulation Data" on page 8-33
- "Variable Viewer" on page 8-18

# Initialize Model Using Operating Point from Logged Simulation Data

This example shows how you can create an `OperatingPoint` object from logged simulation data and then use this operating point to initialize the model for a subsequent simulation run.

1   Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. This model has data logging enabled for the whole model, with the **Workspace variable name** parameter set to `simlog_ssc_dcmotor`.
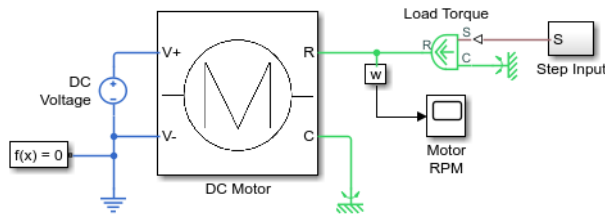


2   Simulate the model to log the simulation data.

3   Examine the simulation results in the Motor RPM scope window.
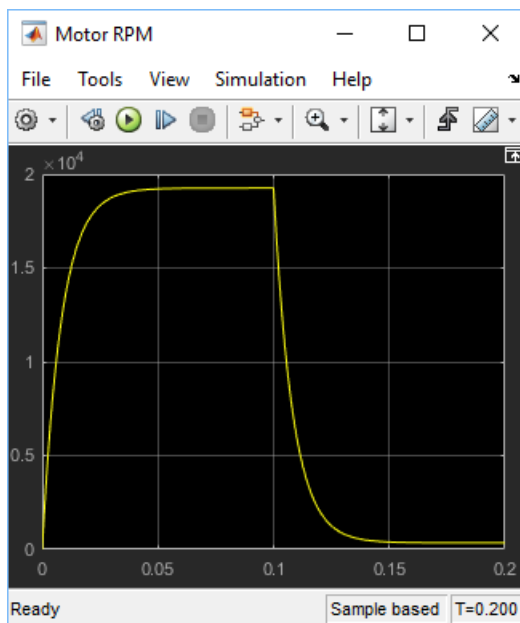


For the first 0.1 seconds, the motor has no external load, and the speed builds up to the no-load value. Then at 0.1 seconds, the stall torque is applied as a load to the motor shaft.

4   Create an operating point from logged simulation data at 0.1 seconds after the start of simulation:

```
op = simscape.op.create(simlog_ssc_dcmotor, 0.1)

op =

  OperatingPoint with children:
```

```
OperatingPoints:

  ChildId              Size
  _____      ____

  'DC Motor'           1x1
  'DC Voltage'         1x1
  'ERef'               1x1
  'Load Torque'        1x1
  'MRRef Motor'        1x1
  'MRRef Torque'       1x1
  'Sensing'            1x1
  'Step Input'         1x1
```

**5** Enable model initialization from operating point:

```
set_param(gcs,'SimscapeUseOperatingPoints','on');
```

This command is equivalent to selecting the **Enable operating point initialization** check box in the **Simscape** pane of the Configuration Parameters dialog box.

**6** Specify the name of operating point:

```
set_param(gcs,'SimscapeOperatingPoint','op');
```

This command is equivalent to entering op in the **Model operating point** textbox.

**7** Simulate the model. The simulation now starts with the full no-load speed.
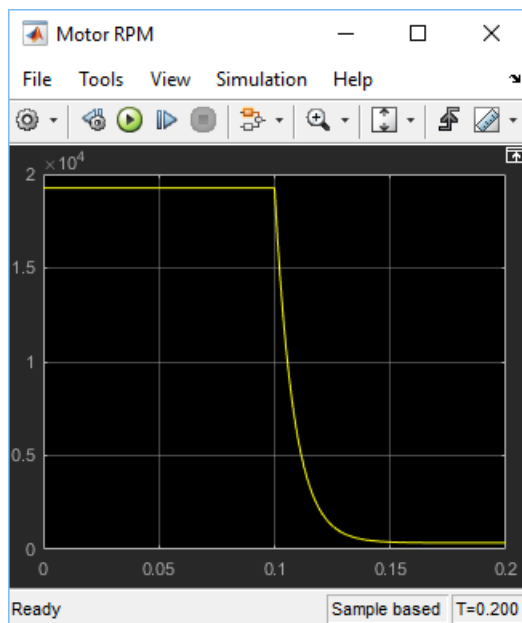


## See Also

## More About

- "Log, Navigate, and Plot Simulation Data" on page 13-20
- "Using Operating Point Data for Model Initialization" on page 8-30

# Indexing into Component Arrays

| In this section... |
| --- |
| "Plotting Logged Simulation Data for Component Array Members" on page 8-35 |
| "Setting Operating Point Targets for Component Array Members" on page 8-37 |

If your model contains blocks with underlying arrays of components, you might want to access individual array members, for example, to set their operating point targets or to plot the logged simulation data for specific nodes. You do this by using command-line interface to index into the array of components and construct the path to a data logging node or an operating point target of the particular member.

The following rules apply:

- Access array members by using the MATLAB matrix indexing techniques. For more information, see "Array Indexing".
- Access scalar elements by regular path (dot-delimited or slash-delimited) indexing.
- A path index cannot cross a nonscalar element.
- Path indexing cannot be combined with matrix indexing.

The last two rules mean that you might need to construct the path in multiple steps by defining intermediate variables, as shown in these examples.

## Plotting Logged Simulation Data for Component Array Members

If your model contains blocks with underlying arrays of components, you can use either Simscape Results Explorer or Simulation Data Inspector to view logged simulation data for individual array members. For more information, see "Data Logging for Component Arrays" on page 13-30.

You can also use command-line interface to index into an array of components, for example, to plot logged simulation data for a particular array member.

Suppose you have a model, `CompArrayExample`, that contains a subsystem, `BatteryPack`, and inside it a block, named `ResistorArray`, with an underlying array of `resistor` components. You want to plot the current through the second resistor in the array.

In the illustration, the Simscape Results Explorer shows the logged simulation data tree structure (in the left pane) and the plot of the current, `i`, through the second resistor (in the right pane).
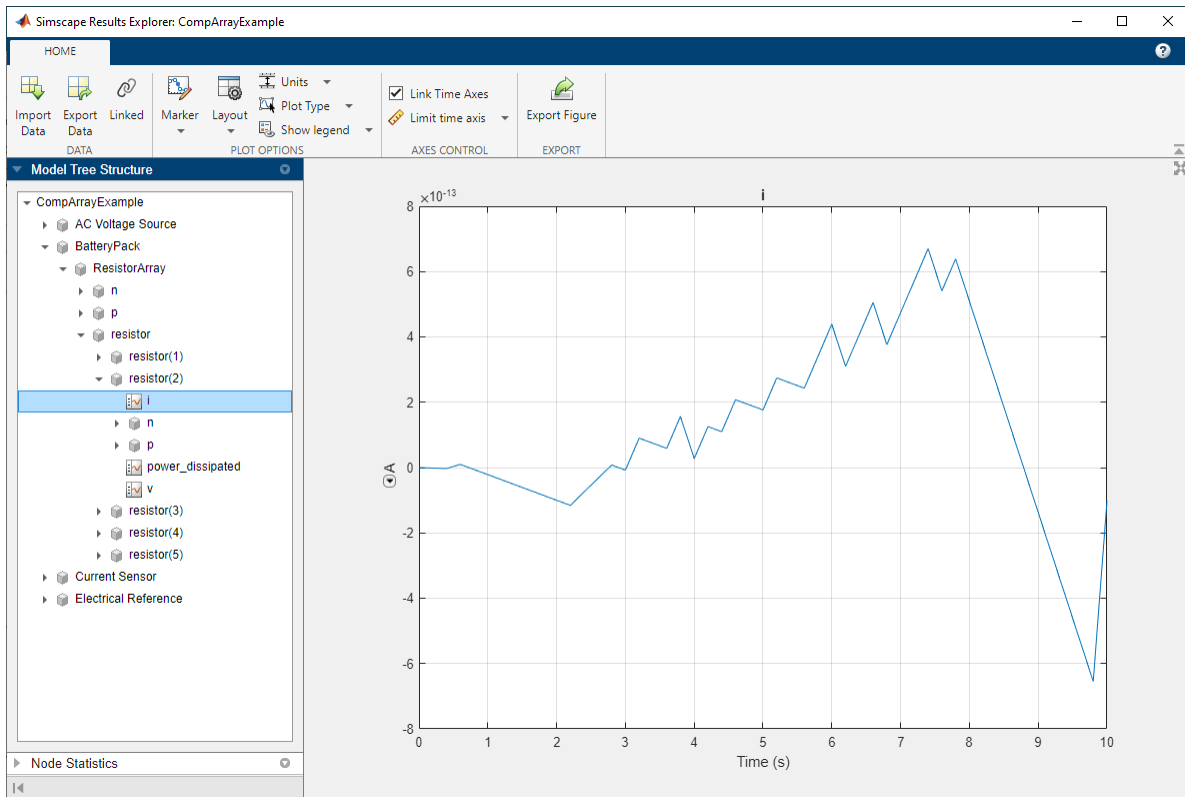
For programmatic access to the same node, you must construct the path to the node using indexing. Because the path index cannot traverse an array, first construct the path to the resistor array by using dot indexing:

```
r = simlog.BatteryPack.ResistorArray.resistor

r =

  1×5 Node array with properties:

    id
    savable
    exportable
```

Next, use matrix indexing to access the second component of the array:

```
r2 = r(2)

r2 =

  Node with properties:

                id: 'resistor'
           savable: 1
        exportable: 0
  power_dissipated: [1×1 simscape.logging.Node]
                 i: [1×1 simscape.logging.Node]
                 v: [1×1 simscape.logging.Node]
                 p: [1×1 simscape.logging.Node]
                 n: [1×1 simscape.logging.Node]
```

Finally, use dot indexing again to access the node for the current, `i`, through the second resistor:

```
i = r2.i

i =

  Node with properties:

            id: 'i'
       savable: 1
    exportable: 0
        series: [1×1 simscape.logging.Series]
```

Plot the node:

```
plot(i)
```
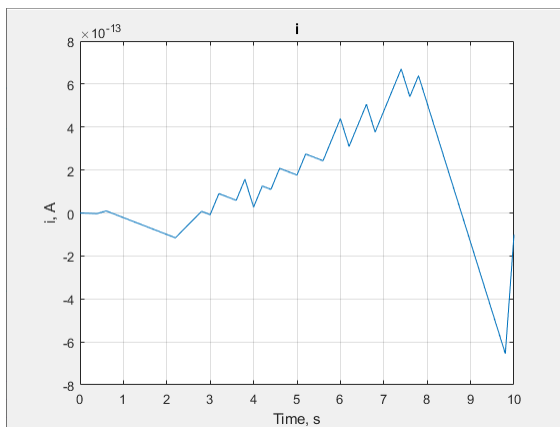


## Setting Operating Point Targets for Component Array Members

For the model discussed in the previous example, "Plotting Logged Simulation Data for Component Array Members" on page 8-35, get and set operating point targets for individual array members.

Create an `OperatingPoint` object named `op` from logged simulation data at 5 seconds after the start of simulation:

```
op = simscape.op.create(simlog, 5)

op =

  OperatingPoint with children:

  OperatingPoints:

   ChildId                 Size
   _____  ____

   'AC Voltage Source'     1x1
   'BatteryPack'           1x1
   'Current Sensor'        1x1
   'Electrical Reference'  1x1
   ----------------------------
```

To set a new operating point target for the voltage, v, at the positive node p of the second resistor in the array, follow a procedure similar to the one described in the previous example, using temporary objects to construct a path through the data tree.

First, extract the operating point data for the array of resistors:

```
r = get(op, 'BatteryPack/ResistorArray/resistor')

r =

  1×5 OperatingPoint array with properties:

    Identifier
    ChildIds
    Children
    Attributes
```

Next, get the target for the voltage. Use matrix indexing to access the second component of the array and then regular slash-delimited path construction from that component to the target:

```
t = get(r(2), 'p/v')

t =

  Target with fields:

         Value: 2.8228e-12 : V
      Priority: None
    Attributes: containers.Map
   Description: Voltage
```

Set a new value for the target:

```
V1 = simscape.Value(2.0000e-12, 'V');
t.Value = V1

t =
  Target with fields:

         Value: 2.0000e-12 : V
      Priority: None
    Attributes: containers.Map
   Description: Voltage
```

Finally, reverse the process to set the new target in the operating point object for the whole model:

```
r(2) = set(r(2), 'p/v', t);
op = set(op, 'BatteryPack/ResistorArray/resistor', r);
```

Initialize the model from the new operating point. Observe that the simulation results have changed compared to the previous example.
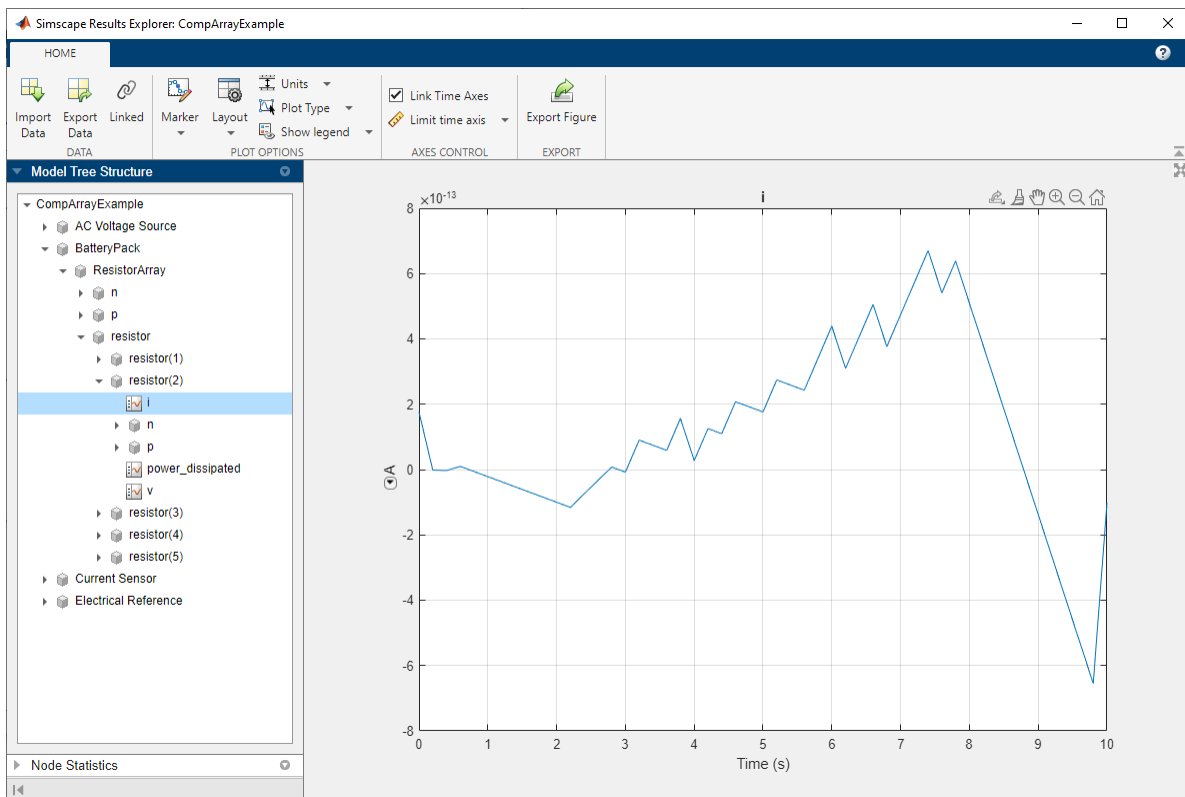
## See Also
`simscape.op.OperatingPoint` | `simscape.op.Target` | `simscape.Value`

## More About
- "Initialize Model Using Operating Point from Logged Simulation Data" on page 8-33

## External Websites
- https://www.mathworks.com/company/newsletters/articles/matrix-indexing-in-matlab.html

# Linearization and Trimming

# Finding an Operating Point

| **In this section...** |
| --- |
| |
| |

## What Is an Operating Point?

An *operating point* of a system is a dynamic configuration that satisfies design and use requirements called *operating specifications*. You can express such operating specifications as requirements on the system state $x$ and inputs $u$. It is not always possible to find a dynamic state that satisfies all operating conditions. Also, a system might have multiple operating points satisfying the same requirements.

Operating points are essential for designing and implementing system controllers. You can optimize a system at an operating point for performance, stability, safety, and reliability.

The most important and common type of operating point is a *steady state*, where some or all of the system dynamic variables are constant.

### Using Operating Points for Linearization

An important motive for finding operating points is *linearization*, which determines the system response to small disturbances at an operating point. Linearization results influence the design of feedback controllers to govern dynamic behavior near the operating point. A full linearization analysis requires one or more system outputs, $y$, in addition to inputs.

See "Linearizing at an Operating Point" on page 9-5.

### Example

A pilot flying an aircraft wants to find, for a given environment, a state of the aircraft engine and control surfaces that produces level, constant-velocity, and constant-altitude flight relative to the ground. The requirements of "level," "constant velocity," "constant altitude," and "relative to the ground" constitute operating specifications. This operating point is a steady state of the aircraft velocity, altitude, and orientation in space.

## Finding Operating Points in Physical Models

You have a number of ways to find an operating point in a Simscape model. You can impose operating specifications and isolate operating points using Simscape and Simulink features.

**Tip**  To find a steady state, the Simscape steady-state solver is the most direct method. For a comprehensive suite of operating point and linearization tools, Simulink Control Design software is recommended.

To analyze operating points, you work with the full state vector of your model, which contains:

- Simulink components, which can be continuous or discrete.

- Simscape components, which are continuous.

Whichever method that you choose to find an operating point, if you want to use it for linearization, you must save the operating point information in the form of an operating point object, a simulation time $t_0$, or a state vector $\boldsymbol{x}_0$ and input vector $\boldsymbol{u}_0$.

- "Simulating in Time to Search for an Operating Point" on page 9-3
- "Using the Simscape Initial Condition Solver" on page 9-3
- "Using Simulink Control Design Techniques to Find Operating Points" on page 9-4
- "Using Sources to Find Operating Points Not Recommended" on page 9-4
- "Simulink trim Function Not Supported with Simscape Models" on page 9-4

**Simulating in Time to Search for an Operating Point**

One way to identify operating points is to simulate your model and inspect its state $\boldsymbol{x}$ and output $\boldsymbol{y}$ as a time series.

1   In your Simscape model, set up sensor outputs for whatever block outputs you want to observe.
2   Connect Scope blocks, To Workspace blocks, or both, to your Simscape block outputs to observe and record simulation behavior.
3   In the **Data Import/Export** pane of your model Configuration Parameters settings, select the **Time**, **States**, and **Output** check boxes to record this simulation information in your workspace.

**Using the Simscape Initial Condition Solver**

Simscape software provides two workflows to initialize a physical model. The first solves for steady state, where all differential variables have zero derivative. Using this approach you can search for multiple steady states with the steady-state solver by varying the model inputs, parameters, and initial conditions. The second approach is to directly specify initial conditions by specifying initialization priority and targets for block variables. For more information on this approach, see "Variable Initialization".

To use the first approach, enable the steady-state solver:

1   In each, some, or all of the physical networks in your Simscape model, open the Solver Configuration block.
2   In each of these block dialog boxes, select the **Start simulation from steady state** check box.
3   In the model Configuration Parameters settings, on the **Data Import/Export** pane, select the **States** check box to record the time series of $\boldsymbol{x}$ values in your workspace.

    If you also have input signals $\boldsymbol{u}$ in the model, you can capture those inputs by connecting To Workspace blocks to the input Simulink signal lines.
4   Close these dialog boxes and start the simulation.

The first vector of values $\boldsymbol{x}(t{=}0)$ that you capture during simulation reflects the steady state $\boldsymbol{x}_0$ that the Simscape solver identified.

**Tip** Finding an initial steady state is part of the nondefault Simscape simulation sequence. See "Initial Conditions Computation" on page 7-8.

You can simplify the initial steady-state computation by setting the simulation time to `0`. The simulation then solves for one time step only (time zero) and returns a single state vector $x(t=0)$.

### Using Simulink Control Design Techniques to Find Operating Points

You can use Simulink Control Design software to find operating points for models with Simscape components. Simulink Control Design provides both command-line and graphical interfaces for finding and analyzing operating points.

For more information, see "Find Steady-State Operating Points for Simscape Models" (Simulink Control Design).

### Using Sources to Find Operating Points Not Recommended

You can impose an operating specification on part of a Simscape model by inserting source blocks from the Simscape Foundation Library. These impose specified values of system variables in parts of the model. You can simulate and save the state vector.

However, you cannot obtain an operating point for the original system (without the source blocks) by saving the state values from the model and then removing the source blocks. In general, the number, order, and identity of state components change after adding and removing Simscape blocks in a model.

### Simulink trim Function Not Supported with Simscape Models

The Simulink `trim` function is not supported for models containing Simscape components.

# Linearizing at an Operating Point

| In this section... |
| --- |
| "What Is Linearization?" on page 9-5 |
| "Linearizing a Physical Model" on page 9-6 |

## What Is Linearization?

Determining the response of a system to small perturbations at an operating point is a critical step in system and controller design. Once you find an operating point, you can linearize the model about that operating point to explore the response and stability of the system. To find an operating point in a Simscape model, see "Finding an Operating Point" on page 9-2.

- "What Is a Linearized Model?" on page 9-5
- "Example" on page 9-5
- "Choosing a Good Operating Point for Linearization" on page 9-6

### What Is a Linearized Model?

Near an operating point, you can express the system state $\boldsymbol{x}$, inputs $\boldsymbol{u}$, and outputs $\boldsymbol{y}$ relative to that operating point in terms of $\boldsymbol{x} - \boldsymbol{x}_0$, $\boldsymbol{u} - \boldsymbol{u}_0$, and $\boldsymbol{y} - \boldsymbol{y}_0$. For convenience, shift the vectors by subtracting the operating point: $\boldsymbol{x} - \boldsymbol{x}_0 \rightarrow \boldsymbol{x}$, and so on.

If the system dynamics do not explicitly depend on time and the operating point is a steady state, the system response to state and input perturbations near the steady state is approximately governed by a *linear time-invariant* (LTI) state space model:

d$\boldsymbol{x}$/dt = $A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u}$

$\boldsymbol{y} = C \cdot \boldsymbol{x} + D \cdot \boldsymbol{u}$.

The matrices $A$, $B$, $C$, $D$ have components and structures that are independent of the simulation time. A system is stable to changes in state at an operating point if the eigenvalues of $A$ are negative.

If the operating point is not a steady state or the system dynamics depend explicitly on time, the linearized dynamics near the operating point are more complicated. The matrices $A$, $B$, $C$, $D$ are not constant and depend on the simulation time $t_0$ , as well as the operating point $\boldsymbol{x}_0$ and $\boldsymbol{u}_0$.

---

**Tip** While you can linearize a closed system with no inputs or outputs and obtain a nonzero $A$ matrix, obtaining a nontrivial linearized input-output model requires at least one input component in $\boldsymbol{u}$ and one output component in $\boldsymbol{y}$.

---

### Example

A pilot is flying, or simulating, an aircraft in level, constant-velocity, and constant-altitude flight relative to the ground. A crucial question for the aircraft pilot and designers is: will the aircraft return to the steady state if perturbed from it by a disturbance, such as a wind gust — in other words, is this steady state stable? If the operating point is unstable, the aircraft trajectory can diverge from the steady state, requiring human or automatic intervention to maintain steady flight.

**Choosing a Good Operating Point for Linearization**

Although steady-state and other operating points (state $x_0$ and inputs $u_0$) might exist for your model, that is no guarantee that such operating points are suitable for linearization. The critical question is: how good is the linearized approximation compared to the exact system dynamics?

- When perturbed slightly, a problematic operating point might exhibit strong asymmetries, with strongly nonlinear behavior when perturbed in one direction and smoother behavior in another.
- Small perturbations might result in a discontinuous change in a state value, making the current state unsuitable for linear approximation.

Operating points with a strongly nonlinear or discontinuous character are not suitable for linearization. You should analyze such models in full simulation, away from any discontinuities, and perturb the system by varying its inputs, parameters, and initial conditions. A common example is actuation systems, which should be linearized away from any hard constraints or end stops.

**Tip** Check for such an unsuitable operating point by linearizing at several nearby operating points. If the results differ greatly, the operating point is strongly nonlinear or discontinuous.

## Linearizing a Physical Model

Use the following methods to create numerical linearized state-space models from a model containing Simscape components.

**Tip** The Simulink Control Design product is recommended for linearization analysis.

- "Independent Versus Dependent States" on page 9-6
- "Linearizing with Simulink Control Design Software" on page 9-7
- "Linearizing with the Simulink linmod and dlinmod Functions" on page 9-7
- "Linearizing with Simulink Linearization Blocks" on page 9-8

**Independent Versus Dependent States**

An important difference from basic Simulink models is that the states in a physical network are not independent in general, because some states have dependencies on other states through constraints.

- The independent states are a subset of system variables and consist of independent (unconstrained) Simscape dynamic variables and other Simulink states.
- The dependent states consist of Simscape algebraic variables and dependent (constrained) Simscape dynamic variables.

For more information on Simscape dynamic and algebraic variables, see "How Simscape Simulation Works" on page 7-6.

The complete, unreduced LTI $A$, $B$, $C$, $D$ matrices have the following structure.

- The A matrix, of size `n_states` by `n_states`, is all zeros except for a submatrix of size `n_ind` by `n_ind`, where `n_ind` is the number of independent states.

- The B matrix, of size `n_states` by `n_inputs`, is all zeros except for a submatrix of size `n_ind` by `n_inputs`.
- The C matrix, of size `n_outputs` by `n_states`, is all zeros except for a submatrix of size `n_outputs` by `n_ind`.
- The D matrix, of size `n_outputs` by `n_inputs`, can be nonzeros everywhere.

**Obtaining the Independent Subset of States**

A minimal linearized solution uses only an independent subset of system states. From the matrices *A*, *B*, *C*, *D*, you can obtain a minimal input-output linearized model with:

- The `minreal` and `sminreal` functions from Control System Toolbox™ software
- Automatically with the Simulink Control Design approach

**Linearizing with Simulink Control Design Software**

**Note** The techniques described in this section require the Simulink Control Design product.

You must use the features of this product on the Simulink lines in your model, not directly on Simscape physical network lines or blocks.

This approach requires that you start with an operating point object saved from trimming the model to an operating specification.

To linearize a model with an operating point object, use the `linearize` function, customizing where necessary. The resulting state-space object contains the matrices *A*, *B*, *C*, *D*.

You can also use the graphical user interface, through the Simulink Toolstrip: on the **Apps** tab, under **Control Systems**, click **Model Linearizer**.

For more information on linearizing Simscape models using Simulink Control Design, see "Linearize Simscape Networks" (Simulink Control Design).

**Linearizing with the Simulink linmod and dlinmod Functions**

You have several ways that you can use the Simulink functions `linmod` and `dlinmod`, and the linearization results can differ depending on the method chosen. To use these functions, you do not have to open the model, just have the model file on your MATLAB path.

For more information about Simulink linearization, see "Linearizing Models".

**Tip** If your model has continuous states, use `linmod`. (Continuous states are the Simscape default.) If your model has mixed continuous and discrete states, or purely discrete states, use `dlinmod`.

Linearizing a model with the local solver enabled (in the Solver Configuration block) is not supported.

**Linearizing with Default State and Input**

You can call `linmod` without specifying state or input. Enter `linmod('`*modelname*`')` at the command line.

With this form of `linmod`, Simulink linearization solves for consistent initial conditions in the same way it does on the first step of any simulation. Any initial conditions, such as initial offset from equilibrium for a spring, are set as if the simulation were starting from the initial time.

`linmod` allows you to change the time of externally specified signals (but not the internal system dynamics) from the default. For this and more details, see the `linmod` function reference page.

**Linearizing with the Steady-State Solver at an Initial Steady State**

You can linearize at an operating point found by the Simscape steady-state solver:

1    Open one or more Solver Configuration blocks in your model.
2    Select the **Start simulation from steady state** check box for the physical networks that you want to linearize.
3    Close the Solver Configuration dialog boxes and save the modified model.
4    Enter `linmod('`*modelname*`')` at the command line.

`linmod` linearizes at the first step of simulation. In this case, the initial state is also an operating point, a steady state.

For more about setting up the steady-state solver, see the Solver Configuration block reference page.

**Linearizing with Specified State and Input — Ensuring Consistency of States**

You can call `linmod` and specify state and input. Enter `linmod('`*modelname*`',x0,u0)` at the command line. The extra arguments specify, respectively, the steady state $x_0$ and inputs $u_0$ for linearizing the simulation. When you specify a state to `linmod`, ensure that it is self-consistent, within solver tolerance.

With this form of `linmod`, Simulink linearization does not solve for initial conditions. Because not all states in the model have to be independent, it is possible, though erroneous, to provide `linmod` with an inconsistent state to linearize about.

If you specify a state that is not self-consistent (within solver tolerance), the Simscape solver issues a warning at the command line when you attempt linearization. The Simscape solver then attempts to make the specified `x0` consistent by changing some of its components, possibly by large amounts.

---

**Tip** You most easily ensure a self-consistent state by taking the state from some simulated time. For example, by selecting the **States** check box on the **Data Import/Export** pane of the model Configuration Parameters dialog box, you can capture a time series of state values in a simulation run.

---

**Linearizing with Simulink Linearization Blocks**

You can generate linearized state-space models from your Simscape model by adding a Timed-Based Linearization or Trigger-Based Linearization block to the model and simulating. These blocks combine time-based simulation, up to specified times or internal trigger points, with state-based linearization at those times or trigger points.

For complete details about these blocks, see their respective block reference pages.

---

**Note** If your model contains PS Constant Delay or PS Variable Delay blocks, or custom blocks utilizing the `delay` operator in the Simscape language, it is recommended that you linearize the

model by using the Timed-Based Linearization or Trigger-Based Linearization block and simulating the model for a time period longer than the specified delay time.

# Linearize an Electronic Circuit

This example shows how to linearize a model of a nonlinear bipolar transistor circuit and create a Bode plot for small-signal frequency-domain analysis.

Depending on the software you have available, use the appropriate sections of this example to explore various linearization and analysis techniques.

## Explore the Model

To open the Nonlinear Bipolar Transistor example model, type `ssc_bipolar_nonlinear` in the MATLAB Command Window.
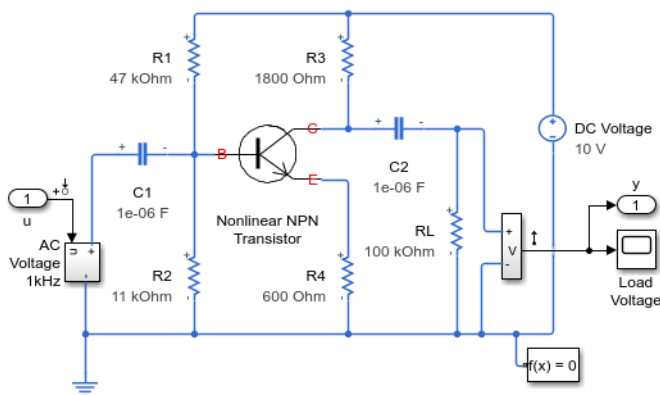


**Nonlinear Bipolar Transistor**

1. Plot voltages at transistor terminals (see code)
2. Linearize circuit to view frequency response (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

The model represents a single-transistor audio amplifier. The transistor is an NPN bipolar device, and as such has a nonlinear set of current-voltage characteristics. Therefore the overall behavior of the amplifier is dependent on the operating point of the transistor. The transistor itself is represented by and Ebers-Moll equivalent circuit implemented using a masked subsystem. The circuit has a sinusoidal input test signal with amplitude 10 mV and frequency 1 kHz. The Load Voltage scope displays the resulting collector output voltage after the DC is filtered out by the output decoupling capacitor.

R1 and R2 set the nominal operating point, and the small signal gain is approximately set by the ratio R3/R4. The decoupling capacitors C1 and C2 have a capacitance of 1uF, to present negligible impedance at 1 kHz.

The model is configured for linearization. You can quickly generate and view the small-signal frequency response by clicking the `Linearize circuit` hyperlink in model annotation. To view the MATLAB script that generates the frequency response, click the next hyperlink in that annotation, `see code`. This documentation provides background information and alternative ways of linearization based on the software you have.

In general, to obtain a nontrivial linearized input-output model and generate a frequency response, you must specify model-level inputs and outputs. The Nonlinear Bipolar Transistor model meets this requirement in two ways, depending on how you linearize:

- Simulink requires top- or model-level input and output ports for linearization with `linmod`. The Nonlinear Bipolar Transistor model has such ports, marked `u` and `y`.

- Simulink Control Design software requires that you specify input and output signal lines with linearization points. The specified lines must be Simulink signal lines, not Simscape physical connection lines. The Nonlinear Bipolar Transistor model has such linearization points specified. For more information on using Simulink Control Design software for trimming and linearization, see documentation for that product.

Open the Solver Configuration block and see that the **Start simulation from steady state** check box is selected. Then open the Load Voltage scope and run the simulation to see the basic circuit behavior. The transistor junction capacitance initial voltages are set to be consistent with the bias conditions defined by the resistors. The output is a steady sinusoid with zero average, its amplitude amplified by the transistor and bias circuit.
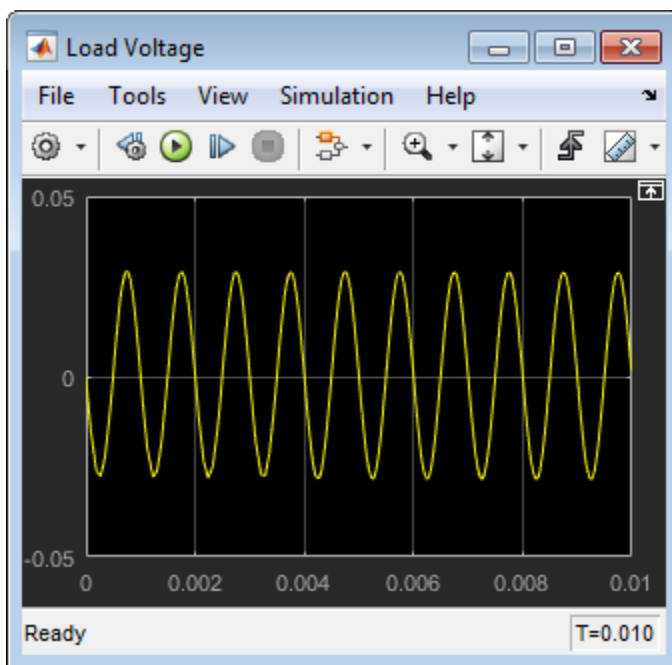


To see the circuit relax from a nonsteady initial state, in the Solver Configuration block, clear the **Start simulation from steady state** check box and click **OK**. With the Load Voltage scope open, simulate again. In this case, the output voltage starts at zero because the transistor junction capacitances start with zero charge.
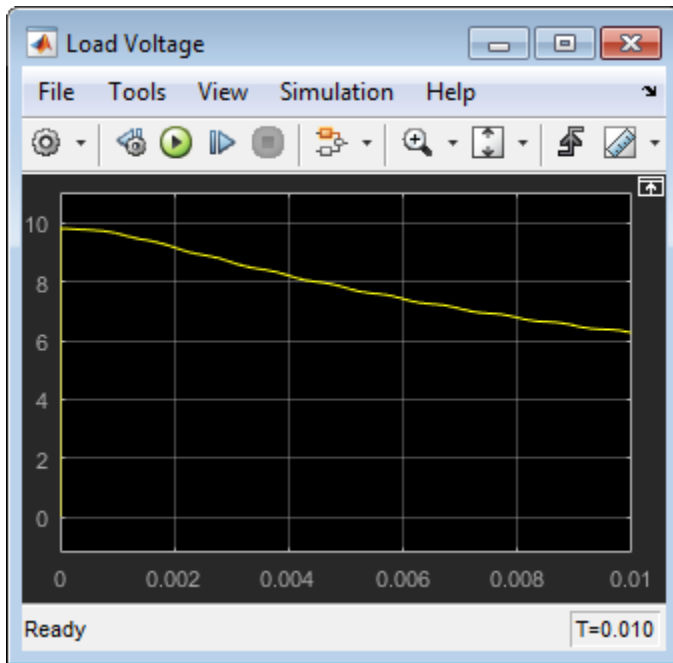
You can get a more comprehensive understanding of the circuit behavior and how it approaches the steady state by long-time transient simulation. Increase the simulation time to 1 s and rerun the simulation. The circuit starts from its initial nonsteady state, and the transistor collector voltage approaches and eventually settles into steady sinusoidal oscillation.
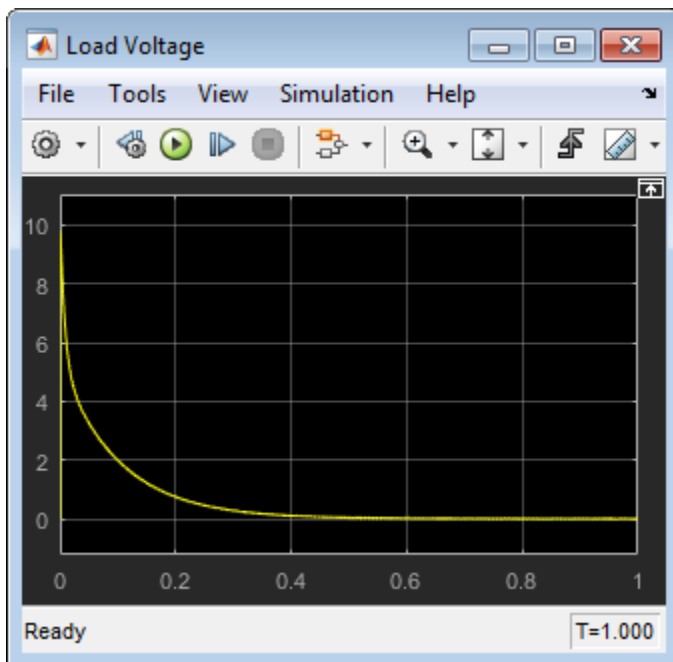


Open the Solver Configuration block, select the **Start simulation from steady state** check box (as it was when you first opened the model), and click **OK**. Change the simulation time back to .01 s and rerun the simulation.

## Linearize with Steady-State Solver and linmod Function

In this example, you:

**1**    Use the Simscape steady-state solver to find an operating point

**2**    Linearize the model using the Simulink `linmod` function

**3**    Generate the Bode plot using a series of MATLAB commands

Open the Solver Configuration block and make sure the **Start simulation from steady state** check box is selected. When you simulate the model with the Simscape steady-state solver enabled, the circuit is initialized at the state defined by the transistor bias resistors. This steady-state solution is an operating point suitable for linearization.

---

**Note** Also make sure that the **Use local solver** check box is cleared. Linearizing a model with the local solver enabled is not supported.

---

To linearize the model, type the following in the MATLAB Command Window:

```
[a,b,c,d]=linmod('ssc_bipolar_nonlinear');
```

You can alternatively call the `linmod` function with a single output argument, in which case it generates a structure with states, inputs, and outputs, as well as the linear time-invariant (LTI) model.

The state vector of the Nonlinear Bipolar Transistor model contains 17 components. The full model has one input and one output. Thus, the LTI state-space model derived from linearization has the following matrix sizes:

- *a* is 17-by-17
- *b* is 17-by-1
- *c* is 1-by-17
- *d* is 1-by-1

To generate a Bode plot, type the following in the MATLAB Command Window:

```
npts = 100; f = logspace(-2,10,npts); G = zeros(1,npts);
for i=1:npts
    G(i) = c*(2*pi*1i*f(i)*eye(size(a))-a)^-1*b +d;
end
subplot(211), semilogx(f,20*log10(abs(G)))
grid
ylabel('Magnitude (dB)')
subplot(212), semilogx(f,180/pi*unwrap(angle(G)))
ylabel('Phase (degrees)')
xlabel('Frequency (Hz)')
grid
```
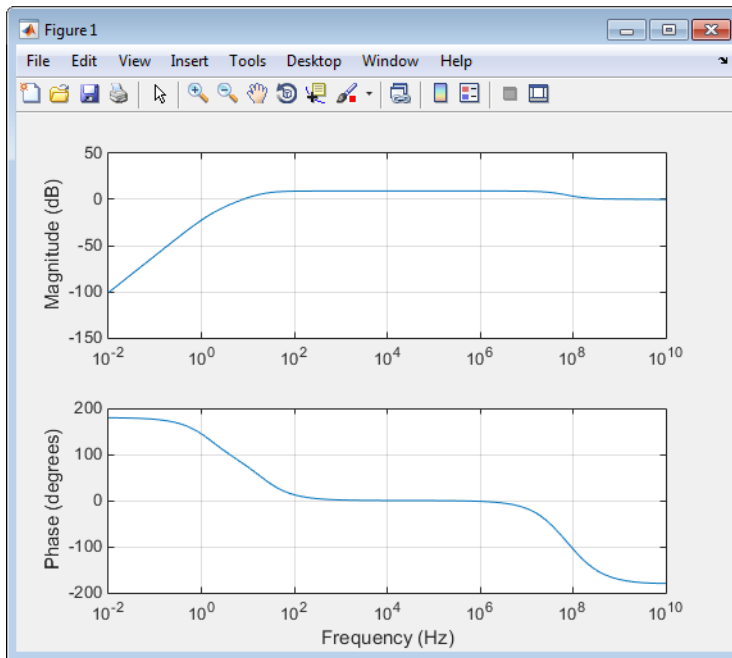
## Linearize with Simulink Control Design Software

---

**Note** To work through this section, you must have a Simulink Control Design license.

---

Simulink Control Design software has tools that help you find operating points and returns a state-space model object that defines state names. This is the recommended way to linearize Simscape models.

**1** In the Simulink Toolstrip of the Nonlinear Bipolar Transistor model window, on the **Apps** tab, under **Control Systems**, click **Model Linearizer**.

**2** In the Model Linearizer window, on the **Linear Analysis** tab, click the **Bode** plot button.
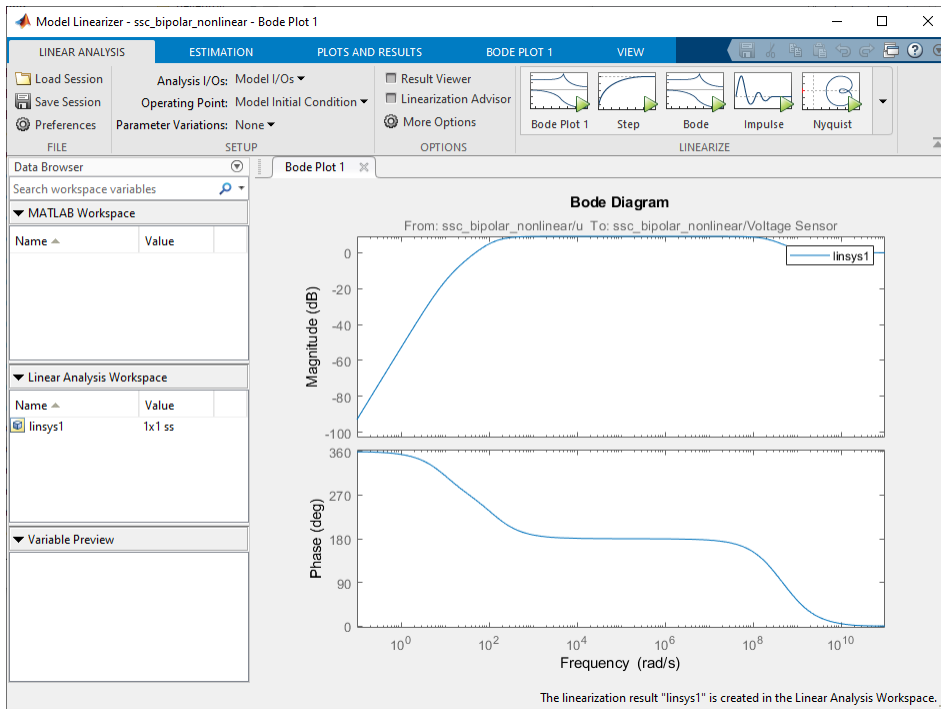
For more information on using Simulink Control Design software for trimming and linearization, see the Simulink Control Design documentation.

## Use Control System Toolbox Software for Bode Analysis

**Note** To work through this section, you must have a Control System Toolbox license.

You can use the built-in analysis and plotting capabilities of Control System Toolbox software to analyze and compare Bode plots of different steady states.

First, use the Simulink `linmod` function to obtain the linear time-invariant (LTI) model.

```
[a,b,c,d]=linmod('ssc_bipolar_nonlinear');
```

Not all the states of the LTI model derived in this example are independent. Confirm this by calculating the determinant of the $a$ matrix, `det(a)`. The determinant vanishes, which implies one or more zero eigenvalues. To analyze the LTI model, reduce the LTI matrices to a minimal realization. Obtain a minimal realization using the `minreal` function.

```
[a0,b0,c0,d0] = minreal(a,b,c,d);
```

```
13 states removed.
```

Extracting the minimal realization eliminates 13 dependent states from the LTI model, leaving four independent states. Analyze the control characteristics of the reduced `a0`, `b0`, `c0`, `d0` LTI model using a Bode plot.

```
bode(a0,b0,c0,d0) % Creates first Bode plot
```

The circuit with R1 changed from 47 to 15 kOhm has a different steady state and response. Double-click the R1 block, change the **Resistance** value to 15 kOhm, and click **OK**. Open the Load Voltage scope and simulate the model. The collector voltage is now no longer amplified relative to the 10 mV AC source but attenuated.
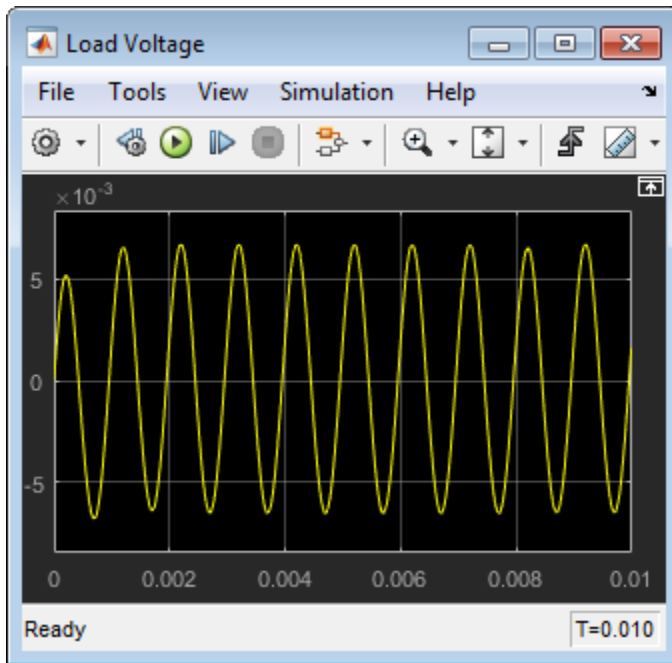


Produce the LTI model at the second steady state, reduce it to a minimal realization, and superpose the second Bode plot on the first one.

```
[a_R1,b_R1,c_R1,d_R1]=linmod('ssc_bipolar_nonlinear');
[a0_R1,b0_R1,c0_R1,d0_R1] = minreal(a_R1,b_R1,c_R1,d_R1); % 13 states removed.
hold on % Keeps first Bode plot open
bode(a0_R1,b0_R1,c0_R1,d0_R1) % Superposes second Bode plot on first
```
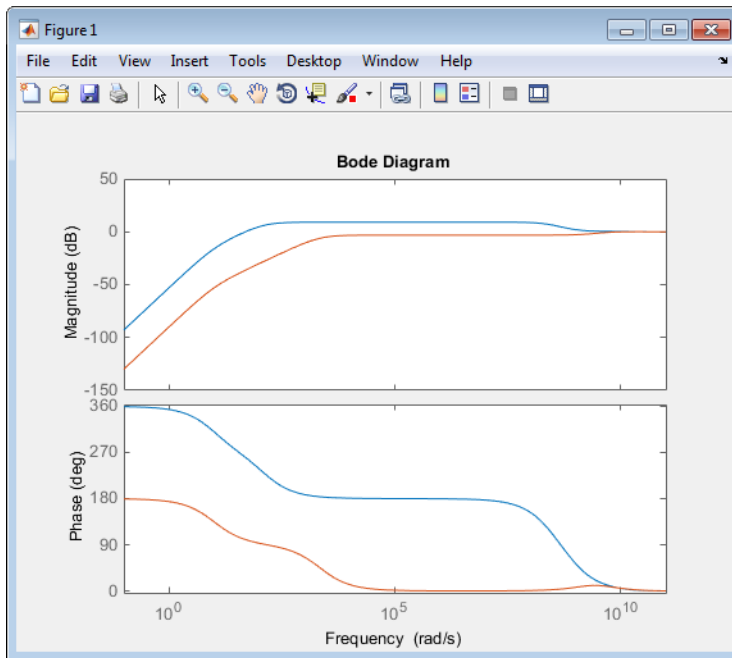
For more information on using Control System Toolbox software for Bode analysis, see the Control System Toolbox documentation.

## See Also

## Related Examples

- "Linearize a Plant Model for Use in Feedback Control Design" on page 9-18

## More About

- "Finding Operating Points in Physical Models" on page 9-2
- "Linearizing a Physical Model" on page 9-6

# Linearize a Plant Model for Use in Feedback Control Design

This example shows how you can linearize a hydraulic plant model to support control system stability analysis and design.

Depending on the software you have available, use the appropriate sections of this example to explore various linearization and analysis techniques.

## Explore the Model

To open the Hydraulic Actuator with Digital Position Controller example model, type `ssc_hydraulic_actuator_digital_control` in the MATLAB Command Window.
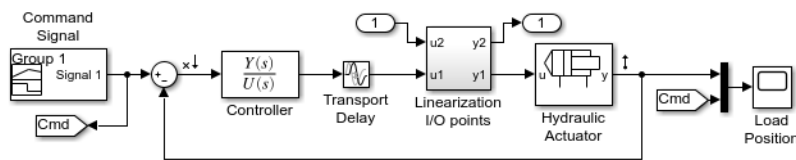


**Hydraulic Actuator with Digital Position Controller**

1. Plot pressures in hydraulic cylinder (see code)
2. Linearize the hydraulic plant (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

The model represents a two-way valve acting in a closed-loop circuit together with a double-acting cylinder. Double-click the Hydraulic Actuator subsystem to see the model configuration.
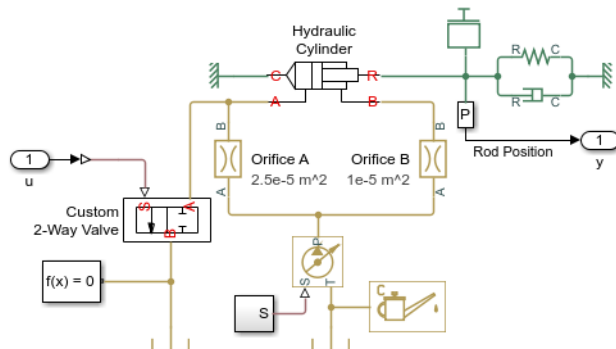


The controller is represented as a continuous-time transfer function plus a transport delay that allows for computational time and a zero-order hold when implemented in discrete time. The Linearization I/O points subsystem lets you easily break and restore the feedback control loop by setting the base workspace variable ClosedLoop to 0 or 1, respectively.
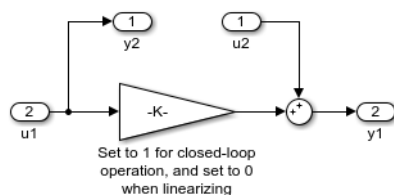
You can quickly generate and view the small-signal frequency response by clicking the `Linearize` hyperlink in model annotation. To view the MATLAB script that generates the frequency response, click the next hyperlink in that annotation, `see code`. This documentation provides background information and alternative ways of linearization based on the software you have.

In general, to obtain a nontrivial linearized input-output model and generate a frequency response, you must specify model-level inputs and outputs. The Hydraulic Actuator with Digital Position Controller model meets this requirement in two ways, depending on how you linearize:

- Simulink requires top- or model-level input and output ports for linearization with `linmod`. The model has such ports, marked `In1` and `Out1`.
- Simulink Control Design software requires that you specify input and output signal lines with linearization points. The specified lines must be Simulink signal lines, not Simscape physical connection lines. The model has such linearization points specified. For more information on using Simulink Control Design software for trimming and linearization, see documentation for that product.

Open the Load Position scope and simulate the model in a normal closed-loop controller configuration.
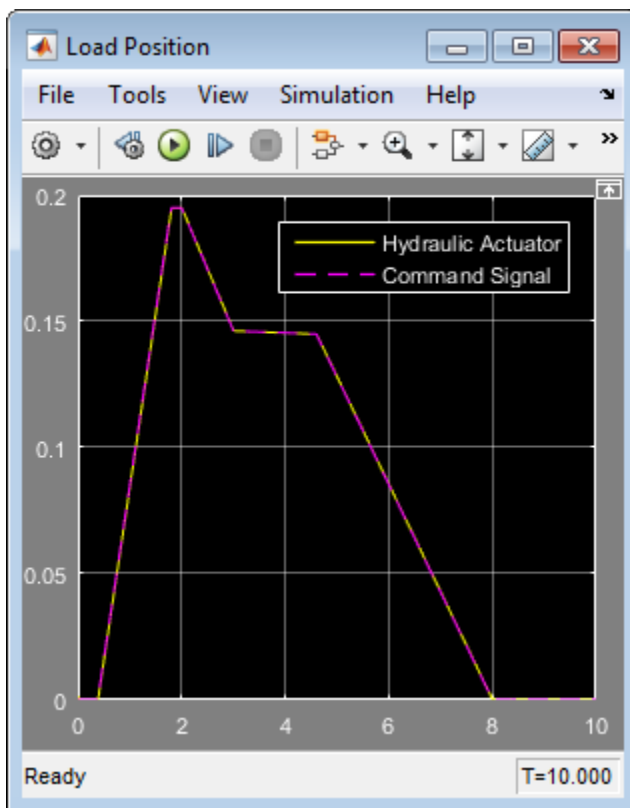


You can see that the model has a quasi-linear steady-state response between 2 and 3 seconds, when the two-way valve is open. Therefore, the state at 2.5 seconds is an operating point suitable for linearization.

## Trim Using the Controller and Linearize with Simulink linmod Function

**1** Set the controller parameters.

To specify sample time for controller discrete-time implementation, type the following in the MATLAB Command Window:

```
ts = 0.001;
```

To specify continuous-time controller numerator and denominator, type:

```
num = -0.5;
den = [1e-3 1];
```

**2** Find an operating point by running closed-loop and selecting the state at 2.5 seconds when the custom two-way valve is open.

To close the feedback loop, type:

```
assignin('base','ClosedLoop',1);
```

To simulate the model and save the operating point information in the form of a state vector *X* and input vector *U*, type:

```
[t,x,y] = sim('ssc_hydraulic_actuator_digital_control');
idx = find(t>2.5,1);
X = x(idx,:); U = y(idx);
```

**3** Linearize the model using the Simulink `linmod` function.

To break the feedback loop, type:

```
assignin('base','ClosedLoop',0);
```

To linearize the model, type:

```
[a,b,c,d] = linmod('ssc_hydraulic_actuator_digital_control',X,U);
```

Close the feedback loop by typing:

```
assignin('base','ClosedLoop',1);
```

**4** To generate a Bode plot with negative feedback convention, type the following in the MATLAB Command Window:

```
c = -c; d = -d;
npts = 100; w = logspace(-3,5,npts); G = zeros(1,npts);
for i = 1:npts
    G(i) = c*(1i*w(i)*eye(size(a))-a)^-1*b +d;
end
subplot(211), semilogx(w,20*log10(abs(G)))
grid
ylabel('Magnitude (dB)')
subplot(212), semilogx(w,180/pi*unwrap(angle(G)))
ylabel('Phase (degrees)')
xlabel('Frequency (rad/s)')
grid
```
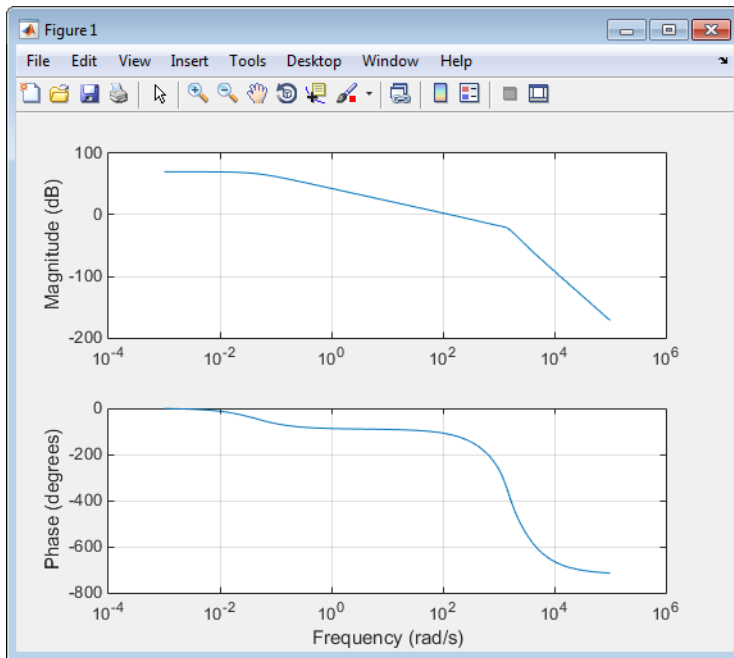
## Linearize with Simulink Control Design Software

**Note** To work through this section, you must have a Simulink Control Design license.

Simulink Control Design software has tools that help you find operating points and returns a state-space model object that defines state names. This is the recommended way to linearize Simscape models.

**1**    In the Simulink Toolstrip of the Hydraulic Actuator with Digital Position Controller model window, on the **Apps** tab, under **Control Systems**, click **Model Linearizer**.

**2**    In the Model Linearizer window, on the **Linear Analysis** tab, in the **Operating Point** drop-down list, select `Linearize At`. Enter simulation snapshot time of `2.5` seconds and click **OK**.

**3**    Click the **Bode** plot button.
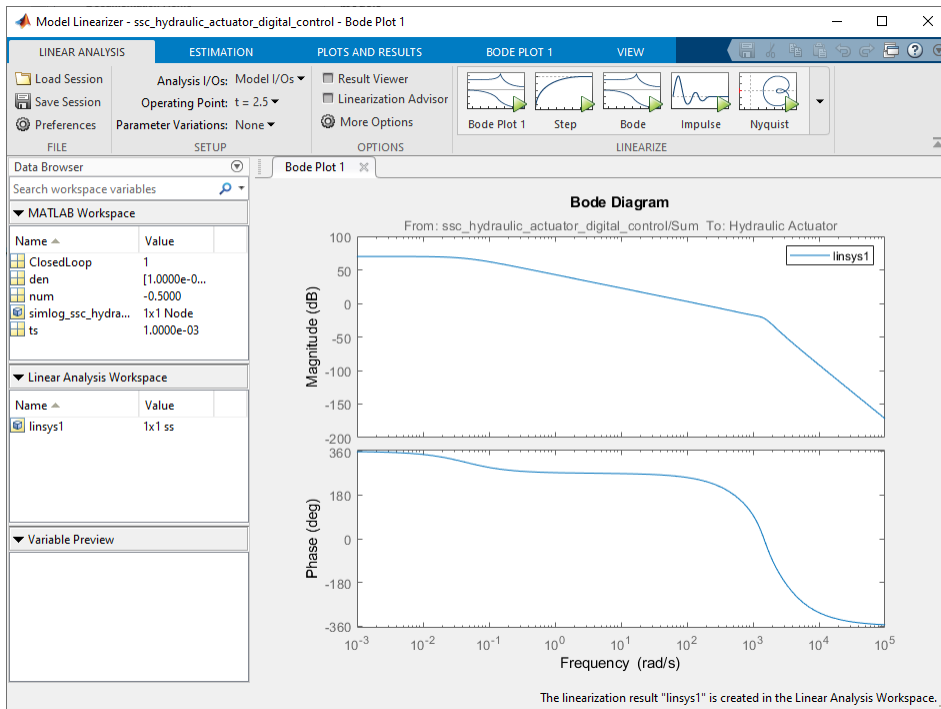
For more information on using Simulink Control Design software for trimming and linearization, see the Simulink Control Design documentation.

## See Also

## Related Examples

- "Linearize an Electronic Circuit" on page 9-10

## More About

- "Finding Operating Points in Physical Models" on page 9-2
- "Linearizing a Physical Model" on page 9-6

# Simscape Run-Time Parameters

# About Simscape Run-Time Parameters

| **In this section...** |
| --- |
| "Enabling Run-Time Configurability" on page 10-2 |
| "Run-Time Configurability for Block-Level Variable Initialization Target Values" on page 10-3 |

Simscape run-time parameters are MATLAB variables or `Simulink.Parameter` objects that are run-time configurable. By default, run-time configurable parameters are noninlined during code generation. Simscape run-time parameters allow you to skip recompiling the model when you change parameter values. You can change parameter values:

- Between fast-restart, iterative simulations on a development computer
- In referenced models on a development computer
- In the generated code in a rapid simulation (RSim) or on real-time target hardware

For more information on using Simscape run-time parameters for these types of simulation, see "Improve Parameter-Sweeping Efficiency Using Simscape Run-Time Parameters" on page 10-12.

By default, all Simscape block parameters are compile-time parameters. You can only change the value of compile-time parameters in the plant model on your development computer.

## Enabling Run-Time Configurability

Simscape supports run-time configurability for most parameters that require a numerical value input. To determine if you can specify a particular parameter as a Simscape run-time parameter, review the settings for the parameter in the block property inspector. If a parameter is run-time configurable, you will see a property inspector set to the default setting, `Compile-time`. You can change this to `Run-time` for the parameters that you want to be run-time configurable. You can change this setting at any time before you generate code from your Simscape model.

To specify a Simscape block parameter as run-time configurable, change the **Configurability** setting, which appears in the block property inspector underneath the parameter name, from `Compile-time` to `Run-time`. The figure shows setting run-time configuration for the **Constant voltage** parameter of a DC Voltage Source block.
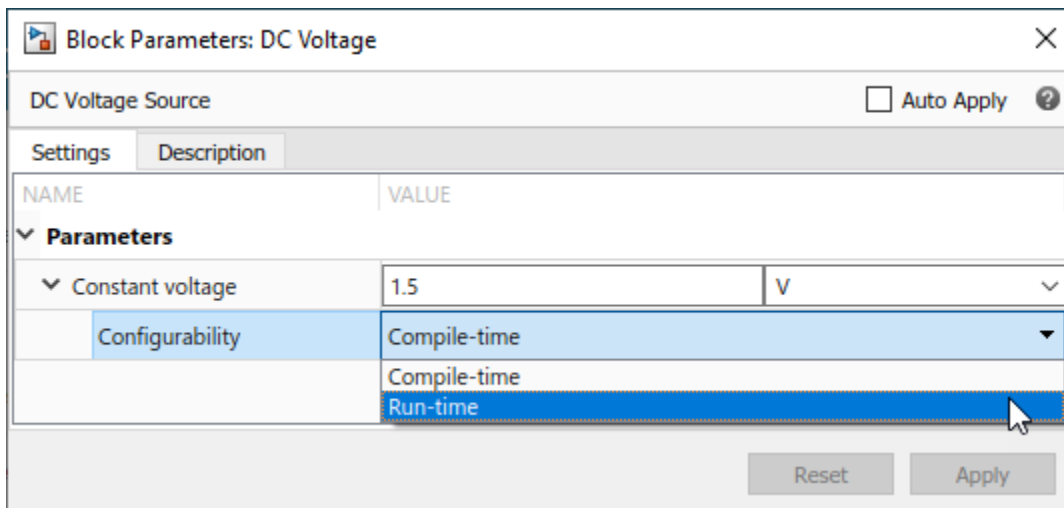
You can specify run-time parameter values numerically in the property inspector. You can also specify the parameter value by entering a name of a variable in the MATLAB workspace, and then tune the parameter by changing the value of the workspace variable. For an example that shows how to specify and change Simscape run-time parameters on development and target computers, see "Specify and Change a Simscape Run-Time Parameter" on page 10-6 and "Change Parameter Values on Target Hardware" on page 11-118.

While Simscape run-time parameters can make iterative simulation more efficient, using them can decrease the efficiency of code that you generate. Code that contains compile-time or inlined run-time parameters is more computationally efficient because it does not have to store or retrieve parameter values. If you set the default parameter behavior for code generation to inlined, the generated code algorithm inlines the numeric values of all block parameters as constants.

For information that can help you decide when to inline Simscape run-time parameters, see "Decrease Computational Cost by Inlining Simscape Run-Time Parameters" on page 10-14. To learn how to inline Simscape run-time parameters, see "Manage Simscape Run-Time Parameters" on page 10-4.

Simscape run-time parameters not the same as Simulink tunable parameters. For information on comparisons between the two types of parameters, see "How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ" on page 10-10.

## Run-Time Configurability for Block-Level Variable Initialization Target Values

Some Simscape blocks have **Initial Targets** settings that allow you to set a target value for block-level variable initialization. For more information, see "Initializing Block Variables for Model Simulation" on page 8-2 and "Set Priority and Initial Target for Block Variables" on page 8-4.

The variables included in the **Initial Targets** settings are run-time configurable by default. You can tune a block-level variable-initialization target value between simulation runs if you specify the target value using a variable that you save to the MATLAB workspace.

## See Also

## More About

- "Specify and Change a Simscape Run-Time Parameter" on page 10-6
- "Change Parameter Values on Target Hardware" on page 11-118
- "Manage Simscape Run-Time Parameters" on page 10-4
- "Troubleshoot Simscape Run-Time Parameter Issues" on page 10-9
- "Improve Parameter-Sweeping Efficiency Using Simscape Run-Time Parameters" on page 10-12
- "Decrease Computational Cost by Inlining Simscape Run-Time Parameters" on page 10-14
- "How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ" on page 10-10

# Manage Simscape Run-Time Parameters

By default, all Simscape parameters are compile-time configurable. You can only change the value of compile-time parameters in the plant model on your development computer.

## Specify Simscape Run-Time Parameters

Some Simscape block parameters are strictly compile-time configurable parameters. To determine if a particular Simscape parameter is supported for run-time configurability, examine the block property inspector settings for the parameter. If a parameter is run-time configurable, you can see an expander symbol in front of its name in the block property inspector. Expanding it exposes the **Configurability** setting for this parameter, which is set to `Compile-time` by default. You can change this to `Run-time` for the parameters that you want to be run-time configurable. You can change this setting at any time before you generate code from your Simscape model.

The figure shows an enabled run-time configurability setting for the **Constant voltage** parameter of a DC Voltage Source block. If there is no expander in front of a parameter name, or if the **Configurability** setting is disabled, then the parameter is strictly compile-time configurable.
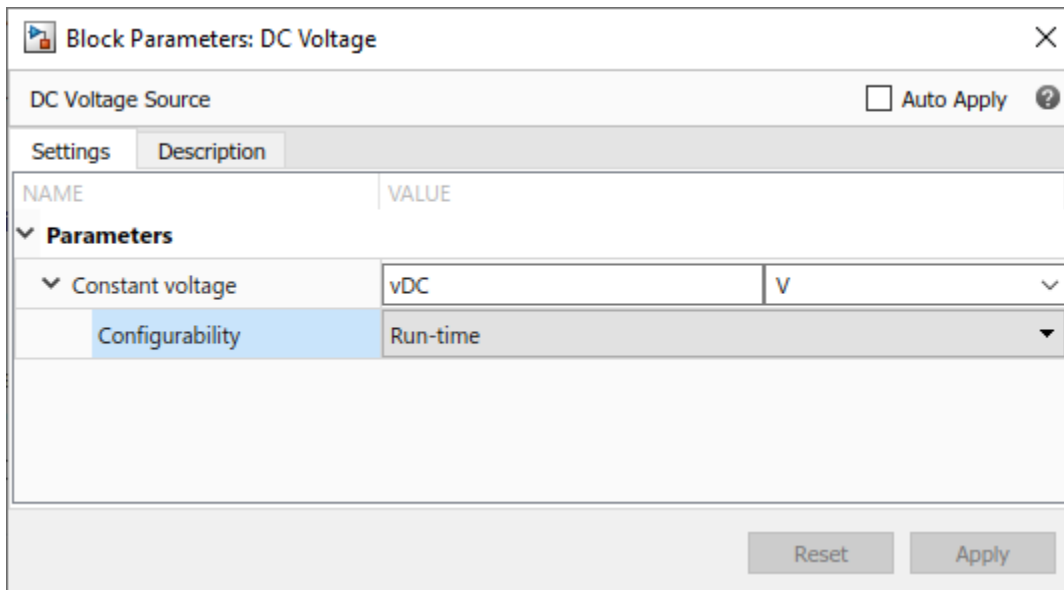


When you set a Simscape parameter to be run-time configurable, you can leave it as a numerical value, or you can specify a variable in the workspace as shown in the figure. To use a variable:

1. Use input tools, such as the command prompt, callbacks, scripts, or MAT-files, to assign a value for the variable in the MATLAB workspace.
2. Specify the variable for the parameter in the block property inspector setting.
3. Use input tools to change the value for the variable as desired.

Either way you specify the parameter, the new value will be incorporated when you restart the simulation. For an example that shows how to specify a Simscape run-time parameter using a variable, see "Specify and Change a Simscape Run-Time Parameter" on page 10-6.

## Set the Default Parameter Behavior for Generated Code

By default, you can change the value of a Simscape run-time parameter while the simulation is stopped without having to recompile the model. If you change the default parameter behavior for code generation to `Inline`, the generated code inlines the numeric values of all the block parameters as constants. The code that you generate using inlined parameters is more computationally efficient because it does not have to store or retrieve parameter values.

To set the default behavior for Simscape run-time parameters:

**1** Open the model Configuration Parameters. On the **Modeling** tab, click **Model Settings** > **Model Settings**.

**2** Select **Code Generation** > **Optimization**.

**3** Select a value for the **Default parameter behavior** parameter:

- `Tunable` (default) — Simulink Coder generates data structures that allow you to change parameters without recompiling between simulation runs.
- `Inline` — Simulink Coder treats the numeric values of all the block parameters as constants in the generated C code, rendering them non-modifiable.

## Selectively Override the Inline Default Behavior

The first run-time configurable parameter that you specify has the highest computational cost, and the added cost for each additional parameter is lower than the last. Therefore, even if the computational cost for your model decreases greatly with inlining, it decreases only slightly for each Simscape run-time parameter that you selectively inline. However, if your model is at risk of overrunning and it contains Simscape run-time parameters, you might decrease the computational cost enough to prevent overruns.

To inline with exceptions:

**1** In the **Code Generation** > **Optimization** settings, set the **Default parameter behavior** to `Inline`.

**2** Click **Configure** to open the Model Parameter Configuration window.

**3** Remove individual parameters from inlining as necessary.

## See Also

## More About

- "Model Configuration Parameters: Code Generation Optimization" (Simulink Coder)
- "Model Callbacks"
- "About Simscape Run-Time Parameters" on page 10-2
- "Specify and Change a Simscape Run-Time Parameter" on page 10-6
- "Change Parameter Values on Target Hardware" on page 11-118
- "Decrease Computational Cost by Inlining Simscape Run-Time Parameters" on page 10-14
- "How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ" on page 10-10

# Specify and Change a Simscape Run-Time Parameter

If you want to generate code, you may need to change the default inlining behavior. The example demonstrates how you can specify and change a Simscape run-time parameter once you have completed this step.

## Prerequisites

### Set the Default Parameter Behavior for Code Generation

Simscape run-time parameters depend on the default parameter behavior setting for code generation. To enable run-time configurability for a Simscape run-time parameter so that you can change its value without recompiling your model, you can:

- Leave the default as `Tunable`.
- Set the default to `Inline`, and override the default behavior for the parameter with the value that you want to change.

For information on setting and overriding the default behavior for Simscape run-time parameters, see "Manage Simscape Run-Time Parameters" on page 10-4.

## Specify a Parameter as Run-Time Configurable

The Permanent Magnet DC Motor example model contains a DC Voltage block. You parameterize the block by specifying the **Constant voltage**. The voltage is a Simscape run-time configurable parameter. To specify **Constant voltage** as a run-time configurable parameter:

**1** To open model, at the MATLAB command prompt, enter:

   `ssc_dcmotor`

**2** Assign a numeric value to the `vDC` variable in the MATLAB workspace:

   `vDC = 5;`

**3** Double-click the DC Voltage block to edit the parameters.

**4** In the **Configurability** drop-down box for **Constant voltage**, select `Run-time`.

**5** Specify the **Constant voltage** parameter value as the variable `vDC`.
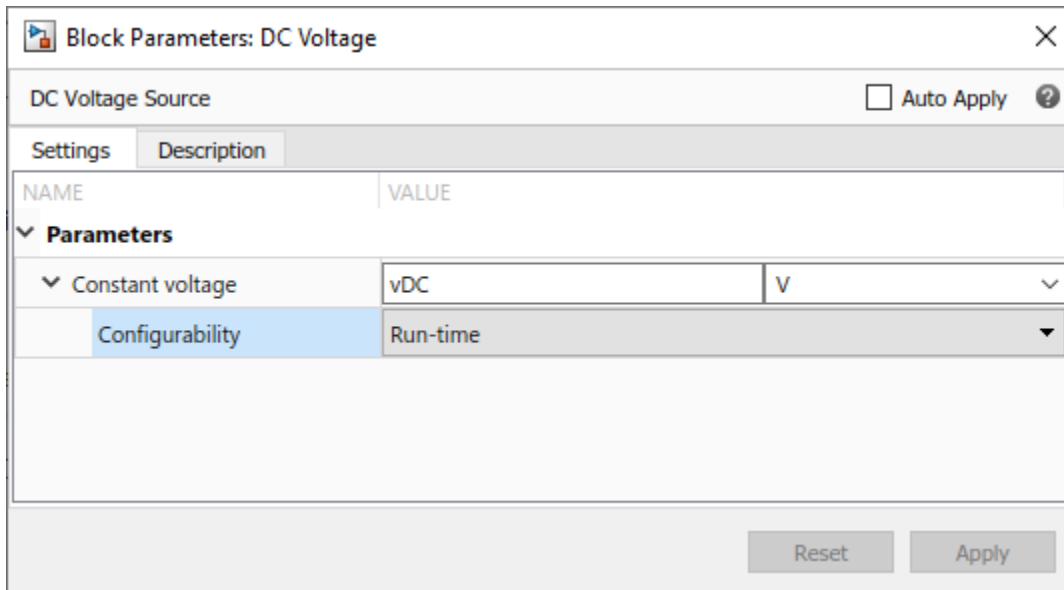
## Change a Simscape Run-Time Parameter Using Fast Restart

To see how altering the value of a run-time configurable parameter can affect simulation results, change the value of the **Constant voltage** parameter between iterative fast-start simulations.

**1**    To enable fast restart, navigate to the **Simulate** section under the **Simulation** tab and click the

**Fast restart** icon  .

**2**    To simulate the model, click **Run**.

**3**    Open the Scope block called Motor RPM. Select **Tools** > **Axes Scaling** > **Automatically Scale Axes Limits** to see the results better.
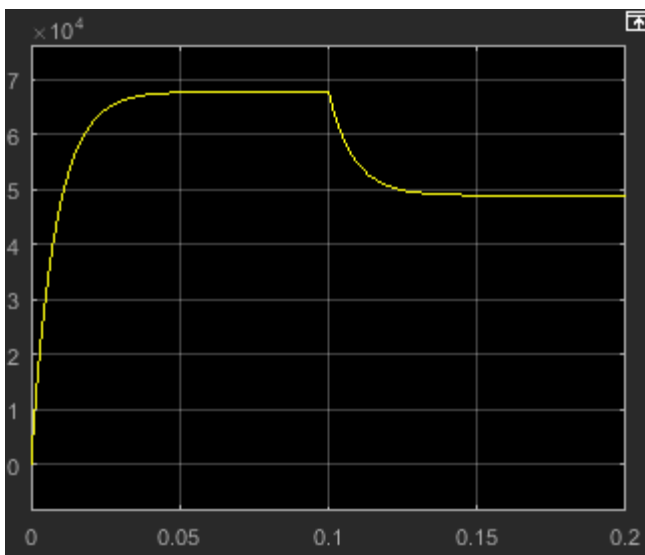


**4**    Assign a different value to the variable that represents voltage:

```
vDC = 1.5;
```

**5**  Simulate the model.

**6**  Open the Scope block.
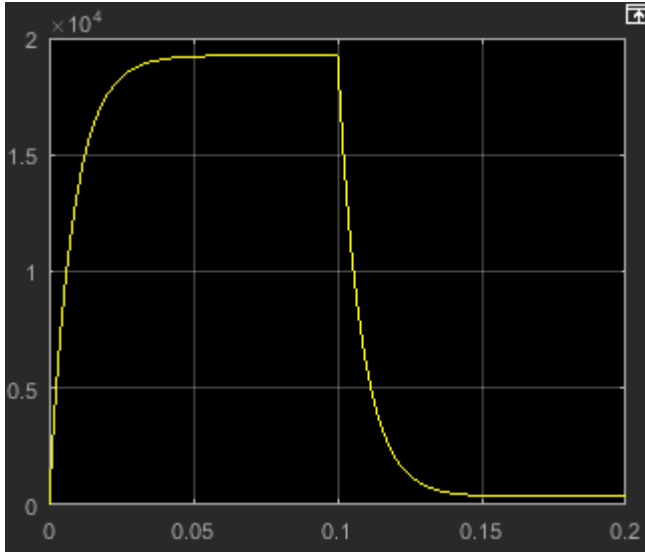


The results reflect the change in value for the run-time parameter.

## See Also

## More About

- "About Simscape Run-Time Parameters" on page 10-2
- "Manage Simscape Run-Time Parameters" on page 10-4
- "Change Parameter Values on Target Hardware" on page 11-118
- "Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File" (Simulink Coder)
- "How Acceleration Modes Work"
- "Get Started with Fast Restart"
- "How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ" on page 10-10

# Troubleshoot Simscape Run-Time Parameter Issues

Below, you can find the solutions to some common issues you may encounter while getting started with run-time parameters.

## Simscape Run-Time Parameter Setting Not Visible

If a Simscape parameter is run-time configurable, its **Configurability** drop-down will show either `Compile-time` or `Run-time`. If the **Configurability** setting is not visible, then the parameter is strictly compile-time configurable. For a detailed explanation of how to enable run-time parameters, see "Specify Simscape Run-Time Parameters" on page 10-4.

## Simulation Does Not Respond to Simscape Run-Time Parameter Change

If you use an expression to define a variable for a Simscape run-time parameter, you might need to recompile the model. When Simulink Coder encounters an unsupported expression, it only codes the current numerical value, and you will see a warning. For information on the limitations for using expressions for run-time configurable parameters, see "Limitations for Block Parameter Tunability in Generated Code" (Simulink Coder).

If you change the value of a Simscape run-time parameter during a simulation, the change will take effect the next time you run the simulation. To see how to change the value of a Simscape run-time parameter, see "Specify and Change a Simscape Run-Time Parameter" on page 10-6 and "Change Parameter Values on Target Hardware" on page 11-118.

## See Also

## More About

- "About Simscape Run-Time Parameters" on page 10-2
- "Manage Simscape Run-Time Parameters" on page 10-4
- "Specify and Change a Simscape Run-Time Parameter" on page 10-6
- "Change Parameter Values on Target Hardware" on page 11-118
- "How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ" on page 10-10

# How Simscape Run-Time Parameters and Simulink Tunable Parameters Differ

Simscape run-time parameters and Simulink tunable parameters both allow you to change the values of parameters on your development or target computer without model recompilation. However, they differ in these important ways:

- You can change the value of a Simulink tunable parameter while a simulation is running, and it will impact the currently running simulation. Simscape run-time parameters are run-time configurable. You can only change the value of a run-time configurable parameter when a simulation is stopped.

- Simulink tunable parameters are tunable by default. Simscape block parameters are only compile-time configurable by default. To make a Simscape block parameter run-time configurable, you must specify it as such.

- For code generation, you specify **Default parameter behavior** as `Tunable` or `Inlined`. You cannot modify inlined parameters in generated code because the compiler specifies them as constants. You can change the values of tunable parameters in generated code because the compiler specifies them as modifiable global variables or structure fields.

  If you set the **Default parameter behavior** to `Tunable`, the compiler specifies all Simscape run-time parameters and Simulink tunable parameters as modifiable entities in the generated code. However, if you set the default behavior to `Inlined`, the compiler inlines only the Simscape run-time parameters. The Simulink tunable parameters are still generated as modifiable entities in the code. To change the value of a particular Simscape run-time parameter in the generated code when the default behavior is inlined, you declare that parameter as an exception to inlining.

The table shows the state, mode, and code section in which you can change a run-time parameter or run-time configurable parameter.

| Machine | Simulink Simulation Mode | Simulation Status | Section of Generated Code That You Modify | Simscape Run-Time Parameter Is Modifiable | Simulink Tunable Parameter Is Modifiable |
|---|---|---|---|---|---|
| Development | Normal | Stopped | Not Applicable | Yes | Yes |
| Development | Normal | Running | Not Applicable | No | Yes |
| Development or Target | Normal, Accelerator, Rapid Accelerator, SIL, PIL, or External | Stopped | Not Applicable | Yes | Yes |
| Development or Target | Normal, Accelerator, Rapid Accelerator, SIL, PIL, or External | Running | Not Applicable | No | Yes |
| Target | Normal, SIL, PIL, or External | Stopped | Setup-Function | Yes | Yes |

| Machine | Simulink Simulation Mode | Simulation Status | Section of Generated Code That You Modify | Simscape Run-Time Parameter Is Modifiable | Simulink Tunable Parameter Is Modifiable |
|---|---|---|---|---|---|
| Target | Normal, SIL, PIL, or External | Running | • Step-Function for a Simulink global variable<br>• External code for a Simulink parameter object | No | Yes |

Between normal mode simulations, as long as your changes do not affect the structure of the model, you can avoid recompiling by using fast restart when you change Simscape run-time and Simulink tunable parameters.

## See Also

## More About

- "Change Parameter Values on Target Hardware" on page 11-118
- "Tune and Experiment with Block Parameter Values"
- "Specify and Change a Simscape Run-Time Parameter" on page 10-6
- "Code Regeneration in Accelerated Models"
- "About Simscape Run-Time Parameters" on page 10-2
- "Manage Simscape Run-Time Parameters" on page 10-4
- "Decrease Computational Cost by Inlining Simscape Run-Time Parameters" on page 10-14

# Improve Parameter-Sweeping Efficiency Using Simscape Run-Time Parameters

Simscape run-time parameters are run-time configurable. They allow you to forgo recompiling if you change parameter values between iterative simulations during parameter surveys. You can only test a single value for a compile-time configurable parameter without recompiling your model.

You can benefit from this advantage when you perform parameter sweeps using fast restart, model referencing, or code generation. Code generation allows you to update Simscape run-time parameters between simulation runs using:

- Rapid simulation (RSim) on development or target hardware
- Real-time simulation on target hardware

## Model Referencing with Run-Time Configurable Parameters

You can use Simulink Model blocks to represent one model within another. Each instance of a Model block represents a reference to another model, called a referenced model. For simulation and code generation, the referenced model effectively replaces the Model block that references it. To change the behavior of a referenced model without recompiling, specify a Simscape run-time parameter value in the referenced model using either global parameters or model arguments.

For information on using and parameterizing referenced models, see "Model Reference Basics" and "Parameterize Instances of a Reusable Referenced Model".

## Code Generation with Run-Time Configurable Parameters

Simscape run-time parameters allow you to test your design over a range of values for plant parameters without recompiling or redeploying code. You can:

- Change the value of a Simscape run-time parameter in both your plant model and in your generated code on your development computer for a rapid or real-time simulation.
- Update a run-time configurable parameter in your deployed code before you run your simulation as an executable on an external target machine.

For an example that uses Simscape run-time parameters for real-time simulation see "Change Parameter Values on Target Hardware" on page 11-118.

**Note** Rapid simulation (RSim) uses portions of the Simulink Coder product to create an executable. These modes replace the interpreted code normally used in Simulink simulations, which shortens model run time. Although RSim uses Simulink Coder code generation technology, you do not need Simulink Coder software on your development computer to accelerate your model with RSim. For more information, see "Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File" (Simulink Coder).

## Fast Restart with Simscape Run-Time Parameters

Changing parameter values does not require you to recompile the model between simulation runs unless the changes alter the model structurally. However, when you use normal mode simulation

without fast restart, each simulation compiles the model. The compiling occurs even if the new values do not change the structure of the model and each recompile increases the overall simulation time.

With Simulink fast restart, you can modify Simscape run-time parameters from the workspace variable, without recompiling. For an example that shows how to specify a Simscape run-time parameter and change the parameter value using fast restart, see "Specify and Change a Simscape Run-Time Parameter" on page 10-6.

## See Also

## More About

- "About Simscape Run-Time Parameters" on page 10-2
- "Manage Simscape Run-Time Parameters" on page 10-4
- "Change Parameter Values on Target Hardware" on page 11-118
- "Get Started with Fast Restart"
- "Decrease Computational Cost by Inlining Simscape Run-Time Parameters" on page 10-14
- "Model Reference Basics"
- "How Fast Restart Improves Iterative Simulations"
- "Choosing a Simulation Mode"
- "Code Regeneration in Accelerated Models"
- "Accelerate, Refine, and Test Hybrid Dynamic System on Host Computer by Using RSim System Target File" (Simulink Coder)

# Decrease Computational Cost by Inlining Simscape Run-Time Parameters

Simscape run-time parameters tend to increase the complexity of your code and, therefore, the computational cost of simulating your model. While the additional cost is not typically problematic for desktop simulation, it can result in a CPU overload during real-time simulation.

Consider inlining Simscape run-time parameters in your model, if your simulation:

- Generates small steps during model preparation. For information, see "Real-Time Model Preparation Workflow" on page 11-4.
- Has a long execution time when you are configuring your model for real-time simulation. For information, see "Real-Time Simulation Workflow" on page 11-72.
- Overruns during real-time simulation on target hardware.

For information on inlining Simscape run-time parameters, see "Manage Simscape Run-Time Parameters" on page 10-4.

## See Also

## More About
- "Estimate Computation Costs" on page 11-87
- "Reduce Computation Costs" on page 11-25
- "Fixed-Cost Simulation for Real-Time Viability" on page 11-71
- "Improving Speed and Accuracy" on page 11-8
- "About Simscape Run-Time Parameters" on page 10-2
- "Manage Simscape Run-Time Parameters" on page 10-4
- "Change Parameter Values on Target Hardware" on page 11-118

# Real-Time Simulation

# Model Preparation Objectives

| **In this section...** |
| --- |
| "Obtain Reference Results" on page 11-2 |
| "Determine Step Size" on page 11-2 |
| "Adjust Model Fidelity or Scope" on page 11-2 |

The main goal of model preparation is to ensure that your model is real-time capable. Your model is real-time capable if it is both:

- Accurate enough to generate simulation results that match your expectations, as based on theoretical models and empirical data
- Fast enough to run on your real-time target machine without overruns

During model preparation, you obtain reference results and determine step size to assess the likelihood that your model is real-time capable. If it is unlikely that your model is real-time capable, you adjust the model scope or fidelity to make real-time simulation with your model feasible.

## Obtain Reference Results

Moving your model from desktop simulation to real-time simulation is an iterative process that can require extensive model reconfiguration. During model preparation, you obtain reference results from a variable-step simulation of your original model. These results provide a baseline against which you can judge the accuracy of your modified models.

## Determine Step Size

In terms of speed, the only way to know if your model is real-time capable is to test for overruns while simulating on real-time hardware. You can, however, analyze solver execution speed using desktop simulation to determine if your model is probably fast enough for real-time simulation. You do so by analyzing the steps of a variable-step solver to find the maximum step size to use for sufficiently accurate real-time simulation results. If the required step size appears small enough to cause an overrun on your real-time hardware, you increase the step size by improving simulation speed.

## Adjust Model Fidelity or Scope

You can adjust the fidelity or scope of your model to increase speed or accuracy. Adjustments include:

- Deleting or adding blocks or modifying block parameters to eliminate or reduce the effects of elements that introduce numerical stiffness or cause discontinuities. Simulations take small steps to calculate accurate solutions for these types of elements.
- Modifying elements or parameters to increase simulation efficiency. For example, simplify graphics that require excessive processing power or including lookup tables instead of utilizing processing power to calculate known values.
- Partitioning independent networks of the model to enable parallel processing.

You can also adjust solver settings to help to make your model real-time capable. For real-time simulation on target hardware, you use a fixed-step, fixed-cost solver that bounds the computation cost, that is, the time the solver takes to execute each time step. You configure the solver parameters

before deploying it to a real-time target machine. The fixed-step solver settings that you adjust to improve the real-time viability of your model include step size, solver type, and number of iterations.

Due to the number of options, it is challenging to find the right combination of model size, model fidelity, and solver parameters to achieve real-time simulation. The relationship between speed and accuracy also makes it hard to find both system and solver configurations that help to make your model real-time capable. If you increase speed, you are likely to decrease accuracy. Conversely, increasing accuracy tends to decrease speed. It is especially difficult to achieve acceptable speed and accuracy if you try to adjust model fidelity and scope while you are changing fixed-step solver settings. A better approach to find the optimal configuration is to change only one or two related settings, analyze how those changes affect simulation speed and accuracy, and then make other adjustments.

The real-time model preparation and the real-time simulation workflows separate the configuration changes into two different step-wise processes. For the real-time model preparation workflow, you adjust only the size or fidelity of your model and use variable-step simulation to analyze the effects of your changes. For the real-time simulation workflow, you adjust only the solver parameters and you use fixed-step, fixed-cost simulation to analyze how the changes affect the speed and accuracy of your model.

## See Also

## Related Examples
- "Determine Step Size" on page 11-12
- "Estimate Computation Costs" on page 11-87
- "Reduce Computation Costs" on page 11-25
- "Reduce Fast Dynamics" on page 11-29
- "Reduce Numerical Stiffness" on page 11-47
- "Reduce Zero Crossings" on page 11-55
- "Partition a Model" on page 11-64
- "Manage Model Variants" on page 11-70

## More About
- "Improving Speed and Accuracy" on page 11-8
- "Fixed-Cost Simulation for Real-Time Viability" on page 11-71
- "Real-Time Model Preparation Workflow" on page 11-4
- "Real-Time Simulation Workflow" on page 11-72

# Real-Time Model Preparation Workflow

| In this section... |
|---|
| "Prepare Your Model for Real-Time Simulation" on page 11-5 |
| "Insufficient Computational Capability for Workflow Completion" on page 11-6 |

The figure shows the real-time model preparation workflow.
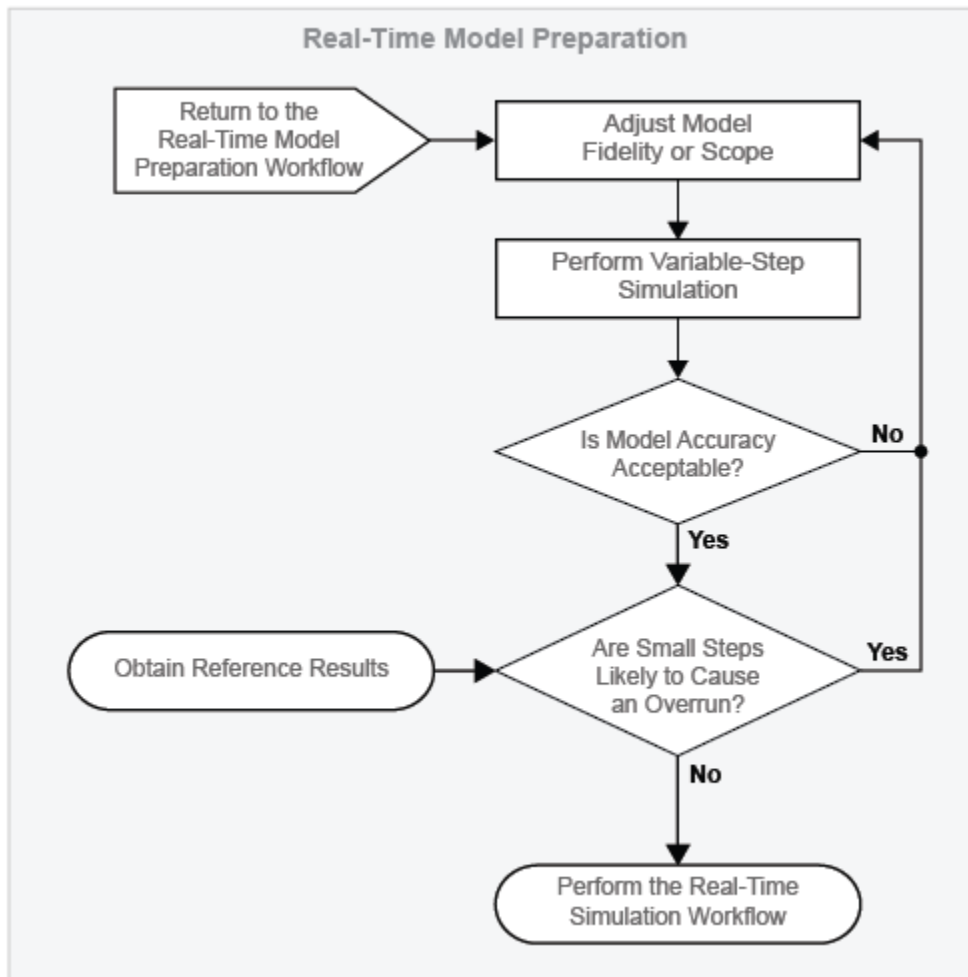


A real-time-capable model is both fast and accurate. When you simulate a real-time-capable model on your real-time target machine, it runs to completion and generates results that match your expectations, as based on theoretical models and empirical data. The only way to determine if your model is real-time capable is to run it on your real-time target machine. You can, however, use desktop simulation, that is, simulation on your development computer, to determine the likelihood that your model is real-time capable before you deploy it.

The real-time model preparation workflow is the first of two workflows that you perform on your development computer to make it more probable that your model is real-time capable. The workflow shows you how to adjust the size or fidelity of your model to improve speed without sacrificing and

accuracy. When you finish this workflow, use the real-time simulation workflow to find the best fixed-step, fixed-cost solver configuration to use for simulating your model in real time.

## Prepare Your Model for Real-Time Simulation

### Obtain Reference Results

Use empirical or theoretical data to design and build your Simscape model. Use a Simulink global variable-step solver to simulate your model. Refine your model as needed to obtain simulation results that the underlying data supports. Reference results provide a baseline to assess model accuracy against throughout all stages of the model preparation and real-time simulation workflows.

### Evaluate Overrun Risk

An overrun occurs when the step size is too small to allow the real-time computer to complete all the processing required for any one step. If your model requires a step size that is so small that it is likely to cause an overrun, then your model is not fast enough for real-time simulation. To determine if small steps are likely to cause an overrun, create a plot of the size of the steps that the variable-step step solver uses to execute the simulation of your model. The step size plot tells you the number and size of the small steps that the solver uses during the simulation.

There are no hard metrics for the size or number of small steps that are likely to cause a real-time-simulation overrun. Moving your model from desktop simulation to real-time simulation is an iterative process. The process, which involves modifying, simulating, and analyzing your model, helps you to determine if the small steps in your model are time limiting or numerous enough to force an overrun.

Experience that you gain by simulating different models on your real-time machine can also help you decide if the small steps in your model are likely to force an overrun. For example, consider a case with two models, M1 and M2, and two different real-time processors, RT1 and RT2. Processors RT1 and RT2 have the same nominal processing speed. Models M1, a mechanical model, and M2, an electrical model, both have a few steps that are 1e-15 seconds. It is possible for model M1 to simulate with sufficiently accurate results on real-time processor RT1, but to incur an overrun or simulate with insufficiently accurate results on real-time processor RT2. It is also possible that model M1 runs to completion with accurate results on RT1 and RT2, whereas model M2 generates an overrun on both processors. These scenarios are possible because the distinct model topologies yield different dynamics and because nominal processing speed is not the only determinant for simulation execution time. Other factors such as the operating system and I/O configuration also affect how simulation execution proceeds on a real-time processor. Familiarity with system dynamics and the processing power of your real-time equipment can guide your decision making when you assess the impact of small step sizes on the real-time viability of a model.

### Adjust Model Fidelity or Scope

Modify the model to increase speed or accuracy if your analysis indicates that real-time simulation with the model is likely to have an overrun or yield insufficiently accurate results.

When you evaluate overrun risk, if you find that the simulation uses too many small steps, use these approaches to improve simulation speed:

- Reduce computation costs.
- Reduce numerical stiffness.
- Reduce zero crossings.

- Reduce fast dynamics
- Partition the model for parallel processing.

When you evaluate model accuracy, if you find that the simulation results do not match the reference results, use these approaches to improve model accuracy:

- Simulink best practices for modeling dynamic systems
- Simscape essential modeling techniques

**Obtain Results with a Variable-Step Solver**

Using a Simulink global variable-step solver, obtain results for the modified version of your model.

The step size plot also helps you to:

- Estimate the maximum step size to use for the fixed-step solver to achieve accurate results during real-time simulation.
- Identify the exact times when discontinuities or fast dynamics slow down the simulation.

**Evaluate Model Accuracy**

Compare the results from simulating on the target computer to your reference results. Are the reference and modified model results the same? If not, are they similar enough that the empirical or theoretical data also supports the results from the simulation of the modified model? Is the modified model representing the phenomena that you want it to measure? Is it representing those phenomena correctly? If you plan on using your model to test your controller design, is the model accurate enough to produce results that you can rely on for system qualification? The answers to these questions help you to decide if your real-time results are accurate enough.

**Perform Real-Time Simulation Workflow**

When variable-simulation results indicate that your model has the speed and accuracy required for real-time processing, you can use the "Real-Time Simulation Workflow" on page 11-72 to configure your model for fixed-step, fixed-cost simulation.

**Return to the Real-Time Model Preparation Workflow**

The connector is an entry point for returning to the real-time model preparation workflow from another workflow (for example, the real-time simulation workflow or the hardware-in-the-loop simulation workflow).

## Insufficient Computational Capability for Workflow Completion

It is possible that your real-time target machine lacks the computational capability for running your model in real time. If, after multiple iterations of the workflow, there is no level of model complexity that makes your model real-time capable, consider these options for increasing processing power:

- "Upgrading Target Hardware" on page 11-10
- "Simulating Parts of the System in Parallel" on page 11-10

## See Also

### Related Examples

- "Determine Step Size" on page 11-12
- "Estimate Computation Costs" on page 11-87
- "Reduce Computation Costs" on page 11-25
- "Reduce Fast Dynamics" on page 11-29
- "Reduce Numerical Stiffness" on page 11-47
- "Reduce Zero Crossings" on page 11-55
- "Partition a Model" on page 11-64
- "Manage Model Variants" on page 11-70

### More About

- "Essential Physical Modeling Techniques" on page 1-12
- "Hardware-In-The-Loop Simulation Workflow" on page 11-106
- "Improving Speed and Accuracy" on page 11-8
- "Model Preparation Objectives" on page 11-2
- "Real-Time Simulation Workflow" on page 11-72

# Improving Speed and Accuracy

| **In this section...** |
|---|

## Why Speed and Accuracy Matter for Real-Time Simulation

Speed and accuracy are the determining factors for making your model real-time capable. Your model is real-time capable if it satisfies both of these conditions when you simulate it on your particular target hardware:

- There are no overruns.
- The simulation results meet your criteria for accuracy.

Speed is objective. The real-time clock determines whether your model is fast enough for real-time simulation. For each step that your solver takes, your real-time hardware system tracks the time that it takes to complete these processing tasks:

- Execute the simulation.
- Process input and output.
- Perform general computer tasks.

An overrun occurs when, for any time step, the time that it takes your system to complete the processing tasks exceeds the real-time limit for the tasks. If your target machine reports any overruns when you use it to simulate your model, your model is not fast enough for real-time simulation.

Your Simscape model is accurate if it produces results that agree with the empirical and theoretical data that are the basis for your model. Accuracy is more subjective when the foundation and simulation data are similar, but are not in absolute agreement. To determine if your model is accurate enough for real-time simulation when the data do not match perfectly, consider these questions:

- Is the model representing the phenomena that you want it to measure?
- Is it representing those phenomena correctly?
- If you plan to use your model to test your controller design, is the model accurate enough to produce results that you can rely on for system qualification?

The only way to test whether your model is real-time capable is to run it on your actual real-time target hardware using fixed-step, fixed-cost solvers. You can, however, estimate whether the model is both fast and accurate enough for real-time simulation by analyzing the results from desktop simulation. To estimate whether your model is real-time capable, see "Determine Step Size" on page 11-12 and "Estimate Computation Costs" on page 11-87.

If the analysis from the desktop simulation indicates that your model likely is not real-time capable, increase model speed or accuracy before deploying your model to your real-time target machine.

Increasing the speed of your simulation tends to decrease the accuracy, and conversely increasing accuracy decreases speed. To make your model real-time capable, maintain a balance between speed and accuracy.

## Balancing Speed and Accuracy

Simulation speed and accuracy correlate to your choices for:

- Model fidelity and scope
- Real-time hardware computing power
- Solver sample time (step size) and number of iterations

To try to increase simulation speed, potentially at the expense of accuracy:

- Decrease model fidelity or scope.
- Increase sample time.
- Decrease the number of solver iterations.

To try to increase simulation accuracy, potentially at the expense of speed:

- Increase model fidelity or scope.
- Decrease sample time.
- Increase the number of solver iterations.

To try to increase both accuracy and speed, or either one without sacrificing the other, increase computing power. To increase computing power, use a faster real-time processor or compute in parallel.

The type of solver that you specify also affects simulation speed and accuracy. For fixed-step simulation, Simscape local solvers are faster and as accurate as Simulink global solvers. Implicit solvers are faster, but less accurate than explicit solvers. However, the numerical stiffness of the network is also a determinant for deciding whether to use an implicit solver or an explicit solver. Explicit solvers yield more accurate results for numerically stiff networks.

For more information on how model complexity affects speed and accuracy, see "Eliminating Effects That Require Intensive Computation" on page 11-9. For more information, on how solver configurations affect speed and accuracy, see "Optimizing Local and Global Solver Configurations" on page 11-10.

It is possible that there is no combination of model complexity and solver settings that can make your model real-time capable. If the simulation does not run in real time on the target machine, or if the accuracy is unacceptable, consider these options for increasing speed and accuracy:

- "Upgrading Target Hardware" on page 11-10
- "Simulating Parts of the System in Parallel" on page 11-10

## Eliminating Effects That Require Intensive Computation

If your desktop simulation analysis indicates that your model likely is not fast enough for real-time simulation, eliminate effects that require intensive computation. Identify elements in your model that cause costly effects, such as discontinuities and rapid changes, that tend to slow down simulations.

Elements that cause discontinuities include:

- Hard stops or backlash
- Stick-slip friction
- Switches or clutches

Elements with small time constants that cause rapid changes include:

- Small masses attached to stiff springs with minimal damping
- Electrical circuits with low capacitance, inductance, and resistance
- Hydraulic circuits with small compressible volumes

To eliminate or modify the elements that are responsible for the effects that slow down your simulation, use these approaches:

- Replace nonlinear components with linearized versions.
- Replace complex equations with lookup tables for their solution.
- Replace complicated components with simplified models.
- Smooth discontinuous functions (step changes) with filters, delays, and other techniques.

## Optimizing Local and Global Solver Configurations

You can also influence the speed and accuracy of your simulation by way of your solver specifications. The level of accuracy that your real-time target machine delivers does not necessarily correlate to a specific step size across all networks in a single model. A real-time target machine can give accurate results for a simple network in your model but inaccurate results for a more complex network. Take advantage of the ability to specify different solver configurations for each network in your Simscape model. To help make your model real-time capable, configure your fixed-step global solver and each local solver individually.

For information on solver options and determining the solvers that help to make your Simscape model real-time capable, see "Solvers for Real-Time Simulation" on page 11-76.

## Upgrading Target Hardware

Different targets give varying levels of accuracy when using the same step size to simulate the same model. You can speed up or increase the accuracy of the real-time simulation by using a faster real-time target computer.

## Simulating Parts of the System in Parallel

Another approach for increasing speed while maintaining accuracy is to configure your model to evaluate multiple physical networks in parallel. You can partition your model if the networks are not dependent upon one another. Work with and experiment with your model, the generated code, and the real-time target machine to use this approach.

## See Also

### Related Examples
- "Determine Step Size" on page 11-12
- "Estimate Computation Costs" on page 11-87
- "Reduce Computation Costs" on page 11-25
- "Reduce Fast Dynamics" on page 11-29
- "Reduce Numerical Stiffness" on page 11-47
- "Reduce Zero Crossings" on page 11-55
- "Partition a Model" on page 11-64

### More About
- "Model Preparation Objectives" on page 11-2
- "Real-Time Model Preparation Workflow" on page 11-4
- "Solvers for Real-Time Simulation" on page 11-76

# Determine Step Size

For the first step in "Real-Time Model Preparation Workflow" on page 11-4, you obtain results from a variable-step simulation of the reference version of your Simscape model. The reference results provide a baseline against which you can assess the accuracy of your model as you modify it. This example shows how to analyze the reference results and the step size that the variable-step solver takes to:

- Estimate the maximum step size that you can use for a fixed-step simulation
- Identify events that have the potential to limit the maximum step size

Discontinuities and rapid changes require small step sizes for accurately capturing these dynamics. The maximum step size that you can use for a fixed-step simulation must be small enough to ensure accurate results. If your model contains such dynamics, then it is possible that the required step size for accurate results, $Ts_{max}$, is too small. A step size that is too small does not allow your real-time computer to finish calculating the solution for any given step in the simulation.

The analysis in this example helps you to estimate the maximum step size that fixed-step solvers can use and still obtain accurate results. You can also use the analysis to determine which elements influence the maximum step size for accurate results. For more information on how obtaining reference results and performing a step-size analysis helps you to prepare your model for real-time simulation, see "Model Preparation Objectives" on page 11-2.

**1** To open the reference model, at the MATLAB command prompt, enter:

```
model = 'ssc_pneumatic_rts_reference';
open_system(model)
```
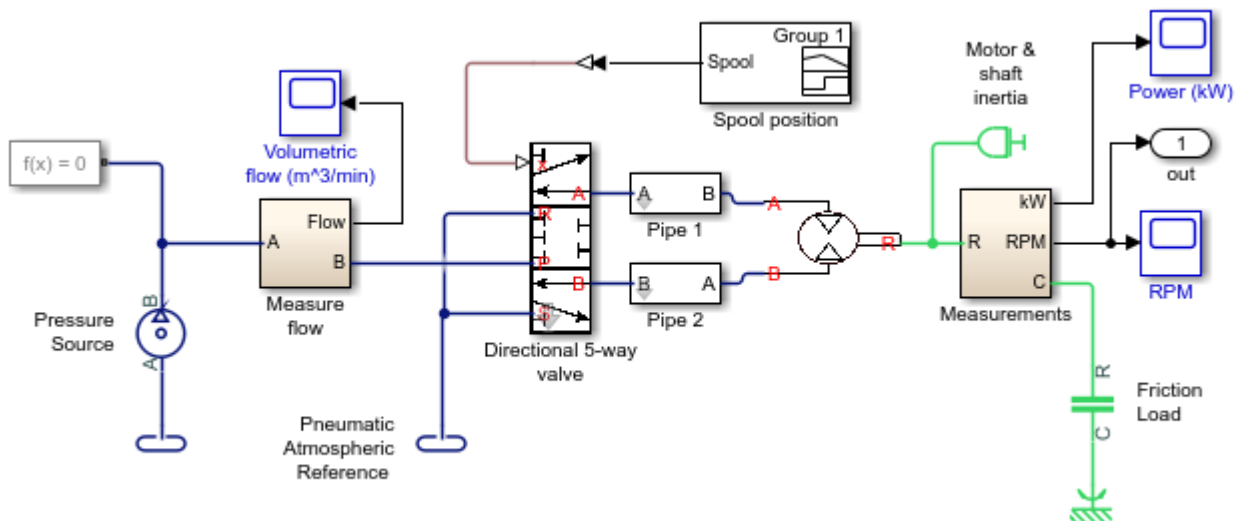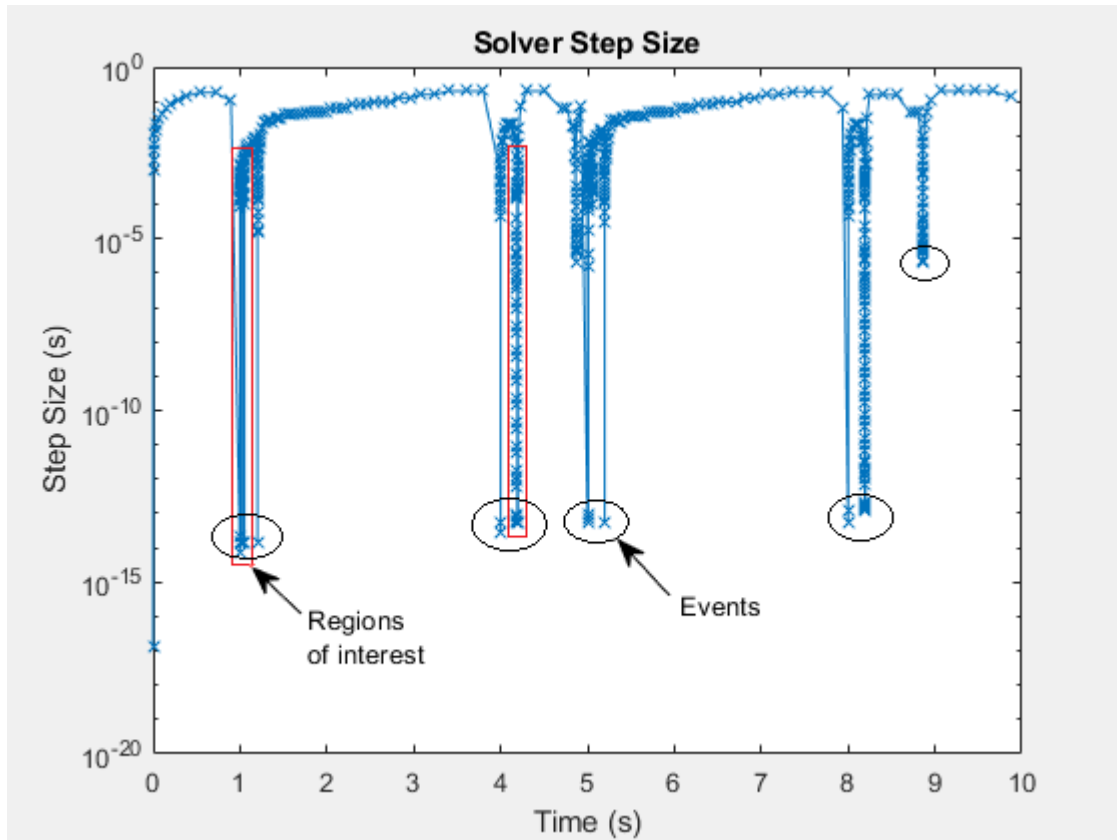


**2** Simulate the model:

```
sim(model)
```

**3** Create a semilogarithmic plot that shows how the step size for the solver varies during the simulation.

```
h1 = figure;
semilogy(tout(1:end-1),diff(tout),'-x')
title('Solver Step Size')
xlabel('Time (s)')
ylabel('Step Size (s)')
```



For much of the simulation, the step size is greater than the value of the $Ts_{max}$ in the plot. The corresponding value, ~0.001 seconds, is an estimated maximum step size for achieving accurate results during fixed-step simulation with the model. To see how to configure the step size for fixed-step solvers for real-time simulation, see "Choose Step Size and Number of Iterations" on page 11-89.

The x markers in the plot indicate the time that the solver took to execute a single step at that moment in the simulation. The step-size data is discrete. The line that connects the discrete points exists only to help you see the order of the individual execution times over the course of the simulation.

A large decrease in step size indicates that the solver detects a zero-crossing event. Zero-crossing detection can happen when the value of a signal changes sign or crosses a threshold. The simulation reduces the step size to capture the dynamics for the zero-crossing event accurately. After the solver processes the dynamics for a zero-crossing event, the simulation step size can increase. It is possible for the solver to take several small steps before returning to the step size that precedes the zero-crossing event. The areas in the red boxes contain variations in recovery time for the variable step solver.
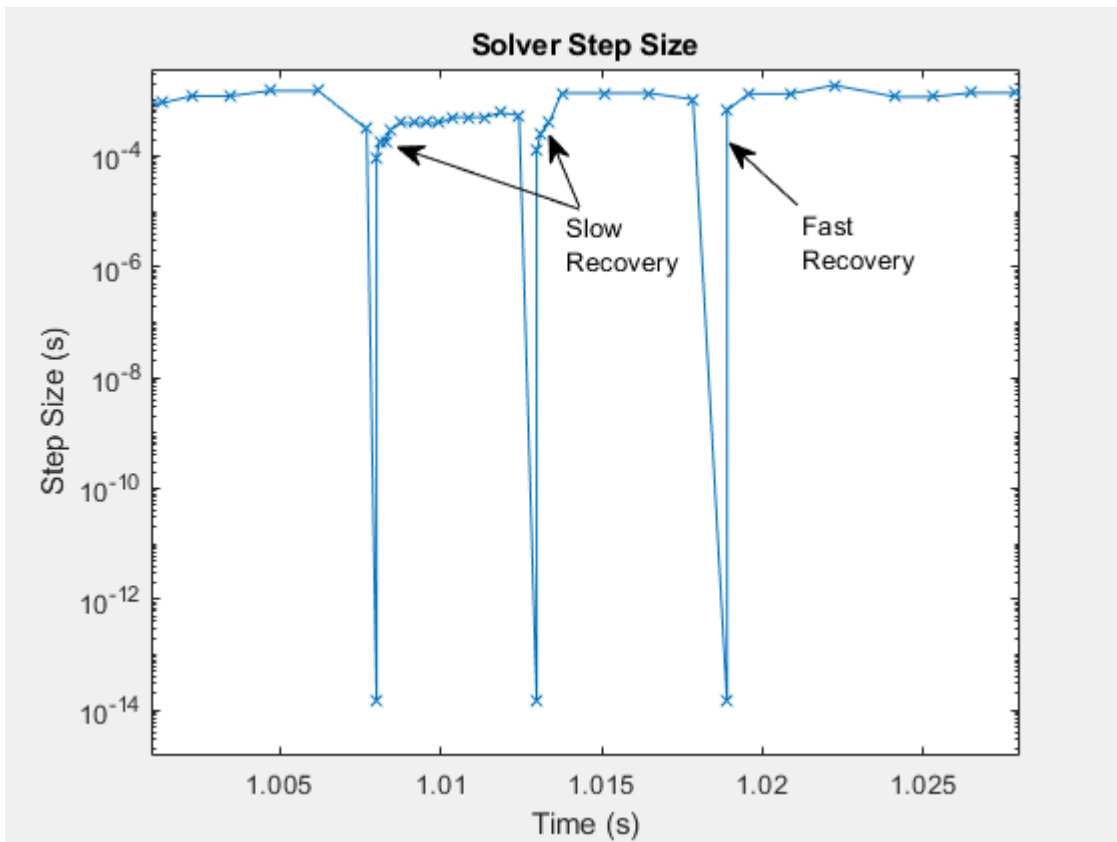
4   To see different post-zero-crossing behaviors, zoom to the region in the red box at time (t) = ~1 second.

**Script for Zooming In**

```
h1;
xStart = 0;
xEnd = 10;
yStart = 0;
yEnd = 10e0;
xZoomStart1 = 1.001;
xZoomEnd1 = 1.028;
yZoomStart1 = 1.5e-15;
yZoomEnd1 = 3.71e-3;
axis([xZoomStart1 xZoomEnd1 yZoomStart1 yZoomEnd1])
```



After $t = 1.005$ seconds, the step size decreases from ~10e-3 seconds to less than 10e-13 seconds to capture an event. The step size increases quickly to ~10e-5 seconds, and then slowly to ~10e-4 seconds. The step size decreases to capture a second event and recovers quickly, and then slowly to the step size from before the first event. The slow rates of recovery indicate that the simulation is using small steps to capture the dynamics of elements in your model. If the required step size limits the maximum fixed-step size to a small enough value, then an overrun might occur when you attempt simulation on your real-time computer.

The types of elements that require small step size are:

- Elements that cause discontinuities, such as hard-stops and stick-slip friction
- Elements that have small time constants, such as small masses with undamped, stiff springs and hydraulic circuits with small, compressible volumes

The step size recovers more quickly after it slows down to process the event that occurs before $t$ = 1.02 seconds. This event is less likely to require small step sizes to achieve accurate results.

**5** To see different types of slow solver recoveries, zoom to the region within the red box at $t$ = ~4.2 seconds.

```
h1;
xZoomStart2 = 4.16;
xZoomEnd2 = 4.24;
yZoomStart2 = 10e-20;
yZoomEnd2 = 10e-1;
axis([xZoomStart2 xZoomEnd2 yZoomStart2 yZoomEnd2]);
```
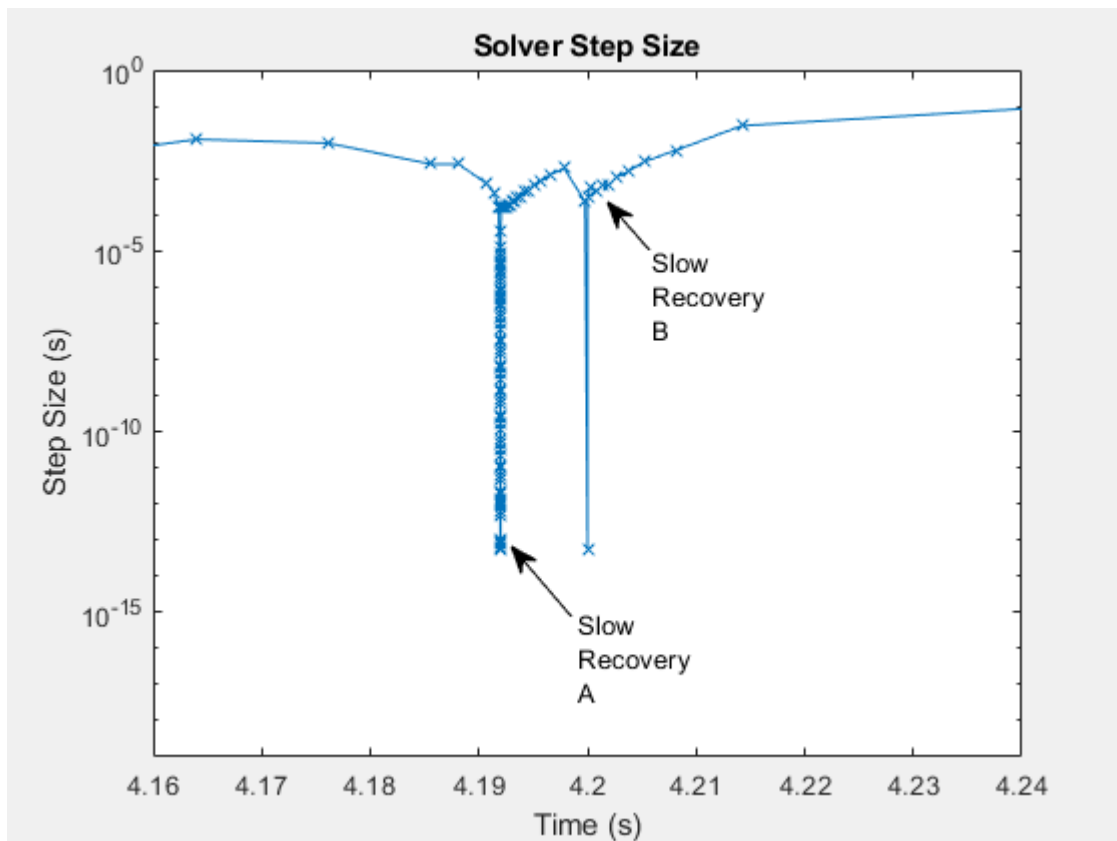


Just as there are different types of events that cause solvers to slow down, there are different types of slow solver recovery. The events that occur just before $t$ = 4.19 and 4.2 seconds both involve zero-crossings. The solver takes a series of progressively larger steps as it reaches the step size from before the event. The large number of very small steps that follow the zero crossing at Slow Recovery A indicate that the element that caused the zero crossing is also numerically stiff.

The quicker step-size increase after the event that occurs at $t$ = 4.2 seconds indicates that the element that caused the zero crossing before Slow Recovery B, is not as stiff as the event at Slow Recovery A.

**6** To see the results, open the Simscape Results Explorer.

```
sscexplore(simlog)
```

**7** Examine the angular speed. In the Simscape Results Explorer window, in the simulation log tree hierarchy, select **Measurements > Ideal Rotational Motion Sensor > [icon]w**.
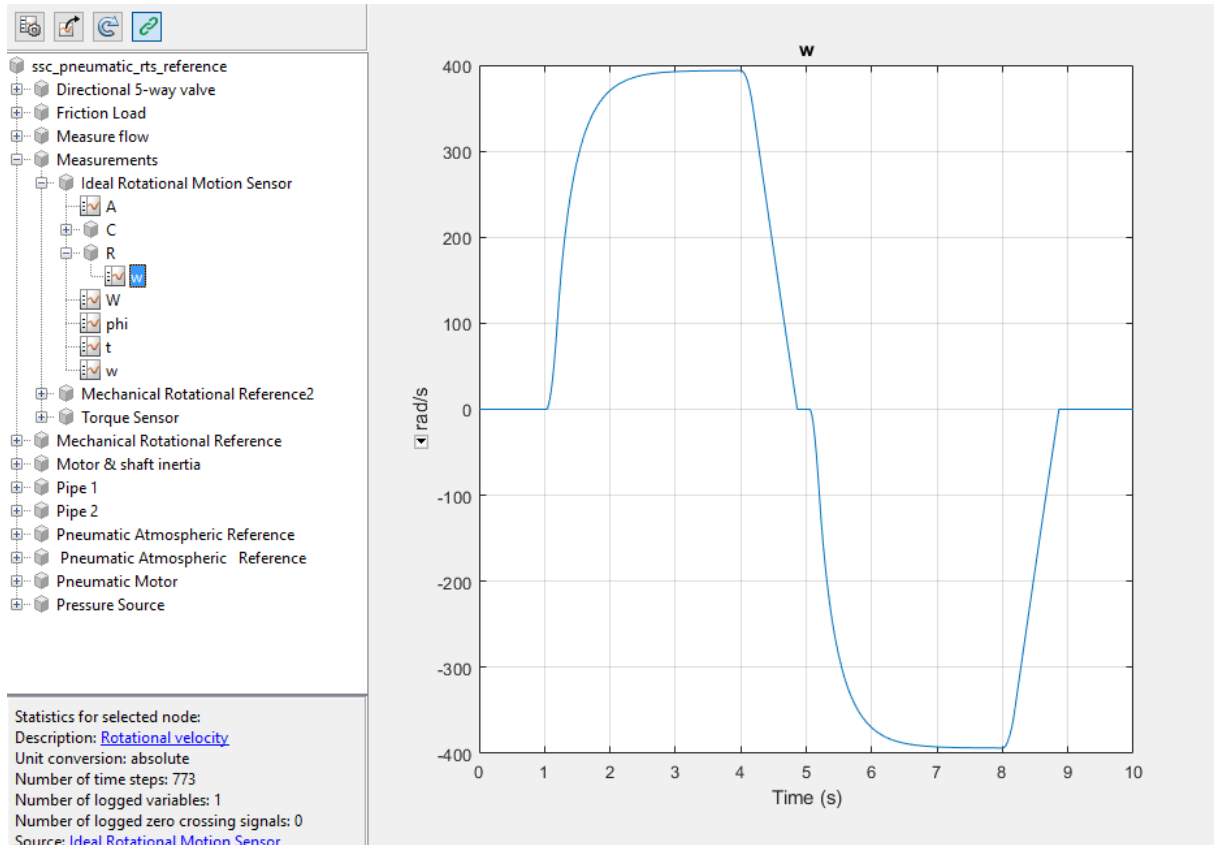


**8** To add a plot of the gas flow, select **Measure Flow > Pneumatic Mass & Heat Flow Sensor** and then, use Ctrl+click to select [icon]**G_ps**.
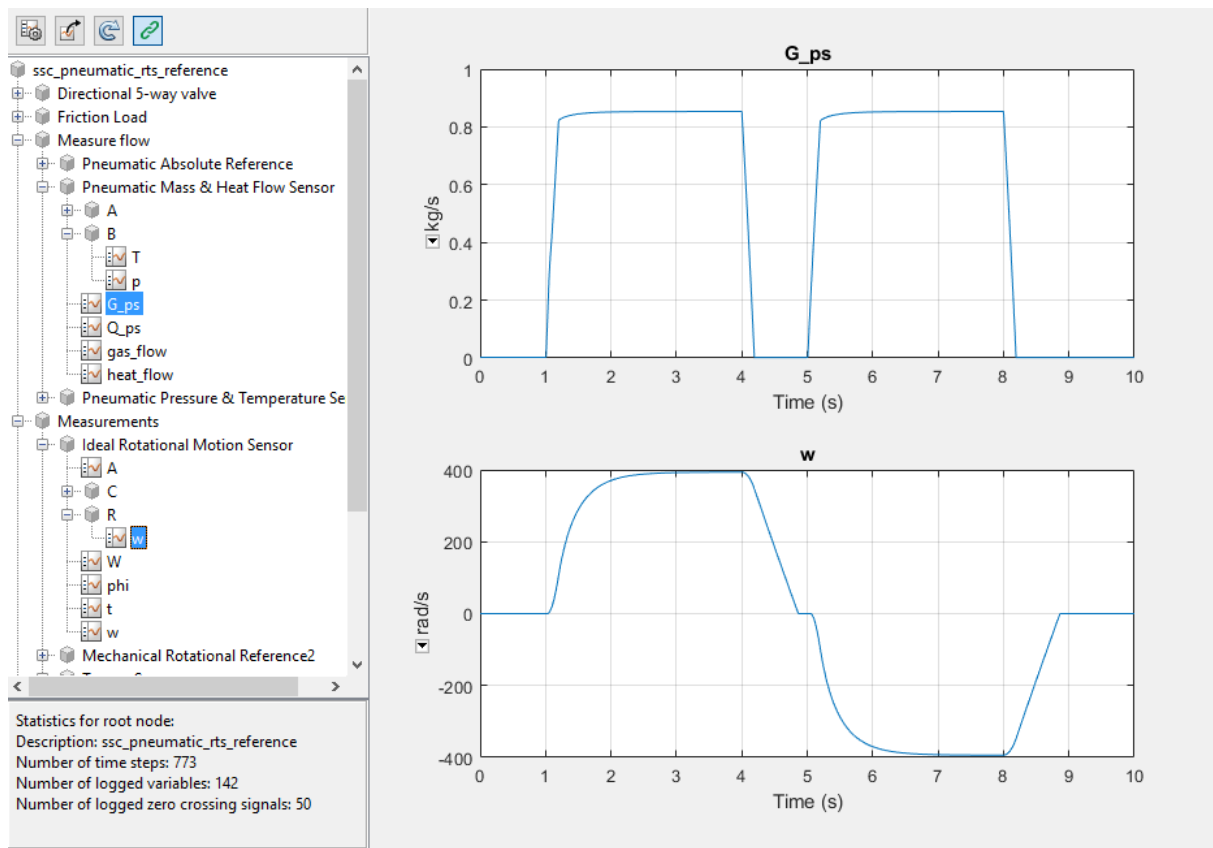
The slow recovery times occur when the simulation initializes, and approximately at $t$ = 1, 4, 5, 8, and 9 seconds. These periods of small steps coincide with these times:

- The motor speed is near zero rpm (simulation time $t$ = ~ 1, 5, and 9 seconds)
- The step change in motor speed is initiated from a steady-state speed to a new speed (time $t$ = ~ 4 and 8 seconds)
- The step change in flow rate is initiated from a steady-state speed to a new flow rate (time $t$ = ~ 4 and 8 seconds)
- The volumetric flow rate is near zero kg/s ($t$ = ~ 1, 4, and 5 seconds)

These results indicate that the slow step-size recoveries are most likely due to elements in the model that involve friction or that have small, compressible volumes. To see how to identify the problematic elements and modify them to increase simulation speed, see "Reduce Numerical Stiffness" on page 11-47 and "Reduce Zero Crossings" on page 11-55.

## See Also
Solver Profiler

## Related Examples
- "Examine Model Dynamics Using Solver Profiler"
- "Estimate Computation Costs" on page 11-87

## More About

# Increase Simulation Speed Using the Partitioning Solver

The Partitioning solver is a Simscape fixed-step local solver that improves performance for certain models by reducing computational cost of simulation. Decreased computational cost yields faster simulation rates for desktop simulation and decreased task execution time (TET) for deployment. The solver converts the entire system of equations for the attached Simscape network into several smaller sets of switched linear equations that are connected through nonlinear functions. Computational cost is reduced because it is more efficient to calculate solutions for several smaller equation systems than it is to calculate the solution for one large system of equations.

The Partitioning solver does not partition models, that is it does not split a model into separate subsystems for multicore processing. To learn how to partition a Simscape model, see "Partition a Model" on page 11-64.

To use the Partitioning solver, open the Solver Configuration block settings and:

**1**   Select the **Use local solver** check box.

**2**   Set the **Solver type** parameter to `Partitioning`.

**3**   Clear the **Start simulation from steady state** check box.

**4**   Set the **Equation formulation** parameter to `Time`.

For real-time simulation, also select the **Use fixed-cost runtime consistency iterations** check box. For more information, see "Make Your Model Real-Time Viable" on page 11-74.

## Limitations

Not all networks can simulate with the Partitioning solver. A simulation that uses the Partitioning solver results in an error if the Simscape network cannot be represented by switched linear equations connected through nonlinear functions. Simulating with the Partitioning solver also yields an error for networks that contain:

- A custom component that uses the Simscape language `delay` operator.
- A block that uses a discrete sample time for periodic events. Examples include the PS Counter, PS Random Number, PS Repeating Sequence, or PS Uniform Random Number blocks from the Simscape/ Physical Signals / Sources library.

Certain Solver Configuration block settings are not compatible with the Partitioning solver. A simulation that uses a Partitioning solver results in an error if the model contains a Solver Configuration block with:

- **Start simulation from steady state** selected
- **Equation formulation** set to `Frequency and time`

## Options

To further improve simulation performance, you can set the **Partition storage method** parameter to `Exhaustive` and specify a value for the **Partition memory budget [kB]** parameter, based on the **Total memory estimate** data in the Statistics Viewer. For more information, see Solver Configuration and "Model Statistics Available when Using the Partitioning Solver" on page 14-19.

## Simulate a Simscape Model Using the Partitioning Solver

This example shows how to compare the speed and the accuracy of a simulation that uses the Partitioning solver to baseline results. It also shows how to compare the speeds of the Partitioning solver and the Backward Euler solver.

1   Open the model. At the MATLAB command prompt, enter the code.

**See Code**

```
model = 'ssc_dcmotor';
open_system(model)
```

2   To return all simulation outputs within a single `Simulink.SimulationOutput` object so that you can later compare simulation times, enable the single-output format of the `sim` command.

```
% Enable single-output format
set_param(model,'ReturnWorkspaceOutputs', 'on')
```

3   Enable the signal that goes to the **Motor RPM** scope block for Simulink data logging and viewing with the Simulation Data Inspector.

**See Code**

```
% Define the w sensor subsystem and the path
%    to it as variables
rpmSensor = 'Sensing';
rpmSensorPath = [model,'/',rpmSensor];

% Mark the output signal from the w sensor subsystem
%   for Simulink(R) data logging
phRpmSensor = get_param(rpmSensorPath,'PortHandles');
set_param(phRpmSensor.Outport(1),'DataLogging','on')
```

The logging badge ⌒• marks the signal in the model.
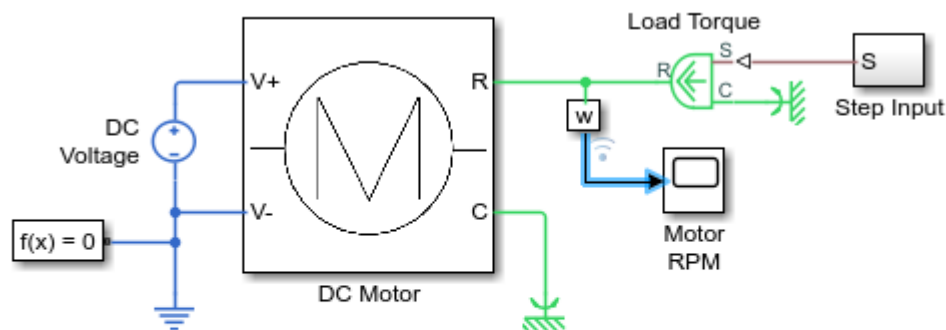


**Permanent Magnet DC Motor**

1. Plot current and load torque (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

**4** Run timed simulations for each of these solvers:

- Variable-step global solver, the original solver for the model
- Fixed-step local Backward Euler solver
- Fixed-step local Partitioning solver

**See Code**

```matlab
% Define the Solver Configuration block and the path
%    to it as variables
solvConfig = 'Solver Configuration';
solvConfigPath = [model,'/',solvConfig];

% Run a timed simulation using the original solver
out = sim(model);
tBaseline  = out.SimulationMetadata.TimingInfo.ExecutionElapsedWallTime;

% Select option Solver Configuration >  Use local solver
set_param(solvConfigPath, 'UseLocalSolver', 'on')

% Set Solver Configuration >  Solver type > Partitioning
set_param(solvConfigPath, 'LocalSolverChoice',...
    'NE_PARTITIONING_ADVANCER')

% Run a timed simulation using the Partitioning solver
out = sim(model);
tPartition = out.SimulationMetadata.TimingInfo.ExecutionElapsedWallTime;

% Set Solver Configuration >  Solver type > Backward Euler
set_param(solvConfigPath, 'UseLocalSolver', 'on',...
    'LocalSolverChoice','NE_BACKWARD_EULER_ADVANCER')

%%
% Run a timed simulation using the Backward Euler solver
out = sim(model);
tBackEuler = out.SimulationMetadata.TimingInfo.ExecutionElapsedWallTime;

% Compute the percent difference for the simulation times
spdPartitionVsBaseline = 100*(tBaseline - tPartition)/tBaseline;
spdBackEulerVsBaseline = 100*(tBaseline - tBackEuler)/tBaseline;
spdPartitionVsBackEuler = 100*(tBackEuler - tPartition)/tBackEuler;

% Reset the solver to the original setting by deselecting Use local solver
set_param(solvConfigPath, 'UseLocalSolver', 'off')

%%
compTimeDiffTable = table({'Baseline';...
    'Partitioning';...
    'Backward Euler'},...
    {tBaseline;tPartition;tBackEuler},...
'VariableNames', {'Solver','Sim_Duration'});

display(compTimeDiffTable);

%%
compPctDiffTable = table({'Partitioning versus Baseline';...
    'Backward Euler versus Baseline';...
    'Partitioning versus Backward Euler'},...
    {spdPartitionVsBaseline;...
    spdBackEulerVsBaseline;...
    spdPartitionVsBackEuler},...
'VariableNames', {'Comparison','Percent_Difference'});

display(compPctDiffTable);

compTimeDiffTable =

  3×2 table

        Solver        Sim_Duration
      _____    _____

      'Baseline'          [0.0319]
```

```
        'Partitioning'        [0.0204]
        'Backward Euler'      [0.0291]

compPctDiffTable =

  3×2 table

                Comparison                      Percent_Difference
    _____        _____

    'Partitioning versus Baseline'                   [35.9128]
    'Backward Euler versus Baseline'                 [ 8.6623]
    'Partitioning versus Backward Euler'             [29.8349]
```

Simulation time on your machine may differ because simulation speed depends on machine processing power and the computational cost of concurrent processes.

The local fixed-step Partitioning and Backward Euler solvers are faster than the variable-step baseline solver. The Partitioning solver is typically, but not always, faster than the Backward Euler solver.

**5** To compare the results, open the Simulation Data Inspector.

**See Code**

```
% Get Simulation Data Inspector run IDs for
%   the last three runs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 2);
runPartition = runIDs(end - 1);
runBackEuler = runIDs(end);

% Open the Simulation Data Inspector
Simulink.sdi.view

compBaselinePartition = Simulink.sdi.compareRuns(runBaseline,...
    runPartition);
```

To see the comparison, click **Compare** and then click **Sensing 1**.
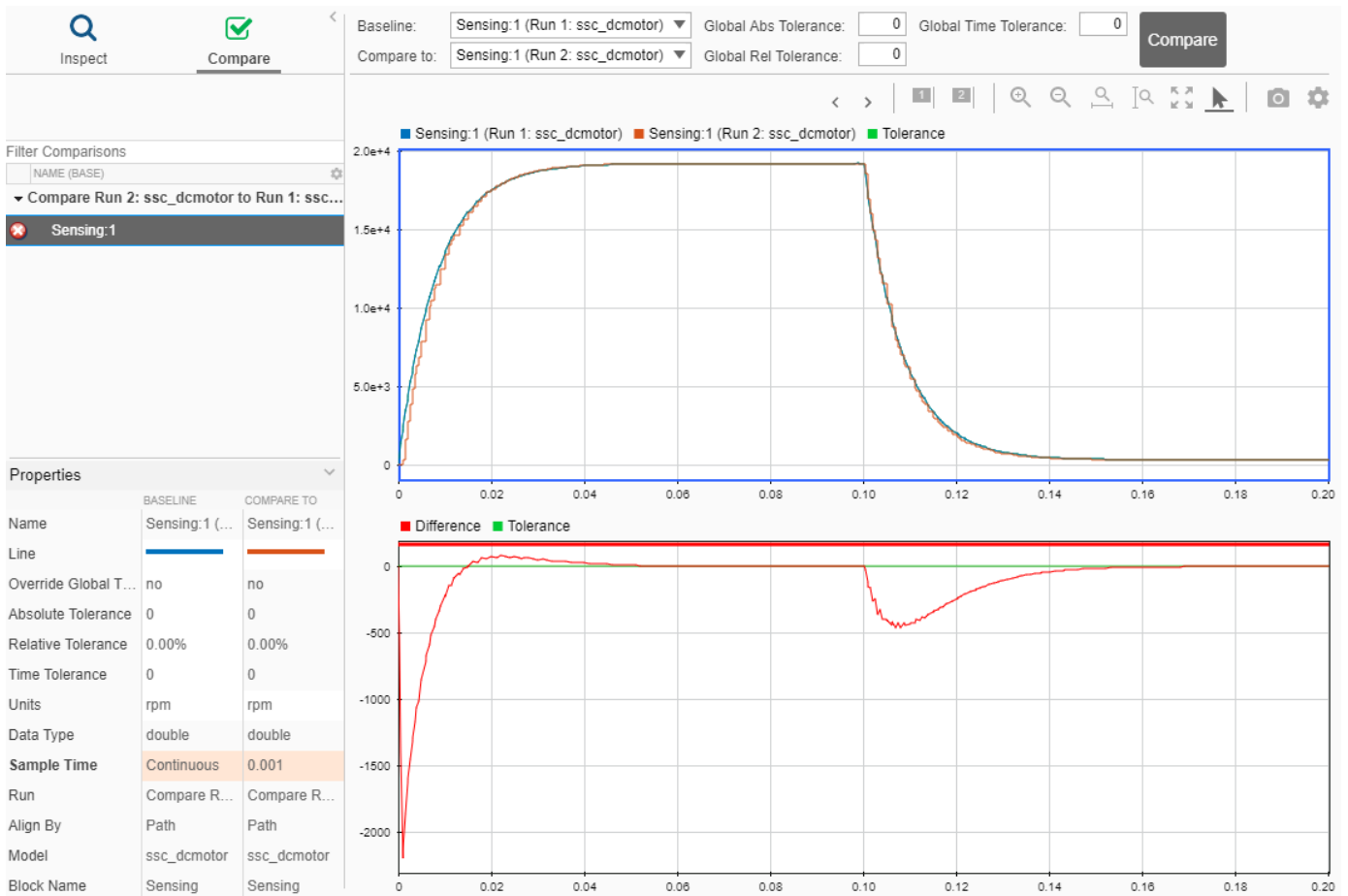
The first plot shows the overlay of the baseline and Partitioning solver simulation results. The second plot shows how they differ. The default tolerance for differences is 0. To determine if the accuracy of the results meet your requirements, you can adjust the relative, absolute, and time tolerances. For more information, see "Compare Simulation Data".

## See Also

### Related Examples

- "Determine Step Size" on page 11-12
- "Estimate Computation Costs" on page 11-87
- "Reduce Fast Dynamics" on page 11-29
- "Reduce Numerical Stiffness" on page 11-47
- "Reduce Zero Crossings" on page 11-55
- "Partition a Model" on page 11-64

### More About

- "Model Statistics Available when Using the Partitioning Solver" on page 14-19

# Reduce Computation Costs

| **In this section...** |
|---|
| "Data Logging and Monitoring Guidelines" on page 11-25 |
| "Improve Data Logging and Monitoring Efficiency" on page 11-25 |
| "Additional Methods for Reducing Computational Cost" on page 11-28 |

Computational cost is a measure of the number and the complexity of tasks that a processor performs per time step during a simulation. Lowering the computational cost of your model increases simulation execution speed and helps you to avoid overruns when you simulate in real time on target hardware.

## Data Logging and Monitoring Guidelines

Data logging and monitoring are interactive procedures that consume memory and processing power. One way to reduce computational cost is to reduce the amount of interactive processing that occurs during simulation. Best practices for limiting computational costs while logging and monitoring data are:

- Use an outport block only if you need to log data for your analysis via the Simulink model on your development computer.
- Use a scope block only if you need to monitor data during real-time simulation via the Simulink model on your development computer.
- If you need to log data or monitor a variable, limit the number or the decimation of data points that you collect whenever your analysis requirements permit you to do so.
- Log data only once.
- If you use Simscape data logging, use local settings to log only the blocks that contain variables that you need for your analysis.

**Note** Simscape simulation data logging is not supported for generated code.

## Improve Data Logging and Monitoring Efficiency

Examine the configuration of the model and the simulation results to determine if the model is logging and monitoring data efficiently.

1. To open the model, at the MATLAB command prompt, enter:

```
model = 'ssc_pneumatic_rts_zc_redux';
open_system(model)
```
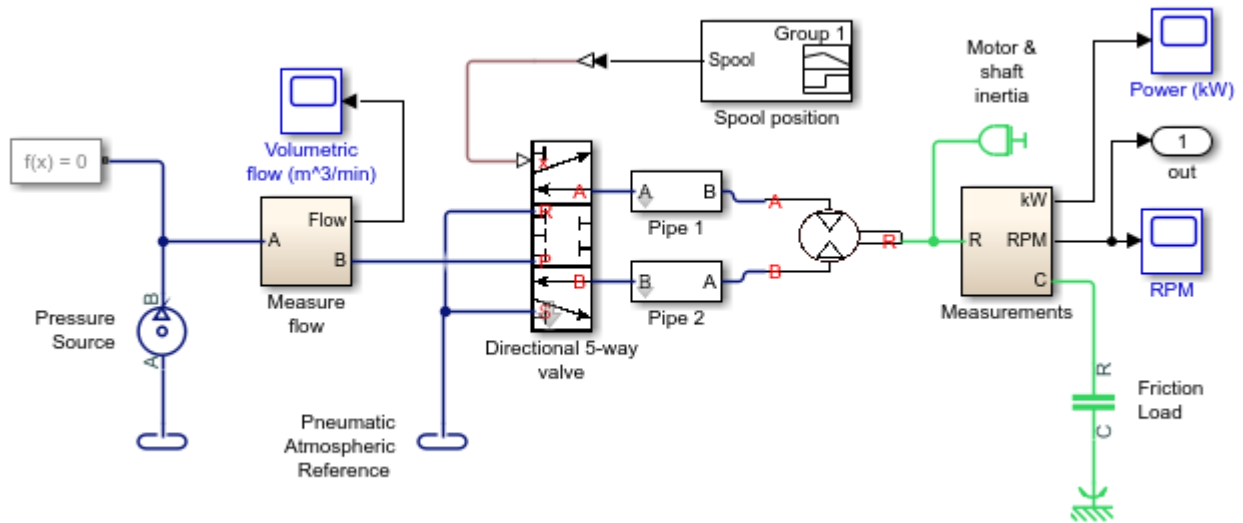
The model contains three scope blocks and one outport block. The Power (kW) scope, RPM scope, and outport block receive data from the Measurements subsystem.

2   Simulate the model:

```
sim(model)
```



The model logs five variables to the workspace, including a Simscape simulation data logging node.

3   To determine the source for the `Pneu_rts_RPM_DATA`, in the MATLAB workspace, open the structure. Alternately, at the command line, enter:

```
Pneu_rts_RPM_DATA.blockName
```

```
ans =

    'ssc_pneumatic_rts_zc_redux/RPM'
```

The `blockName` variable shows that the RPM scope logs the data. In the model, the outport that logs data to `yout` connects to the signal between the Measurements subsystem and the RPM scope block.

4   To compare the data that `Pneu_rts_RPM_DATA` and `yout` log, plot both data sets to a single figure.

```
h1 = figure;
plot(tout,yout)
```

```
h1;
hold on
plot(Pneu_rts_RPM_DATA.time,Pneu_rts_RPM_DATA.signals.values,'r--')
title('Speed')
xlabel('Time (s)')
ylabel('Speed (rpm)')
h1Leg = legend({'yout','Pneu-rts-RPM-DATA'});
```
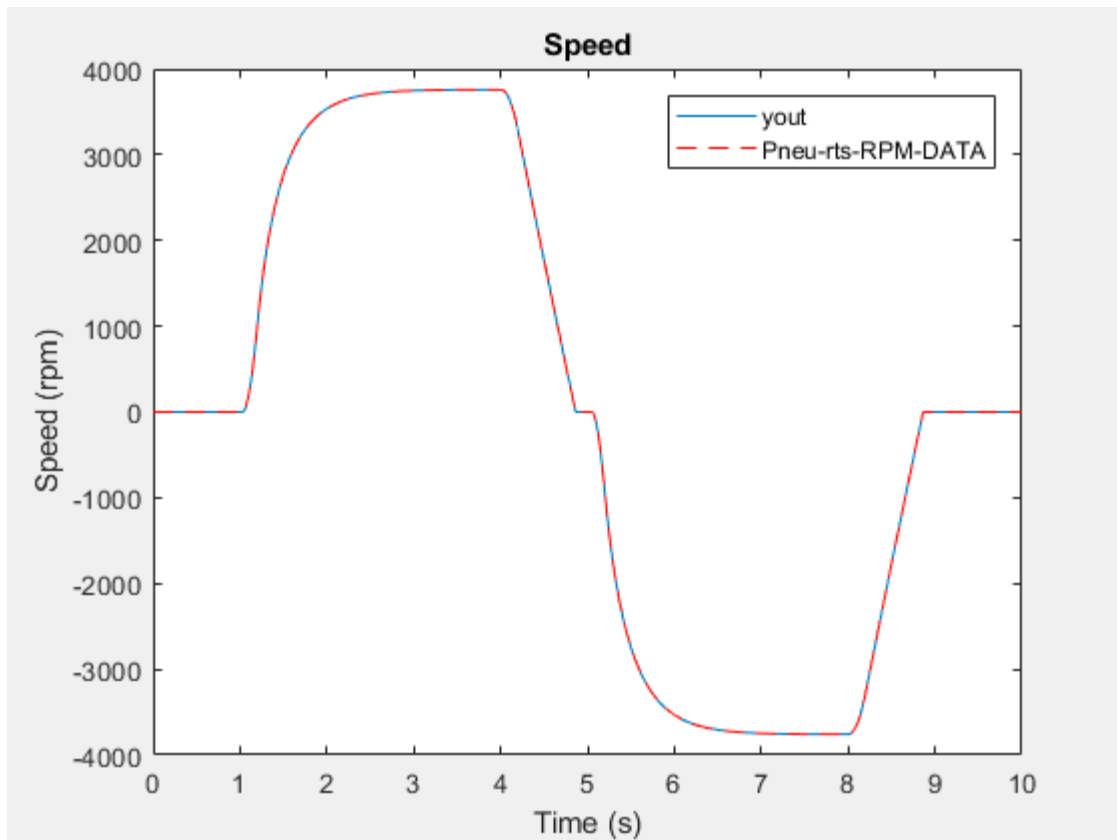


The data is the same, which means that you are logging the same data twice.

To reduce the computational cost for logging or monitoring the speed data via the Simulink model on your development computer during real-time simulation:

*   If you only need to log the speed data, delete the RPM scope block.
*   If you need to log and monitor the speed data, delete the outport block.
*   If you only need to monitor the speed data, delete the outport block and disable data logging for the RPM scope.

If you do not need to log or monitor the speed data via the Simulink model on your development computer during real-time simulation with target hardware, delete both the RPM scope block and the outport block.

If you want to reduce costs by deleting the scope and outport blocks, but you want to log data while you prepare your model for real-time simulation, configure the model to log only the data that you need. To do so, use a `simlog` node in the MATLAB workspace. For information, see "Log Data for Selected Blocks Only" on page 13-5.

## Additional Methods for Reducing Computational Cost

In addition to reducing the number of logged and monitored signals, you can use these methods for decreasing the number and complexity of tasks that the processor performs per time step during simulation:

- Avoid using large images and complex graphics.
- Disable unnecessary error and warning diagnostics.
- Reconfigure tolerances.
- Simplify complex subsystems or replace them with lookup tables.
- Linearize nonlinear effects.
- Eliminate redundant calculations, for example, multiplication by one.
- Reduce the number of differential algebraic equations (DAEs).

## See Also

## Related Examples

## More About

# Reduce Fast Dynamics

You can make your model real-time capable by identifying and reducing sources of fast or high-frequency dynamics. A real-time capable model is one that produces acceptable results on a real-time machine without generating overruns. The example shows you how to identify fast dynamics by examining the frequency response and pole locations of a linearized model. It also shows how to identify and remove sources of the fast dynamics.

## Why Reduce Fast Dynamics

Models with fast dynamics typically have a high computational cost. Removing fast dynamics decreases the computational cost and increases the minimum step size that you can specify for a fixed-step, fixed-cost simulation. Using a larger step size increases the likelihood that your model is real-time capable.

## Frequency-Response Analysis

*Frequency response* describes the steady-state response of a system to sinusoidal inputs. For a linear system, a sinusoidal input results in an output that is a sinusoid with the same frequency, $\omega$, but with a different amplitude and phase, $\theta$.
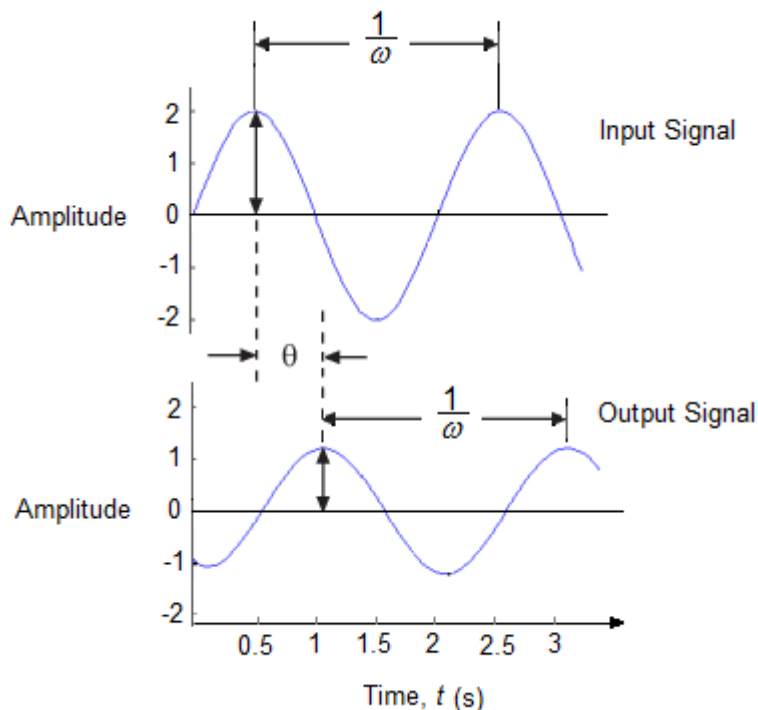


Frequency analysis shows how amplitude and phase change over a given range of frequencies. For a small change in frequency, a large magnitude or phase change indicates that a system has fast dynamics. This example uses Bode plots, which allow you to see how the amplitude, in terms of magnitude in dB, and phase vary as a function of frequency.

## Pole Analysis

Fast poles are also indicative of fast dynamics. Fast poles are poles that respond or oscillate rapidly. Poles that have real components that are far to the left of the imaginary axis on the complex plane have a fast response speed. Complex pole pairs that have imaginary components that are far from the real axis oscillate rapidly. For example, the real pole at -1500 has a faster response speed than the real pole at -1000 and the complex pole pair at -500 ± 1500$i$ has a faster oscillation speed than the complex pole pair at -500 ± 500$i$.
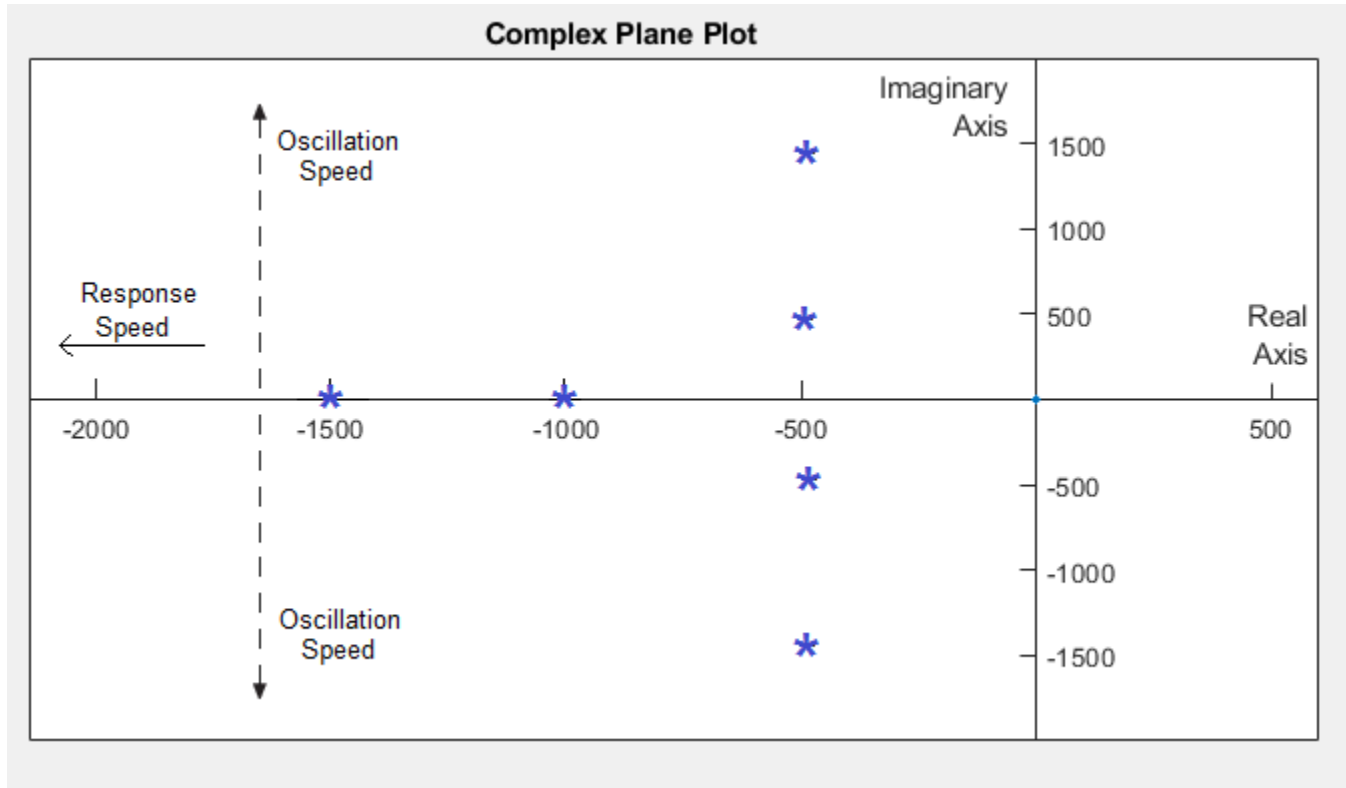


For state-space models, the poles are the eigenvalues of the *A*-matrix. This example shows you how to examine pole speed by determining the state-space model and then, calculating and plotting the eigenvalues of the *A*-matrix values.

## Linearize the Model

The model in this example is not linear. Before performing the frequency-response and pole analyses, trim, that is extract and specify operating points for linearization, and linearize the model.

1    Open and examine the model. At the MATLAB command prompt, enter:

```
%% Open the model
open_system('ssc_hydraulic_actuator_digital_control')
```
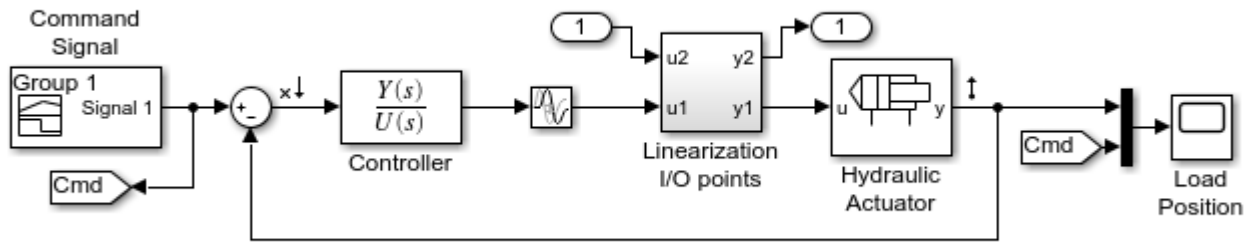
In addition to signal-generation, operation, routing, and visualization blocks, the model contains these blocks:

- **Controller**— A Transfer Fcn block that defines a continuous time representation of the control system. A callback function for the model saves the numerator, *num*, and denominator, *den*, of the transfer function as variables in the workspace.

- — A Transport Delay block to represent the delays associated with computational delay and the sample-and-hold function when deploying a discrete-time implementation of the continuous time control system.

- **Linearization I/O points**— A subsystem that allows you to configure the system as a closed loop, for trimming, or as an open loop, for linearization. The callback function for the model configures the system as closed loop by setting *ClosedLoop* to 1 in the workspace.

- **Hydraulic Actuator** — A subsystem that contains the physical model of the plant.

**2** Find a suitable operating point for linearizing the system. Simulate the model, extract the data from the Simscape logging nodes, then plot and examine the results.

### Script for Simulating the Model and Plotting the Results

```matlab
%% Simulate the model and plot the simulation results
% Simulate the model
sim('ssc_hydraulic_actuator_digital_control')

% Extract simulation results from the Simscape logging nodes
simlog1 = simlog_ssc_hydraulic_actuator_digital_control;
pCylA1 = simlog1.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_A.A.p.series;
pCylB1 = simlog1.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_B.A.p.series;
aValve1 = simlog1.Hydraulic_Actuator.Custom_2_Way_Valve.Calculate_Orifice_Area.PS_Saturation.O.series;

% Plot simulation results
h3_ssc_hydraulic_actuator_digital_control=figure('Name',...
    'ssc_hydraulic_actuator_digital_control',...
    'Outerposition', [1244 673 576 513]);

% Plot pressures
ah(1) = subplot(2,1,1);
plot(pCylA1.time,pCylA1.values,'LineWidth',6,'Color','b');
hold on
plot(pCylB1.time,pCylB1.values,'LineWidth',3,'Color','b');
hold off
grid on
ylabel('Pressure (Pa)');
title('Cylinder Pressures');
legend({'Side A','Side B'},'Location','Best');
```
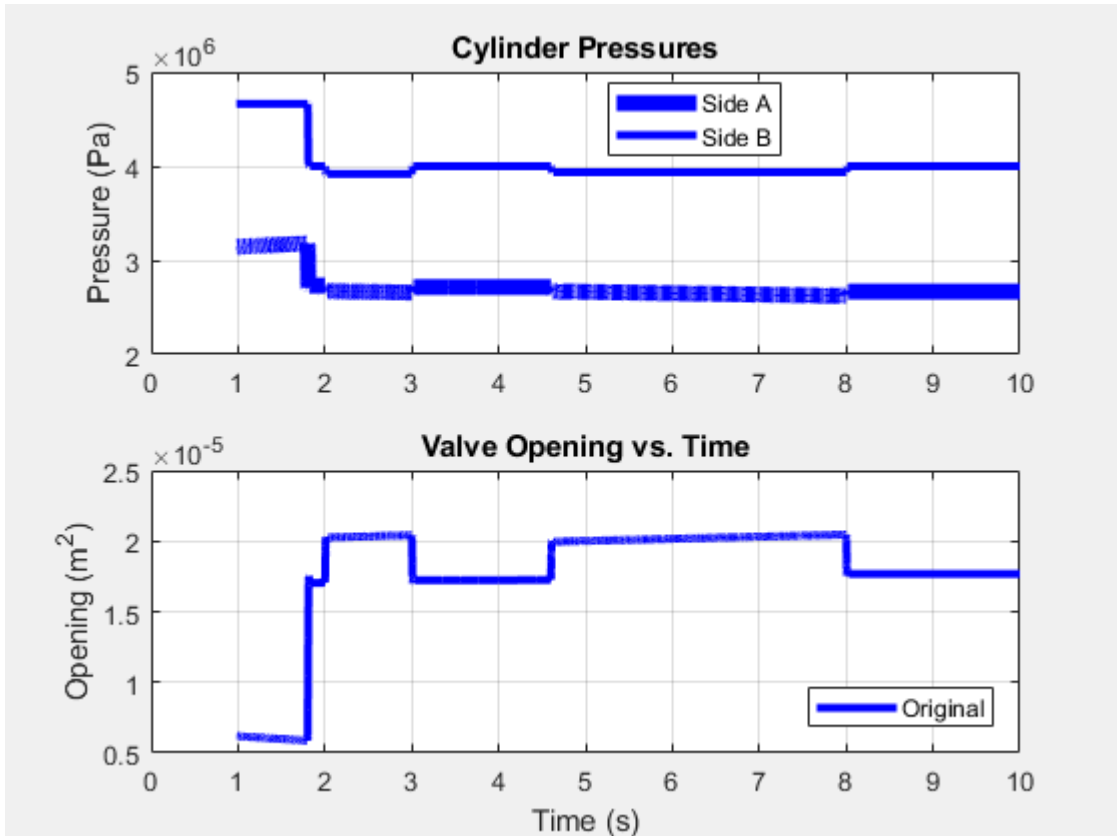
```matlab
% Plot valve activity
ah(2) = subplot(2,1,2);
plot(aValve1.time,aValve1.values,'LineWidth',3,'Color','b');
grid on
title('Valve Opening vs. Time');
ylabel('Opening (m^2)');
xlabel('Time (s)');
linkaxes(ah,'x');

% Create legend
legend('Original','Location','southeast');
```



The custom two-way valve is open when the simulation time, $t$, is 2–3 seconds.

**3**   Trim the model. Perform closed-loop simulation, using $t = 2.5$ seconds, when the valve is open, for the operating point.

**Script for Trimming the Model**

```matlab
%% Trim the model
assignin('base','ClosedLoop',1); % Close the feedback loop
[t,x,y] = sim('ssc_hydraulic_actuator_digital_control');
idx = find(t>2.5,1);
X = x(idx,:);
U = y(idx);
```

**4**   Linearize an open-loop configuration of the continuous time model and save the state variables, $a$, $b$, $c$, and $d$ in the workspace using the linmod function.

**Script for Linearizing the Model**

```
%% Linearize the model
assignin('base','ClosedLoop',0); % Break the feedback loop
[a,b,c,d]=linmod('ssc_hydraulic_actuator_digital_control',X,U);
assignin('base','ClosedLoop',1); % Close the feedback loop
```

# Perform Frequency-Response and Pole-Speed Analyses

**1** Generate a Bode plot.

**Script for Generating a Bode Plot**

```
%% Calculate and plot the frequency response
% Generate data
npts = 100;
w = logspace(-3,5,npts);
G = zeros(1,npts);
for i=1:npts % Use negative feedback convention ( c:=-c, d:=-d)
    G(i) = (-c)*(2*pi*1i*w(i)*eye(size(a))-a)^-1*b +(-d);
end

% Create Bode plot figure
h4_ssc_hydraulic_actuator_digital_control=figure...
    ('Name','hydraulic_actuator_digital_control');

% Plot magnitude
ah(1) = subplot(2,1,1);
magline_h=semilogx(w,20*log10(abs(G)));
xlim(ah(1),[w(1),w(end)]);
grid on
ylabel('Magnitude (dB)');
title('Frequency Response (Hydraulic Actuator)');

% Plot phase
ah(2) = subplot(2,1,2);
phsline_h=semilogx(w,180/pi*unwrap(angle(G)));
set([magline_h,phsline_h],'LineWidth',1);
ylabel('Phase (deg)');
xlabel('Frequency (Hz)');
grid on
linkaxes(ah,'x');

% Create legend
legend('Original','Location','southwest');
```
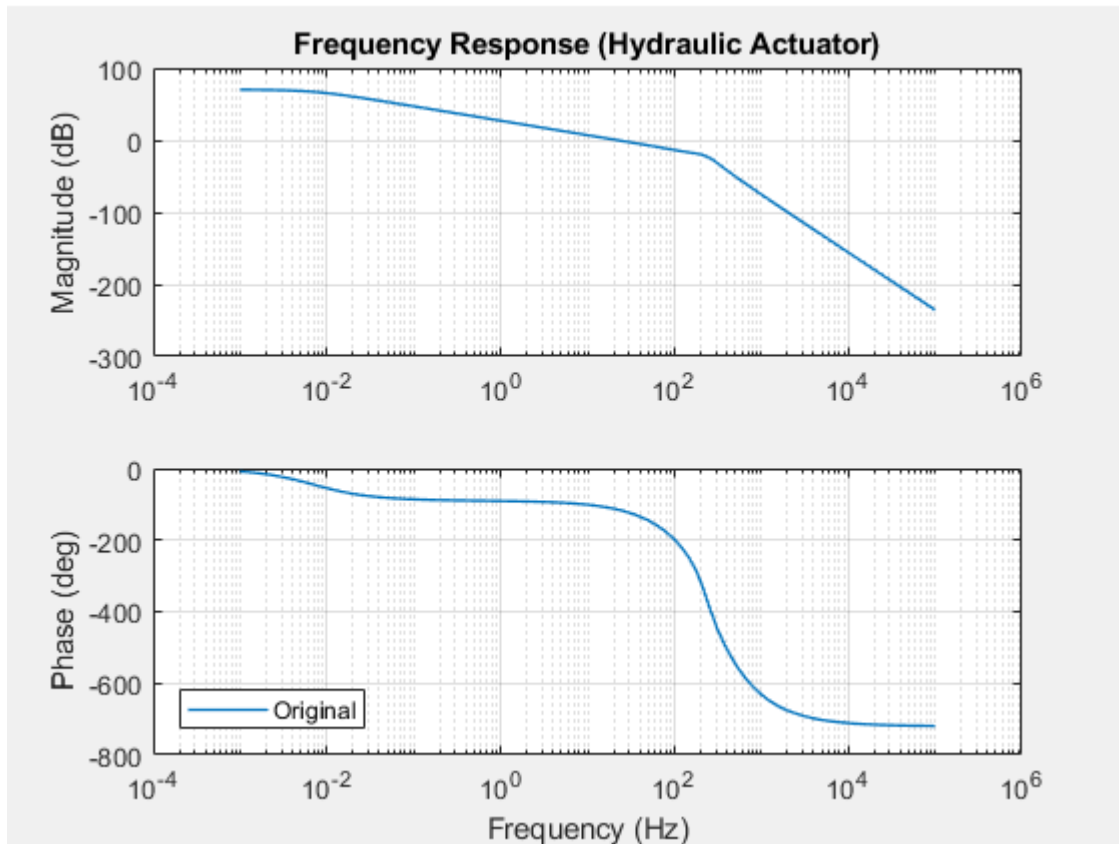
When the frequency, $\omega$, is between $10^2$ and $10^3$ Hz, the phase drops by approximately 600 degrees. The rapid change in the phase shift, $\theta$, indicates that the system has fast dynamics.

2   Calculate eigenvalues of the *a* matrix using the `eig` function and plot the poles in the complex plane.

**Script for Calculating and Plotting the *A*-Matrix Eigenvalues**

```
%% Calculate and plot the poles
% Extract the poles
poles = eig(a);

% Extract the real and imaginary coordinates
X1 = real(poles);
Y1 = imag(poles);

% Create complex plane plot figure
h5_ssc_hydraulic_actuator_digital_control = figure('Name',...
    'hydraulic_actuator_digital_control',...
    'OuterPosition',[668 150 1137 512]);

% Create axes
axes1 = axes('Parent',h5_ssc_hydraulic_actuator_digital_control,...
    'Position',[0.0315 0.112 0.932 0.815]);
hold(axes1,'on');

% Create title
title('Complex Plane Plot');
```

```
% Plot poles
plot(X1,Y1,'DisplayName','Original Poles','Marker','*',...
    'MarkerSize',3,'LineStyle','none');

% Limit and label axes
xlim(axes1,[-3000 500]);
xlabel({'Real','Axis'});
ylim(axes1,[-2000 2000]);
ylabel({'Imaginary',' Axis'});
set(axes1,'XAxisLocation','origin','YAxisLocation','origin');
box(axes1,'on');

% Create legend
legend(axes1,'show','Original','Location','southwest');
```
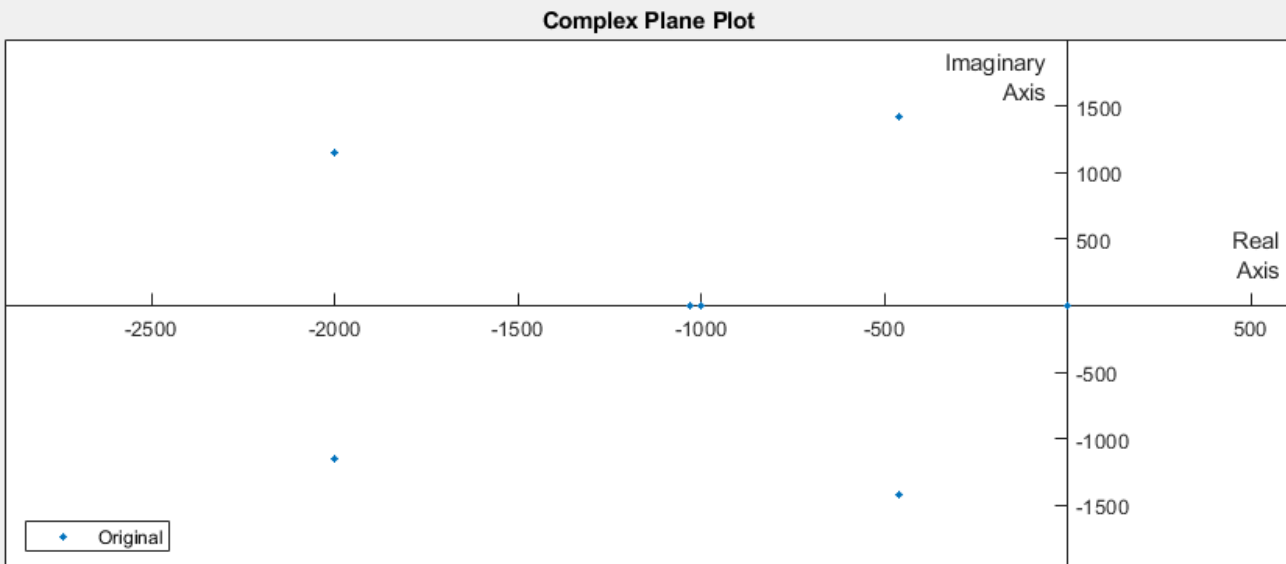


There are six fast poles, including two potential oscillating pole pairs.

**3**    Confirm that there are pole pairs. Print the values of the six fast poles to the command window using the `eigs` function.

**Script for Printing Pole Values**

```
%% Print fast-pole values
eigs(a,6)

ans =

   1.0e+03 *

  -2.0000 + 1.1547i
  -2.0000 - 1.1547i
  -0.4614 + 1.4208i
  -0.4614 - 1.4208i
```

**11-35**

```
      -1.0314 + 0.0000i
      -1.0000 + 0.0000i
```

There are two sets of pole pairs.

## Identify and Eliminate the Sources of Fast Dynamics

Examine the model for potential sources of fast dynamics.

1   To linearize the model, this example uses the `linmod` function. The documentation for the `linmod` advises against using the function to linearize a model that contains a Transport Delay block. The documentation for the Transport Delay block indicates that the Pade approximation for a linearization routine can add dynamic states to a model. Determine if the block is the source of the fast poles that result in the linearized model.

2   To simulate the model without the effects of the Transport Delay block, comment through the block.

**Script for Commenting Out Transport Delay Block**

```
%% Eliminate Transport Delay block from the network
set_param('ssc_hydraulic_actuator_digital_control/Transport Delay',...
    'Commented','through')
```
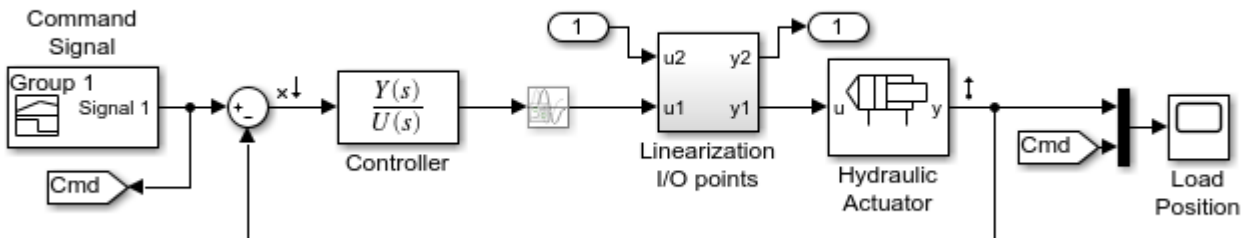


The icon for the Transport Delay fades to indicate that it is commented through.

3   To examine the frequency response of the model without the effects of the Transport Delay block, trim, linearize, and simulate the model, and then, update the Bode plot.

**Script for Trimming and Linearizing the Model and Updating the Bode Plot**

```
%% Trim, Linearize, and update the Bode plot.
% Trim
assignin('base','ClosedLoop',1); % Close the feedback loop
[t,x,y] = sim('ssc_hydraulic_actuator_digital_control');
idx = find(t>2.5,1);
X = x(idx,:);
U = y(idx);

% Linearize
assignin('base','ClosedLoop',0); % Break the feedback loop
[a,b,c,d]=linmod('ssc_hydraulic_actuator_digital_control',X,U);
assignin('base','ClosedLoop',1); % Close the feedback loop
```

```matlab
% Generate data
npts = 100; w = logspace(-3,5,npts); G = zeros(1,npts);
for i=1:npts % Use negative feedback convention ( c:=-c, d:=-d)
    G(i) = (-c)*(2*pi*1i*w(i)*eye(size(a))-a)^-1*b +(-d);
end

% Update the Bode plot figure
figure(h4_ssc_hydraulic_actuator_digital_control);
hold on
ah(1) = subplot(2,1,1);
hold on
magline_h=semilogx(w,20*log10(abs(G)),'--');
xlim(ah(1),[w(1),w(end)]);
ah(2) = subplot(2,1,2);
hold on
phsline_h=semilogx(w,180/pi*unwrap(angle(G)),'--');
set([magline_h,phsline_h],'LineWidth',1);
linkaxes(ah,'x');
legend('Original','No Transport Delay','Location','southwest');
```
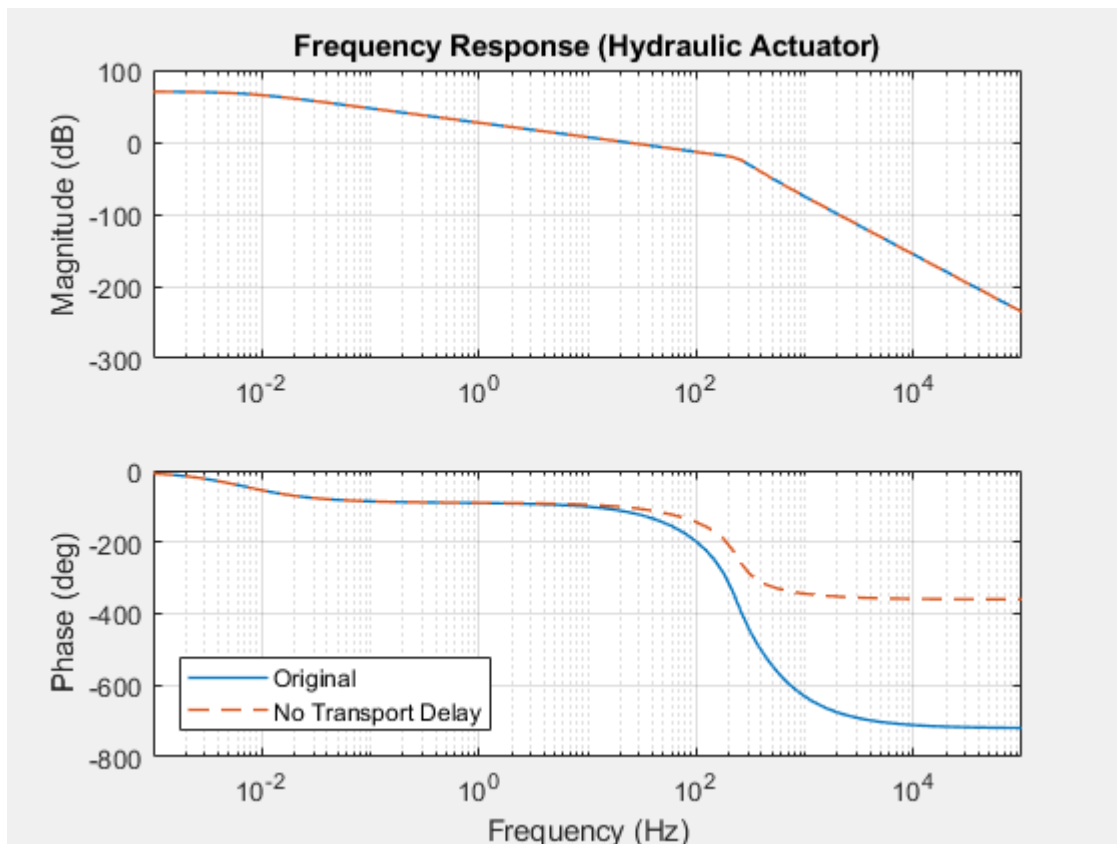


When the frequency, $\omega$, is between $10^2$ and $10^3$ Hz, the phase drops by only by ~250 degrees.

**4** Calculate and plot the fast poles.

**Script for Calculating and Plotting the *A*-Matrix Eigenvalues**

```matlab
%% Calculate and plot the poles
% Extract the poles
```

```
poles = eig(a);

% Extract the real and imaginary coordinates
X1 = real(poles);
Y1 = imag(poles);

% Update the complex plane plot figure
figure(h5_ssc_hydraulic_actuator_digital_control)
hold on
plot(X1,Y1,'DisplayName','No Transport Delay','Marker','d',...
    'MarkerSize',5,'LineStyle','none');
```
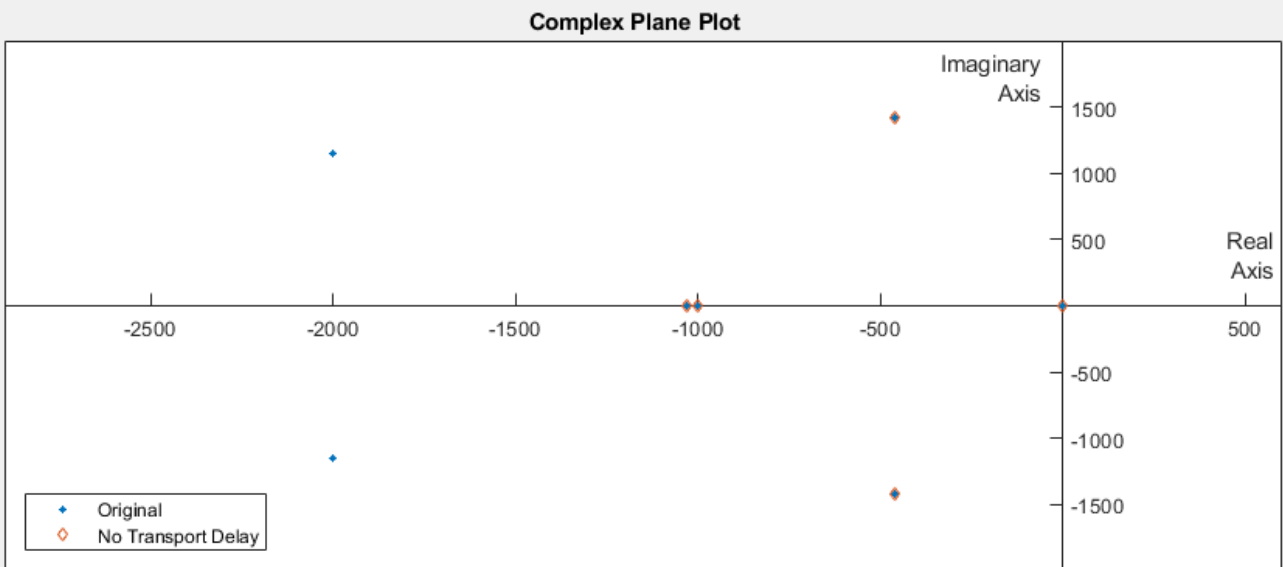


The Transport Delay block is responsible for the missing oscillatory pole pair at -2000 ± 1.1547$i$ rad/sec

5    Plot the simulation results to see if they adequately match the original results.

### Script for Simulating the Model and Plotting the Results

```
%% Plot the simulation results
% Get simulation results
simlog2 = simlog_ssc_hydraulic_actuator_digital_control;
pCylA = simlog2.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_A.A.p.series;
pCylB = simlog2.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_B.A.p.series;
aValve = simlog2.Hydraulic_Actuator.Custom_2_Way_Valve.Calculate_Orifice_Area.PS_Saturation.O.series;

% Update figure
figure(h3_ssc_hydraulic_actuator_digital_control)
hold on
ah(1) = subplot(2,1,1);
hold on
plot(pCylA.time,pCylA.values,'LineWidth',3,'Color','r','LineStyle','-');
hold on
plot(pCylB.time,pCylB.values,'LineWidth',1,'Color','r','LineStyle','-');
legend({'Side A','Side B'},'Location','Best');
hold off
ah(2)= subplot(2,1,2);
hold on
plot(aValve.time,aValve.values,'LineWidth',1,'Color','r','LineStyle','-');
hold off
legend({'Original','No Transport Delay'},'Location','southeast');
```
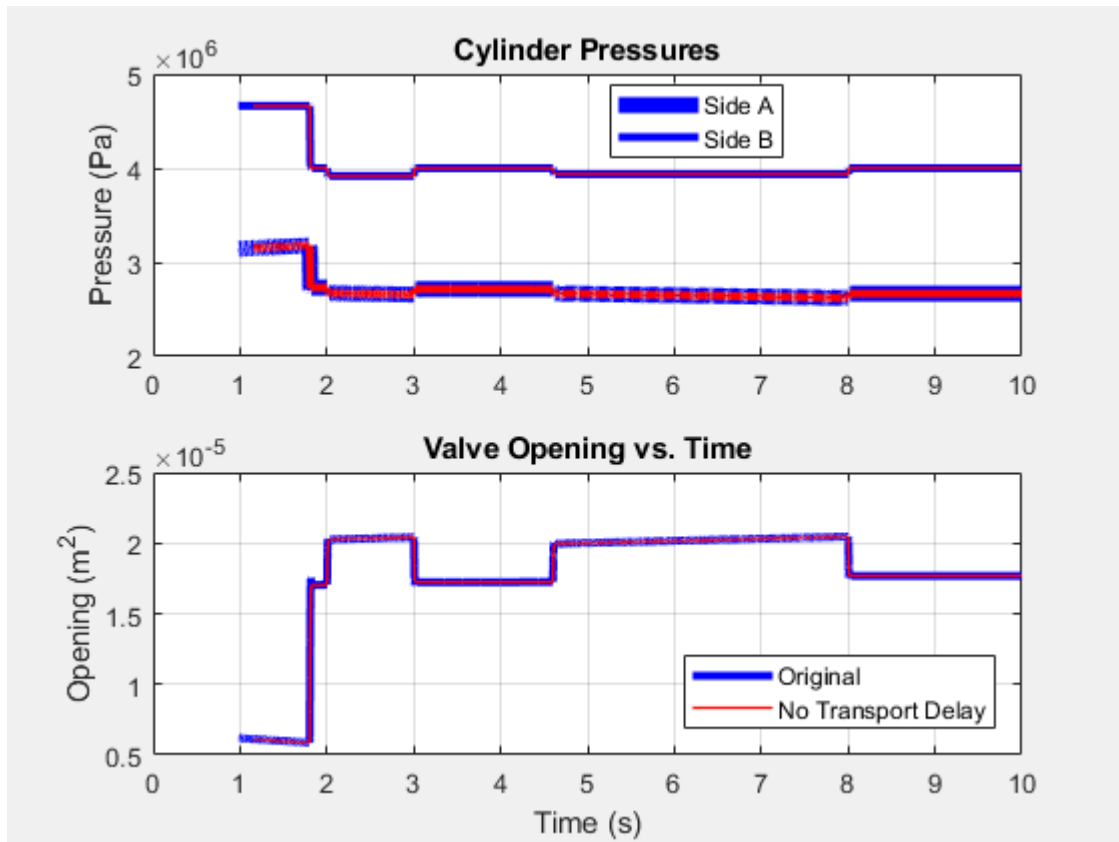
The results appear similar.

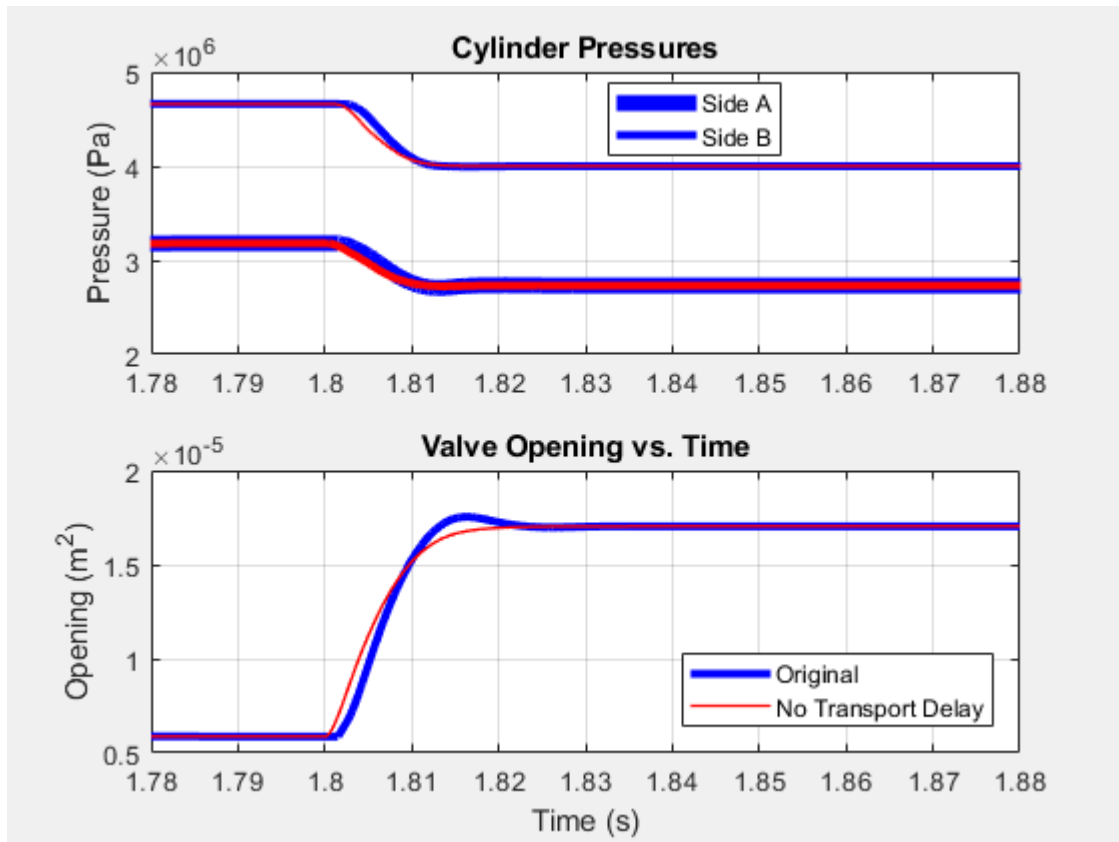**6** Zoom to evaluate accuracy in more detail.

**Script for Zooming In**

```
%% Zoom the figure
% Define the x-axis limits
xStart = 0;
xEnd = 10;
xZoomStart1 = 1.78;
xZoomEnd1 = 1.88;

% Zoom in
figure(h3_ssc_hydraulic_actuator_digital_control);
ah(2);
xlim([xZoomStart1, xZoomEnd1]);
```

At this level, you can see a small difference in the results for the modified model. However, the simulation is accurate enough that the results meet expectations based on empirical and theoretical data.

7   The model includes hydraulic compressibility, that is, oil column resonance. To confirm that the column resonance is responsible for the second oscillatory pole pair, turn off compressibility and repeat the frequency response and pole analyses.

**Script for Eliminating Compressibility and Performing the Frequency Response and Pole Analysis**

```
%% Remove Compressibility
model = 'ssc_hydraulic_actuator_digital_control';
subsystem = 'Hydraulic Actuator';
composite_block = 'Hydraulic Cylinder';

CylA = [model,'/',subsystem,'/',composite_block,'/Chamber A'];
CylB = [model,'/',subsystem,'/',composite_block,'/Chamber B'];
set_param(CylA, 'Compressibility', '0')
set_param(CylB, 'Compressibility', '0')

%% Trim
assignin('base','ClosedLoop',1); % Close the feedback loop
[t,x,y] = sim('ssc_hydraulic_actuator_digital_control');
idx = find(t>2.5,1);
X = x(idx,:); U = y(idx);

%% Linearize
assignin('base','ClosedLoop',0); % Break the feedback loop
[a,b,c,d]=linmod('ssc_hydraulic_actuator_digital_control',X,U);
assignin('base','ClosedLoop',1); % Close the feedback loop
```

```matlab
%% Update Bode plot
% Generate data
npts = 100;
w = logspace(-3,5,npts);
G = zeros(1,npts);
for i=1:npts % Use negative feedback convention ( c:=-c, d:=-d)
    G(i) = (-c)*(2*pi*1i*w(i)*eye(size(a))-a)^-1*b +(-d);
end

figure(h4_ssc_hydraulic_actuator_digital_control);
hold on
ah(1) = subplot(2,1,1);
hold on
magline_h=semilogx(w,20*log10(abs(G)));
xlim(ah(1),[w(1),w(end)]);
% figure(h4_ssc_hydraulic_actuator_digital_control);
% hold on

ah(2) = subplot(2,1,2);
hold on
phsline_h=semilogx(w,180/pi*unwrap(angle(G)));
set([magline_h,phsline_h],'LineWidth',1);
linkaxes(ah,'x');
legend('Original','No Transport Delay','No Compressibility',...
    'Location','southwest');

%% Update complex plane plot
poles = eig(a);

figure(h5_ssc_hydraulic_actuator_digital_control)
hold on
X1 = real(poles);
Y1 = imag(poles);
plot(X1,Y1,'DisplayName','No Compressibility','Marker','d',...
    'MarkerSize',8,'LineStyle','none');
```
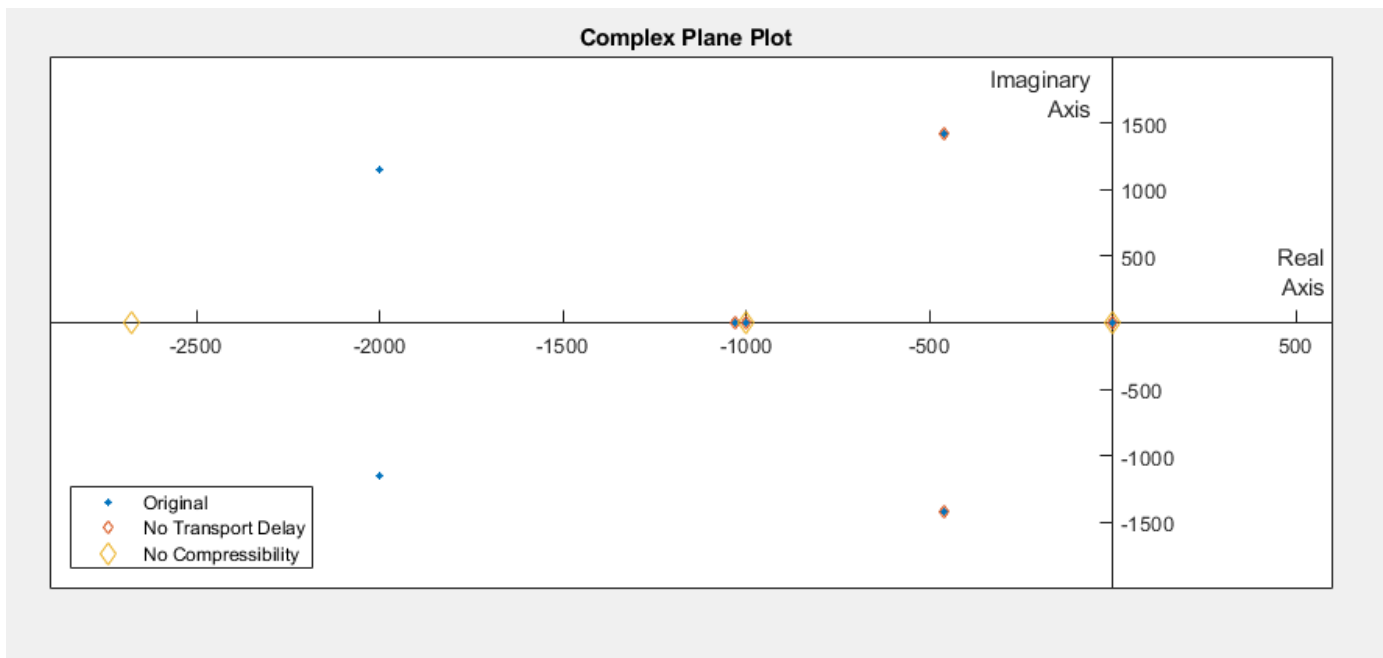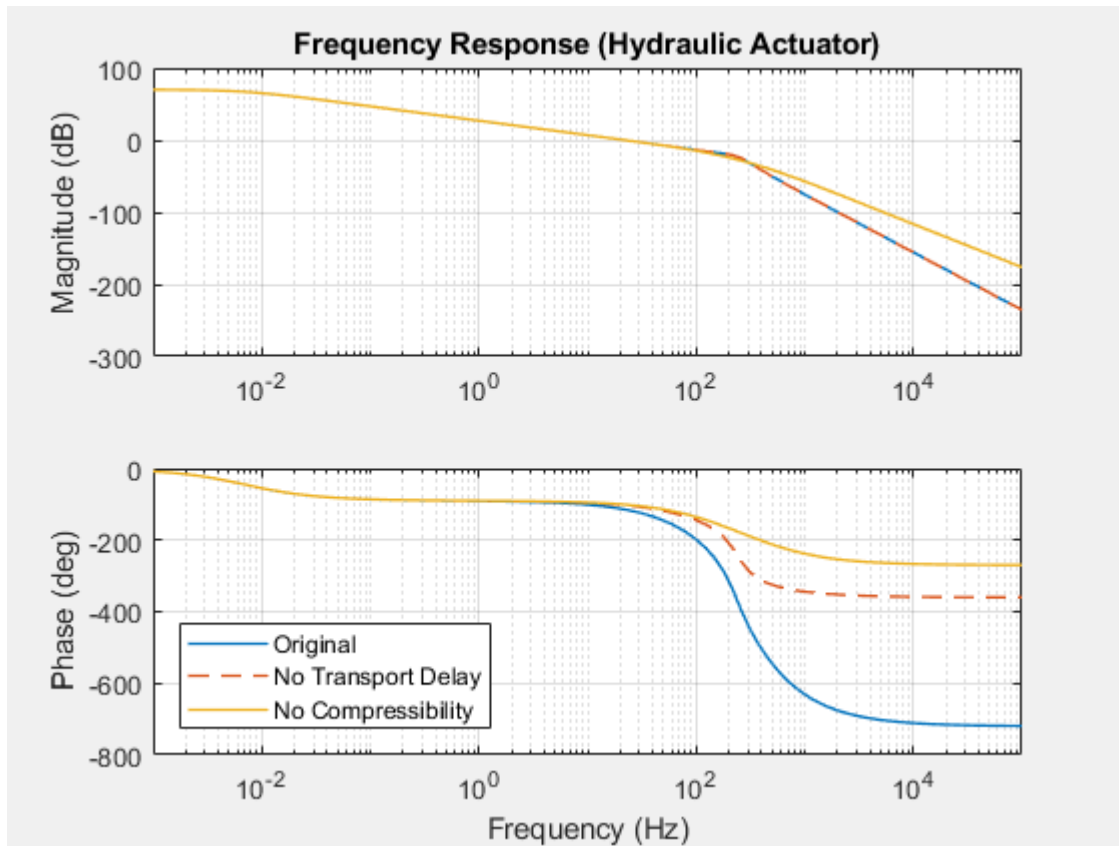
**11-41**

The further decrease in the phase drop reflects the reduction in high-frequency dynamics. There are now two fast poles. Even though one of the fast poles has moved further away from the

imaginary axis, there are fewer fast dynamics because the oscillating pole pair at 458.8 ± 1.4273*i* rad/sec is eliminated.

**8** Print the values of the remaining two fast poles to the command window.

**Script for Printing Pole Values**

```
%% Print the pole values
eigs(a,2)

two_fast_poles =

   1.0e+03 *

   -2.6767
   -1.0000
```

The fast pole at -2677 rad/s corresponds to the load mass and hydraulic damping introduced by the two orifice components in the hydraulic subsystem. These dynamics are central to the simulation results. The fast pole at -1000 rad/s corresponds to the controller denominator, 0.001 s+1. The transfer function is integral to the controller design. No more dynamic modes can be removed without changing important system-level behavior.

**9** Plot the simulation results to see if they adequately match the original results.

**Script for Plotting the Simulation Results**

```
%% Print simulation results
% Get simulation results
simlog3 = simlog_ssc_hydraulic_actuator_digital_control;
pCylA = simlog3.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_A.A.p.series;
pCylB = simlog3.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_B.A.p.series;
aValve = simlog3.Hydraulic_Actuator.Custom_2_Way_Valve.Calculate_Orifice_Area.PS_Saturation.O.series;

% Update results
figure(h3_ssc_hydraulic_actuator_digital_control)
hold on
ah(1)= subplot(2,1,1);
hold on
plot(pCylA.time,pCylA.values,'LineWidth',1,'Color','y','LineStyle','--');
hold on
plot(pCylB.time,pCylB.values,'LineWidth',1,'Color','y','LineStyle','--');
legend({'Side A','Side B'},'Location','Best');
hold off
ah(2)= subplot(2,1,2);
hold on
plot(aValve.time,aValve.values,'LineWidth',1,'LineStyle','--');
legend({'Original','No Transport Delay','Compressibility'},'Location','southeast');
hold off

% Zoom out
figure(h3_ssc_hydraulic_actuator_digital_control);
ah(2);
xlim([xStart, xEnd]);
```
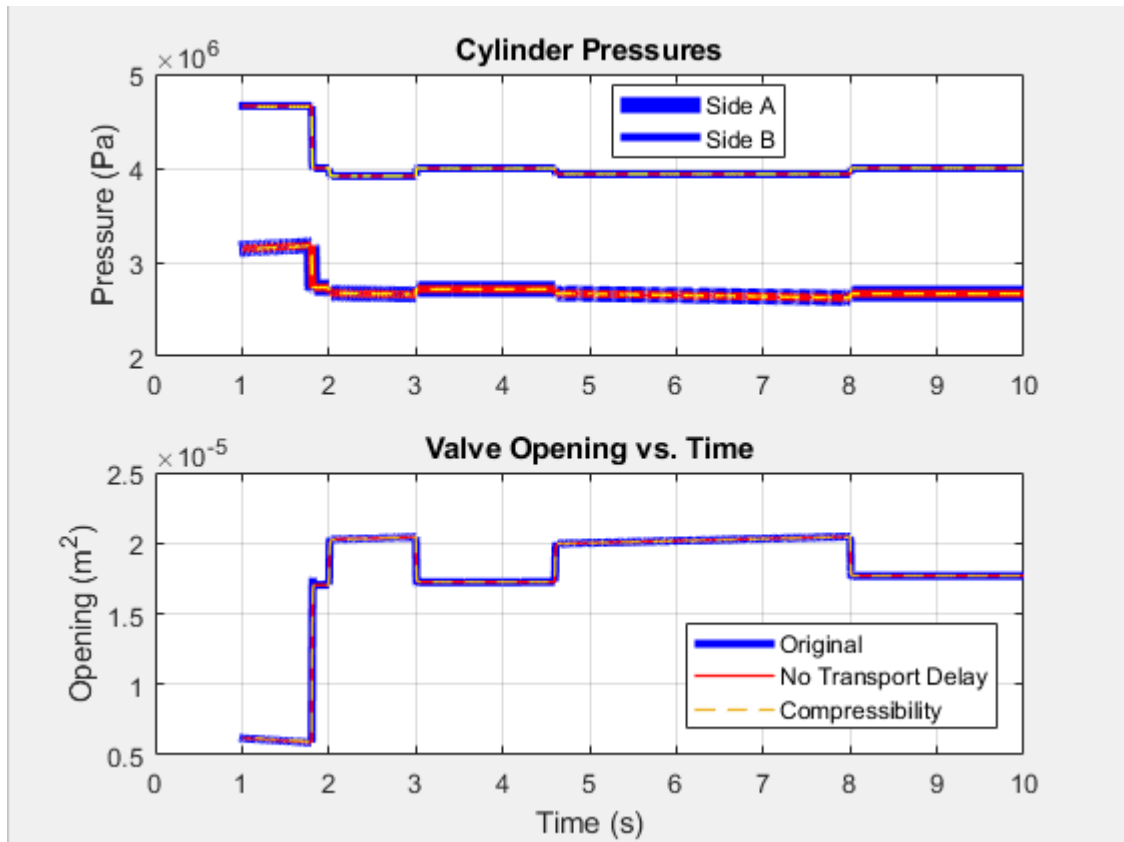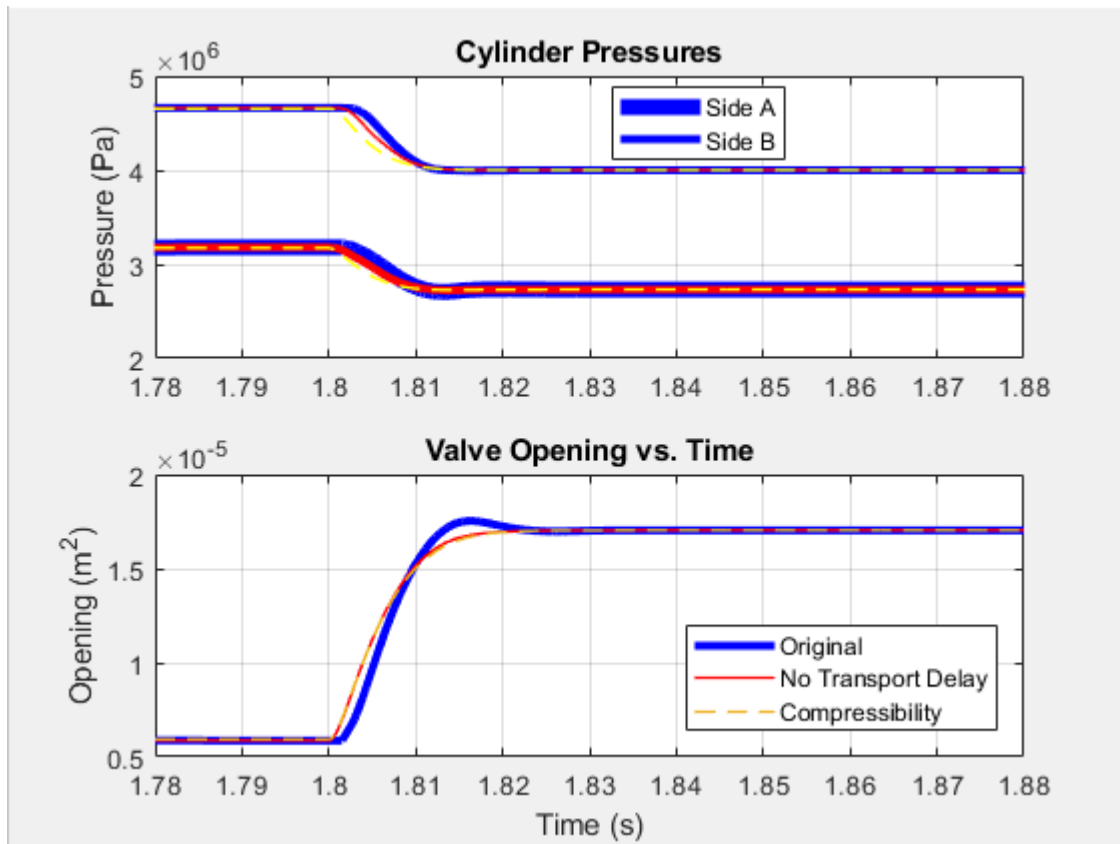
The accuracy of the updated model seems acceptable.

**10**  Zoom to evaluate accuracy in more detail.

**Script for Zooming In**

```
%% Zoom the figure
% Zoom in
figure(h3_ssc_hydraulic_actuator_digital_control);
ah(2);
xlim([xZoomStart1, xZoomEnd1]);
```

At this level, you can see that there is only a small additional difference in the results for the modified model. The accuracy of the simulation is acceptable.

## See Also

**Blocks**
Transport Delay

**Functions**
eig | eigs | linmod

## Related Examples

- "Determine Step Size" on page 11-12
- "Estimate Computation Costs" on page 11-87
- "Reduce Computation Costs" on page 11-25
- "Reduce Numerical Stiffness" on page 11-47
- "Reduce Zero Crossings" on page 11-55

## More About

- "Model Preparation Objectives" on page 11-2

- "Real-Time Model Preparation Workflow" on page 11-4
- "Linearizing Models"

# Reduce Numerical Stiffness

| **In this section...** |
|---|
| "Why Reduce Stiffness?" on page 11-47 |
| "Review Reference Results" on page 11-47 |
| "Identify and Modify a Stiff Element" on page 11-49 |
| "Analyze Results" on page 11-50 |

This example helps you to complete the steps outlined in "Real-Time Model Preparation Workflow" on page 11-4 and to meet the goals described in "Model Preparation Objectives" on page 11-2.

In "Determine Step Size" on page 11-12, you use the results of a variable-step simulation of your Simscape model to identify when step size decreases to capture behavior accurately at discontinuities and for rapid dynamics in numerically stiff systems. These types of events often require solvers to take steps that are too small to support real-time simulation. This example shows how to use the results from "Determine Step Size" on page 11-12 to identify a numerically stiff element in your model. It also shows how to modify the element for faster simulation without sacrificing accuracy.

## Why Reduce Stiffness?

Numerical stiffness can prevent your model from being real-time capable. A real-time-capable model is one that produces acceptable results without incurring overruns when you simulate it on your target processor. Stiff systems contain dynamics that vary both quickly and slowly. When solvers take large steps, they usually capture slowly changing dynamics, but they tend to miss rapid changes unless they are taking small steps. Small step sizes cause overruns when they do not provide enough time for a real-time computer to complete calculating solutions during a single step.

To you reduce numerical stiffness, you eliminate rapid changes. If there are no rapid changes, the solver can take larger steps and still obtain accurate simulation results. The larger the step size, the less likely it is that your model generates an overrun during real-time simulation.

## Review Reference Results

**1**   To open the model, at the MATLAB command prompt, enter:

```
model = 'ssc_pneumatic_rts_reference';
open_system(model)
```
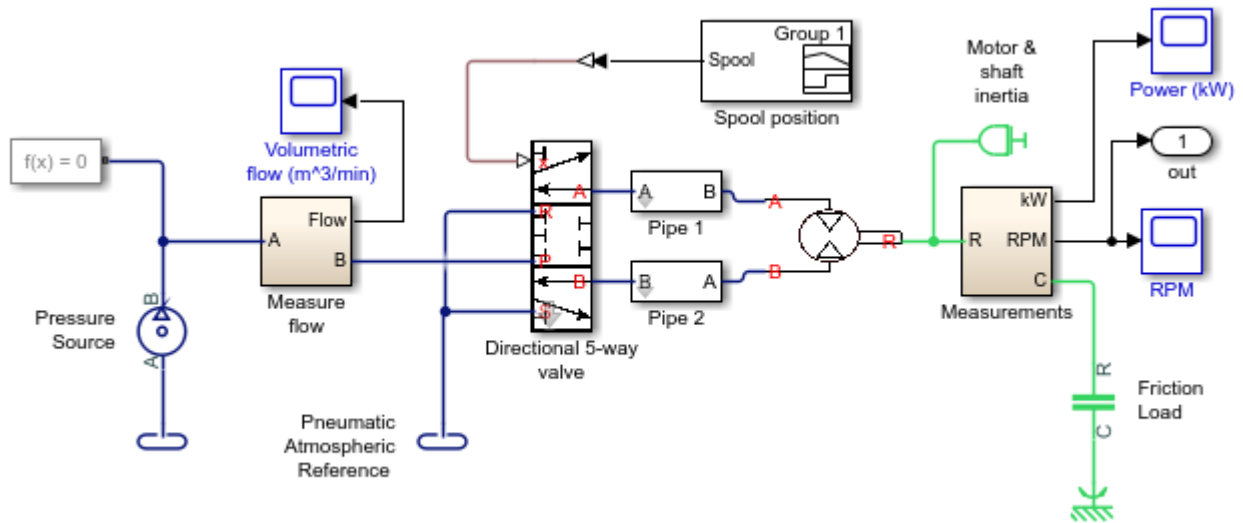
2   Simulate the model:

```
sim(model)
```

3   Create a figure that contains a semilogarithmic plot of the solver step size, a plot of the motor speed results, and a plot of the gas flow results.

```
h1 = figure;
subplot(3,1,1)
semilogy(tout(1:end-1),diff(tout),'-x')
title('Solver Step Size and Results')
ylabel('Step Size (s)')
subplot(3,1,2)
plot(tout,Pneu_rts_RPM_DATA.signals.values)
ylabel('Speed (rpms)')
subplot(3,1,3)
plot(tout,Pneu_rts_Vol_Flow_DATA.signals.values)
xlabel('Time (s)')
ylabel('Flow Rate (m^3/min)')
```
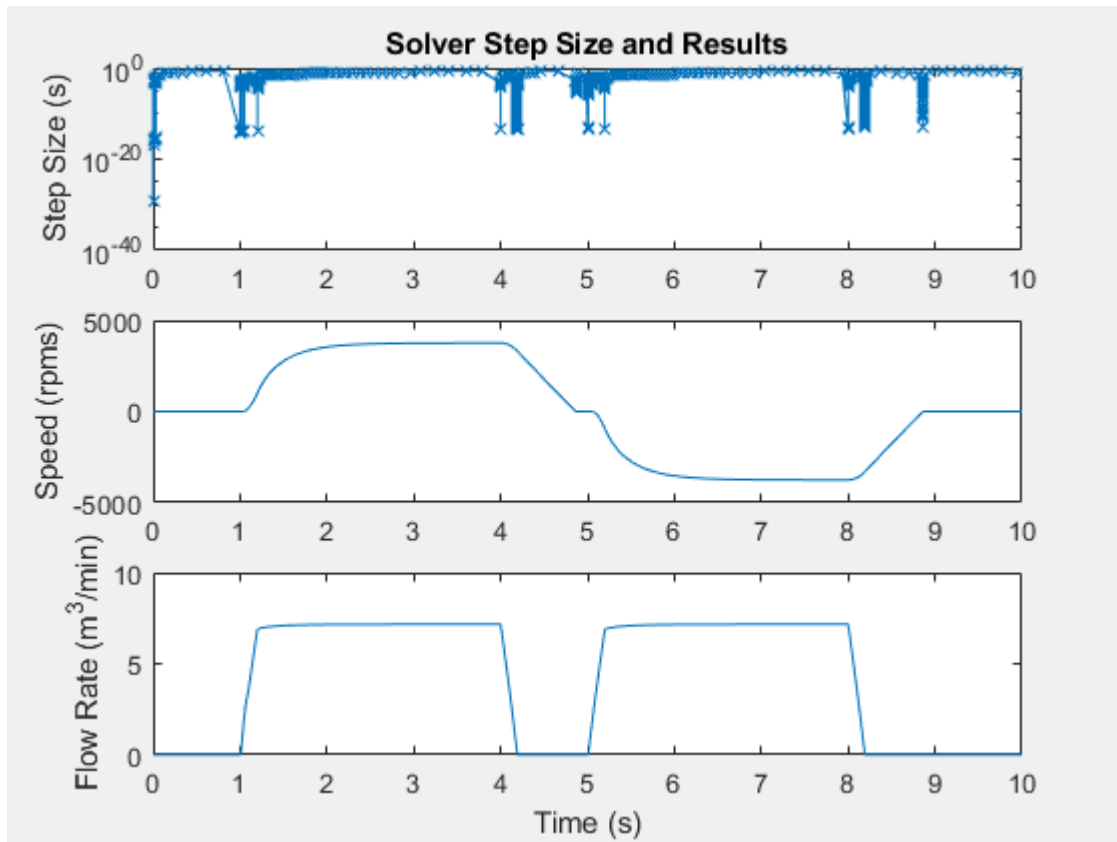
The simulation takes steps smaller than 1e-10 seconds when:

- The motor speed is near zero rpm (simulation time $t = \sim 1$, 5, and 9 seconds)

- The step change in motor speed is initiated from a steady-state speed to a new speed (time $t = \sim 4$ and 8 seconds)

- The step change in flow rate is initiated from a steady-state speed to a new flow rate (time $t = \sim 4$ and 8 seconds)

- The volumetric flow rate is near zero m^3/min ($t = \sim 1$, 4, and 5 seconds)

The results indicate that small step sizes are required to achieve accuracy when the simulation is capturing dynamics that involve friction or small, compressible volumes. Elements that generate zero crossings might also be responsible for the small steps and slow recovery times.

**4** Assign the simulation results to new variables in the MATLAB workspace so that you can compare the data to results from a model that you modify.

```
timeRef   = tout;
simlogRef = simlog;
```

## Identify and Modify a Stiff Element

Examine the friction load in the model to determine if it incorporates discontinuities or has a small time constant that causes numerical stiffness. Modify the element if it causes any rapidly changing dynamics that require a small step size.

**11-49**

**1** Save the model as `rts_stiffness_model` in a writable folder on the MATLAB path.

**2** Open the Friction Load block property inspector, a Rotational Friction block. The figure shows the friction torque/relative velocity characteristic for the simple approximation of continuous friction that the block models.
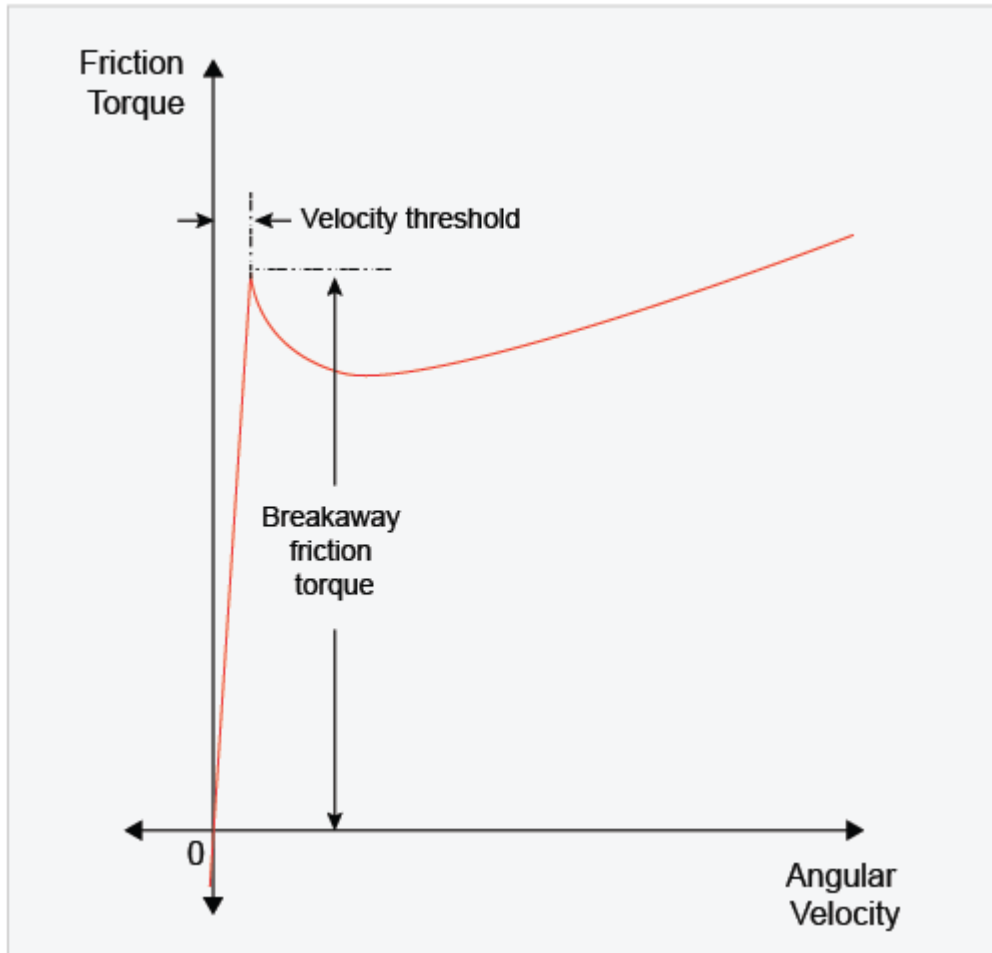


The breakaway torque is modeled as a function of the velocity threshold. When velocity is close to zero, a small change in velocity yields a large change in torque. When velocity is not close to zero, the torque change is more gradual. This block represents a stiff element. To make the element less stiff, specify a higher value for the **Breakaway friction velocity**.

**3** On the **Parameters** tab of the property inspector, change the **Breakaway friction velocity** from `0.059137` to `0.1` rad/s.

**4** Simulate the modified model.

## Analyze Results

To see how modifying the velocity threshold for the friction block affects the stiffness of the component, compare the step sizes for the two simulations. The reference results meet expectations based on empirical and theoretical data. You can assess the accuracy of the modified model by comparing the speed results from the modified model to the results from the original version of the model.

**1**  Plot the step size for the reference results for modified model to the figure that contains the reference data.

```
h2 = figure;
semilogy(timeRef(1:end-1),diff(timeRef),'-x',...
    'LineWidth',1,'MarkerSize',7)
hold on
semilogy(tout(1:end-1),diff(tout),'--x','Color','r',...
    'LineWidth',.1,'MarkerSize',5)
title('Solver Step Size')
xlabel('Time (s)')
ylabel('Step Size (s)')
h1Leg = legend({'Reference','Modified'},'Location','best');
```
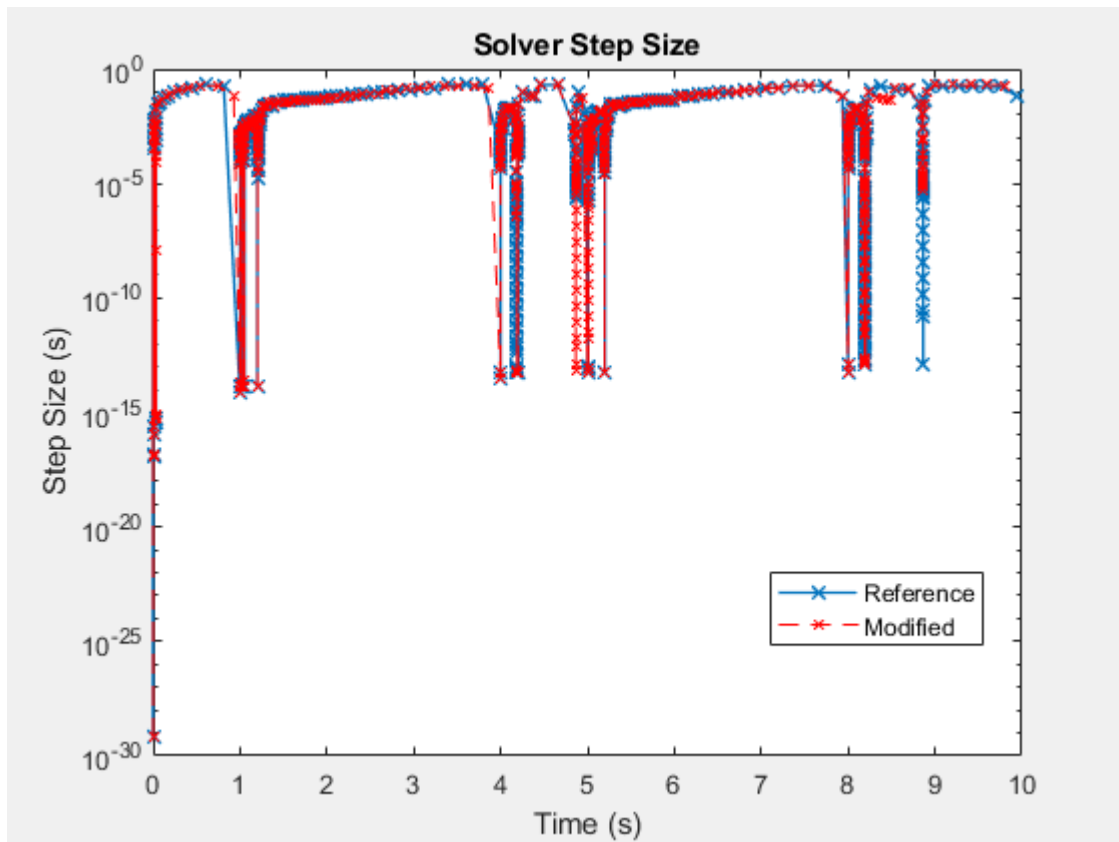


The step size recovers more quickly from the event that occurs at simulation time $t = 4$ and 9 seconds. The simulation is less stiff at these times.

**2**  Extract the speed and time data from the logging nodes for the original and modified models.

```
speedRefNode = simlogRef.Measurements.Ideal_Rotational_Motion_Sensor.R.w;
speedRef = speedRefNode.series.values('rpm');
timeRef = speedRefNode.series.time;
speedModNode = simlog.Measurements.Ideal_Rotational_Motion_Sensor.R.w;
speedMod = speedModNode.series.values('rpm');
timeMod = speedModNode.series.time;
```

**3**  Plot and compare the results for the speed data for both simulations to make sure that the modified model is accurate.

```
h3 = figure;
plot(timeRef,speedRef)
h3;
hold on
plot(timeMod,speedMod,'r--')
title('Speed')
xlabel('Time (s)')
ylabel('Speed (rpms)')
h3Leg = legend({'Reference','Modified'},'Location','best');
```
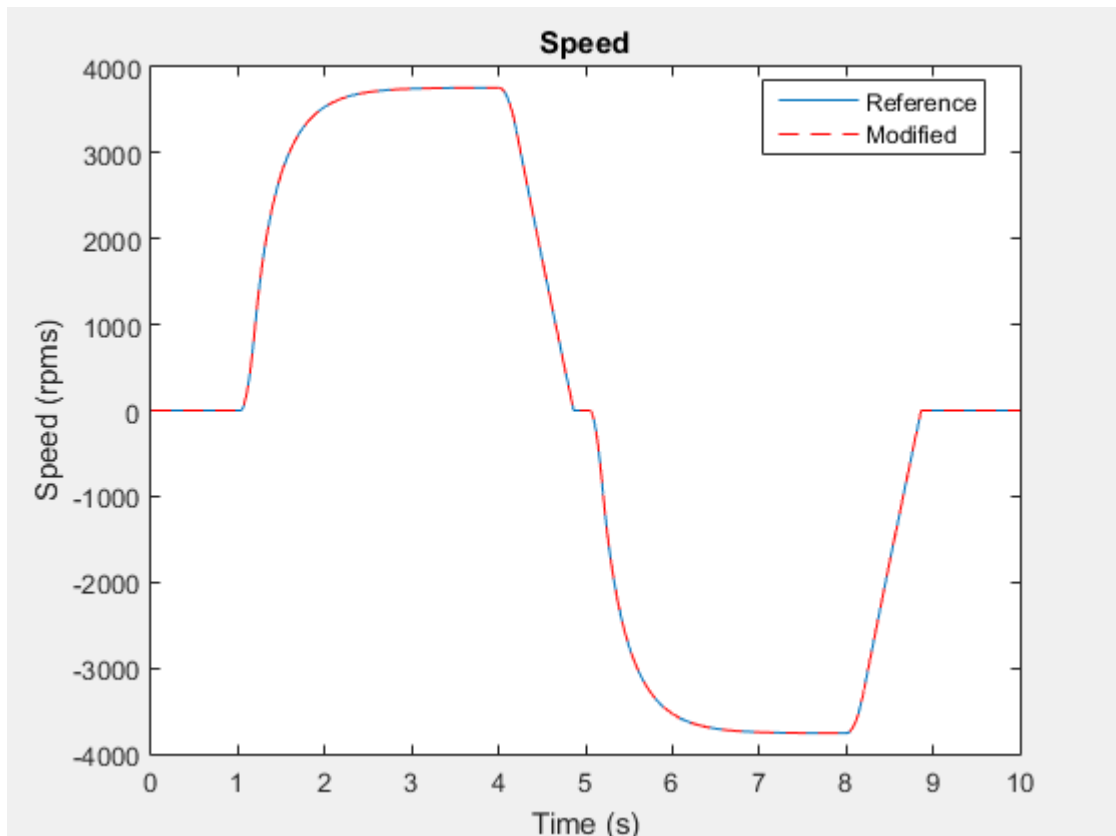


**4** Zoom for a closer look at the inflection point at time ($t$) = ~5 seconds.

```
h3;
xStart = 0;
xEnd = 10;
yStart = -4000;
yEnd = 4000;
xZoomStart1 = 4.8;
xZoomEnd1 = 5.2;
yZoomStart1 = -400;
yZoomEnd1 = 150;
axis([xZoomStart1 xZoomEnd1 yZoomStart1 yZoomEnd1])
```
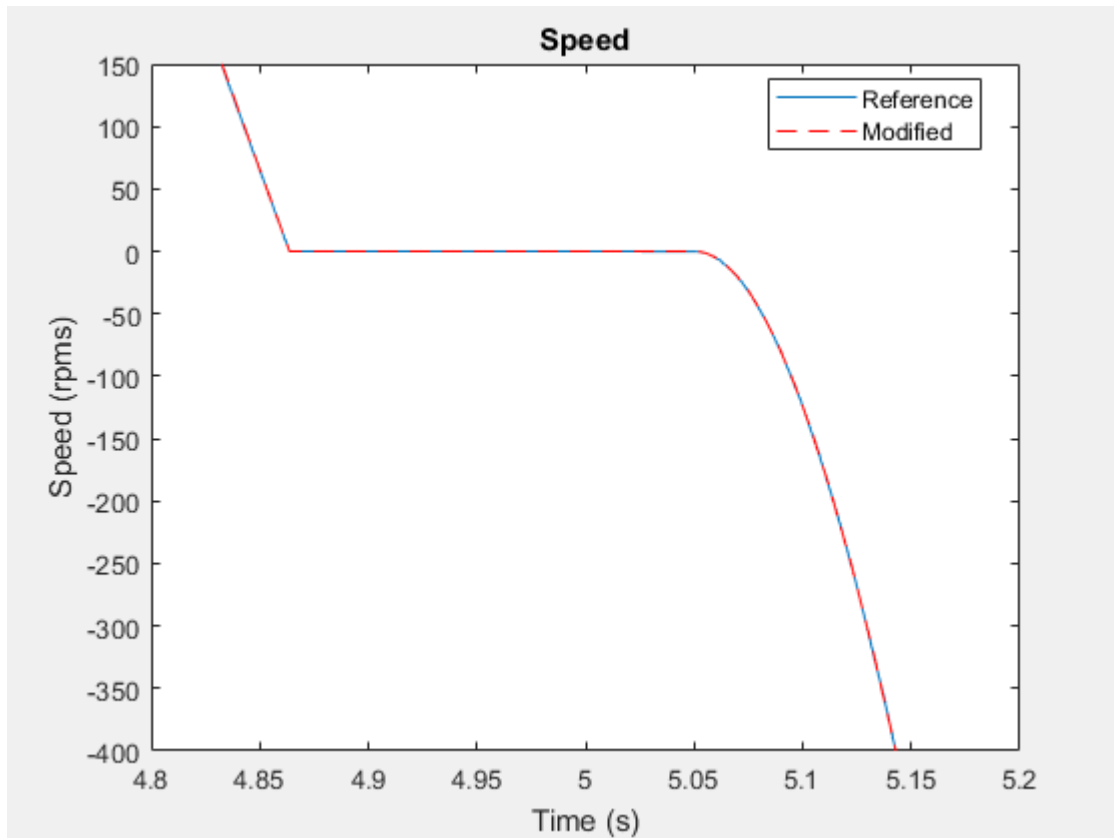
At this zoom level, you can see that the simulation results for the modified model are accurate enough to meet expectations based on empirical and theoretical data.

The Friction Load is now less numerically stiff. The figure of step size during simulation shows that other elements in the model are also responsible for slow recovery times. Reduce more slow-recovery steps by examining and modifying the other elements that cause stiffness.

You can also increase speed by modifying the model using methods in "Reduce Computation Costs" on page 11-25 and "Reduce Zero Crossings" on page 11-55. If you can eliminate all small steps that might generate an overrun, you can attempt to run a fixed-step simulation using the methods in "Choose Step Size and Number of Iterations" on page 11-89.

## See Also
Rotational Friction

## See Also

## Related Examples
- "Determine Step Size" on page 11-12
- "Estimate Computation Costs" on page 11-87
- "Reduce Computation Costs" on page 11-25
- "Reduce Fast Dynamics" on page 11-29

• "Reduce Zero Crossings" on page 11-55
• "Determine System Stiffness" on page 11-82

## More About

• "About Simulation Data Logging" on page 13-2
• "Events and Zero Crossings" on page 7-3
• "Log and Plot Simulation Data" on page 13-8
• "Model Preparation Objectives" on page 11-2
• "Real-Time Model Preparation Workflow" on page 11-4
• "Stiffness" on page 7-2

# Reduce Zero Crossings

| In this section... |
| --- |
| "Why Reduce Zero Crossings?" on page 11-55 |
| "Obtain Reference Results and Plot Simulation Step Size" on page 11-55 |
| "Identify and Modify Elements That Cause Zero Crossings" on page 11-58 |
| "Analyze the Results of the Modified Model" on page 11-60 |

## Why Reduce Zero Crossings?

Real-time deployment requires using a fixed-step solver. You typically use a variable-step solver for desktop simulation. Variable-step solvers take smaller steps when they detect a zero-crossing event. Smaller steps help to capture the dynamics that cause the zero crossing accurately. Fixed-step solvers do not vary the size of the steps that they take. If your model relies heavily on detecting zero crossings, you might need to specify a very small fixed-step size to capture the dynamics accurately. A small step size can lead to overruns during real-time simulation. By reducing the number of zero crossings, you can configure your solver to use a larger step size for both variable-step and fixed-step deployment while generating results that are accurate enough.

## Obtain Reference Results and Plot Simulation Step Size

Simulate your model to generate data that you can use to:

- Decide which model elements to change to reduce the number of zero-crossing events.
- Assess the accuracy of your modified model.

**1**   To open the model, at the MATLAB command prompt, enter:

```
model = 'ssc_pneumatic_rts_stiffness_redux';
open_system(model)
```
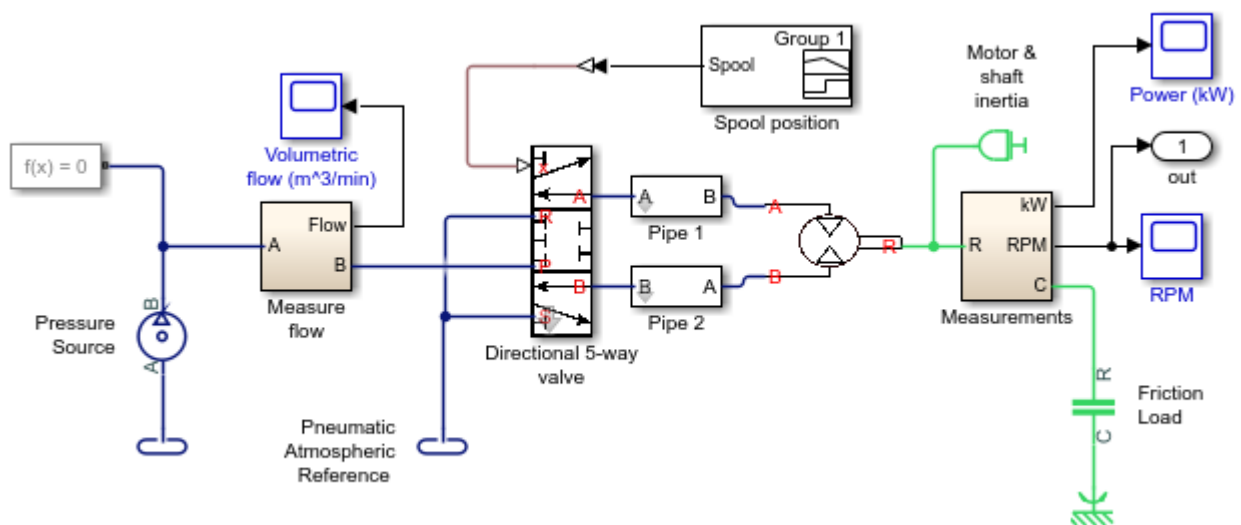
**2** Simulate the model:

```
sim(model)
```

**3** Save the data to the workspace.

```
simlogRef = simlog;
timeRef = tout;
```

**4** Plot the step size against the simulation time.

```
h1 = figure;
semilogy(timeRef(1:end-1),diff(timeRef),'-x')
title('Solver Step Size')
xlabel('Time (s)')
ylabel('Step Size (s)')
```
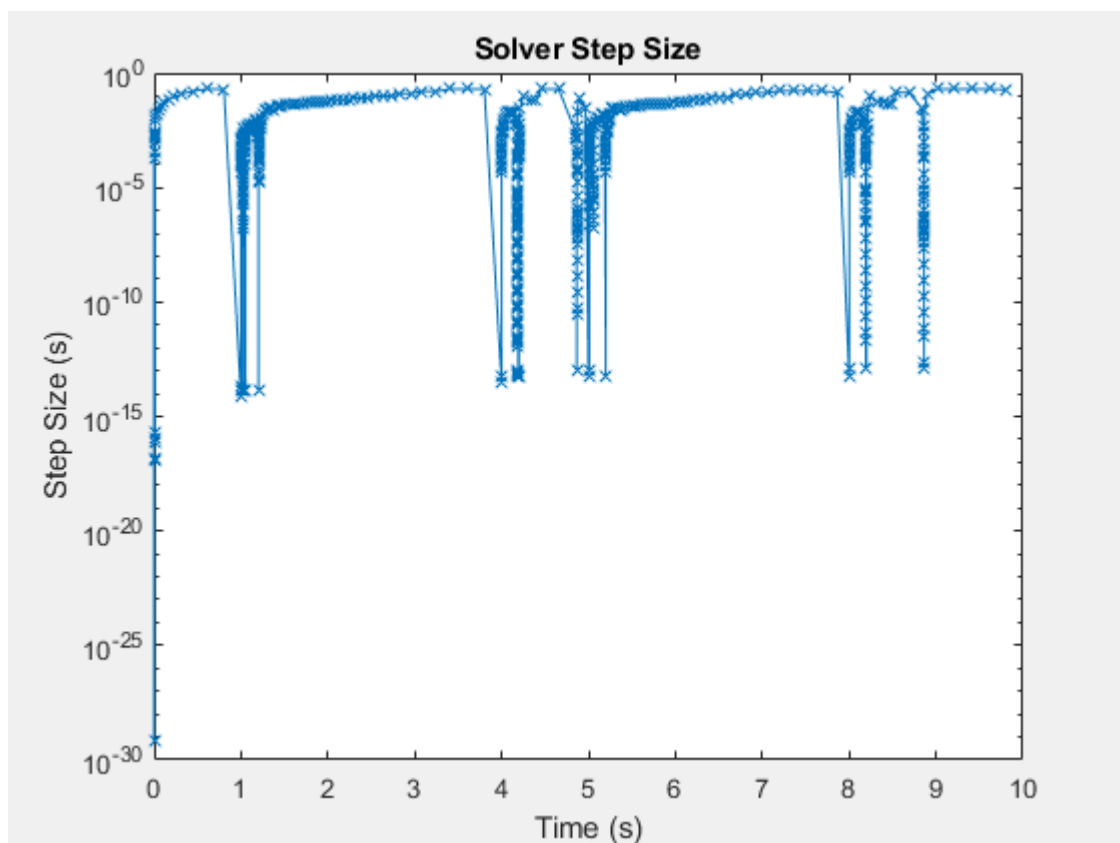


The simulation slows down repeatedly at the beginning of the simulation and at time $t = $ ~1, 4, 5, 8, and 9 seconds.

**5** Zoom to examine the data between time $t = 0.8$ and 1.03 seconds.

```
h1;
xStart = 0;
xEnd = 10;
yStart = 0;
yEnd = 10e0;
xZoomStart1 = 0.8;
xZoomEnd1 = 1.03;
```

```
yZoomStart1 = 10e-20;
yZoomEnd1 = 10e-1;
axis([xZoomStart1 xZoomEnd1 yZoomStart1 yZoomEnd1])
```
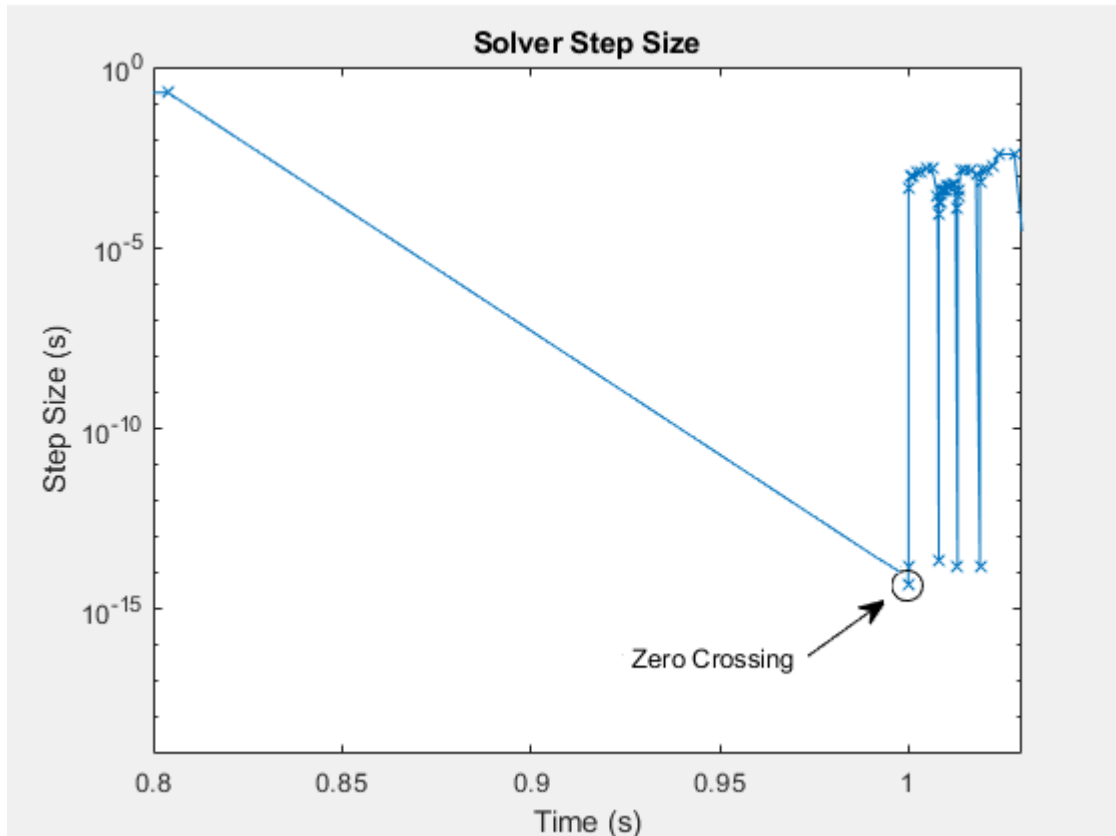


The blue x markers in the figure indicate that the simulation has completed executing a step. The circled markers indicate a very small step size and represent zero-crossing events. The step size decreases to approximately 10e-15 seconds for each zero-crossing detection.

**6**  To obtain the reference results for motor speed, open the Measurements subsystem. Select the

Ideal Rotational Motion Sensor block,  . With the block selected, use the `simscape.logging.findNode` function to find and save the node that contains the data for W, the signal for the angular velocity of the motor.

```
nRef = simscape.logging.findNode(simlogRef,gcbh)

nRef =

  Node with properties:

            id: 'Ideal_Rotational_Motion_Sensor'
       savable: 1
    exportable: 0
           phi: [1×1 simscape.logging.Node]
             C: [1×1 simscape.logging.Node]
             R: [1×1 simscape.logging.Node]
             A: [1×1 simscape.logging.Node]
```

**11-57**

```
            w: [1×1 simscape.logging.Node]
            t: [1×1 simscape.logging.Node]
            W: [1×1 simscape.logging.Node]
```

**7**  Use the `simscape.logging.plot` function to plot the reference results for W.

```
simscape.logging.plot(nRef.W);
```
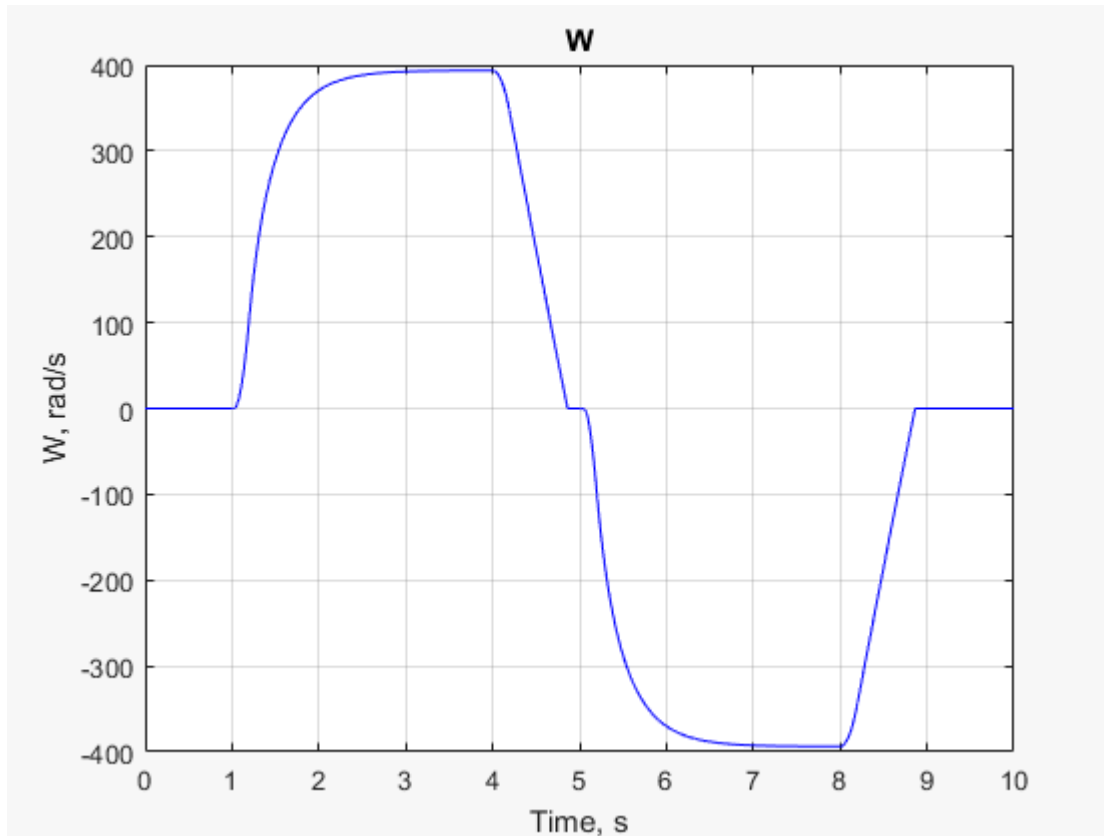


## Identify and Modify Elements That Cause Zero Crossings

Analyze the simulation data to determine the elements responsible for the zero crossings. Modify the model to reduce the number of zero crossings that those elements cause.

**1**  Use the Simscape `sscprintzcs` function to print zero-crossing information for logged simulation data.

```
sscprintzcs(simlogRef)
```

```
ssc_pneumatic_rts_stiffness_redux (42 signals, 50 crossings)
  +-Directional_5_way_valve (38 signals, 46 crossings)
  | +-Variable_Area_Orifice_1 (9 signals, 13 crossings)
  | +-Variable_Area_Orifice_2 (9 signals, 10 crossings)
  | +-Variable_Area_Orifice_3 (9 signals, 14 crossings)
  | +-Variable_Area_Orifice_4 (11 signals, 9 crossings)
  +-Pipe_1 (2 signals, 0 crossings)
  | +-Constant_Chamber (2 signals, 0 crossings)
  +-Pneumatic_Motor (2 signals, 4 crossings)
```

The results show that the 50 detected zero crossings occur in the Directional 5-way valve block (46 crossings) and the Pneumatic Motor block (4 crossings).

**2** Use the `sscexplore` function to open the Simscape Results Explorer to interact with logged simulation data.

`sscexplore(simlogRef)`

**3** In the results tree, click **Pneumatic Motor** to see the results for the motor.
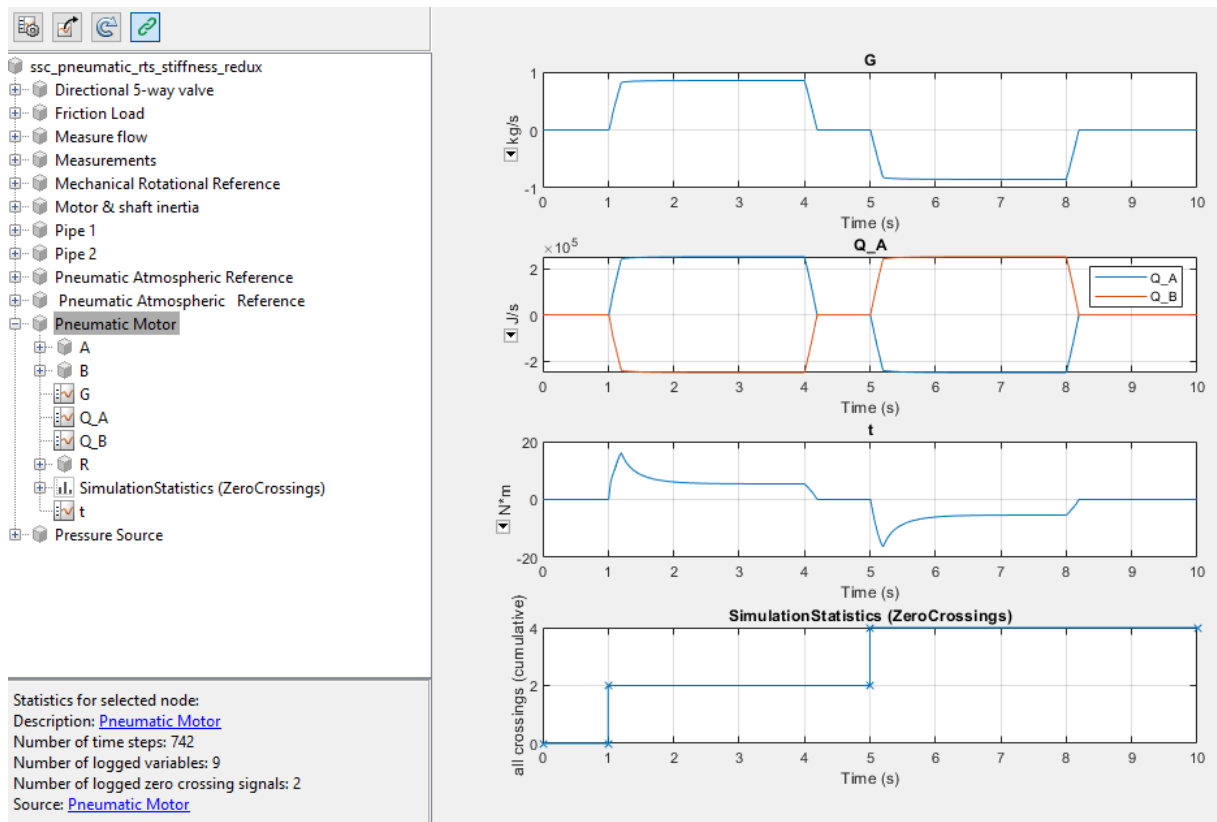


Most of the zero crossings occur between $t = 0$ and $t = 1$ seconds, when the other signals in the block are near zero. The few remaining zero crossings occur at approximately $t = 5$ seconds.

**4** To identify the source code that triggers some of the zero crossings, select **Directional 5-way valve > Variable Area Orifice 2 > SimulationStatistics (ZeroCrossings) >** ✦ **zc_1 - 8 crossings**. Click the **Pneumatic.Elements.VariableOrifice** link that appears in the lower, left corner of the window.
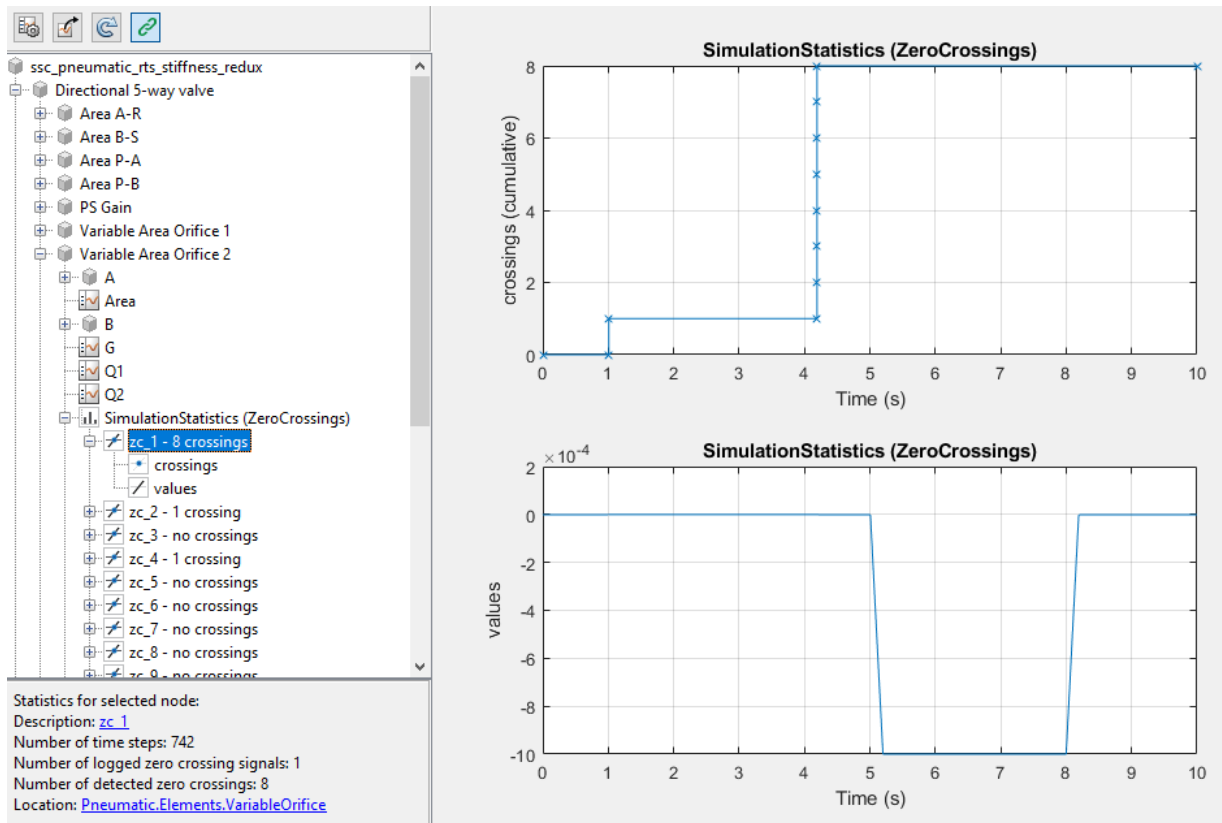
The source code for the Pneumatic Motor block opens with the cursor at this code:

```
% Area - limit to be greater than Area0
AreaL = if Area<Area0, Area0 else Area end;
```

The conditional statement that is responsible for the zero crossings is related to the orifice area.

5    Decrease the number of zero crossings, by decreasing the maximum orifice area of the Directional 5-way valve. Open the Directional 5-way valve block property inspector and specify 995 for the **Maximum orifice area (mm^2)** parameter.

## Analyze the Results of the Modified Model

Compare the results to the reference results to ensure the accuracy of your modified model. Confirm that your modified model has fewer zero crossings.

1    Simulate the model and print the zero crossing data.

```
sim(model)
sscprintzcs(simlog)

ssc_pneumatic_rts_stiffness_redux (42 signals, 30 crossings)
  +-Directional_5_way_valve (38 signals, 26 crossings)
  | +-Variable_Area_Orifice_1 (9 signals, 7 crossings)
  | +-Variable_Area_Orifice_2 (9 signals, 6 crossings)
  | +-Variable_Area_Orifice_3 (9 signals, 8 crossings)
  | +-Variable_Area_Orifice_4 (11 signals, 5 crossings)
  +-Pipe_1 (2 signals, 0 crossings)
```

```
| +-Constant_Chamber (2 signals, 0 crossings)
+-Pneumatic_Motor (2 signals, 4 crossings)
```

The overall number of zero crossings has decreased from 50 to 30.

**2** Compare the results using the `simscape.logging.plot` function to plot the reference results and the results from the modified model to a single plot:

```
simscape.logging.plot...
    ({simlogRef.Measurements.Ideal_Rotational_Motion_Sensor.W...
    simlog.Measurements.Ideal_Rotational_Motion_Sensor.W}, ...
    'names', {'Reference','Modified'})
```
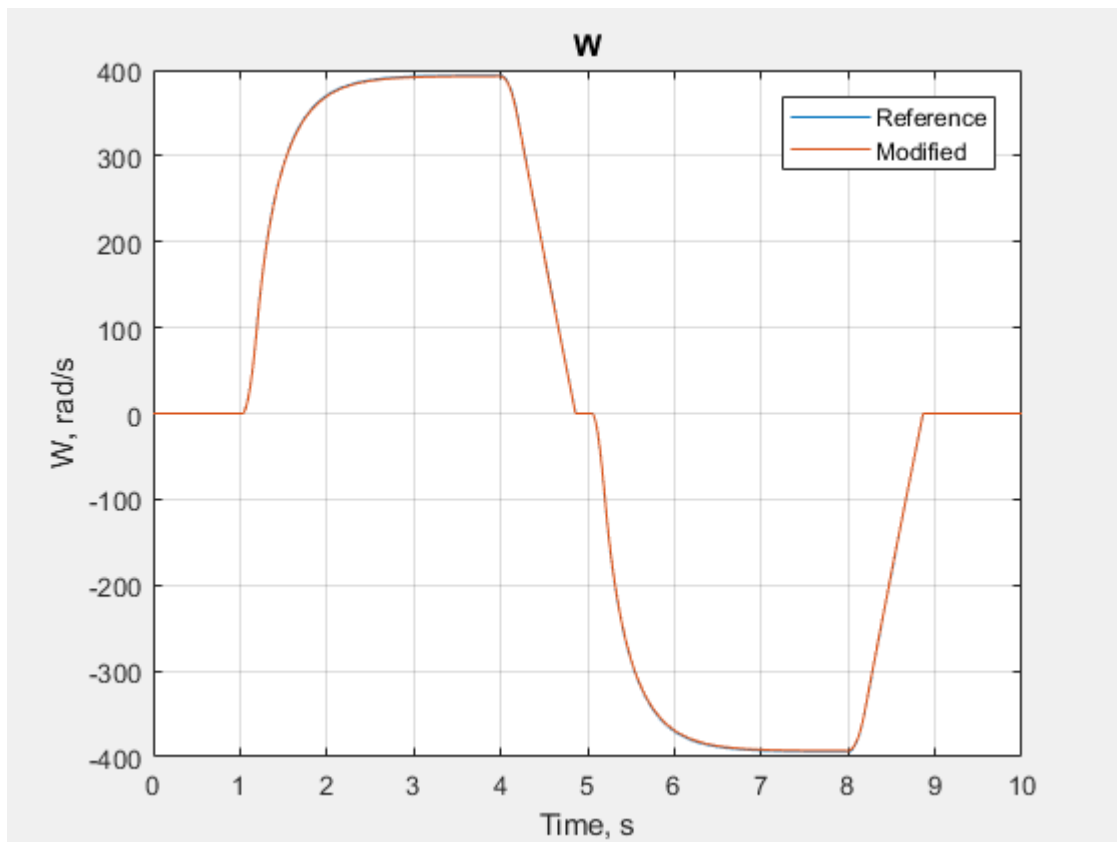


The results look the same.

**3** Zoom control for a closer look at the inflection point at t = ~ 5 seconds.

```
xStart = 0;
xEnd = 10;
yStart = -400;
yEnd = 400;
xZoomStart1 = 4.75;
xZoomEnd1 = 5.15;
yZoomStart1 = -20;
yZoomEnd1 = 30;
axis([xZoomStart1 xZoomEnd1 yZoomStart1 yZoomEnd1])
```
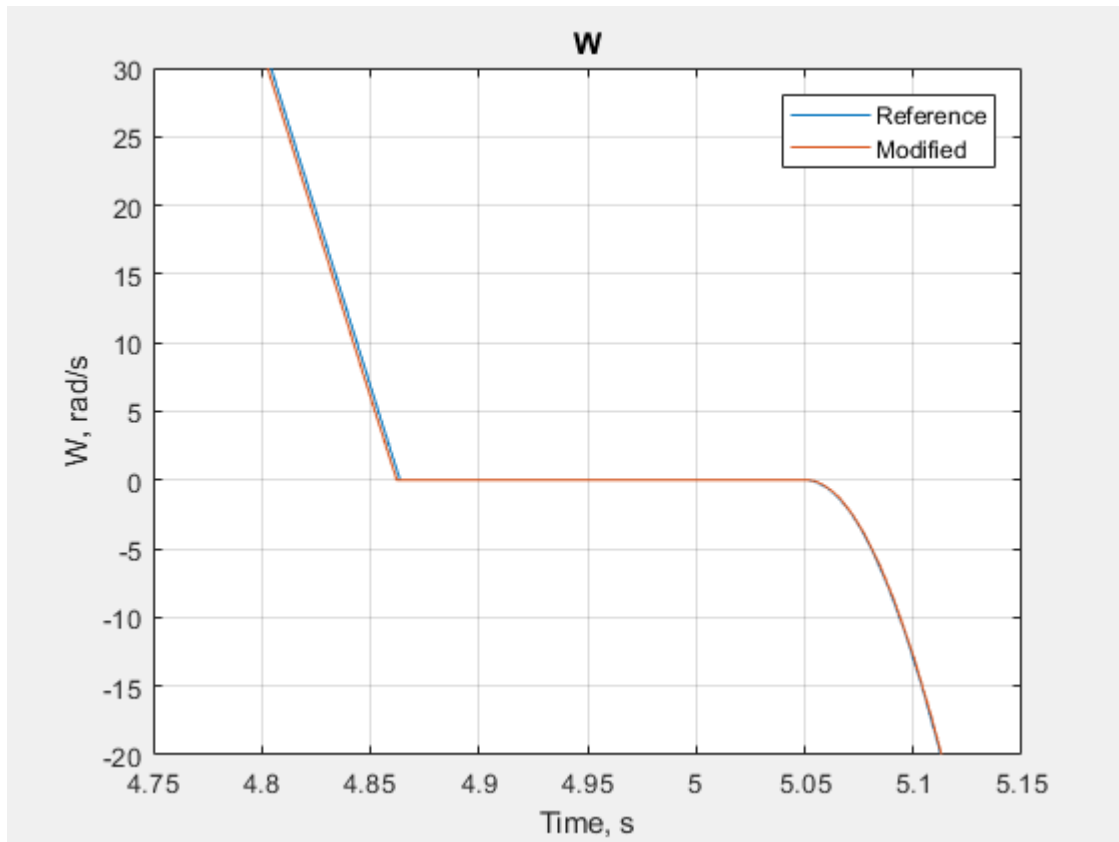
At this zoom level, you can see a small difference in the results for the modified model. However the simulation is accurate enough that the results meet expectations based on empirical and theoretical data.

To improve simulation speed further before performing the real-time simulation workflow with this model, try:

- Repeating the method shown in this example to identify and adjust other elements that cause zero crossings that are responsible for the small steps
- Reducing any numerical stiffness that is responsible for the small steps

## See Also
`simscape.logging.plotxy` | `simscape.logging.findNode` | `sscexplore` | `sscprintzcs`

## See Also

## Related Examples
- "Determine Step Size" on page 11-12
- "Estimate Computation Costs" on page 11-87
- "Reduce Computation Costs" on page 11-25
- "Reduce Fast Dynamics" on page 11-29

- "Reduce Numerical Stiffness" on page 11-47
- "Log, Navigate, and Plot Simulation Data" on page 13-20

## More About

- "About Simulation Data Logging" on page 13-2
- "About the Simscape Results Explorer" on page 13-24
- "Events and Zero Crossings" on page 7-3
- "Model Preparation Objectives" on page 11-2
- "Improving Speed and Accuracy" on page 11-8
- "Real-Time Model Preparation Workflow" on page 11-4
- "Using Conditional Expressions in Equations"
- "Zero-Crossing Detection"

# Partition a Model

You can make your model real-time capable by dividing the computational cost for simulation between multiple processors via model partitioning. Computational cost is a measure of the number and complexity of tasks that a central processing unit (CPU) performs per time step during a simulation. A high computational cost can slow simulation execution speed and cause overruns when you simulate in real time on a single CPU.

Typically, you can lower computational costs enough for real-time simulation on a single processor by adjusting model fidelity and solver settings using methods described in "Real-Time Model Preparation Workflow" on page 11-4. However, it is possible that there is no combination of model complexity and solver settings that can make your model real-time capable on a single CPU on your target machine. If your real-time simulation using a single CPU does not run to completion, or if the results from the simulation are not acceptable, partition your model. You can run a partitioned model using a single, multi-core target machine or multiple, single-core target machines.

This example shows you how to partition your model into two discrete subsystems, one that contains the plant, and one that contains the controller, for parallel processing on individual real-time CPUs.

1. Open the model. At the MATLAB command prompt, enter

   ```
   model = 'ssc_hydraulic_actuator_digital_control';
   open_system(model)
   ```
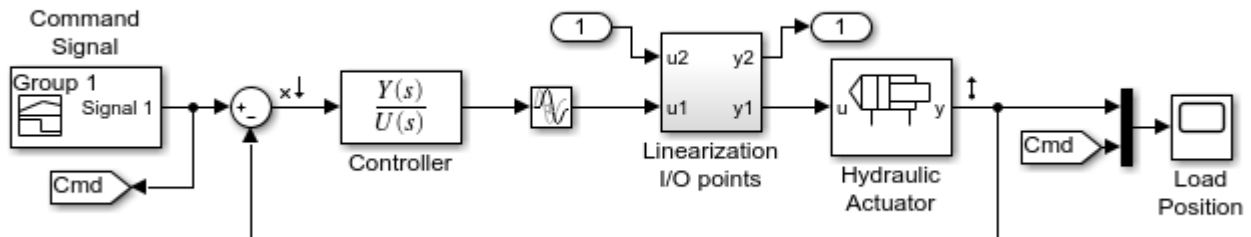


In addition to signal routing and monitoring blocks, the model contains these blocks:

- Command Signal — A Signal Builder block that generates the input reference signal, $r$.
- Sum — A block that compares the reference signal, $r$, from the Command Signal block to the output signal, $y$, from the Hydraulic Actuator to generate the error, $x$, that is $r - y = x$.
- Controller — A continuous Transfer Fcn block. The **Numerator coefficients** and **Denominator coefficients** parameters for this block are specified by the variables num and den.
- Transport Delay — A block that simulates time delay for a continuous input signal.

   **Note** By default, Simulink Editor hides the automatic block names in model diagrams. To display hidden block names for training purposes, clear the **Hide Automatic Block Names** check box. For more information, see "Manage Block Names and Ports".

- Linearization I/O — A subsystem that linearizes the model about an operating point.

- Hydraulic Actuator — A subsystem that contains the Simscape plant model.

2    Examine the variables in the workspace by clicking each variable in turn.

- The variable for sample time, *ts* = `0.001`.
- The **Numerator coefficients** parameter, *num* = `-0.5`.
- The **Denominator coefficients** parameter, *den* = `[0.001 1]`.
- The variable *ClosedLoop* = `1`.

3    Simulate the model and open the Load Position scope to examine the results.
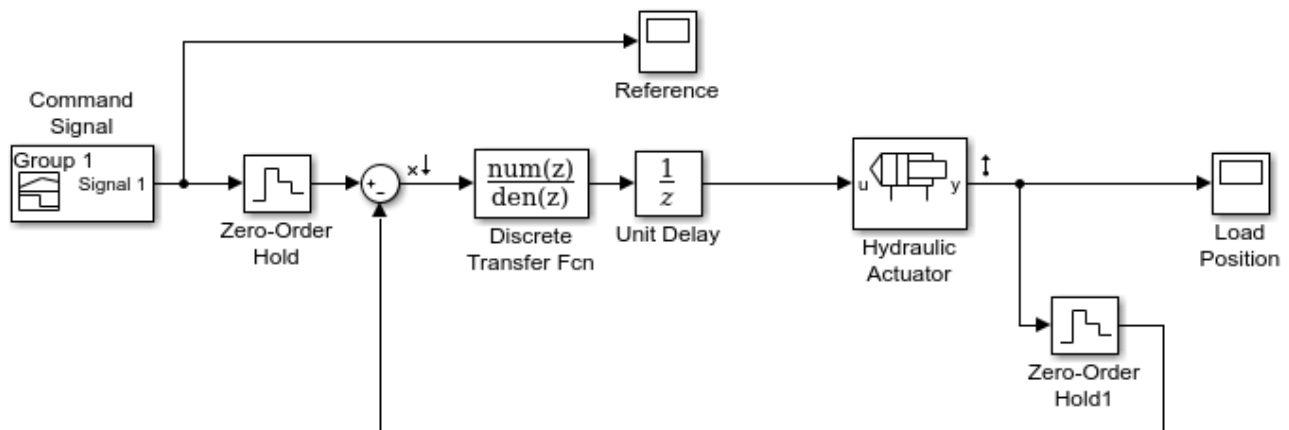
```
sim(model)
open_system([model, '/Load Position'])
```
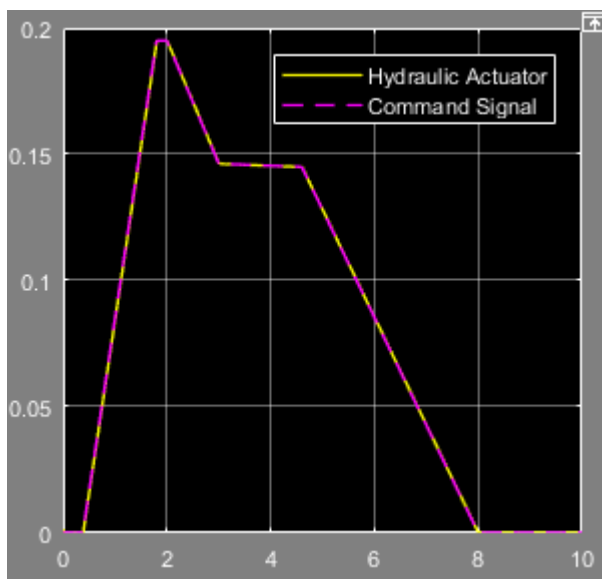


The output from the hydraulic actuator matches the command signal.

4    Eliminate items that add to the computational cost but which do not affect the results of real-time simulation. In the example model, because the closed loop gain is 1, such items include the Linearization I/O points, In1, and In2 blocks. Delete the three blocks and the lines that interconnect them.

5    Configure the model for visualization.

   a    Delete the Mux block.
   b    Delete the Goto and From blocks that are named Cmd.
   c    Connect the Load Position Scope block to the output signal from the Hydraulic Actuator.
   d    Add a second Scope block.
   e    Connect the new Scope block to the unconnected connection line from the Command Signal.
   f    Change the name of the new Scope block to `Reference`.

6    Replace the Transport Delay block with a Unit Delay block.

   a    Delete the Transport Delay block and the open ended connection line that is connected to the outport of the block.

    **b**    Add the Unit Delay block from the Simulink Discrete library and connect it to the input port of the Hydraulic Actuator Subsystem.

    **c**    For the **Sample time (-1 for inherited)** parameter of the Unit Delay block, specify `ts`.

**7**    Replace the Controller block with a Discrete Transfer Fcn block from the Simulink Discrete library.

    **a**    Delete the Controller block.

    **b**    Click in the model window and enter `discrete transfer fcn`. When the dropdown menu that contains the block appears, click `Discrete Transfer Fcn`.

    **c**    Connect the new block to the open-ended connection line from the Sum block.

    **d**    Connect the outport of the new block to the inport of the Unit Delay block.

    **e**    Specify parameters for the discrete controller using a Tustin transformation of the original, continuous transfer function.

        **i**    At the MATLAB command line, save new variables based on the original coefficients:

```
k = num;
alpha = den(1,1);
```

        **ii**    For the Discrete Transfer Fcn block **Numerator** parameter, specify `[k*ts k*ts]`.

        **iii**    For the **Denominator** parameter, specify `[2*alpha+ts ts-2*alpha]`.

        **iv**    For the **Sample time (-1 for inherited)** parameter, specify `ts`.

**8**    Provide digital sampling for continuous time measurements using Zero-Order Hold blocks.

    **a**    Add Zero-Order Hold blocks to both signals that are input to the Sum block.

    **b**    For the **Sample time (-1 for inherited)** parameter of both Zero-Order Hold blocks, specify `ts`.

**9**    Connect the blocks as shown in the figure.



**10**    Simulate the model and open the Load Position scope to see how the modifications affect the results.

```
sim(model)
open_system([model, '/Load Position'])
```
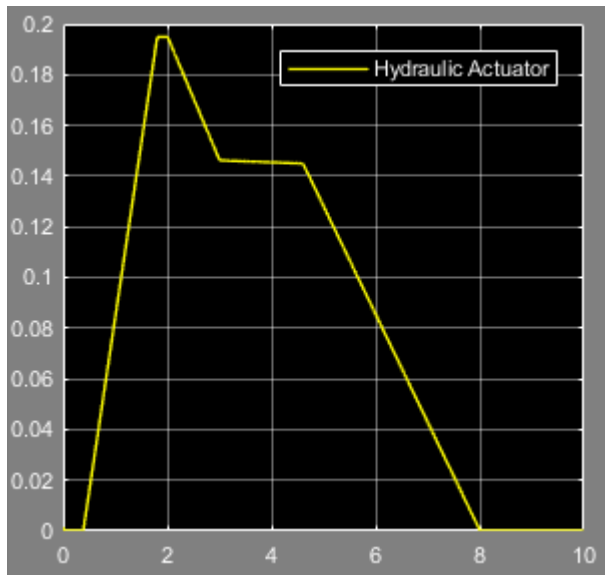
The output from the hydraulic actuator matches the original results.

**11** Configure the solvers.

**a** To configure the global solver, open the model configuration parameters, and in the **Solver** pane:

- Set the solver **Type** to `Fixed-step`.
- Set the **Solver** to `discrete (no continuous states)`.
- Specify `ts` for the **Fixed-step size (fundamental sample time)** parameter.
- Click **OK**.

**b** To configure the local solver, open the Hydraulic Actuator subsystem and update these parameters for the Solver Configuration block:

- Select the option to **Use local solver**.
- Specify `ts` for the **Sample time**.
- Select the option to **Use fixed-cost runtime consistency iterations**.
- Click **OK**.

**12** Partition the model into two subsystems:

**a** Create a subsystem that contains these blocks:

- Command Signal
- Reference
- Zero-Order Hold
- Sum
- Discrete Transfer Fcn
- Unit Delay

**b** Label the subsystem `Controller Subsystem`.

**c** Open the Controller Subsystem.

**11-67**

**d** Rename the Out1 Outport block as `u`.

**e** Rename the In1 Inport block as `y`.

**f** Navigate to the top model.

**g** Create a second subsystem that contains these blocks:

- Hydraulic Actuator
- Zero-Order Hold1
- Load Position

**h** Label the subsystem `Plant Subsystem`.

**i** Open the Plant Subsystem.

**j** Rename the Out1 Outport block as `u_plant`.

**k** Rename the In1 Inport block as `y_plant`.

**l** To see the partitioned subsystems, navigate to the top model.
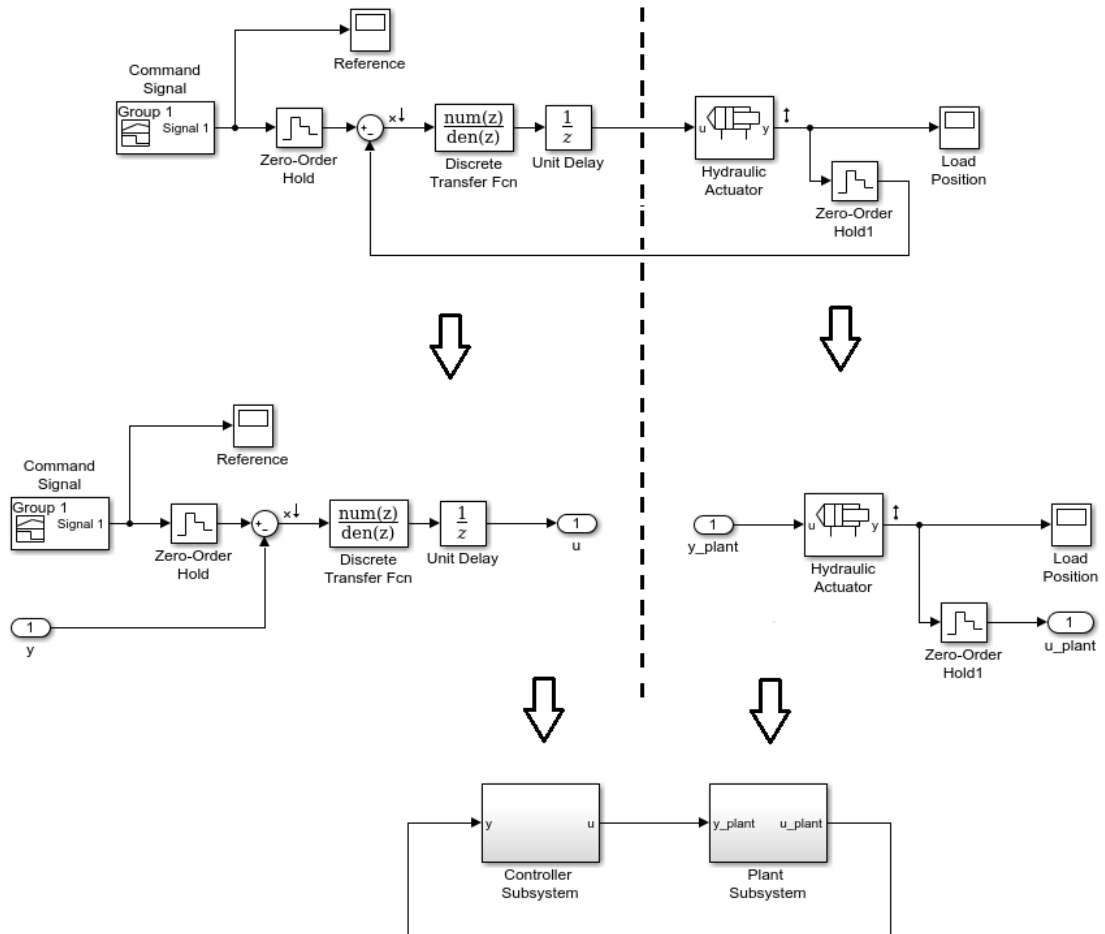


This model is partitioned for concurrent execution. To learn how to add tasks, and map individual tasks to partitions, see "Partition Your Model Using Explicit Partitioning".

## See Also

Discrete Transfer Fcn | Unit Delay | Zero-Order Hold

## Related Examples

- "Determine Step Size" on page 11-12
- "Estimate Computation Costs" on page 11-87
- "Reduce Computation Costs" on page 11-25

## More About

- "Real-Time Model Preparation Workflow" on page 11-4
- "Model Preparation Objectives" on page 11-2
- "Implicit and Explicit Partitioning of Models"
- "Multicore Programming with Simulink"

## External Websites

- Concurrent Execution with Simulink Real-Time and Multicore Target Hardware

# Manage Model Variants

Variant blocks allow you to create a single model that caters to multiple variant requirements. Such models have a fixed common structure and a finite set of variable components. The variable components are activated depending on the variant choice that you select. Thus, the resultant active model is a combination of the fixed structure and the variable components based on the variant choice. The use of variant blocks in a model helps in reusability of the model for different conditional expressions called variant choices. For more information and examples, see "Variant Subsystems".

However, you cannot simulate on real-time target hardware using code that does not specify default variant choices. Before you generate code for real-time simulation, use the Variant Manager to identify variant blocks in your model and to manage the variation points that are modeled using those blocks. To learn how to use the variant manager, see "Variant Manager Overview".

## Limitations

Simscape does not support conditional compilation for model variants.

## See Also
Variant Subsystem, Variant Model

## More About
- "Prepare Variant-Containing Model for Code Generation"
- "Variant Manager Overview"
- "Variant Subsystems"
- "What Are Variants and When to Use Them"
- "Working with Variant Choices"

# Fixed-Cost Simulation for Real-Time Viability

The step size and number of iterations that you specify affect the computational cost of your real-time simulation. As you decrease the step size or increase the number of iterations, the results become more accurate, but the simulation costs more so it can take longer to simulate. Simulation overrun occurs if the step size is too small or if there are too many iterations for the solver to calculate a solution in a single real-time computational frame.

Limit the computational cost by specifying the solver step size and, for implicit solvers, the number of iterations for the Simulink global solver and for each Simscape local solver in your model.

For best results when specifying the step size of a fixed-step solver for real-time simulation:

- Specify a sample time that results in time steps that are no greater than the maximum step size.
- Specify the sample time for each local solver independently and as an integer multiple of the sample time that you specify for the global solver.
- Choose a step size that is larger than the minimum step size for required speed and smaller than the maximum step size for required accuracy.

To configure the number of iterations for real-time simulation with a fixed-step solver:

- For local solvers, specify the number of nonlinear iterations for each independently configured Solver Configuration block.
- For global solvers ode14x and ode1be, specify the number of Newton's iterations.

To optimize the number of iterations your model uses for fixed-cost, use the `simscape.getLocalSolverFixedCostInfo` function.

## See Also

`simscape.getLocalSolverFixedCostInfo`

## Related Examples

- "Choose Step Size and Number of Iterations" on page 11-89

## More About

- "Simulating with Fixed Cost" on page 7-19
- "Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers" on page 7-18
- "Solvers for Real-Time Simulation" on page 11-76

# Real-Time Simulation Workflow

| **In this section...** |
|---|
| "Make Your Model Real-Time Viable" on page 11-74 |
| "Insufficient Computational Capability for Real-Time Viability" on page 11-75 |

The figure shows the real-time simulation workflow. The connectors are exit points for returning to the real-time model preparation workflow.
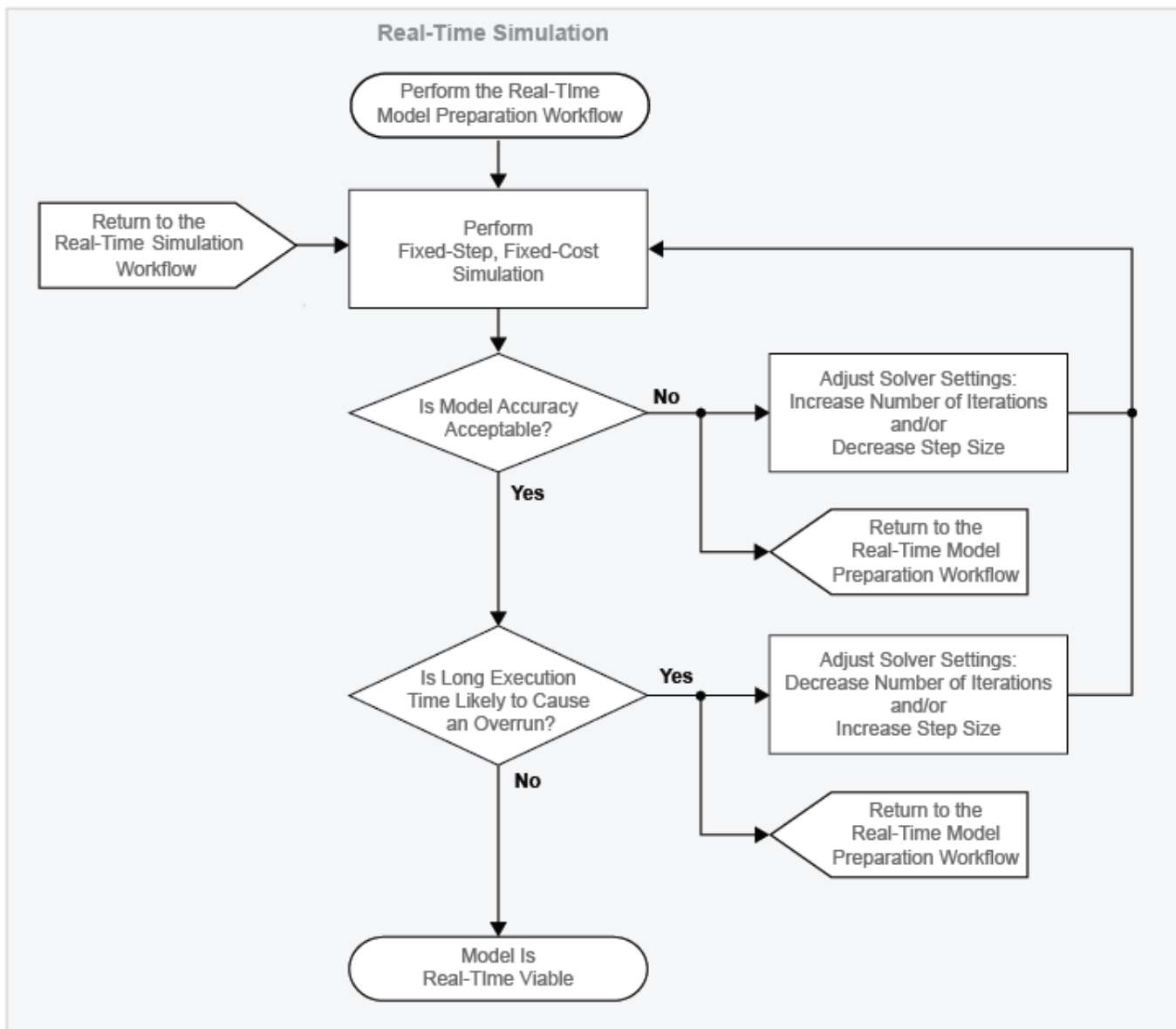


The figure shows the real-time model preparation workflow. The connector is an entry point for returning to the real-time model preparation workflow from other real-time workflows (for example, the real-time simulation workflow or the hardware-in-the-loop simulation workflow).
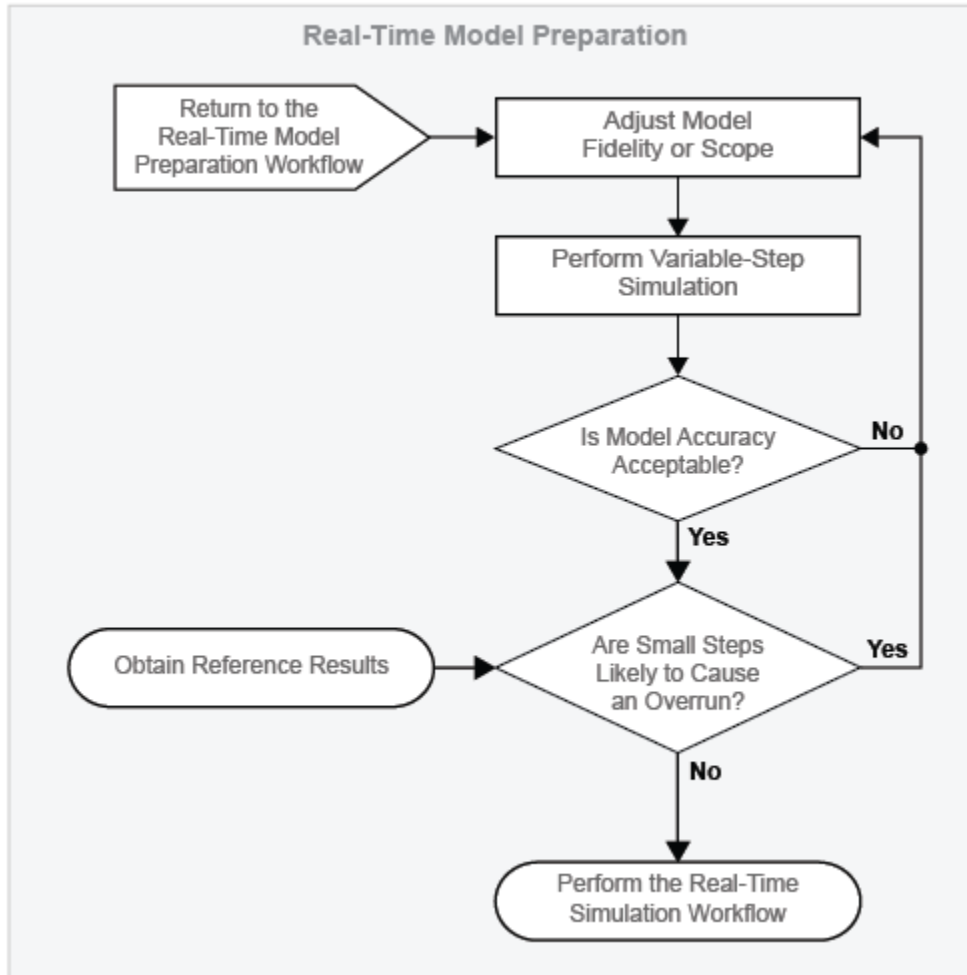
Before performing this workflow, prepare your model for real-time simulation using the "Real-Time Model Preparation Workflow" on page 11-4. The real-time model preparation workflow shows you how to obtain reference results, determine the maximum step size, and modify your model to simulate quickly and produce accurate results.

Use the real-time simulation workflow to increase the likelihood that your model is real-time capable. Your model is real-time capable if it meets both of these criteria when you simulate it on your real-time computer:

- The results match your expectations, based on empirical data or theoretical models.
- The model simulates without incurring an overrun.

The real-time simulation workflow uses bounded, that is fixed-step, fixed-cost, simulation. Fixed-step, fixed-cost simulation sets an upper boundary on computational cost by limiting both the step size and the number of iterations that the solver uses.

## Make Your Model Real-Time Viable

### Perform Fixed-Step, Fixed-Cost Simulation

Run your model on a desktop computer using fixed-step, fixed-cost configurations for the global solver and local solvers. For more information on specifying fixed-step, fixed-cost solver configurations for real-time simulation, see "Choose Step Size and Number of Iterations" on page 11-89 and "Fixed-Cost Simulation for Real-Time Viability" on page 11-71.

### Evaluate Model Accuracy

Compare the results from the simulation on the target computer to your reference results. Are the reference and modified model results the same? If not, are they similar enough that the empirical or theoretical data also supports the results from the simulation of the modified model? Is the modified model representing the phenomena that you want it to measure? Is it representing those phenomena correctly? If you plan on using your model to test your controller design, is the model accurate enough to produce results that you can rely on for system qualification? The answers to these questions help you to decide if your real-time results are accurate enough.

### Improve Accuracy by Adjusting Solver Settings

If your fixed-step, fixed-cost simulation results do not match your reference results, try to improve accuracy by adjusting solver configurations. Increasing the number of iterations or decreasing the step size can improve accuracy.

For an implicit global solver (ode14x, ode1be), increase the number of Newton's iterations. For a Backward Euler or Trapezoidal Rule local solver, increase the number of nonlinear iterations.

For the global solver, and for any local solvers, decrease the step size. Configure the step size for each local solver as an integer multiple of the step size you specify for the global solver.

### Return to the Real-Time Model Preparation Workflow

If changing solver configurations does not improve or speed enough, try to make your model real-time capable by returning to the real-time model preparation workflow.

Adjust the fidelity or scope of your model, and then step through the other processes and decisions in the real-time model preparation workflow. Iterate on adjusting, simulating, and analyzing your model until it is fast and accurate enough for you to attempt the real-time simulation workflow again. For information, see "Real-Time Model Preparation Workflow" on page 11-4.

### Evaluate Overrun Risk

In terms of speed, the only method for definitively determining that your model is real-time capable is to test for overruns during simulation on your target hardware. You can, however, use fixed-step, fixed-cost simulation to estimate the likelihood that your solver executes quickly enough for real-time simulation. For information on estimating simulation time, see "Estimate Computation Costs" on page 11-87.

### Improve Simulation Speed by Adjusting Solver Settings

If your computational cost estimate indicates that your model executes too slowly to avoid an overrun on a real-time target machine, try to increase simulation speed by adjusting solver configurations. Decreasing the number of iterations or increasing the step size can improve accuracy.

For an implicit global solver (ode14x, ode1be), decrease the number of Newton's iterations. For either a Backward Euler or Trapezoidal Rule local solver, decrease the number of nonlinear iterations.

For the global solver, and for any local solvers, increase the step size. Configure the step size for each local solver as an integer multiple of the step size you specify for the global solver.

**Model Is Real-Time Viable**

When fixed-step, fixed-cost simulation results indicate that your model is likely real-time capable, you can attempt real-time simulation on the target hardware. For information on how you can use real-time simulation to test your controller hardware, see "Basics of Hardware-In-The-Loop simulation" on page 11-103.

**Return to the Real-Time Simulation Workflow**

The connector is an entry point for returning to the real-time simulation workflow from another workflow (for example, the hardware-in-the-loop simulation workflow).

## Insufficient Computational Capability for Real-Time Viability

It is possible that your real-time target machine lacks the computational capability for running your model in real time. If, after multiple iterations of the workflow, there is no combination of model complexity and solver settings that makes your model real-time viable, consider these options for increasing processing power:

- "Upgrading Target Hardware" on page 11-10
- "Simulating Parts of the System in Parallel" on page 11-10

## See Also

## Related Examples

- "Choose Step Size and Number of Iterations" on page 11-89
- "Determine System Stiffness" on page 11-82
- "Estimate Computation Costs" on page 11-87

## More About

- "Fixed-Cost Simulation for Real-Time Viability" on page 11-71
- "Hardware-In-The-Loop Simulation Workflow" on page 11-106
- "Improving Speed and Accuracy" on page 11-8
- "Real-Time Model Preparation Workflow" on page 11-4
- "Solvers for Real-Time Simulation" on page 11-76
- "Basics of Hardware-In-The-Loop simulation" on page 11-103

# Solvers for Real-Time Simulation

| In this section... |
| --- |
| "Choosing Between Discrete and Continuous Solvers" on page 11-76 |
| "Computational Cost for Continuous Solvers" on page 11-77 |
| "How Numerical Stiffness Affects Solver Choice" on page 11-77 |
| "Using Simscape Local Fixed-Step Solvers" on page 11-78 |

To run your model on a real-time target machine, configure your model for fixed-step, fixed-cost simulation. The type of fixed-step solver, step size, and number of iterations that you specify affect the speed and accuracy of your real-time simulation.

Each distinct Simscape physical network in your model has its own Simscape Solver Configuration block. You can set the solver choice differently for each physical network. If you do not check the local solver option for a physical network, then that network uses the Simulink global solver that you specify.

When choosing a fixed-step solver type, the main factors to consider for each network in your model are:

- Whether the network is discrete or continuous
- The computational cost of the solver
- The numerical stiffness of the network

The following table summarizes the types of fixed-step solvers in the Simulink and Simscape libraries.

| Realm | Type | Numerical Method | Solver |
| --- | --- | --- | --- |
| Simulink global solver | Continuous | Explicit | ode1 (Euler's method) |
| | | | ode2 (Huen's method) |
| | | | ode3 (Bogacki-Shampine) |
| | | | ode4 (Fourth-Order Runge-Kutta, RK4) |
| | | | ode5 (Dormand-Prince,RK5) |
| | | | ode8 (Dormand-Prince, RK8) |
| | | Implicit | ode14x (extrapolation) |
| | | | ode1be (Backward Euler) |
| | Discrete | Not applicable | Discrete (no continuous states) |
| Simscape local network | Continuous | Implicit | Backward Euler |
| | | | Trapezoidal Rule |
| | | | Partitioning |

## Choosing Between Discrete and Continuous Solvers

To perform real-time simulation on a discrete model, for example, for the design of a digital controller, specify the Simulink global discrete solver. If the network that contains the controller has any continuous states, discretize the network. For an example that shows how to discretize the

controller for the hydraulic actuator, see "Hydraulic Actuator Configured for HIL Testing" on page 20-116.

---

**Note** A physical network using a local solver appears to the global Simulink solver as if it has discrete states.

---

If your controller model does contain continuous states, for example, if you are modeling an analog controller, use a Simulink global continuous solver.

## Computational Cost for Continuous Solvers

Computation cost is the number of calculations per time step that a processor performs. Real-time readiness varies inversely with computation cost. The lower the computational cost of a model is, the more likely it is that a real-time simulation of the model proceeds without overruns and generates sufficiently accurate results.

The figure shows the normalized computational cost of most global and local continuous fixed-step solvers. The data comes from a series of fixed-step, fixed-cost simulations using the different solver types. The model is nonlinear and contains one physical network. Although the solver type varies, the simulations use the same step size and a similar setting for the total number of solver iterations. They do so because the step size and number of iterations also affect the computational cost of a simulation.
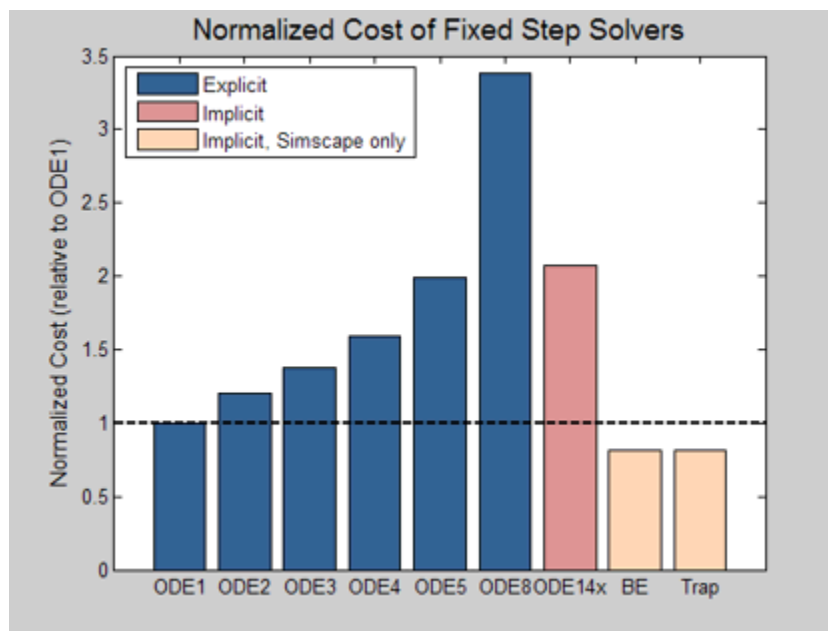


For a given accuracy, explicit global solvers generally have a lower computational cost than implicit global solvers. Local (Simscape only) solvers are less costly than global solvers.

## How Numerical Stiffness Affects Solver Choice

To determine whether to use an explicit or implicit fixed-step solver for simulating your model in real time, consider these two factors:

- The numerical stiffness of the system
- The computational cost of the solver

To determine if your system is stiff or nonstiff, simulate with different fixed-step solver configurations and compare results from each to the reference results. If the step size is too large, stiff systems can produce oscillations because they contain dynamics that vary both quickly and slowly. For more information, see "Stiffness of System" and "Determine System Stiffness" on page 11-82.

Explicit solvers are faster than implicit solvers, but they provide less accurate solutions for numerically stiff systems because they tend to damp out oscillations. Implicit solvers can better capture the oscillations that occur in stiff systems because they are more robust than explicit solvers. However, implicit solvers deliver better accuracy at the expense of speed.

If your controller model is continuous and numerically stiff, use the implicit solver ode14x. If ode14x does not allow your model to simulate fast enough for real-time simulation, at the expense of accuracy, you can:

- Improve simulation speed by increasing the step size or decreasing the number of iterations.
- Use the ode1be Backward Euler solver.
- Reduce the stiffness of your model and specify an explicit solver instead of ode14x.

To determine the explicit solver that is the best choice for your less stiff or numerically nonstiff, continuous controller model, perform bounded simulation using each of the explicit continuous solvers. Configure each solver to use the same step size and a similar number of solver iterations. Compare the simulation results and choose the solver that provides the best combination of accuracy and speed.

To increase the accuracy of the results that your explicit solver provides, at the expense of speed, decrease the step size or increase the number of iterations. For more information on configuring your model for fixed-step, fixed-cost simulation, and evaluating the results of bounded simulation, see "Choose Step Size and Number of Iterations" on page 11-89.

## Using Simscape Local Fixed-Step Solvers

You can usually further minimize computational cost by using a Simscape local solver for each independent physical network in your model. For similar levels of accuracy, local solvers have a lower computational cost than Simulink global solvers.

Simscape allows you to specify a different solver configuration for each independent physical system (subsystem) in your model. You can use an implicit fixed-step solver on the stiff local networks and an explicit fixed-step solver on the nonstiff local networks. Optimizing solvers for each network minimizes the overall number of computations done per time step and makes it more likely that the model can run in real time without generating an overrun.

Choose between three Simscape fixed-step solvers for real-time simulation.

- Backward Euler
- Trapezoidal Rule
- Partitioning

The Backward Euler solver is more robust, and therefore more stable than the Trapezoidal Rule solver. It tends to damp oscillations. The Trapezoidal Rule solver is more accurate, but less stable

than the Backward Euler solver. It tends to capture oscillations, like the sinusoid AC waveforms that are common to electrical systems. The Partitioning solver is also more robust than the Trapezoidal Rule solver, however, it cannot simulate certain models. For more information, see "Increase Simulation Speed Using the Partitioning Solver" on page 11-19. Regardless of the local solver you choose, the simulation uses the Backward Euler whenever numerical stability is at risk:

- At the start of simulation.
- After an instantaneous change, when the corresponding block undergoes an internal discrete change.

## See Also

Solver Configuration

## Related Examples

- "Determine System Stiffness" on page 11-82
- "Reduce Numerical Stiffness" on page 11-47
- "Choose Step Size and Number of Iterations" on page 11-89

## More About

- "Fixed-Cost Simulation for Real-Time Viability" on page 11-71
- "Making Optimal Solver Choices for Physical Simulation" on page 7-18
- "Compare Solvers"
- "Fixed Step Solvers in Simulink"

# Troubleshooting Real-Time Simulation Issues

| In this section... |
| --- |
| "Avoid Computer Overloads and Unacceptable Simulation Results" on page 11-80 |
| "Optimize Real-Time Application Execution Using Simscape Checks" on page 11-80 |

## Avoid Computer Overloads and Unacceptable Simulation Results

A model is not real-time capable if, during simulation on real-time target hardware, it overloads the CPU or produces results that do not match your theoretical calculations or experimental data. To make your model real-time capable, use the workflows in "Real-Time Model Preparation Workflow" on page 11-4 and "Real-Time Simulation Workflow" on page 11-72. For examples that show how to:

- Find step-size limits and configure solvers for real-time simulation, see "Determine Step Size" on page 11-12 and "Choose Step Size and Number of Iterations" on page 11-89.

- Analyze and modify the fidelity of your model for real-time simulation, see "Estimate Computation Costs" on page 11-87, "Reduce Computation Costs" on page 11-25, "Determine System Stiffness" on page 11-82, "Reduce Numerical Stiffness" on page 11-47, and "Reduce Zero Crossings" on page 11-55.

If you cannot find a combination of solver settings and model fidelity that makes your model real-time capable, consider one of these options:

- Execute your real-time application on a faster target machine.

- Configure the networks in your model so that they are independent of each other, and then partition them for parallel simulation on individual target computers. For information, see "Multicore Programming with Simulink"

## Optimize Real-Time Application Execution Using Simscape Checks

If you have a Simulink Real-Time™ license, you can optimize your model for real-time execution using the `Execute real-time application` activity mode in Performance Advisor. This mode includes several checks specific to physical models. Use Simulink Performance Advisor to identify:

- Simscape Solver Configuration blocks with settings that are suboptimal for real-time simulation. For optimal results, Solver Configuration blocks should have the following options selected: **Use local solver** and **Use fixed-cost runtime consistency iterations**.

- Simscape blocks that have a **Fluid dynamic compressibility** option that is suboptimal for real-time simulation. For optimal results, the **Fluid dynamic compressibility** option should be set to **Off**.

To access the checks, in the Performance Advisor window, under **Activity**, select `Execute real-time application`. In the left pane, expand the **Real-Time** folder, and then the **Simscape checks** folder. Run the Simscape checks.

For more information, see "Use Performance Advisor to Improve Simulation Efficiency".

**See Also**

**More About**

- "Model Preparation Objectives" on page 11-2
- "Real-Time Model Preparation Workflow" on page 11-4
- "Real-Time Simulation Workflow" on page 11-72

# Determine System Stiffness

| **In this section...** |
| --- |
| "Obtain Reference Results" on page 11-82 |
| "Simulate with an Implicit Fixed-Step Solver" on page 11-83 |
| "Simulate with an Explicit Fixed-Step Solver" on page 11-84 |
| "Analyze the Results" on page 11-85 |

Determining the numerical stiffness of your model helps you to decide between using an implicit or an explicit fixed-step solver for real-time simulation. To determine numerical stiffness, first use the real-time model preparation workflow to optimize the speed and accuracy of your model. Then, simulate your model using both explicit and implicit fixed-step solvers. Compare the simulation results to see how the solvers behave. If your model is numerically stiff, an explicit solver typically exhibits small oscillations around the desired solution.

Implicit solvers are more robust than explicit solvers, however, explicit solvers are faster. For robust results when performing real-time simulation with numerically stiff model, use an implicit fixed-step solver. If your model is not stiff, use an explicit solver to maximize simulation speed.

In this example, you obtain reference results by simulating a pneumatic model with a variable-step solver. You also configure and simulate the model using an implicit and then an explicit fixed-step global Simulink solver. Then you compare the results from all three simulations to determine if the pneumatic model is numerically stiff.

## Obtain Reference Results

1  To open the model, at the MATLAB command prompt, enter:

   ```
   ssc_pneumatic_rts_reference
   ```
2  Save the model as `stiffness_model` to a writable folder on the MATLAB path.
3  Simulate the model.
4  Assign the simulation results to new variables.

   ```
   yRef = yout;
   tRef = tout;
   ```
5  Plot the results of the variable-step simulation.

   ```
   h1 = figure;
   plot(tRef,yRef)
   h1Leg = legend({'Reference'});
   title('Speed')
   xlabel('Time (s)')
   ylabel('Speed (rpm)')
   ```
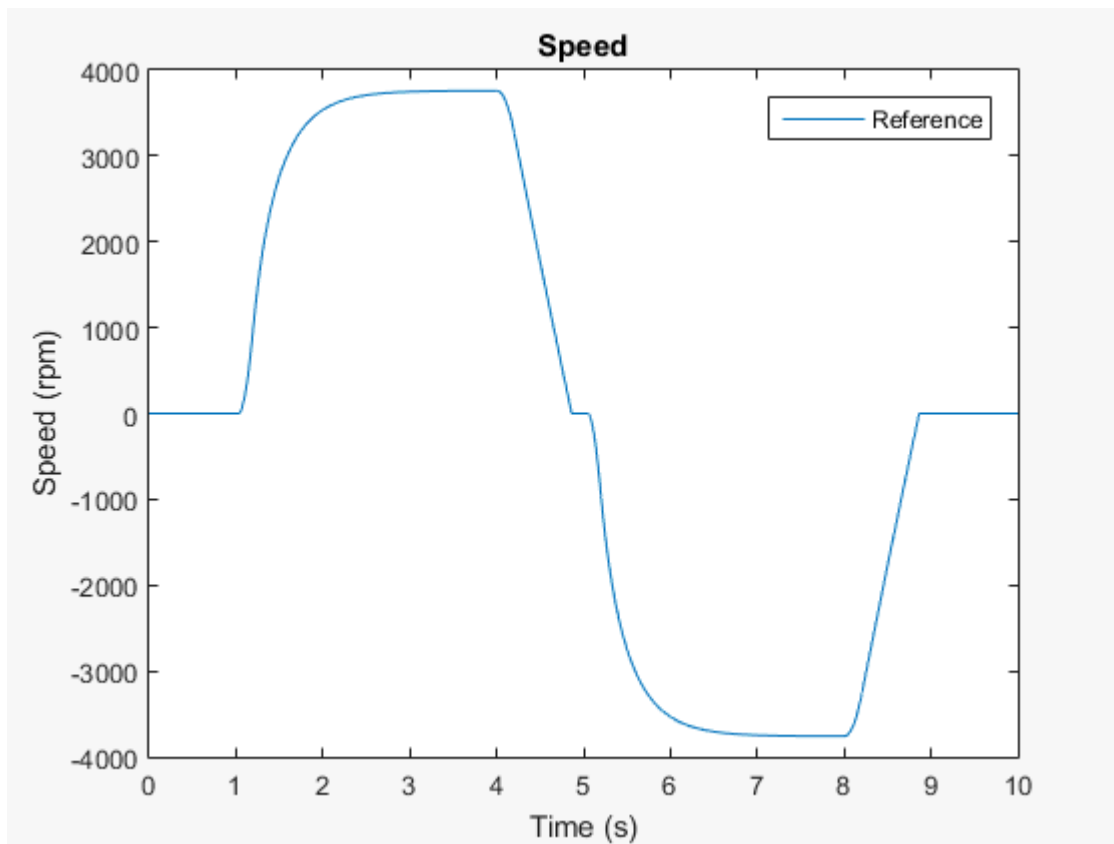
## Simulate with an Implicit Fixed-Step Solver

**1**   Configure the model for fixed-step simulation with implicit solver ode14x. In the configuration parameters Solver pane, set:

- **Type** to `Fixed-step`
- **Solver** to `ode14x (extrapolation)`
- Under **Additional options**, **Fixed-step size (fundamental sample time)** to `1e-3`
- **Number of Newton's iterations** to `1`.

Click **Apply**.

**2**   Simulate the model.

**3**   Assign the simulation results to new variables.

```
yOde14x = yout;
tOde14x = tout;
```

**4**   Use the `stairs` function to plot the results of the implicit fixed-step simulation so you can see how the solver behaves when it executes each step in the simulation.

```
h1
hold on
stairs(tOde14x,yOde14x,'g--')
h1Leg = legend({'Reference','Implicit Solver'});
```

**11-83**

The results appear the same.
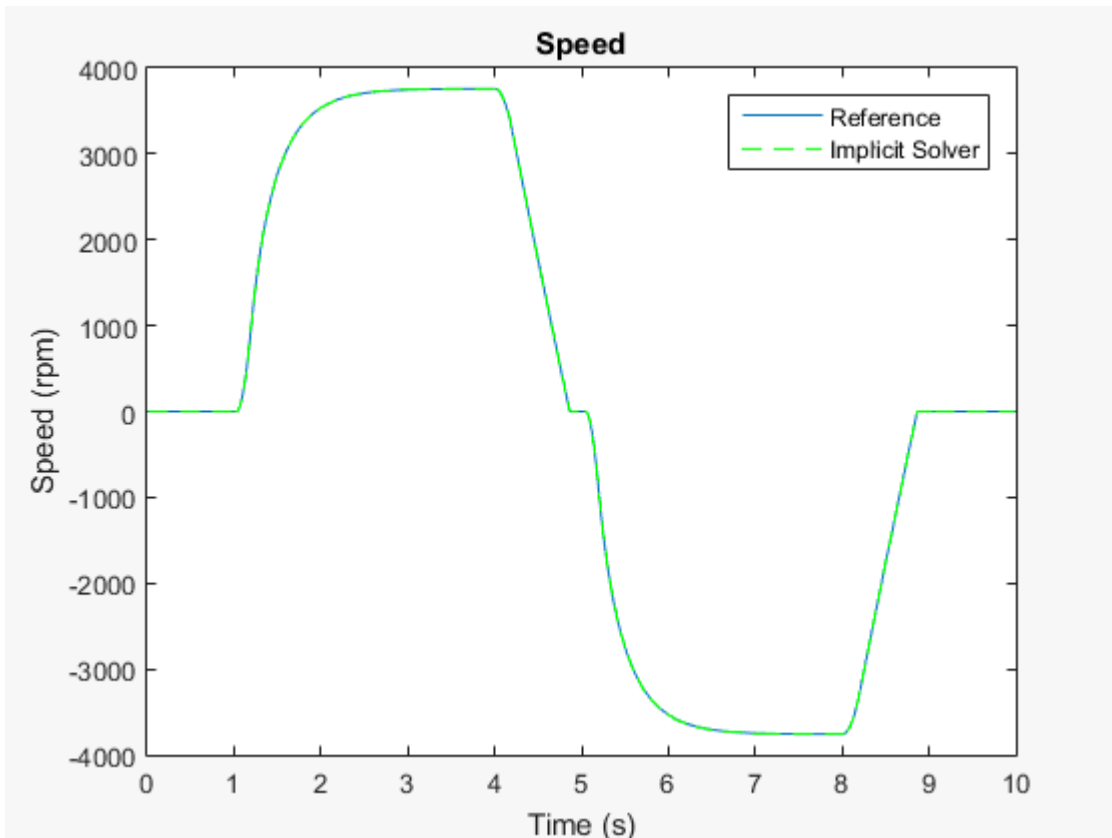


## Simulate with an Explicit Fixed-Step Solver

1   Configure the model for fixed-step simulation with explicit fixed-step solver ode5. In the configuration parameters Solver pane, set:

   - **Type** to `Fixed-step`
   - **Solver** to `ode5 (Dormand-Prince)`

   Click **OK**.

2   Filter the input signal to provide the required input derivative for the explicit solver. In the PS-S Simulink Converter block property inspector, on the **Input Handling** tab, set **Filtering and derivatives** to `Filter Input`. Click **OK**.

3   Simulate the model.

4   Assign the simulation results to new variables.

```
yOde5 = yout;
tOde5 = tout;
```

5   Use the `stairs` function to plot the results of the explicit fixed-step simulation.

```
h1
hold on
stairs(tOde5,yOde5,'r-')
h1Leg = legend({'Reference','Implicit Solver','Explicit Solver'});
```
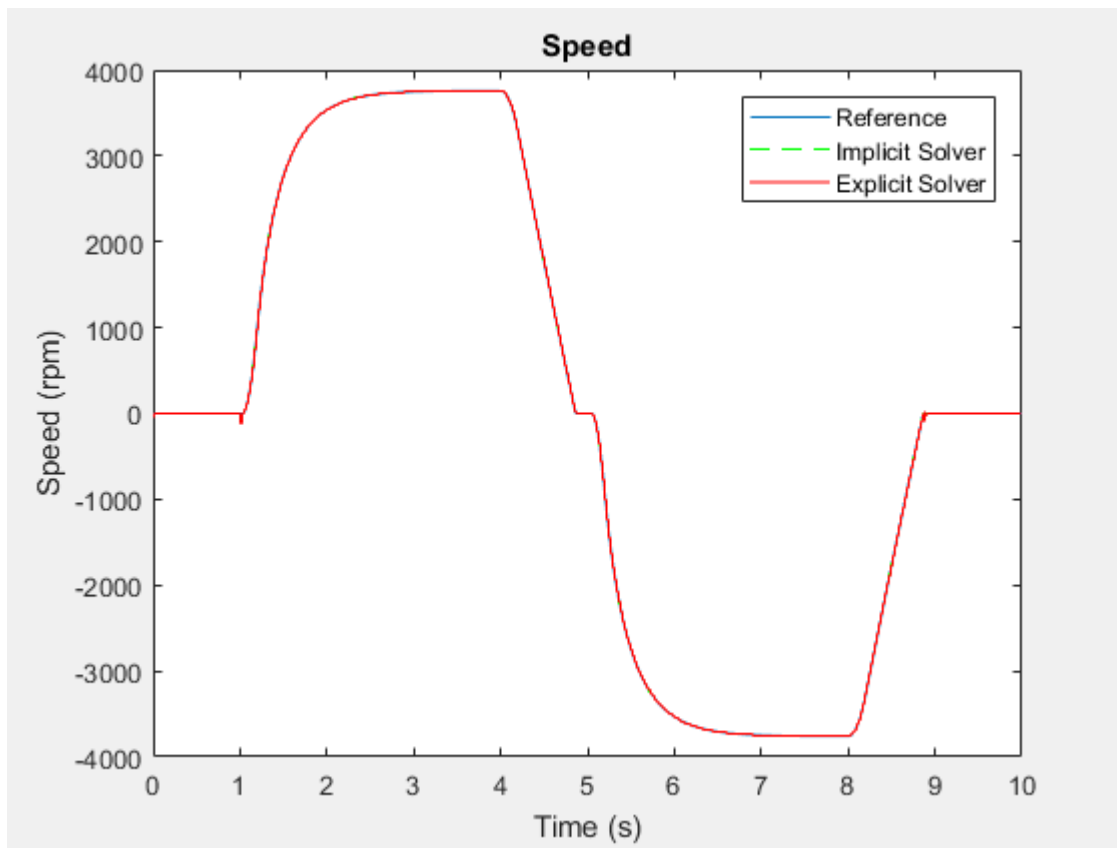
The results differ at the inflection points.

## Analyze the Results

**1** To see the results more closely, zoom to the inflection point just after time $t = \sim 1$ second.
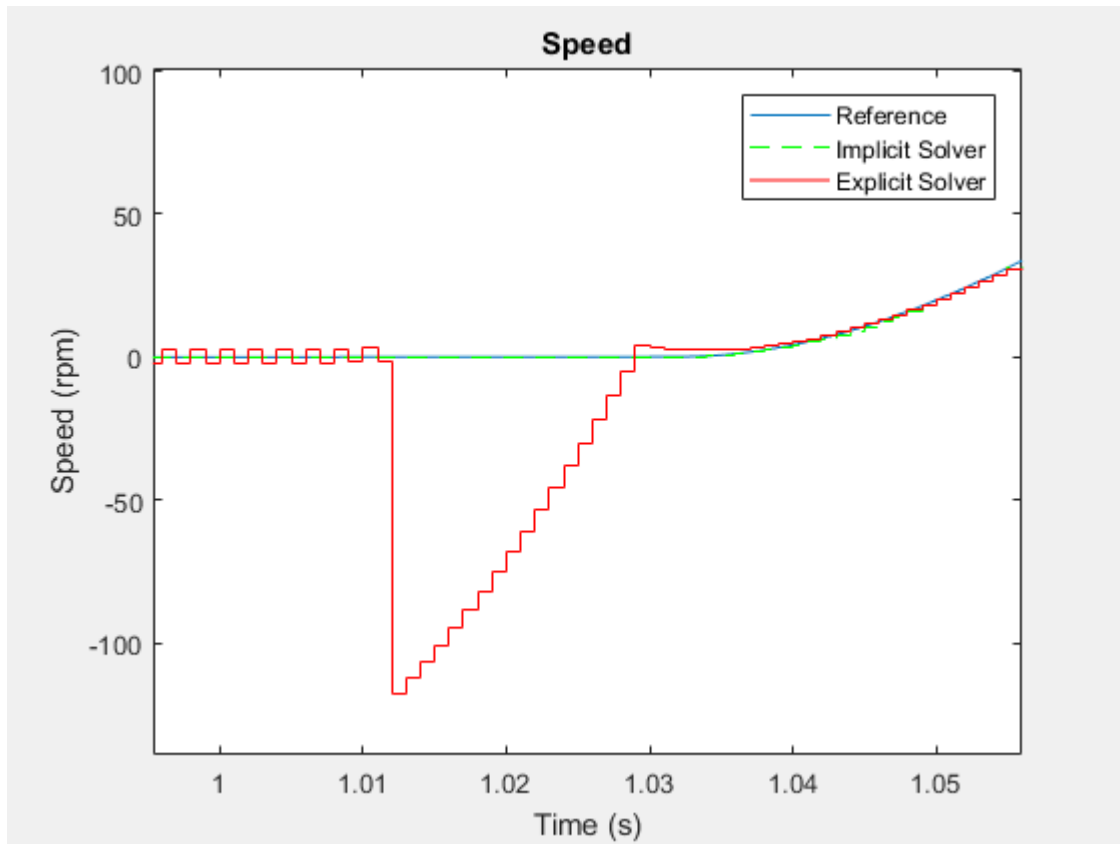
The implicit solver follows a path that is similar to the path that the variable-step solver takes when generating the reference results. The oscillations that the explicit solver exhibits indicate that the model is numerically stiff. The oscillations also indicate that the explicit solver is more computationally costly than the implicit solver for simulating the stiff model. Use a global or local implicit fixed-step solver for real-time simulation with numerically stiff models to avoid unnecessary computational cost.

## See Also

`stairs`

## Related Examples

- "Reduce Numerical Stiffness" on page 11-47
- "Determine Step Size" on page 11-12

## More About

- "Filtering Input Signals and Providing Time Derivatives" on page 7-29
- "Real-Time Model Preparation Workflow" on page 11-4
- "Solvers for Real-Time Simulation" on page 11-76
- "Stiffness" on page 7-2

# Estimate Computation Costs

Estimating computational cost helps you to determine if your model is likely to cause an overrun when you simulate it on your real-time processor. Computational cost is the execution time per time step during simulation. To estimate the time that it takes for your model to execute on real-time hardware, estimate the simulation execution-time budget for your real-time target machine.

To estimate the simulation execution-time, first, measure the execution time of desktop simulation for a particular model. Then determine the average execution time per time step on the real-time target machine for the same model. Knowing how these execution times compare for one model means that you can estimate execution time on the real-time target machine from desktop simulation execution time when you test other models. Having an estimate for the execution-time budget helps you to choose a feasible combination of solver settings for fixed-step, fixed-cost simulation.

During each time step, the real-time target machine must perform the procedures that the figure shows.
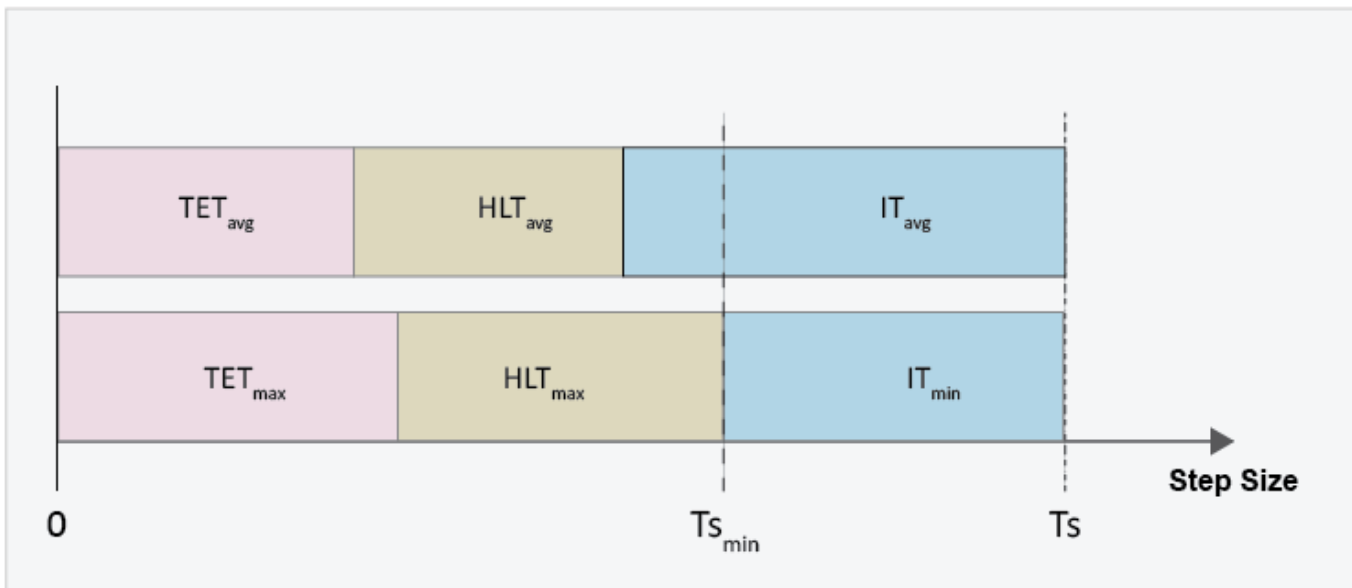


The equation for determining the minimum step size to specify for the fixed-step solver to avoid simulation overrun is

$$Ts_{\min} = TET_{\max} + HLT_{\max},$$

where

- *TET* is the task execution time. Task execution time involves calculating the simulation results for the time step, processing inputs from and writing outputs to the development computer, and performing general computing tasks such as buffering data and accessing memory.
- *HLT* is the hardware latency time. Hardware latency time includes scheduling, interrupt, and input/output (I/O) latency.
- $Ts_{min}$ is the minimum step size.

**11-87**

If the time that it takes for the target machine to execute the simulation and handle latency processes is less than the specified time step, the processor remains idle during the remainder of the step. That is,

$$Ts = TET_{\max} + HLT_{\max} + IT,$$

where

- $Ts$ is the step size that you specify for the fixed-step solver.
- $IT$ is the idle time.

This equation can be rearranged as:

$$TET_{\max} = Ts - HLT_{\max} - IT,$$

The task execution, hardware latency, and idle times vary, but you can implement a safety margin by specifying the idle time in the budget calculation as a function of the step size for the fixed-step solver. For example, if you specify a step size of 1e-5 for the solver, and you want a 20% safety margin, then $IT = (0.2)*(1e-5)$.

Therefore, the amount of time available for simulation execution can be calculated as follows:

$$TET_{\max} = Ts - HLT_{\max} - [(SMT)*(Ts)],$$

where

- $SMT$ is the desired safety margin, specified as a percent.

## See Also

### Related Examples
- "Reduce Computation Costs" on page 11-25

### More About
- "Fixed-Cost Simulation for Real-Time Viability" on page 11-71
- "Simulation Phases in Dynamic Systems"

# Choose Step Size and Number of Iterations

| **In this section...** |
|---|
| |
| |
| |
| |
| |

The step size and number of iterations that you specify for solvers in your model affect the speed and accuracy of your real-time simulation. If you decrease the step size or increase the number of iterations, the results are more accurate, but the simulation runs slower. If you increase the step size or decrease the number of iterations, the simulation runs faster, but the results are less accurate.

To optimize your model for simulation on a real-time target machine, specify a combination of step size ($Ts$) and number of iterations ($N$) that provides acceptable accuracy and the speed to avoid an overrun. As with solver type, you can specify different combinations of $Ts$ and $N$ values for the Simulink global solver and for each independent Simscape network in your model.

This workflow helps you to select the step size and number of iterations for real-time simulation:

- Obtain reference results by performing variable-step simulation on a model of a hydraulic actuator.
- Use a modified version of the model to determine the maximum step size to use to achieve accurate enough results from a fixed-step, fixed-cost simulation. Fixed-step, fixed-cost simulation is required for real-time simulation.
- Specify global and local fixed-step, fixed-cost solver settings for the modified version of the model.
- Perform a timed simulation with the modified model and evaluate the accuracy of the results.
- Adjust the step size and number of iterations to find solver settings that provide the required speed and accuracy for real-time simulation.

## Obtain Reference Results

To obtain reference results, simulate the original version of the hydraulic actuator model.

**1** To open the hydraulic actuator model, at the MATLAB command prompt, enter:

```
model = 'ssc_hydraulic_actuator_digital_control';
open_system(model)
```
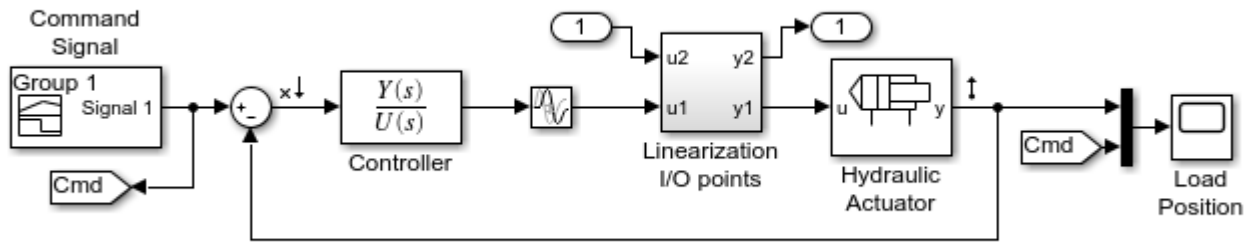
2  The model is configured to limit data points. To configure the model to log all data points, open the model configuration parameters, and in the **Simscape** pane, clear the **Limit data points** check box.

3  Simulate the model.

```
sim(model)
```

4  Extract the data for pressure and simulation-step time from the logged Simscape node.

```
simlogRef = simlog_ssc_hydraulic_actuator_digital_control;
pRefNode = simlogRef.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_A.A.p;
pRef = pRefNode.series.values('Pa');
tRef = pRefNode.series.time;
```

5  Plot the step size.

```
h1 = figure;
semilogy(tRef(1:end-1),diff(tRef),'-x')
title('Solver Step Size')
xlabel('Time (s)')
ylabel('Step Size (s)')
```
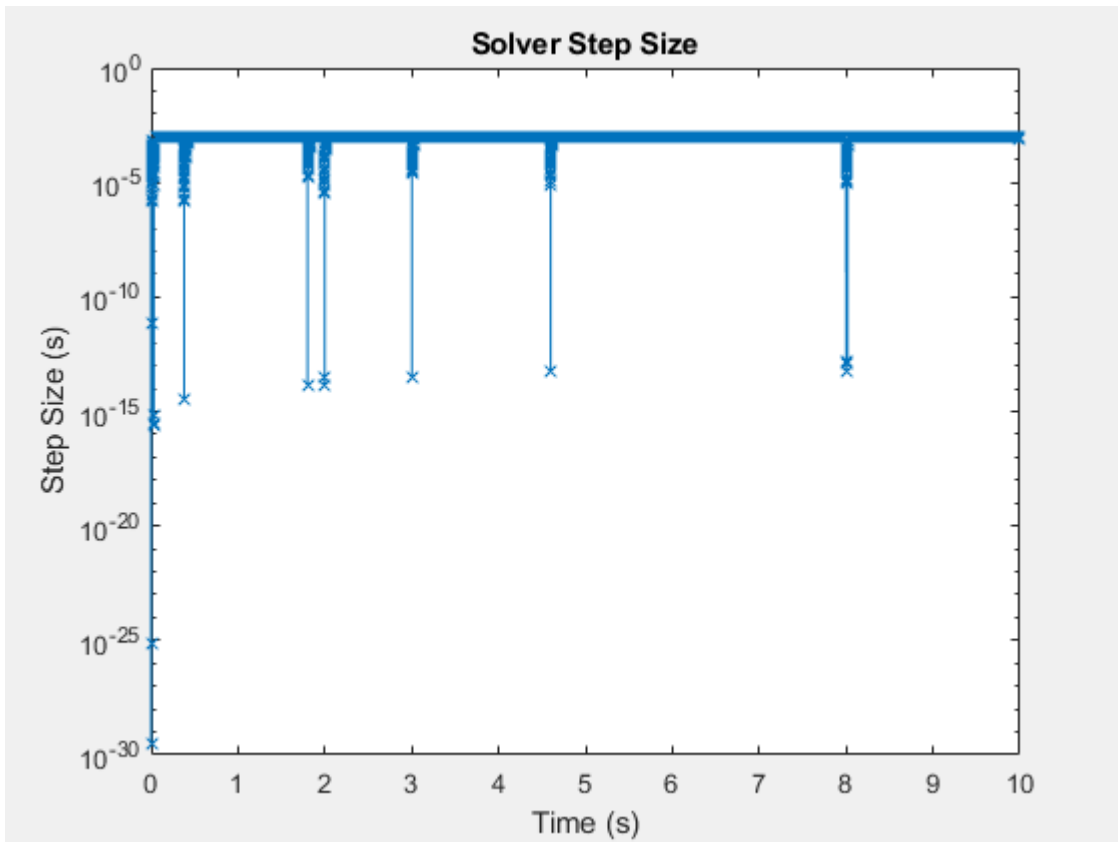
The maximum step size ($Ts_{max}$) for obtaining accurate real-time results for the original model is approximately 1e-2 seconds. For information on determining $Ts_{max}$, see "Determine Step Size" on page 11-12.

**6**  Plot the simulation results.

```
h2 = figure;
plot(tRef,pRef, 'b-')
h2Legend1 = legend({'Reference'},'Location','southoutside');
title('Cylinder Pressure')
xlabel('Time (s)')
ylabel('Pressure (Pa)')
```
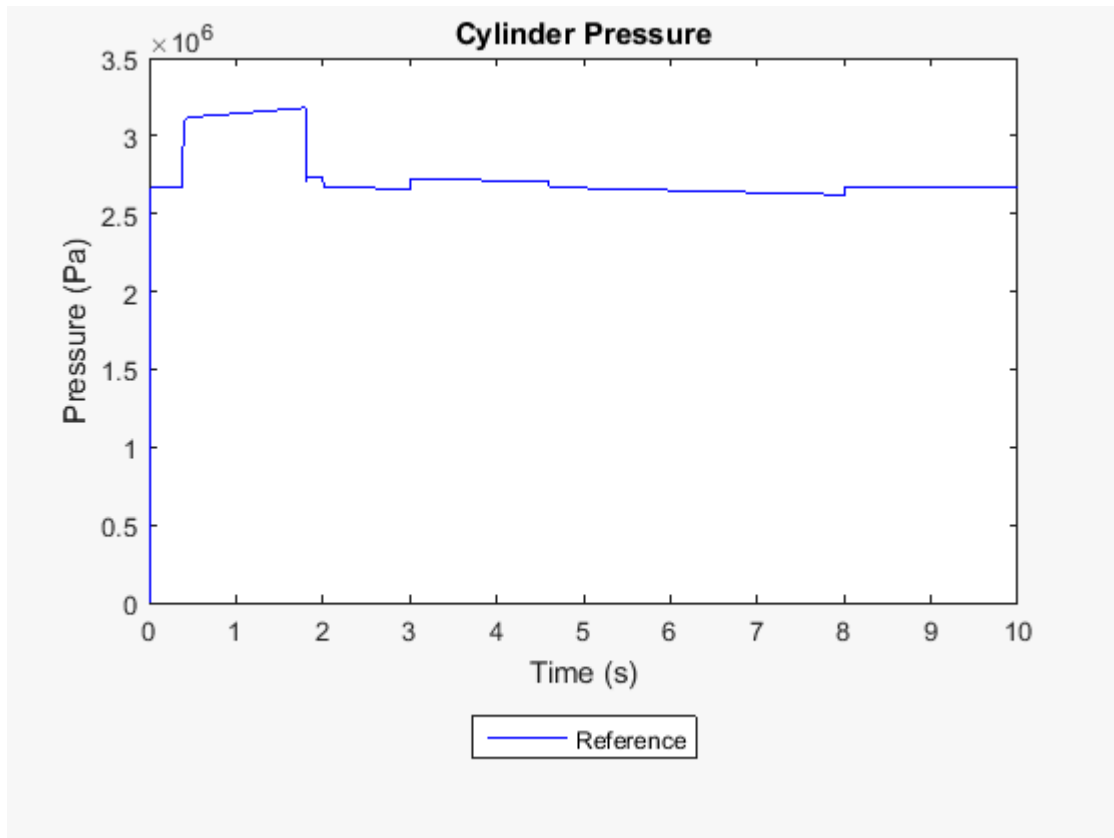
## Determine Maximum Step Size for Accurate Results

In a modified version of the hydraulic actuator model, you can change the value of $Ts_{max}$, the maximum step size for achieving accurate real-time simulation results.

1  Open the modified hydraulic actuator model.

    ssc_hydraulic_actuator_HIL
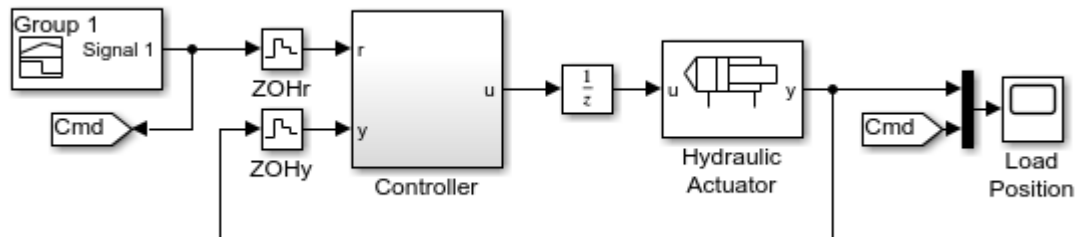


This version of the hydraulic actuator contains a discretized, partitioned controller. The local solver for the hydraulic actuator subsystem is enabled for fixed-step, fixed-cost simulation. The step size is parameterized (*ts*) so that you can make solver adjustments that decrease the likelihood of generating an overrun. For an example that shows how to discretize the controller for the hydraulic actuator, see "Hydraulic Actuator Configured for HIL Testing" on page 20-116.

**2** To determine the maximum step size to use for achieving accurate real-time simulation results, you simulate with a global, variable-step solver. To configure the modified model for variable-step simulation using the global solver, disable the local solver configuration. In the Hydraulic Actuator subsystem, in the Solver Configuration block property inspector, clear the **Use local solver** check box.

**3** Simulate the model.

**4** Extract the data for pressure and time from the logged Simscape node.

```
simlog0 = simlog_ssc_hydraulic_actuator_HIL;
pNodeSim0 = simlog0.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_A.A.p;
pSim0 = pNodeSim0.series.values('Pa');
tSim0 = pNodeSim0.series.time;
```

**5** Plot the step size to the figure that contains the step-size data for the original model.

```
figure(h1)
hold on
semilogy(tSim0(1:end-1),diff(tSim0),'--x', 'Color','r',...
    'LineWidth',.1,'MarkerSize',5)
title('Solver Step Size')
xlabel('Time (s)')
ylabel('Step Size (s)')
h1Legend1 = legend({'Reference','Modified'},...
    'Location','southoutside');
```
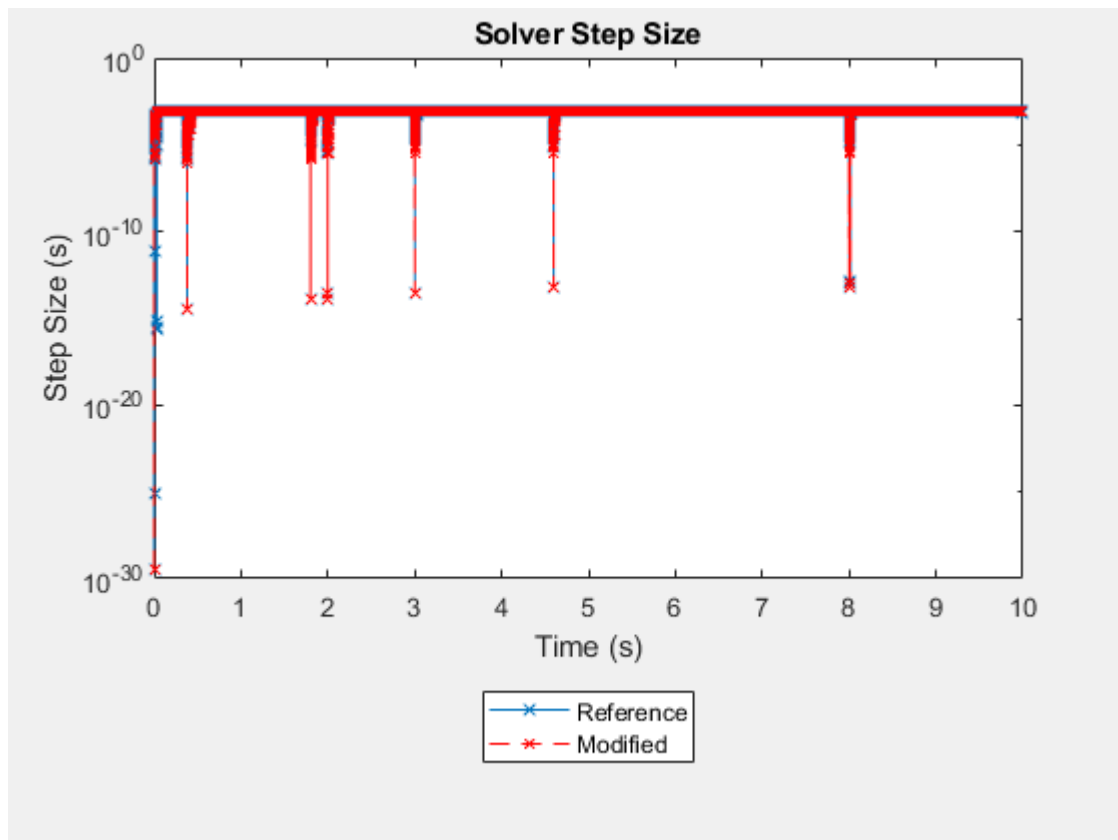


For the discretized model, $Ts_{max}$ is between 1e-2 and 1e-3 seconds.

## Parameterize Global and Local Solver Settings

To reduce the number of steps for finding the optimal real-time-simulation solver settings, parameterize the solver configuration with workspace variables. In the Hydraulic Actuator Discrete Model, the step size for the local solver configuration is specified as the workspace variable *ts*. For this example, you also use workspace variables to parameterize the global step size (*tsG*) and the local number of nonlinear iterations *(N)*.

1   For the modified model, in the model configuration parameters property inspector, specify these settings:

| Pane | Parameter | Value | Purpose |
|------|-----------|-------|---------|
| **Solver** | **Type** | `Fixed-step` | Configure the global solver of the modified model for fixed-step simulation. |
| | **Solver** | `discrete (no continuous states)` | Configure the global solver to match the state of the controller. |
| | `Additional options` > **Fixed-step size (fundamental sample time)** | `tsG` | Parameterize the global step size. |
| **Simscape** | **Limit data points** | Clear the check box. | As you decrease the solver step size, the number of data points that the simulation generates increases. Clear the option to ensure that you collect all the data that you need for evaluating simulation accuracy. |

2   Configure the local solver for fixed-step simulation. In the Hydraulic Actuator subsystem, in the Solver Configuration block property inspector, select **Use local solver**.

3   To parameterize the cost of the simulation, set **Nonlinear iterations** to `N`.

## Perform Fixed-Step, Fixed-Cost Simulation

You can determine if your solver settings are appropriate for real-time simulation by simulating the model and then evaluating the accuracy of the results and the speed of the simulation. To evaluate accuracy, compare the results to the reference results and to the results of other fixed-step, fixed-cost simulations. To evaluate simulation speed, compare the elapsed time to the specified simulation time and to the simulation execution budget. If the speed or accuracy is not acceptable, adjust the step size and number of iterations to make your model real-time capable.

The simulation execution-time budget for this example is four seconds. For information on determining the execution-time budget for your model, see "Estimate Computation Costs" on page 11-87.

1   For the first simulation, specify both the global and local step size as the largest possible value of $Ts_{max}$ from the step plot. Specify a relatively large value for the step size for both solvers and three for the number of nonlinear iterations for the local solver.

```
ts = 1e-2;
tsG = 1e-2;
N = 3;
```

2   Perform a timed fixed-step, fixed-cost simulation.

```
tic; sim('ssc_hydraulic_actuator_HIL'); tSim1 = toc;
time1 = max(tSim1);
```

3   Extract the data for pressure and simulation time from the logged Simscape node.

```
simlog1 = simlog_ssc_hydraulic_actuator_HIL;
pNodeSim1 = simlog1.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_A.A.p;
pSim1 = pNodeSim1.series.values('Pa');
tSim1 = pNodeSim1.series.time;
```

4   Plot the simulation results to the figure that contains the reference results. Write the elapsed time to the figure legend.

```
figure(h2)
hold on
plot(tSim1, pSim1, 'g--')
delete(h2Legend1)
configSim1L = ['Local: Ts= ',num2str(ts),'s, N= ',num2str(N),'.'];
configSim1G = [' Global: Ts= ',num2str(tsG),'s.'];
timeSim1T = ['Time=',num2str(time1)];
cfgSim1 = [configSim1L,configSim1G,timeSim1T];
h2Legend2 = legend({'Reference',num2str(cfgSim1)},...
    'Location','southoutside');
```
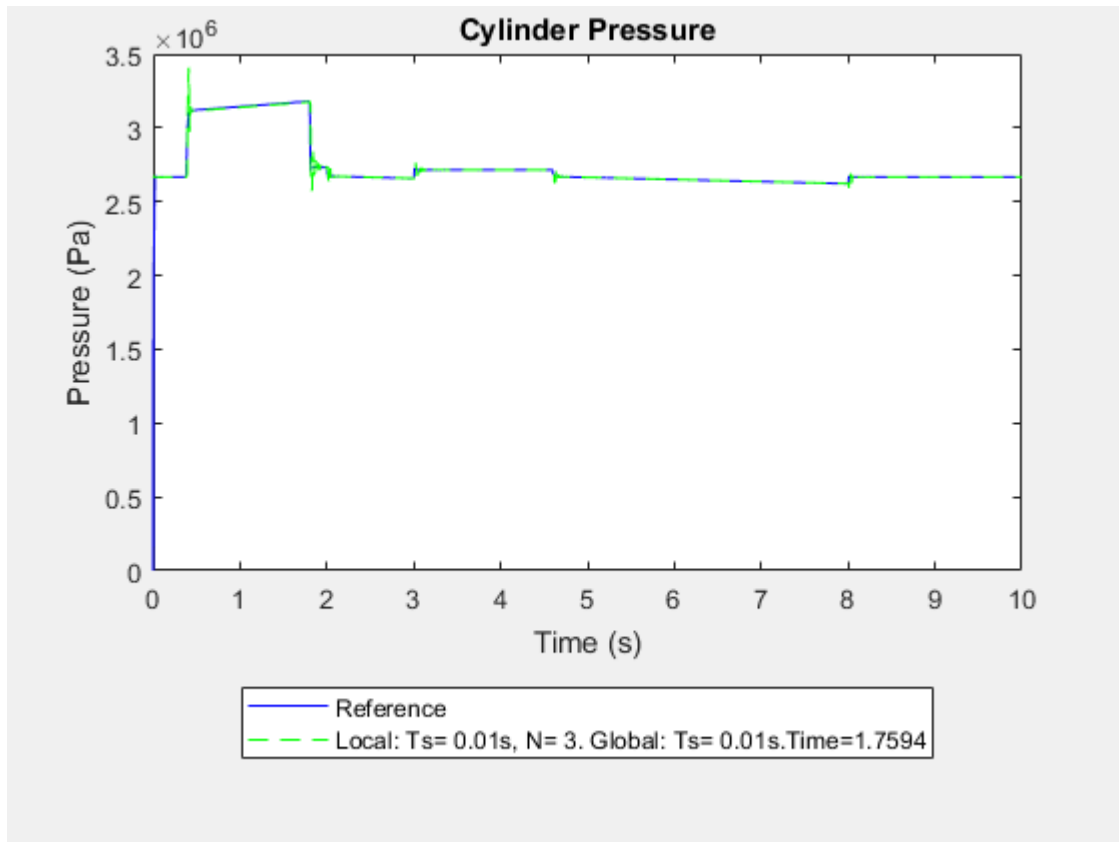
The elapsed time varies because it depends on the immediate computational capacity of the computer that runs the simulation. The elapsed times in the legend are from simulation on a 3.6-GHz Intel® CPU with a 16-GB memory. Your legend contains the elapsed time for the simulation on your computer.

The simulation took less time to complete than the specified simulation time (10 s) so it runs faster than real time on the development computer. The elapsed time is also less than the simulation execution-time budget for this example (four seconds). Therefore, the specified solver configuration provides an acceptable safety margin for real-time simulation on the target machine that provided the budget data.

**5**  Zoom to an inflection point to evaluate the accuracy of the results.

```
figure(h2)
xStart = 0;
xEnd = 10;
yStart = 0;
yEnd = 3.5e6;
xZoomStart = 0.3;
xZoomEnd = 0.6;
yZoomStart = 2.6e6;
yZoomEnd = 3.5e6;
axis([xZoomStart xZoomEnd yZoomStart yZoomEnd])
```
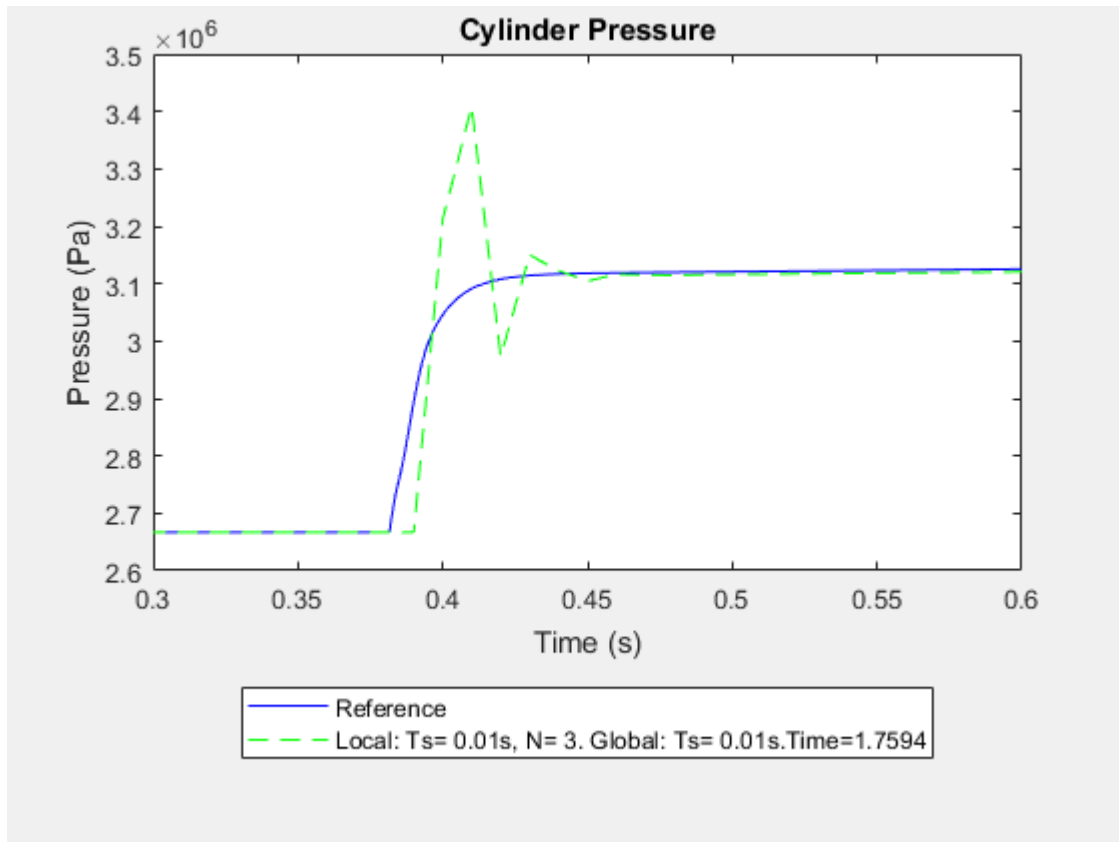
Theoretical and empirical data support the reference results. The accuracy of the simulation results is not acceptable because the solver oscillates before it converges on the solution in the reference data.

If you can achieve acceptable result accuracy, but the simulation runs too slowly for a given execution-time budget, increase speed by increasing the step size or decrease the number of iterations.

When you find a combination of solver settings that provide accurate enough results and a simulation speed that fits your execution-time budget, you can attempt to run your model on a real-time target machine by performing the hardware-in-the-loop simulation workflow. If you cannot find the right combination of solver settings, perform the real-time model preparation workflow or increase your real-time computing capability to improve simulation speed and accuracy. To increase your real-time computing capability, upgrade your target hardware or partition your model for parallel processing.

## Adjust Solver Settings to Improve Accuracy

You can generally improve accuracy by increasing the number of iterations or by decreasing the step size.

1  Try to improve accuracy by increasing the number of iterations (*N*) to 10.

```
N = 10;
```

2  Run a timed simulation.

```
tic; sim('ssc_hydraulic_actuator_HIL'); tSim2 = toc;
time2 = max(tSim2);
```

**3** Extract the pressure and simulation time data.

```
simlog2 = simlog_ssc_hydraulic_actuator_HIL;
pNodeSim2 = simlog2.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_A.A.p;
pSim2 = pNodeSim2.series.values('Pa');
tSim2 = pNodeSim2.series.time;
```

**4** Plot the results.

```
figure(h2)
hold on
plot(tSim2, pSim2, 'r:')
delete(h2Legend2)
axis([xStart xEnd yStart yEnd])
configSim2L = ['Local: Ts= ',num2str(ts),'s, N= ',num2str(N),'.'];
configSim2G = [' Global: Ts= ',num2str(tsG),'s.'];
timeSim2T = ['Time=',num2str(time2)];
cfgSim2 = [configSim2L,configSim2G,timeSim2T];
h2Legend3 = legend({'Reference',num2str(cfgSim1),num2str(cfgSim2)},...
    'Location','southoutside');
```
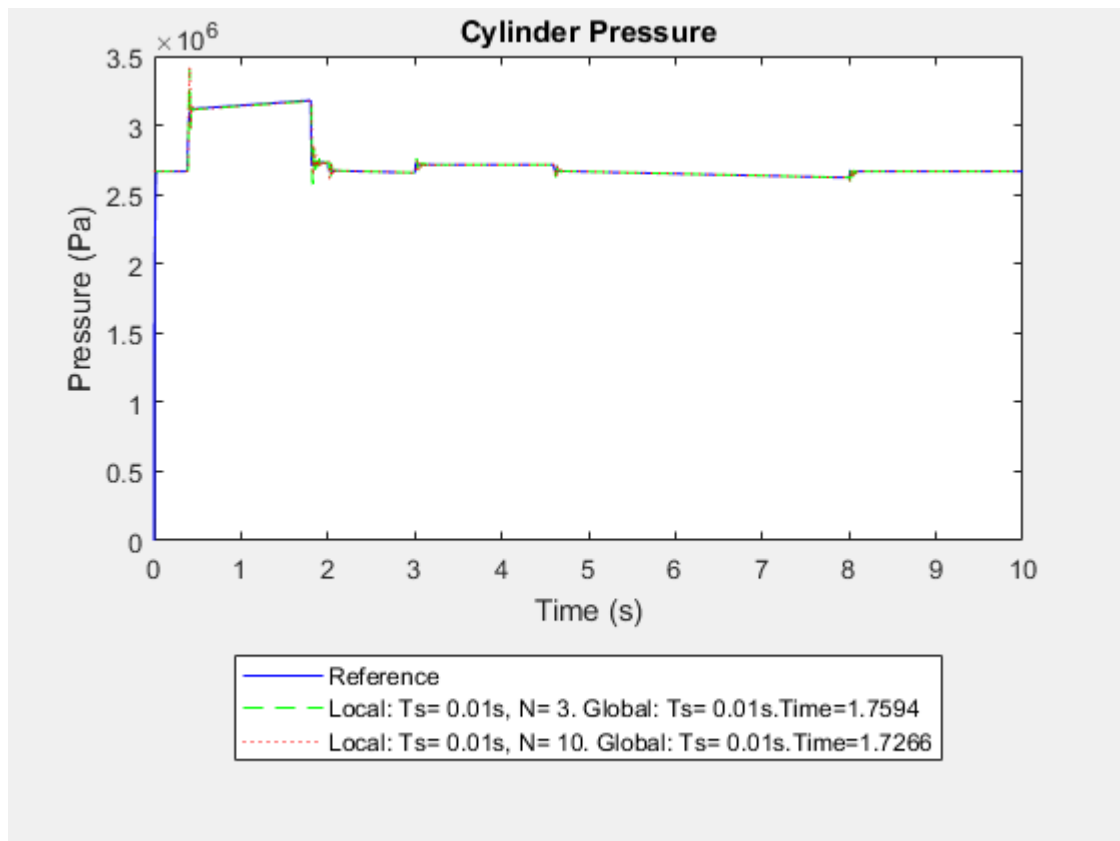


The simulation is fast enough for real-time simulation because it took less time to run than the four-second simulation execution budget.

**5** Zoom to evaluate accuracy.

```
figure(h2)
axis([xZoomStart xZoomEnd yZoomStart yZoomEnd])
```
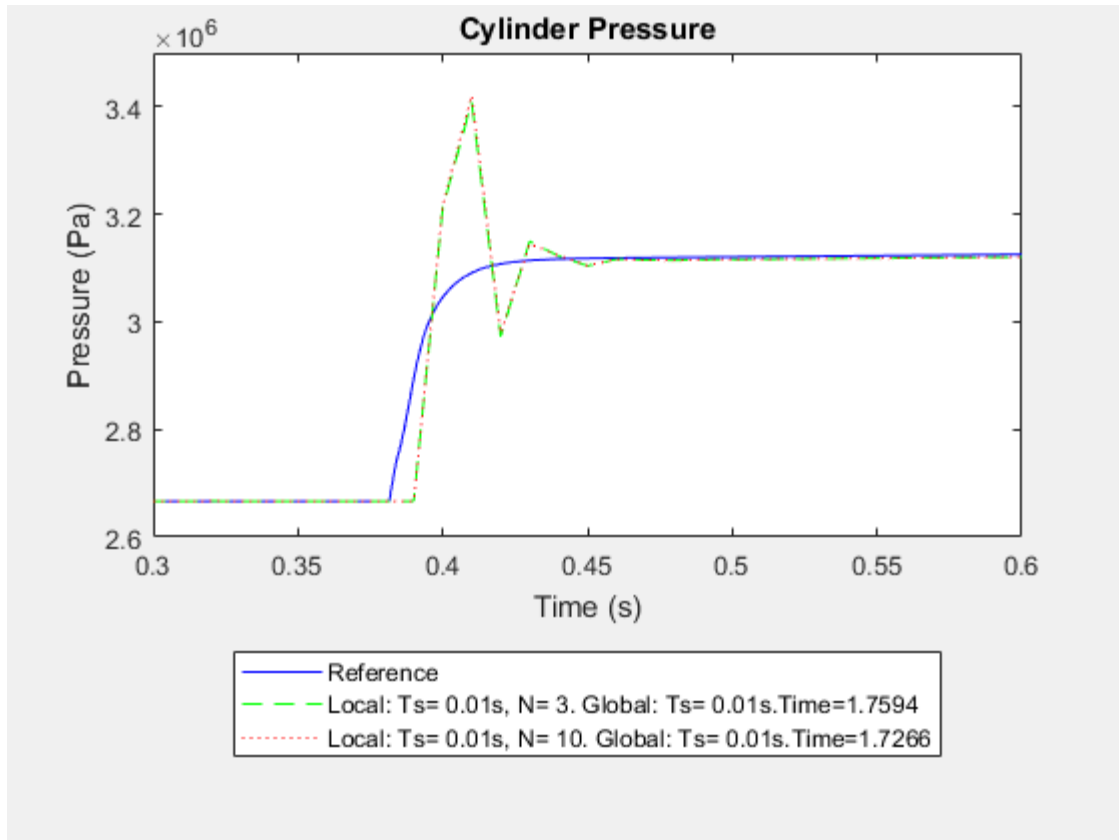


Overall, the results are not much more accurate than the results from the simulation with fewer iterations.

**6** Try to improve accuracy by decreasing the step size to 1e-3 seconds for the local and global solvers. Specify 3 for the number of iterations (*N*).

```
ts = 1e-3;
tsG = 1e-3;
N = 3;
```

**7** Run a timed simulation.

```
tic; sim('ssc_hydraulic_actuator_HIL'); tSim3 = toc;
time3 = max(tSim3);
```

**8** Extract the pressure and simulation time data.

```
simlog3 = simlog_ssc_hydraulic_actuator_HIL;
pNodeSim3 = simlog3.Hydraulic_Actuator.Hydraulic_Cylinder.Chamber_A.A.p;
pSim3 = pNodeSim3.series.values('Pa');
tSim3 = pNodeSim3.series.time;
```

**9** Plot the results.

```
figure(h2)
hold on
plot(tSim3, pSim3, 'k--')
delete(h2Legend3)
```

```
axis([xStart xEnd yStart yEnd])
configSim3L = ['Local: Ts= ',num2str(ts),'s, N= ',num2str(N),'.'];
configSim3G = [' Global: Ts= ',num2str(tsG),'s.'];
timeSim3T = ['Time=',num2str(time3)];
cfgSim3 = [configSim3L,configSim3G,timeSim3T];
h2Legend4 = legend...
    ({'Reference',num2str(cfgSim1),num2str(cfgSim2),num2str(cfgSim3)},...
    'Location','southoutside');
```
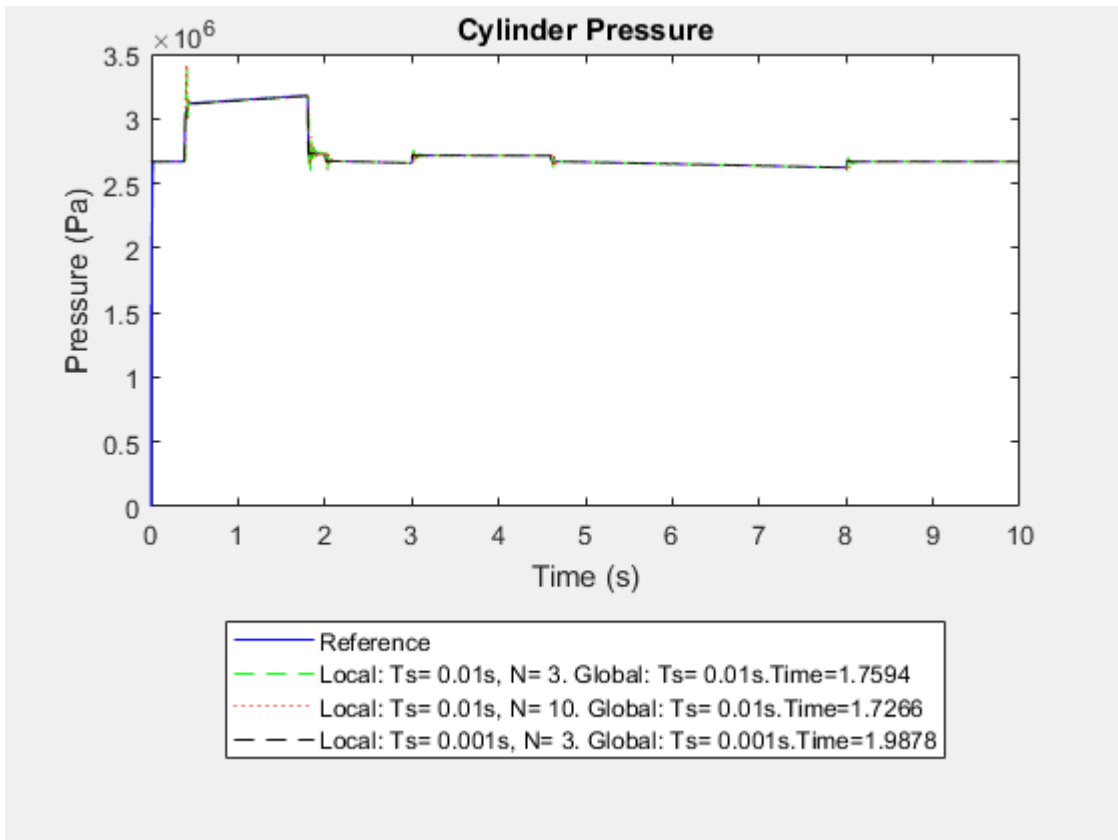


The simulation takes longer but is fast enough given the four-second simulation execution-time budget.

**10** Zoom to evaluate accuracy better.

```
figure(h2)
axis([xZoomStart xZoomEnd yZoomStart yZoomEnd])
```
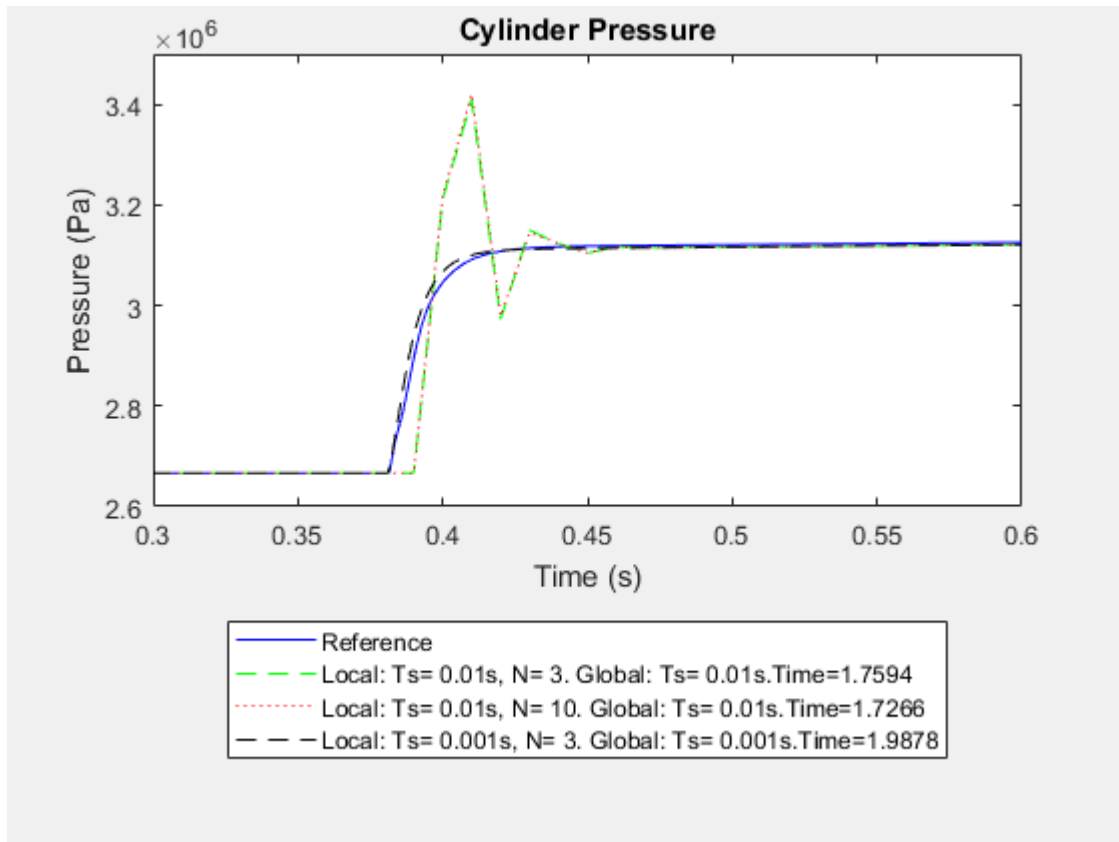
The accuracy of the results is acceptable. For real-time simulation with the modified model, use the solver settings that provided acceptable speed and accuracy:

- Three nonlinear iterations
- Global and local step sizes of 1e-3 seconds

If you can achieve accurate enough results, but the simulation runs too slowly for your execution-time budget, improve speed by increasing the step size or decreasing the number of iterations.

When you find a combination of solver settings that provides accurate enough results and a simulation speed that is less than your execution-time budget, you can run your model on a real-time target machine. To run your model on a real-time target machine, perform the hardware-in-the-loop simulation workflow.

If you cannot find the right combination of solver settings for real-time simulation, improve simulation speed and accuracy by modifying the scope or fidelity of your model. For more information, see "Real-Time Model Preparation Workflow" on page 11-4.

If you cannot make your model real-time capable by changing the scope or fidelity of your model, increase your real-time computing capability. For more information, see "Upgrading Target Hardware" on page 11-10 and "Simulating Parts of the System in Parallel" on page 11-10.

## See Also

Solver Profiler | `simscape.getLocalSolverFixedCostInfo`

## Related Examples

- "Determine Step Size" on page 11-12
- "Determine System Stiffness" on page 11-82
- "Estimate Computation Costs" on page 11-87

## More About

- "Filtering Input Signals and Providing Time Derivatives" on page 7-29
- "Fixed-Cost Simulation for Real-Time Viability" on page 11-71
- "Hardware-In-The-Loop Simulation Workflow" on page 11-106
- "Improving Speed and Accuracy" on page 11-8
- "Log and Plot Simulation Data" on page 13-8
- "Real-Time Model Preparation Workflow" on page 11-4
- "Solvers for Real-Time Simulation" on page 11-76
- "Basics of Hardware-In-The-Loop simulation" on page 11-103

# Basics of Hardware-In-The-Loop simulation

Hardware-in-the-loop (HIL) simulation is a type of real-time simulation. You use HIL simulation to test your controller design. HIL simulation shows how your controller responds in real time to realistic virtual stimuli. You can also use HIL to determine if your physical system (plant) model is valid.

In HIL simulation, you use a real-time computer as a virtual representation of your plant model and a real version of your controller. The figure shows a typical HIL simulation setup. The desktop computer (development hardware) contains the real-time capable model of the controller and plant. The development hardware also contains an interface with which to control the virtual input to the plant. The controller hardware contains the controller software that is generated from the controller model. The real-time processor (target hardware) contains code for the physical system that is generated from the plant model.
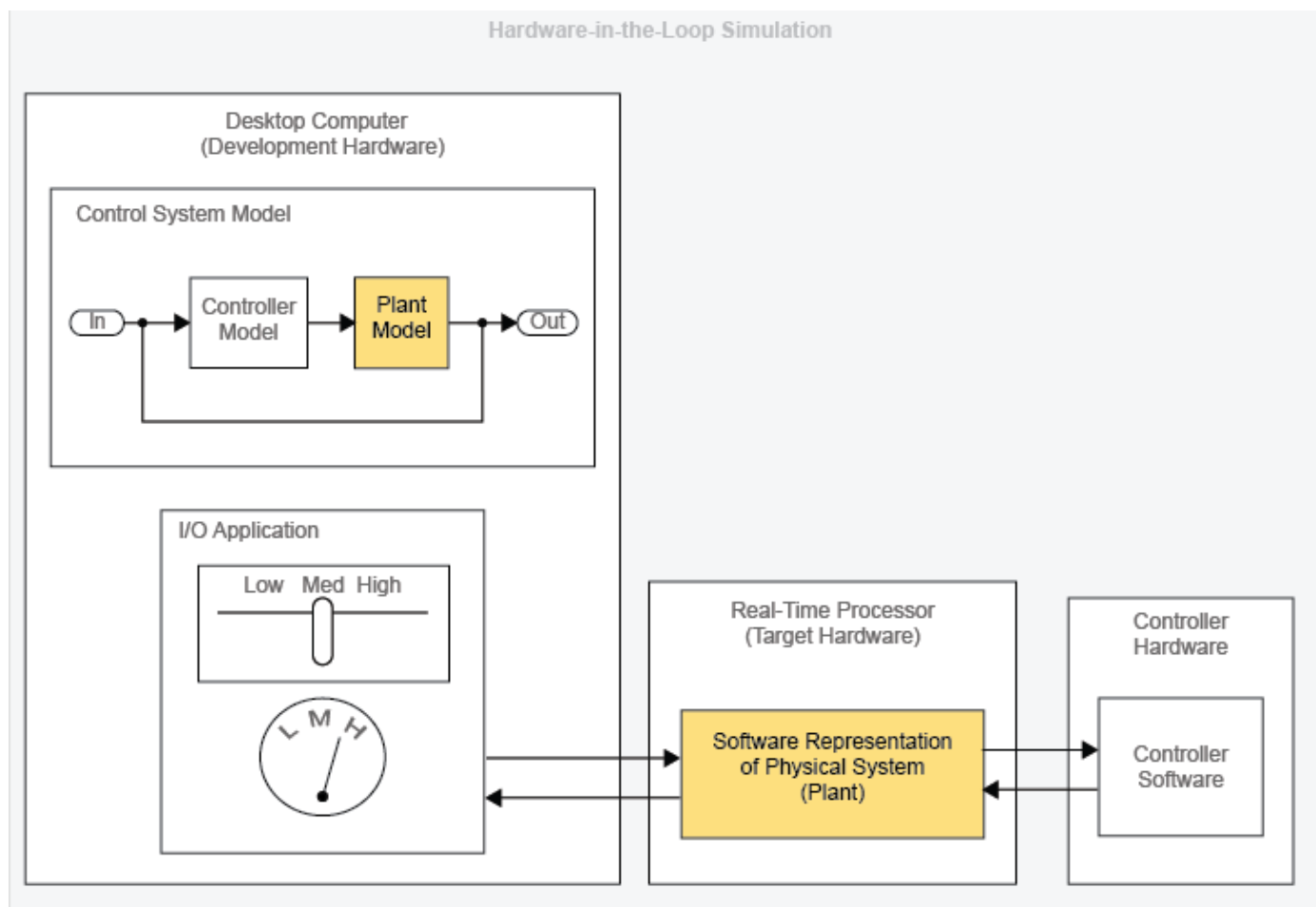


## When to use Hardware-In-The-Loop Simulation

Use HIL simulation to test the design of your controller when you are performing Model-Based Design (MBD). The figure shows where HIL simulation fits into the MBD design-to-realization workflow.
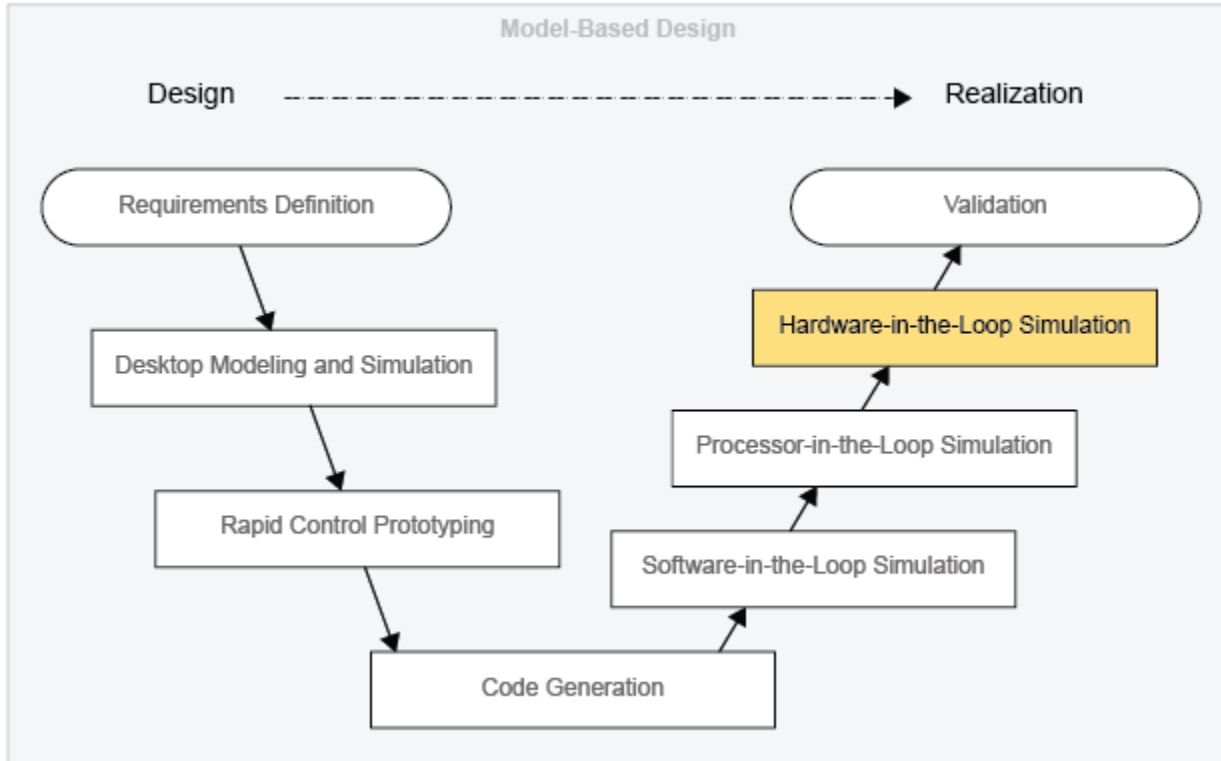
Validation involves using actual plant hardware to test your controller in real-life situations or in environmental proxies (for example, a pressure chamber). In HIL simulation, you do not have to use real hardware for your physical system (plant). You also do not have to rely on a naturalistic or environmental test setup. By allowing you to use your model to represent the plant, HIL simulation offers benefits in cost and practicality.

There are several areas in which HIL simulation offers cost savings over validation testing. HIL simulation tends to be less expensive for design changes. You can perform HIL simulation earlier than validation in the MBD workflow so you can identify and redesign for problems relatively early the project. Finding problems early includes these benefits:

- Your team is more likely to approve changes.
- Design changes are less costly to implement.

In terms of scheduling, HIL simulation is less expensive and more practical than validation because you can set it up to run on its own.

HIL simulation is more practical than validation for testing your controller's response to unusual events. For example, you can model extreme weather conditions like earthquakes or blizzards. You can also test how your controller responds to stimuli that occur in inaccessible environments like deep sea or deep space.

## See Also

## Related Examples

- "Generate, Download, and Execute Code" on page 11-117

## More About

- "Hardware-In-The-Loop Simulation Workflow" on page 11-106

# Hardware-In-The-Loop Simulation Workflow

| In this section... |
| --- |
| "Perform Hardware-In-The-Loop Simulation" on page 11-109 |
| "Insufficient Computational Capability for Hardware-In-The-Loop Simulation" on page 11-110 |

This figure shows the hardware-in-the loop simulation workflow. The connectors are exit points for returning to the real-time model preparation workflow.
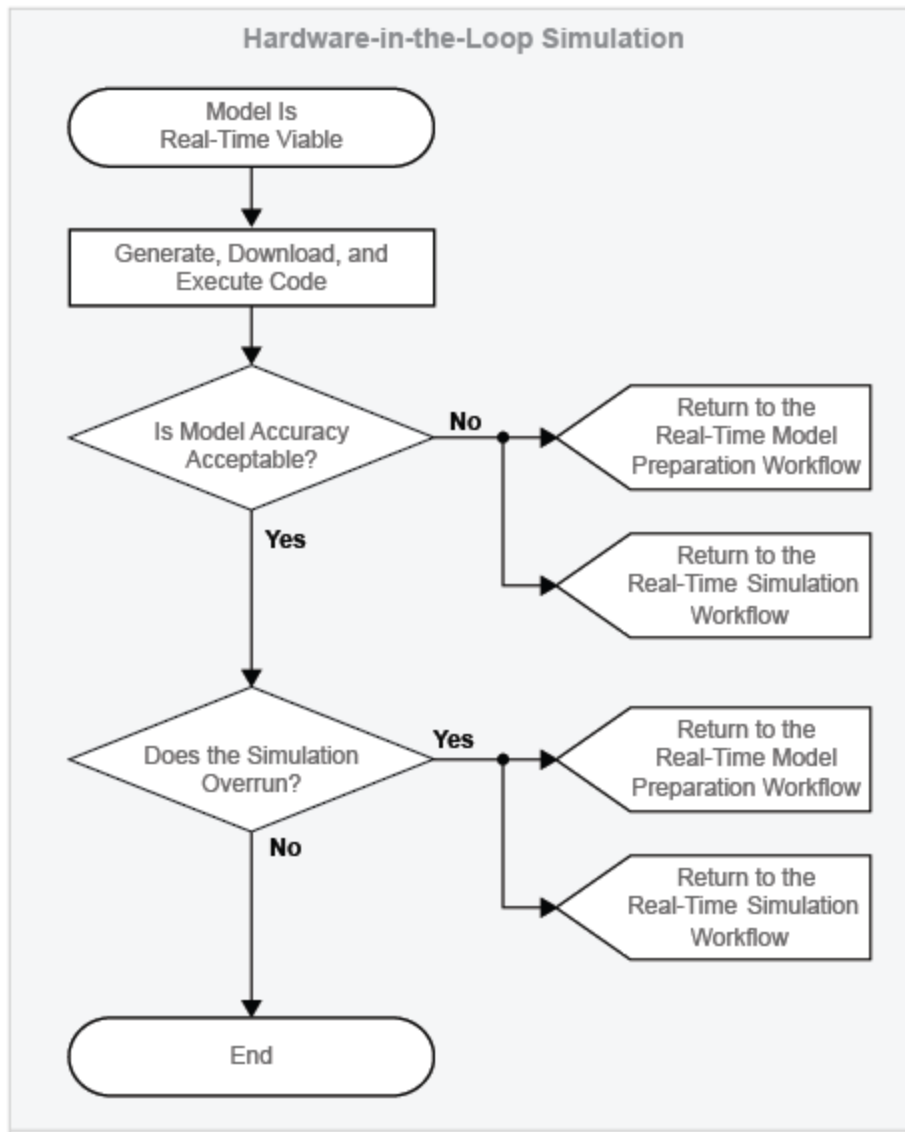


This figure shows the real-time model preparation workflow. The connector is an entry point for returning to the real-time model preparation workflow from other real-time workflows such as the hardware-in-the-loop simulation workflow.
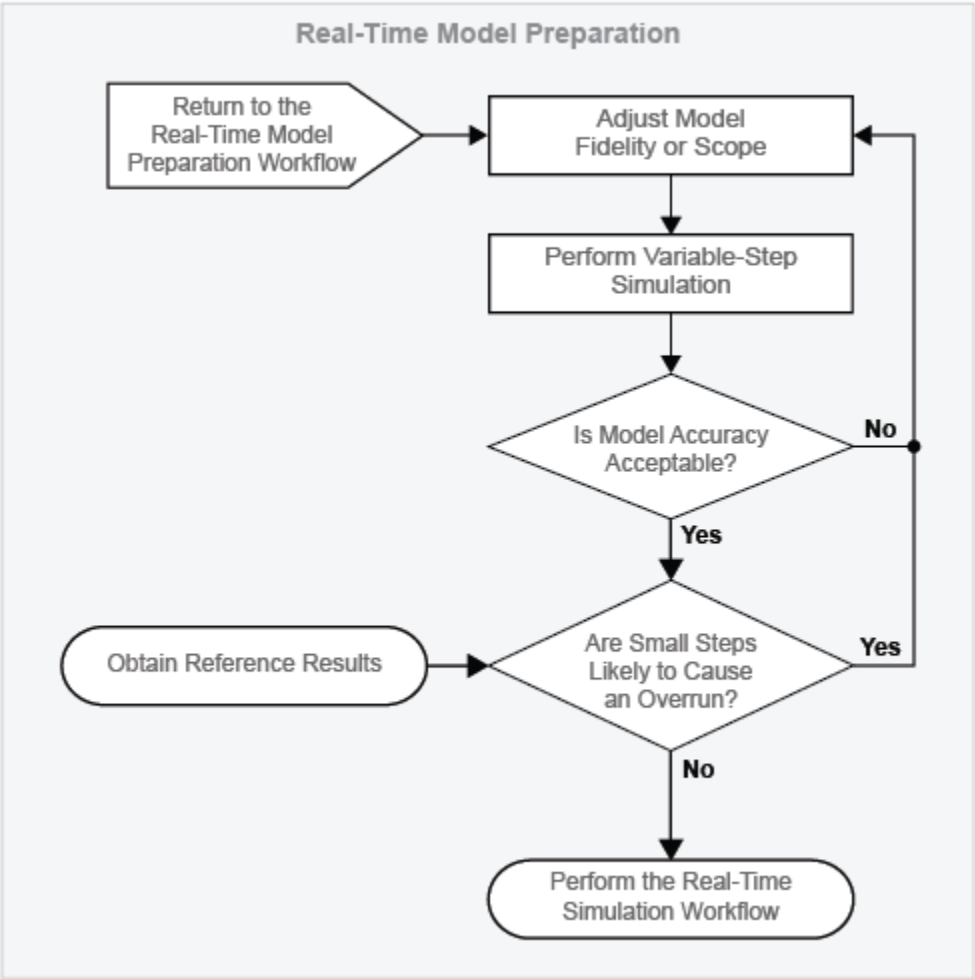
This figure shows the real-time simulation workflow. The connectors are exit points for returning to the real-time model preparation workflow.
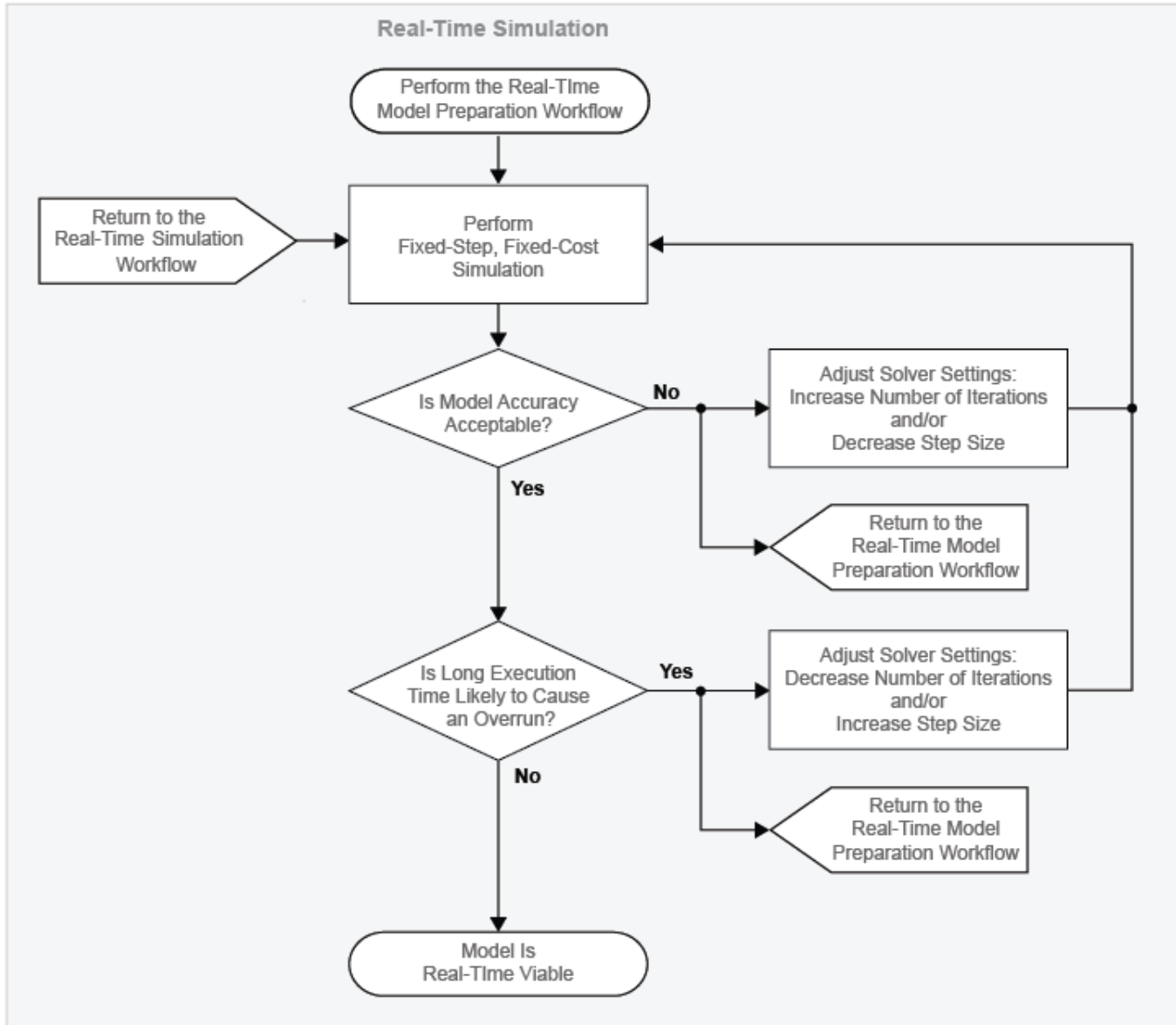
Before performing the hardware-in-the-loop (HIL) simulation workflow:

1 Prepare and configure your model for real-time simulation. For information, see "Real-Time Model Preparation Workflow" on page 11-4 and "Real-Time Simulation Workflow" on page 11-72.

2 Set up and configure the software, I/O interfaces, and connectivity for your development computer, target computer, and I/O board. For information, see "Get Started with Simulink Real-Time" (Simulink Real-Time).

3 If you are performing HIL simulation to test your controller:

- Configure your controller.
- Connect your controller to the real-time computer.

# Perform Hardware-In-The-Loop Simulation

### Generate, Download, and Execute Code

Use Simulink Real-Time to:

- Generate and compile code on the development computer.
- Download the real-time application to the target computer.
- Execute the real-time application remotely from the development computer.

For information, see "Generate, Download, and Execute Code" on page 11-117.

### Evaluate Accuracy

Compare the results from the simulation on the target computer to your reference results. Are the reference and modified model results the same? If not, are they similar enough that the empirical or theoretical data also supports the results from the simulation of the modified model? Is the modified model representing the phenomena that you want it to measure? Is it representing those phenomena correctly? If you plan on using your model to test your controller design, is the model accurate enough to produce results that you can rely on for system qualification? The answers to these questions help you to decide if your real-time results are accurate enough.

### Evaluate Speed

To find out if your simulation generates an overrun, examine the task execution time (TET) report that Simulink Real-Time generates for the simulation.

### Return to the Real-Time Model Preparation Workflow

Your model is not real-time capable if simulation on your real-time target machine generates an overrun or produces results that do not match your reference results closely enough. To make your model real-time capable by adjusting model fidelity, return to the real-time model preparation or real-time simulation workflow.

Adjust the fidelity or scope of your model, and then step through the other processes and decisions in the real-time model preparation workflow. Iterate on adjusting, simulating, and analyzing your model until it is fast and accurate enough for you to perform the real-time simulation workflow. Perform the real-time simulation workflow, and then attempt the hardware-in-the-loop simulation workflow again. For information, see "Real-Time Model Preparation Workflow" on page 11-4 and "Real-Time Simulation Workflow" on page 11-72.

### Return to the Real-Time Simulation Workflow

Your model is not real-time capable if simulation on your real-time target machine generates an overrun or produces results that do not match your reference results closely enough. To make your model real-time capable by adjusting the simulation solver settings, return to the real-time simulation workflow.

Perform the real-time simulation workflow, and then attempt the hardware-in-the-loop simulation workflow again. For information, see "Real-Time Simulation Workflow" on page 11-72.

## Insufficient Computational Capability for Hardware-In-The-Loop Simulation

Your real-time target machine can lack the computational capability for running your model in real time. If your model fails to run in real time or produces unreliable results on your target machine after multiple iterations of the real-time workflows, consider these options for increasing processing power:

- "Upgrading Target Hardware" on page 11-10
- "Simulating Parts of the System in Parallel" on page 11-10

## See Also

## Related Examples
- "Generate, Download, and Execute Code" on page 11-117

## More About
- "Real-Time Model Preparation Workflow" on page 11-4
- "Basics of Hardware-In-The-Loop simulation" on page 11-103
- "Real-Time Simulation Workflow" on page 11-72
- "Code Generation Requirements" on page 11-111

# Code Generation Requirements

| In this section... |
|---|
| "Hardware Requirements" on page 11-111 |
| "Software Requirements" on page 11-111 |

Code generation for hardware-in-the-loop (HIL) simulation with Simulink Real-Time requires specific hardware and software.

## Hardware Requirements

The minimum hardware requirements for HIL simulation with Simulink Real-Time are:

- Development computer with a network or serial interface. For information on development computer specifications, see Development Computer Requirements (Simulink Real-Time).
- Real-time target computer system, containing this hardware:

  - CPU.
  - I/O board. For information on supported I/O boards, see Supported Hardware: Hardware Drivers.
  - Protocol interface

- Controller, preconfigured with code from your controller model
- Connection cable for linking the development computer to the real-time target machine.
- Peripherals that provide a way to:

  - Boot the real-time target computer
  - Transfer the Simulink Real-Time operating system and executable code to the real-time target computer

- Wiring harness to connect the real-time target computer to the controller.

## Software Requirements

For information on the minimum software requirements for HIL simulation with Simulink Real-Time, see Simulink Real-Time Software Requirements. The Simulink Real-Time requirements include a C compiler.

---

**Note** If you want more configuration options for code optimization, use Embedded Coder® to generate code for your real-time computer. For information, see "Embedded Coder Product Description" (Embedded Coder).

---

## See Also
**Simulink Real-Time Explorer**

## Related Examples
- "Set Up and Configure Simulink Real-Time" (Simulink Real-Time)

- "Create and Run Real-Time Application from Simulink Model" (Simulink Real-Time)

## More About

# Software and Hardware Configuration

Before simulating your Simscape model on your target hardware using Simulink Real-Time, configure your development and target computers for code generation and real-time simulation:

**1** Install Simulink Real-Time software on your development computer. The development computer downloads the kernel software and real-time application to your target machine at run time. Code generation and hardware-in-the-loop (HIL) simulation with Simulink Real-Time require specific hardware and software, including a C compiler. For information, see "Code Generation Requirements" on page 11-111. For Simulink Real-Time installation and configuration information, see Simulink Real-Time Software Requirements and "Install Development Computer Software" (Simulink Real-Time).

**2** Configure your real-time target computer for HIL with Simulink Real-Time. For Simulink Real-Time to work properly on the target computer, you configure the target settings to match the settings of the development computer and boot the target machine. For information, see "Target Computer Settings" (Simulink Real-Time).

**3** Configure your target computer with the required I/O modules. You can link to I/O boards via connection cables or install them in PCI slots of the target computer. The I/O boards provide a direct interface to the sensors, actuators, and other devices for real-time control or signal processing system models.

For information on supported I/O boards, see Supported Hardware: Hardware Drivers.

**4** Configure your development computer for Ethernet communication with your target computer. For information, see "Enable Development Computer Communication (Windows)" (Simulink Real-Time).

**5** Physically connect your target computer to your development computer. For information, see "Set Up Target Computer Ethernet Connection" (Simulink Real-Time).

**6** Configure the Simulink Real-Time target settings. For a configuration procedure that uses Simulink Real-Time, see "Target Computer Settings" (Simulink Real-Time).

## See Also
**Simulink Real-Time Explorer**

## More About
- "Code Generation Requirements" on page 11-111
- "Target Computer Settings" (Simulink Real-Time)
- "Select and Configure C or C++ Compiler" (Simulink Coder)
- "Basics of Hardware-In-The-Loop simulation" on page 11-103
- "Hardware-In-The-Loop Simulation Workflow" on page 11-106

# Signal and Parameter Visualization and Control

The Simulink Real-Time environment contains an interface for your target computer. The interface provides these capabilities for signal acquisition:

- Create, save, and load signal groups.
- Monitor signals.
- Add and configure host, target machine, or file scopes.
- Attach signals to or remove signals from scopes.
- Start and stop scopes.
- Attach signals to instruments.

The Simulink Real-Time interface also provides these capabilities for changing parameters:

- Create, save, and load parameter groups.
- Display and change parameter values.
- Attach parameters to instruments.

You can use the Simulink Real-Time interface from MATLAB or Simulink. You can also use other development environments to create custom standalone client applications outside of MATLAB. Therefore, there are several ways that you can visualize and control signals and parameters in a real-time application from development and target computers.

This table compares the interfaces that Simulink Real-Time supports for signal visualization and parameter control.

| Interface | Signal Acquisition | Parameter Control | Can Run Outside of MATLAB |
|---|---|---|---|
| **Simulink Real-Time Explorer** | Yes | Yes | Yes |
| "Simulink Real-Time Command-Line Interface in MATLAB" (Simulink Real-Time) | Yes | Yes | No |
| "Simulink External Mode Interface" (Simulink Real-Time) | Yes | Yes | No |
| "Visualize Signals in Simulink Real-Time Models" (Simulink Real-Time) | Yes | No | No |
| "Target Computer Command Line" (Simulink Real-Time) | Yes | Yes | Yes |

## See Also
slrealtime | slrtExplorer

## Related Examples
- "Monitor Signals by Using Simulink Real-Time Explorer" (Simulink Real-Time)
- "Tune Parameters by Using Simulink Real-Time Explorer" (Simulink Real-Time)

## More About

- "Tunable Block Parameters and Tunable Global Parameters" (Simulink Real-Time)
- "Signal Monitoring Basics" (Simulink Real-Time)
- "Signal Tracing Basics" (Simulink Real-Time)
- "Target and Application Objects" (Simulink Real-Time)

# Troubleshoot Hardware-in-the-Loop Simulation Issues

If your real-time application generates an overrun, to improve application execution time:

- Use the processes described in "Real-Time Model Preparation Workflow" on page 11-4, "Real-Time Simulation Workflow" on page 11-72, and "Hardware-In-The-Loop Simulation Workflow" on page 11-106.

- Run the Simulink Real-Time Performance Advisor Checks. Use the `Execute real-time application` activity mode in Performance Advisor, which includes checks specific to physical models. The mode helps you optimize your Simscape model for real-time execution. The checks are organized in folders. The checks in the **Simscape checks** folder are applicable to all physical models. Subfolders contain checks that target blocks from add-on products such as Simscape Electrical and Simscape Driveline™.

  To access the checks:

  **1** Open the Performance Advisor. On the **Debug** tab, click **Performance Advisor** > **Performance Advisor**.

  **2** In the Performance Advisor window, under **Activity**, select `Execute real-time application`.

  **3** In the left pane, expand the **Real-Time** folder, and then the **Simscape checks** folder.

  **4** Run the top-level Simscape checks. If your model contains blocks from an add-on product, also run the checks in the subfolder corresponding to that product.

  For more information, see "Troubleshoot Unsatisfactory Real-Time Performance" (Simulink Real-Time).

A Simulink Real-Time simulation can also fail due to development and target computer issues, changes in underlying system software, I/O module issues, and procedural errors. To address these issues, follow the workflow in "Troubleshooting Basics" (Simulink Real-Time). For more information, see "Troubleshooting in Simulink Real-Time" (Simulink Real-Time).

## See Also

## More About

- "Real-Time Model Preparation Workflow" on page 11-4
- "Real-Time Simulation Workflow" on page 11-72
- "Hardware-In-The-Loop Simulation Workflow" on page 11-106
- "Troubleshooting Real-Time Simulation Issues" on page 11-80
- "Automated Performance Optimization"

# Generate, Download, and Execute Code

| In this section... |
|---|
| "Requirements for Building and Executing Simulink Real-Time Applications" on page 11-117 |
| "Create, Build, Download, and Execute a Real-Time Application" on page 11-117 |

To perform hardware-in-the-loop simulation on target hardware, use Simulink Real-Time to:

- Generate and compile code on the development computer.
- Download the real-time application to the target computer.
- Execute the real-time application remotely from the development computer.

## Requirements for Building and Executing Simulink Real-Time Applications

Before building and executing your real-time application:

1 Prepare and configure your model for real-time simulation. For information, see "Real-Time Model Preparation Workflow" on page 11-4 and "Real-Time Simulation Workflow" on page 11-72.

2 Set up and configure the software, I/O interfaces, and connectivity for your development computer, target computer, and I/O board. For information, see "Get Started with Simulink Real-Time" (Simulink Real-Time).

## Create, Build, Download, and Execute a Real-Time Application

For an example that shows you how to generate code for the model on your development computer, transfer the code to your real-time computer, and execute the code on your real-time computer, see "Create and Run Real-Time Application from Simulink Model" (Simulink Real-Time).

## See Also
slrtExplorer

## Related Examples

- "Set Up and Configure Simulink Real-Time" (Simulink Real-Time)
- "Create and Run Real-Time Application from Simulink Model" (Simulink Real-Time)

## More About

- "Hardware-In-The-Loop Simulation Workflow" on page 11-106
- "Basics of Hardware-In-The-Loop simulation" on page 11-103
- "About Code Generation from Simscape Models" on page 12-2
- "How Simscape Code Generation Differs from Simulink" on page 12-5
- "Code Generation Requirements" on page 11-111

# Change Parameter Values on Target Hardware

| In this section... |
| --- |
| "Prerequisites" on page 11-118 |
| "Configure the Simscape Model for Deployment" on page 11-118 |
| "Deploy the Model to the Real-Time Target Machine" on page 11-119 |
| "Change Parameters and See Results Using Simulink Real-Time Explorer" on page 11-119 |

This example shows how to:

- Configure a Simscape model to generate code that supports signal visualization and changes to Simscape run-time parameters.
- Use Simulink Real-Time and Simulink Coder to deploy an executable version of the model to a real-time target machine.
- Use Simulink Real-Time Explorer on your development computer to change the value of a Simscape run-time parameter on the target machine and to see the effects of the parameter change.

## Prerequisites

This example requires an active connection between your development computer and a real-time target machine. For information on configuring and connecting your development computer to target hardware, see "Get Started with Simulink Real-Time" (Simulink Real-Time).

## Configure the Simscape Model for Deployment

To enable your development computer to change parameter values on a real-time target machine, configure the Simscape run-time and code generation parameters for your Simscape model.

1   To open the reference model, at the MATLAB command prompt, enter:

    `ssc_resistive_ac_circuit`

    The model opens and the PreLoadFcn loads parameters for the model to the MATLAB workspace. The peak voltage, `A_peak_voltage_src`, is 3 V, the resistance, `R_resistor`, is 10 Ohms, and the step size is 1e-5.

2   To allot enough time to see the effects of parameter-tuning on the target machine, configure the application to run until you stop the simulation by setting the simulation stop time to `inf`.

3   Adjust the step size for real-time simulation. At the MATLAB command prompt, enter:

    `ts = 8e-5;`

4   Configure the model for code generation using Simulink Coder and Simulink Real-Time.

    a   Open the Configuration Parameters window. In the Simulink Editor, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters window opens.

    b   In the **Code Generation** pane, to the right of **System target file**, click **Browse** and select `slrealtime.tlc`.

    c   In the System Target File Browser window, click **OK**.

**d** Open the **Code Generation** > **Report** pane.

**e** To display a code generation report select **Create code generation report** and **Open report automatically**.

**f** Click **OK**.

**5** Enable signal logging on the signal that you want to view in the Simulation Data Inspector. Click the signal named Current and, from the action menu, select **Enable Data Logging**.
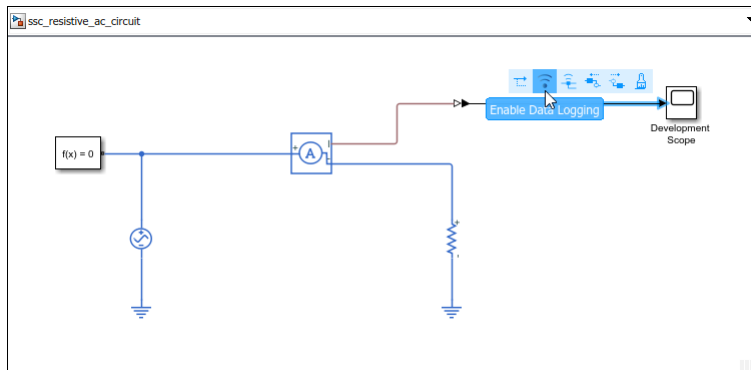


## Deploy the Model to the Real-Time Target Machine

Build an executable application to be deployed on the target machine.

**1** Check that you are connected to the real-time target machine:

```
tg = slrealtime
```

**2** To build the code to be deployed, in the Simulink Editor, open the **Real-Time** tab and click **Run on Target** > **Build Application**.

The code report opens after the code download.

**3** Verify that the generated code represents the Simscape run-time variables in a data structure.

**a** In the Code Generation Report, in the left pane, in the **Data files** node, open `ssc_resistive_ac_circuit_data.cpp`.

**b** Search for the section of the code that contains the parameter variables. In the **Find** box, enter `Block parameters (default storage)`.

**c** Verify that the `A_peak_voltage_src` and the `R_resistor` variables are represented in the `P_ssc_resistive_ac_circuit_T ssc_resistive_ac_circuit_P` data structure.

## Change Parameters and See Results Using Simulink Real-Time Explorer

Use Simulink Real-Time Explorer to change Simscape run-time parameters between runs of your real-time application on target hardware. Visualize the simulation results on a scope in the Explorer window.

**1** To open Simulink Real-Time Explorer, on your development computer, at the MATLAB command prompt, enter:

```
slrtExplorer
```

**2** Select the target computer in the **Targets Tree** panel. To connect to the target computer, click **Disconnected**, to toggle it to **Connected**.
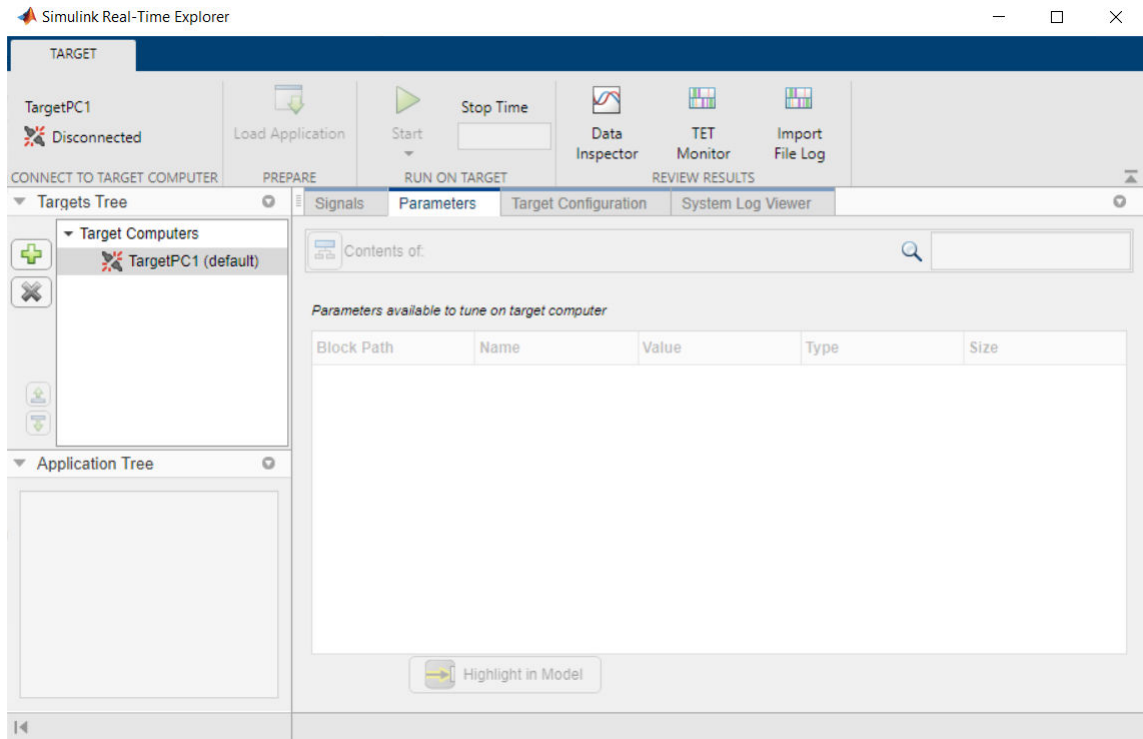


**3** To load the real-time application built earlier, click **Load Application**. In the **Application on host computer** pane, click **File Selector** and select the `ssc_resistive_ac_circuit.mldatx` file. Click **Load**.

**4** To select the signals for streaming, on the **Signals** tab, select the signal **Current**, click the Add selected signals button  to add the signal to the list in the right pane, and then click the **Start Streaming** button.
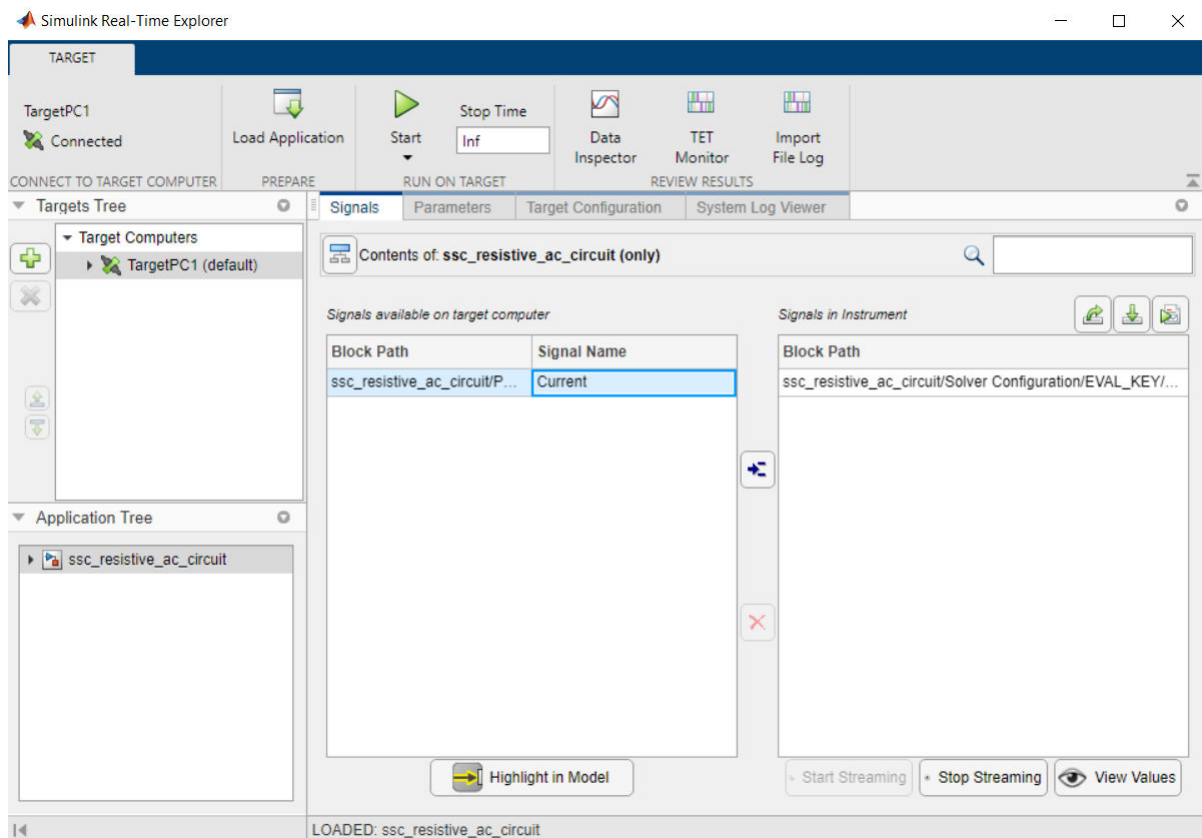
**5** To view the Simscape run-time parameters in Simulink Real-Time Explorer, open the Signals and Parameters **Parameters** tab and click the Show contents of current system and below button

.
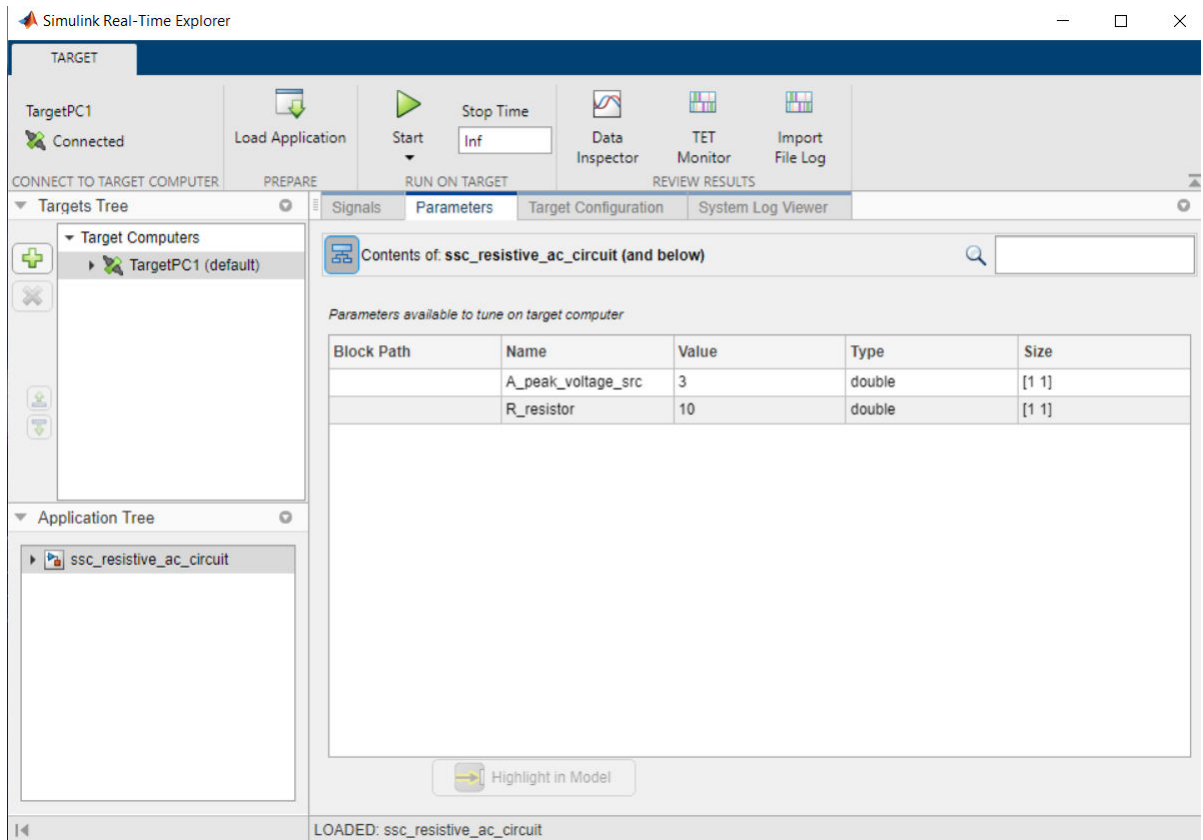
**6**    To run the application with the original peak amplitude value, click **Start**.

**7**    To view the streaming signals, click **Data Inspector**.
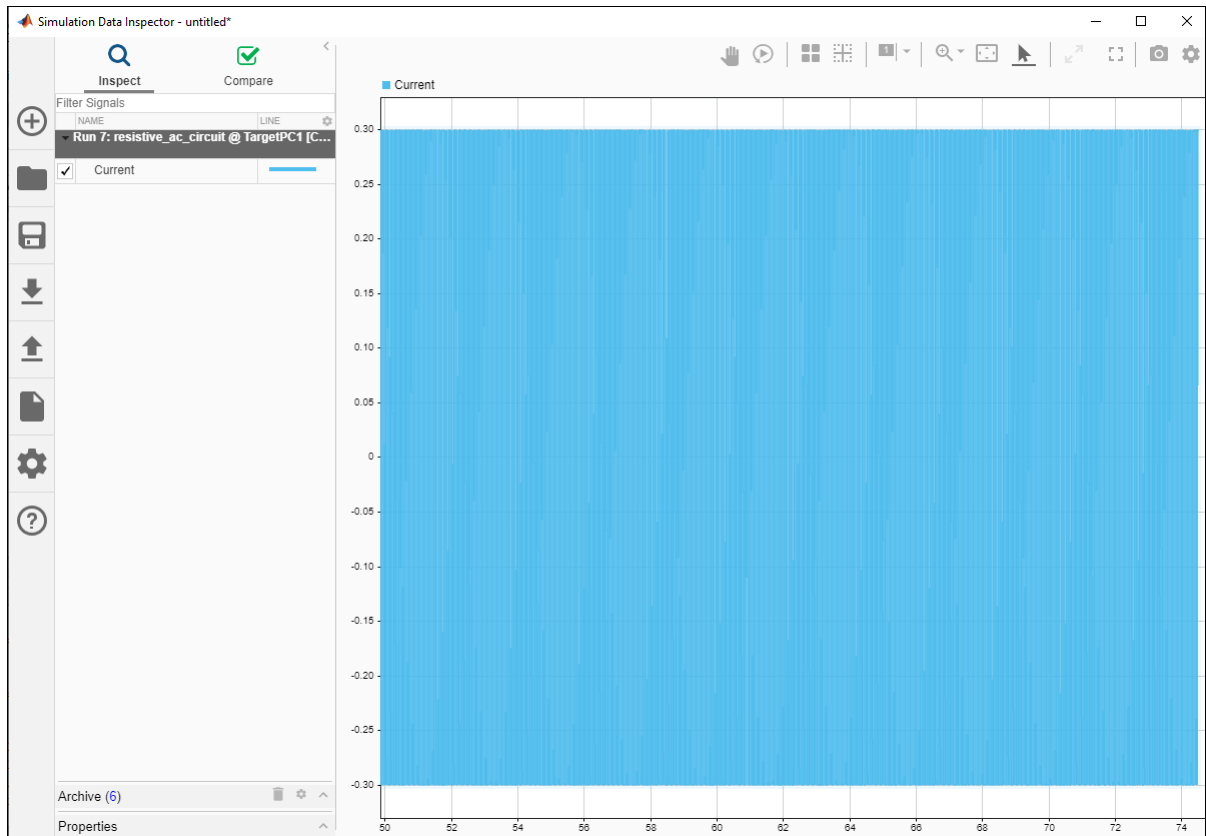
The streamed data shows that the current is approximately 0.3 A. The defining equation for the circuit in the model is $I = V/R$. The results are correct for the given voltage (10 V) and resistance (3 Ohms).

**8** Change the `A_peak_voltage_src` parameter, which represents the peak amplitude for the Voltage Source block. Because Simscape run-time parameters are run-time configurable, you cannot change the parameter value during simulation. Instead, you stop the simulation, change the value of the parameter, and apply the parameter change. Then, you restart the simulation to see how changing the parameter affects the results.

    **a** To stop execution, in the Simulink Real-Time Explorer window, click **Stop**.

    **b** Click **Value** box for the **A_peak_voltage_src** parameter and enter 50.



    **c** Click the **Start** button to simulate with the modified peak amplitude value.
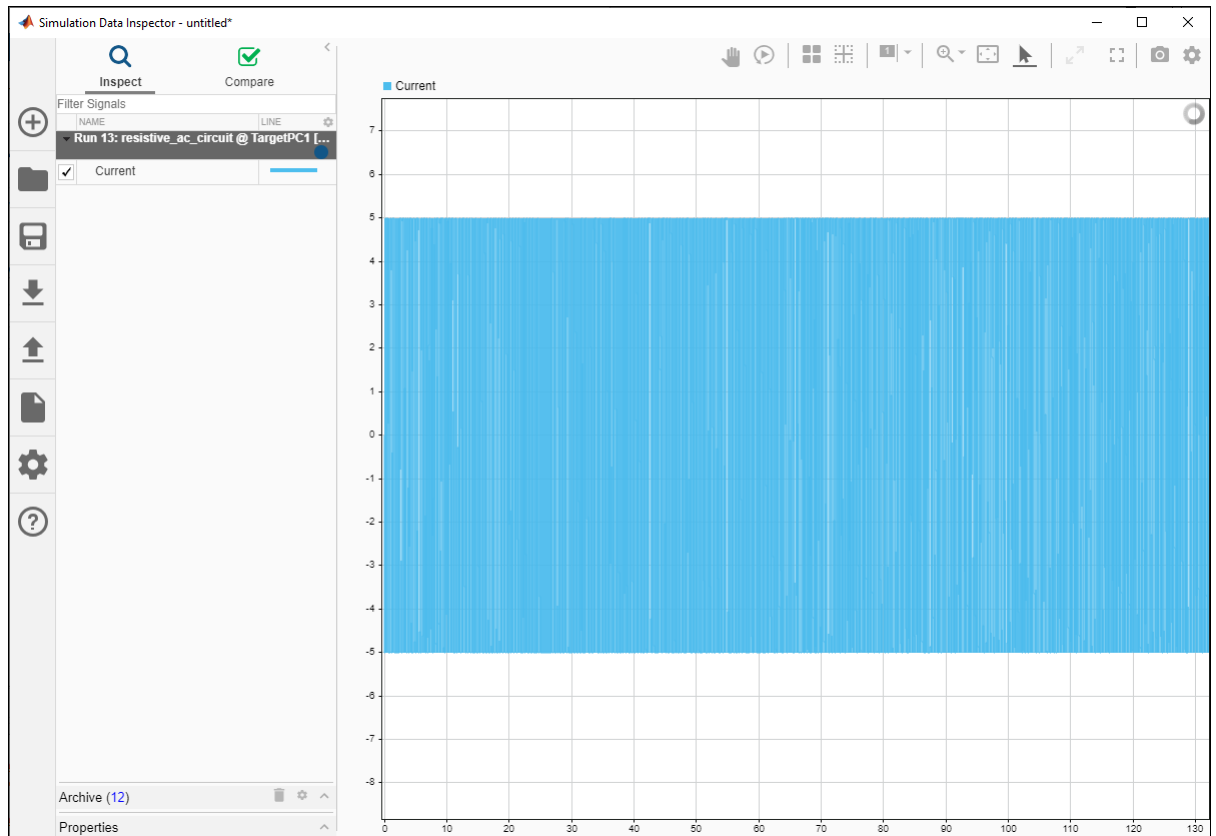
The streamed data shows that the current is approximately 5 A when the peak amplitude is 50 V. The results reflect the change in value for the voltage, given that the resistance is 10 Ohms.

## See Also

slrealtime | slrtExplorer

## Related Examples

- "Generate, Download, and Execute Code" on page 11-117
- "Set Up and Configure Simulink Real-Time" (Simulink Real-Time)
- "Configure and Control Real-Time Application by Using Simulink Real-Time Explorer" (Simulink Real-Time)
- "External Mode Simulation with TCP/IP or Serial Communication" (Simulink Coder)

## More About

- "Manage Simscape Run-Time Parameters" on page 10-4
- "Tunable Block Parameters and Tunable Global Parameters" (Simulink Real-Time)
- "Signal Monitoring Basics" (Simulink Real-Time)
- "Target and Application Objects" (Simulink Real-Time)
- "Model Callbacks"

# Requirements for Using Alternative Platforms

| **In this section...** |
| --- |
| "Hardware Requirements" on page 11-125 |
| "Software Requirements" on page 11-125 |

Simulink Real-Time creates standalone, real-time applications for performing hardware-in-the-loop (HIL) simulation on dedicated hardware. Creating and executing a standalone, real-time application using an alternative platform requires specific hardware and software.

## Hardware Requirements

The minimum hardware requirements for HIL simulation with a custom application are:

- Development computer with a network, serial, or USB interface for communicating with the real-time processor
- A real-time capable target CPU or computer that supports 64-bit precision floating-point arithmetic and 32-bit integer size
- I/O board supported by the real-time target machine
- Controller preconfigured with code from your controller model
- Peripheral for transferring code to the real-time target machine
- Wiring harness to connect the real-time target machine to the controller

**Note** Your real-time target machine may also require a real-time operating system (RTOS).

## Software Requirements

The minimum software requirements for HIL simulation with a custom application are:

- Embedded Coder and the Embedded Coder Software Requirements.
- Simulink Coder and the Simulink Coder Software Requirements.
- Template example main function that you can manually or automatically combine with generated code. For information, see "Incorporate Generated Code Using an Example Main Function" (MATLAB Coder).
- I/O driver. Options are:
  - C code I/O drivers for the code generation build
  - Precompiled static or dynamic library with the necessary documentation
- Compiler requirements
  - C compiler.
  - Cross compiler that supports 64-bit precision floating-point arithmetic and 32-bit integer size.

  For details on supported compiler versions, see Supported and Compatible Compilers.

  By default, Simulink Coder uses ISO®/IEC 9899:1990 (C89/C90 [ANSI]) library to produce C code. Not all compilers support this library. To learn how to enable the code generator to use a

**11-125**

different math extensions library in a model, see "Configure Standard Math Library for Target System" (Simulink Coder).

## See Also

### More About

- "About Code Generation from Simscape Models" on page 12-2
- "How Simscape Code Generation Differs from Simulink" on page 12-5
- "Hardware-In-The-Loop Simulation Workflow" on page 11-106
- "Limitations" on page 13-3
- "Basics of Hardware-In-The-Loop simulation" on page 11-103
- "Configure Run-Time Environment Options" (Simulink Coder)

# Extending Embedded and Generic Real-Time System Target Files

Simulink Coder and Embedded Coder use system target files (STFs) to generate code for interfacing with specific real-time operating systems. The Target Language Compiler (TLC) uses STFs and various other target files to convert a model into generated code. In addition to including STFs for ready-to-run configurations, Simulink Coder and Embedded Coder allow you to extend STFs to support third-party and custom target hardware. For more information on TLC files and STFs, including a list of available STFs, see "Configure a System Target File" (Simulink Coder) and "Target Language Compiler Basics" (Simulink Coder).

Generic real-time (GRT) is a Simulink Coder STF that you can use when you generate code from a Simscape model for hardware-in-the-loop (HIL) simulation. To generate code for HIL simulation, you must configure your Simscape model to use a fixed step, local solver. To learn about Simscape solver configurations that support HIL simulation, see "Solvers for Real-Time Simulation" on page 11-76.

Embedded real-time (ERT) is an Embedded Coder STF for deploying production-quality code for real-time execution of the algorithm for your Simulink controller. Do not deploy code that you generate from a Simscape model to production platforms. Simscape models contain constructs that are not compatible with performance-related Embedded Coder Model Advisor checks, such as "Check for blocks not recommended for C/C++ production code deployment". For more information, see "Embedded Coder Model Advisor Checks for Standards, Guidelines, and Code Efficiency" (Embedded Coder).

To extend ERT or GRT system target files and create hardware-specific, standalone applications, use the toolchain build process approach. The toolchain approach generates optimized makefiles and supports custom toolchains. For information, see "Customize System Target Files" (Simulink Coder) and "Support Toolchain Approach with Custom Target" (Simulink Coder). To get started extending system target files, see "Sample Custom Targets" (Simulink Coder).

Third-party vendors supply additional system target files for the Simulink Coder product. For more information about third-party products, see the MathWorks Connections Program Web page: https://www.mathworks.com/products/connections.

## See Also

### More About

- "Compare System Target File Support Across Products" (Simulink Coder)
- "Template Makefiles and Make Options" (Simulink Coder)
- "Customize System Target Files" (Simulink Coder)
- "Custom Target Optional Features" (Simulink Coder)
- "Configure Production and Test Hardware" (Simulink Coder)
- "Target Language Compiler Basics" (Simulink Coder)
- "Configure Generated Code with TLC" (Simulink Coder)
- "Target Language Compiler Library Functions Overview" (Simulink Coder)

# Precompiled Static Libraries

Simscape and its add-on products provide static libraries precompiled for compilers supported by Simulink Coder software. For details on supported compilers, see Supported and Compatible Compilers. For all other compilers, the static libraries needed by code generated from Simscape models are compiled once per model during the code generation build process.

To save time during the build process, precompile new or updated S-function libraries (MEX-files) for a model by using the MATLAB language function `rtw_precompile_libs`. You can also use the `rtw_precompile_libs` function to recompile a precompiled S-function library. Recompiling a precompiled library allows you to customize compiler settings for various platforms or environments. For details on using `rtw_precompile_libs`, see "Precompile S-Function Libraries" (Simulink Coder).

Typically, a target machine places cross-compiled versions of the precompiled libraries in a default location as specified in an rtwmakecfg.m file. The default file suffix and file extension used by the Simulink Coder code generator to name precompiled libraries during the build process are:

- On Windows® systems, *model*`_rtwlib.lib`
- On UNIX® or Linux® systems, *model*`_rtwlib.a`

You can control the file destination, location, suffix, and extension by customizing the system target file (STF) for your target hardware. For more information, see "Control Library Location and Naming During Build" (Simulink Coder) and "Use rtwmakecfg.m API to Customize Generated Makefiles" (Simulink Coder).

## See Also

`rtw_precompile_libs`

## Related Examples

- "Use rtwmakecfg.m API to Customize Generated Makefiles" (Simulink Coder)
- "Control Library Location and Naming During Build" (Simulink Coder)
- "Customize Template Makefiles" (Simulink Coder)
- "Precompile S-Function Libraries" (Simulink Coder)

# Initialization Cost

Initialization occurs at the beginning of simulation when simulation time, *t*, is at the model Start Time. During the first call to ModelOutputs, Simscape performs model initialization. The solver determines the simulation starting point by iteratively computing the initial values for all system variables. Finding a solution that satisfies the model equations for a nonlinear system can take more time than the time step allows. If the computation time exceeds the time step, a central processing unit (CPU) overload occurs. Real-time processors typically respond to CPU overloads by terminating model execution.

If your real-time application terminates when *t* is at the model Start Time, disable the automatic shutdown response on your real-time hardware to CPU overloads for that time period. To determine how to disable such processes, check with your hardware vendor.

## See Also

## More About

- "How Simscape Simulation Works" on page 7-6

# Generate HDL Code Using the Simscape HDL Workflow Advisor

With an HDL Coder license and the Simscape HDL Workflow Advisor, you can generate HDL code from your Simscape model for deployment onto FPGA platforms. The advisor generates a Simulink implementation of your Simscape, which it uses to generate HDL code using HDL Coder.

Converting your Simscape model to HDL code allows you to:

- Take advantage of the Simscape physical system modelling capabilities.
- Rapidly prototype models using the reconfigurability and parallelism capabilities of the FPGA.
- Simulate the HDL implementation in real time with hardware-in-the-loop (HIL).

Before running the Simscape HDL Workflow Advisor, configure your network to exclude:

- Events
- Mode charts
- Delays
- Enabled run time parameters
- Periodic sources
- Simulink-PS or PS-Simulink blocks performing unit conversions

The only supported units for the Simulink-PS input and PS-Simulink are 1, V, A, or `Inherit`.

## Generate HDL Code for a Simscape Model Using the Simscape HDL Workflow Advisor

This example shows how to convert your Simscape model to HDL code using the Simscape HDL Workflow Advisor and the Backward-Euler solver. To configure your Simscape network and Simulink model for real-time simulation and HDL code generation, see "Model Preparation" on page 11-130. To open a version of the model that is already prepared for using the Simscape HDL Workflow Advisor, see "Generate HDL Code by Using the Simscape HDL Workflow Advisor" on page 11-136.

---

**Note** This example involves reconfiguring a model to replace nonlinear blocks, which is necessary when you want to use the Backward-Euler solver. To avoid reconfiguring the model, you can set **Solver type** to `Partitioning solver`.

---

### Model Preparation

To prepare your Simscape model for FPGA deployment:

**1**   Open the `ssc_bridge_rectifier` model. At the MATLAB command prompt, enter

```
baselineModel = 'ssc_bridge_rectifier';
load_system(baselineModel)
open_system(baselineModel)
```

**2**   To compare the baseline simulation results to subsequent iterations, remove the data point limitation on the Scope block labeled Load Voltage scope block and capture the signal that inputs data to the Scope block by enabling data logging to the Simulation Data Inspector.

**a** Open the Scope block. Click **View > Configuration Properties**. On the **Logging** tab, clear **Limit data points to last**.

**b** Right-click the connection to the Scope block and select `Log selected signals`. The logging badge appears above the signal.
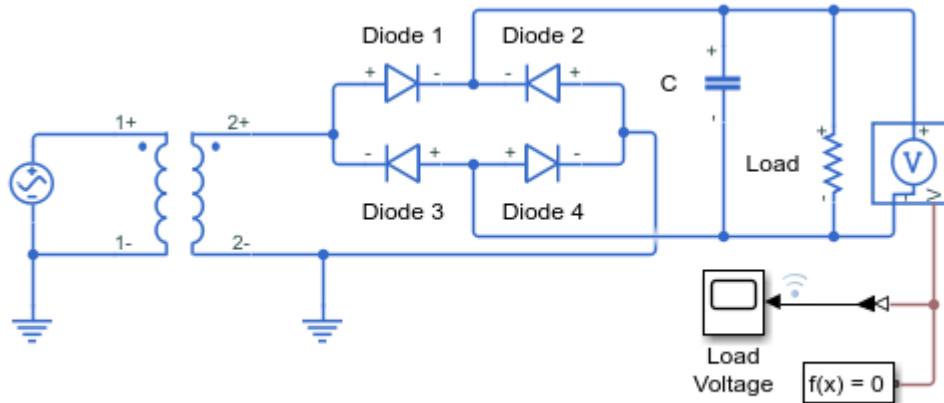


**3** Simulate the model and view the results in the Simulation Data Inspector.

```
%% Simulate baseline model
sim(baselineModel)

%% Get Simulation Data Inspector run IDs for
runIDs = Simulink.sdi.getAllRunIDs;
runID = runIDs(end);
run = Simulink.sdi.getRun(runID);
signal1 = run.getSignalByIndex(1);
% run.signalCount
signal1.checked = true;
Simulink.sdi.view
```

As needed, press the space bar on your keyboard to fit the Simulation Data Inspector plot to view.
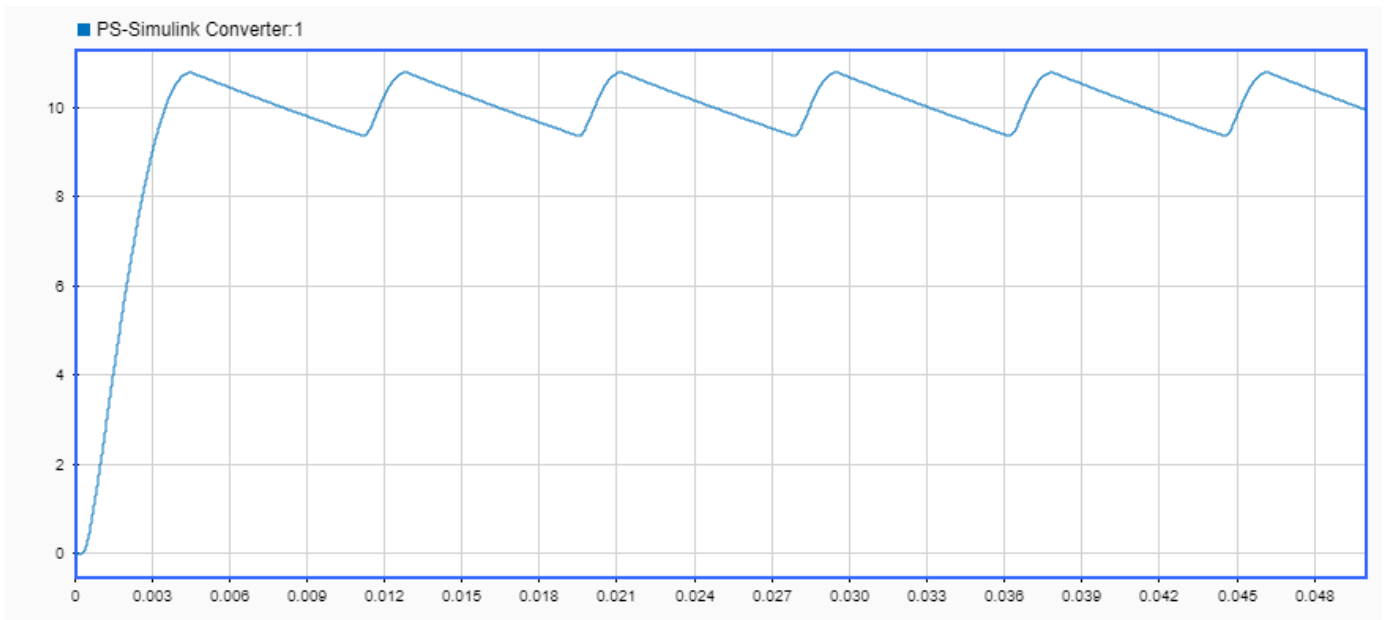
The baseline simulation returns the expected results for the full-wave bridge rectifier load voltage.

4. Before running the advisor, identify and replace blocks that cause your network to be nonlinear. To identify the blocks, use the `simscape.findNonlearBlocks` function.

```
simscape.findNonlinearBlocks(baselineModel)
```

```
Found network that contains nonlinear equations in the following blocks:
    'ssc_bridge_rectifier/AC Voltage Source'

The number of linear or switched linear networks in the model is 0.
The number of nonlinear networks in the model is 1.

ans =

  1×1 cell array

    {'ssc_bridge_rectifier/AC Voltage Source'}
```

The model contains an AC Voltage Source block, a periodic source that yields nonlinear equations.

5. You can replace the periodic source with a Controlled Voltage Source block in the Simscape network with a Sine Wave block outside the network.

   a. Delete the AC Voltage Source block.

   b. Add a Sine Wave block from the **Simulink > Sources** library.

   c. Add a Simulink-PS Converter block from the **Simscape > Utilities** library.

   d. Add a Controlled Voltage Source block from the **SimscapeFoundation LibraryElectricalElectrical Sources** library.

   e. Connect the Sine Wave block to the Simulink-PS Converter block and the Simulink-PS Converter block to the Controlled Voltage Source block.

**Full-Wave Bridge Rectifier**
1. Plot voltages and currents of rectifier (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example
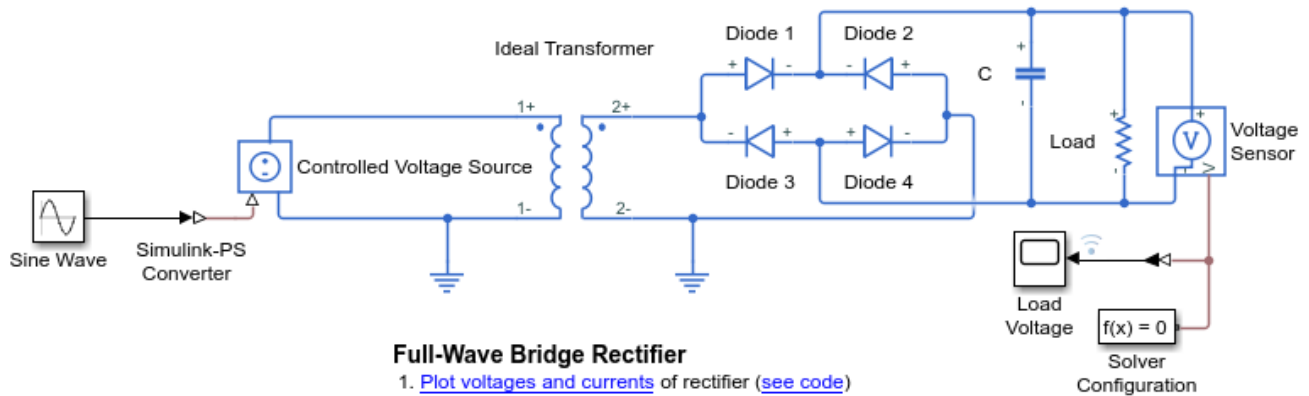
**6** Configure the Sine Wave block to match the parameters of the AC Voltage Source block that you removed.

**a** Set the **Amplitude** parameter to `sqrt(2)*120`.

**b** Set the **Frequency (rad/sec)** parameter to `60*2*pi`.

**c** Set the **Sample time** parameter to `1e-5`. Then click the three-dots icon ⠿ next to the **Sample time** box, and select **Create variable**. Name the variable `Ts` and click **Create**. You can now view and edit this variable in you workspace.

**7** Ensure that there are no blocks that cause your network to be nonlinear.

```
% Simulate
sim(baselineModel)

% Check for nonlinear blocks
simscape.findNonlinearBlocks(baselineModel)
```

```
The number of linear or switched linear networks in the model is 1.
```

```
ans =

  0×0 empty cell array
```

The model contains only blocks that yield linear or switched linear equations. Now the model is ready for the Backward-Euler solver.

**8** Simulate the model and compare the results to the baseline results in the Simulation Data Inspector.

```
% Get Simulation Data Inspector run IDs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 1);
runSwitchedLinear = runIDs(end);

% Open the Simulation Data Inspector
Simulink.sdi.view

compBaseline1 = Simulink.sdi.compareRuns(runBaseline,...
    runSwitchedLinear);
```

**11-133**
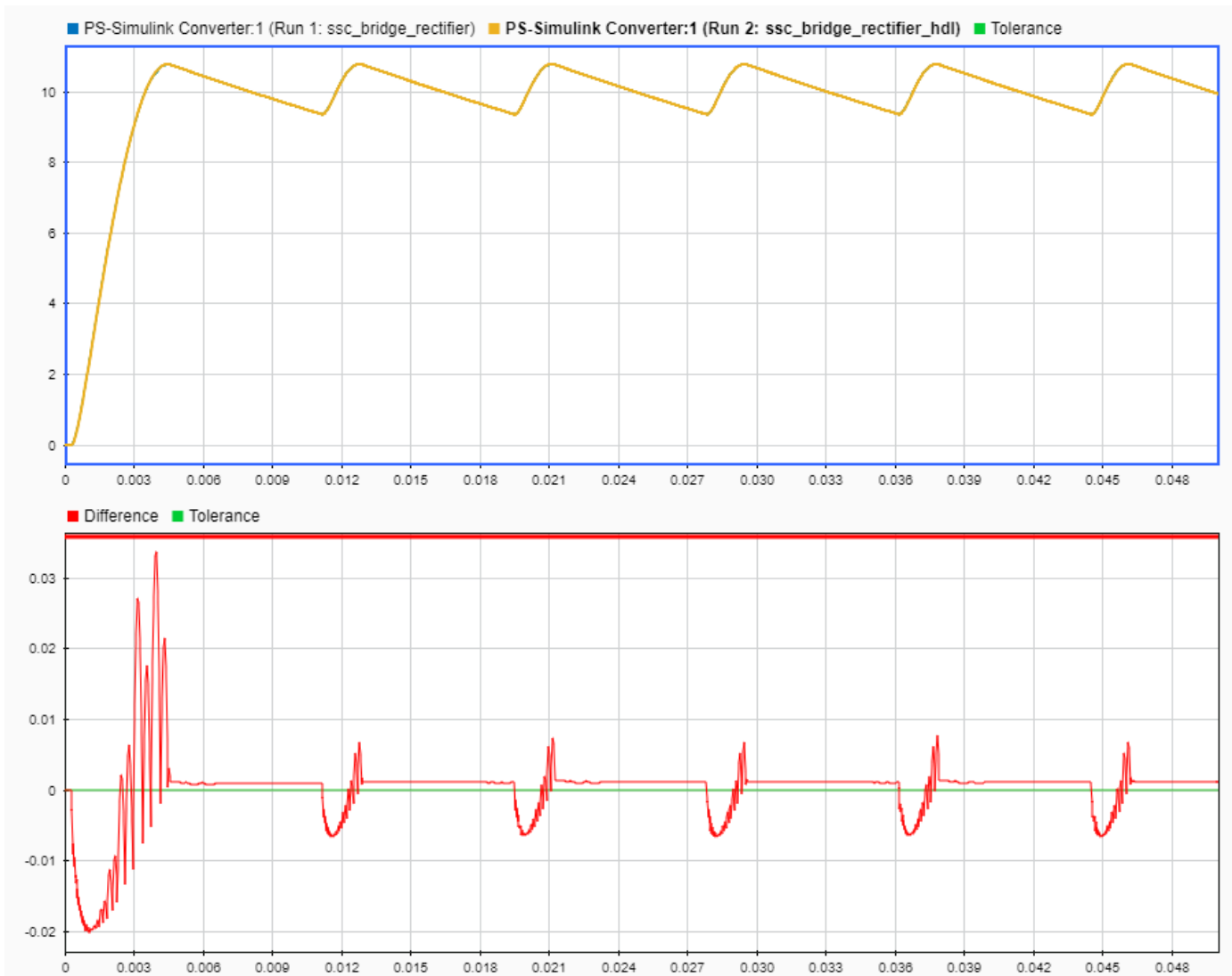
The results are similar to the baseline results.

**9**   To perform future progress checks for the Simscape HDL Workflow Advisor, add and connect a Digital Clock block from the **Simulink > Sources** library and a Display block from the **SimulinkSinks** library, as shown in the figure. For the Digital Clock, set the **Sample time** parameter to Ts.

**Full-Wave Bridge Rectifier**
1. Plot voltages and currents of rectifier (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example
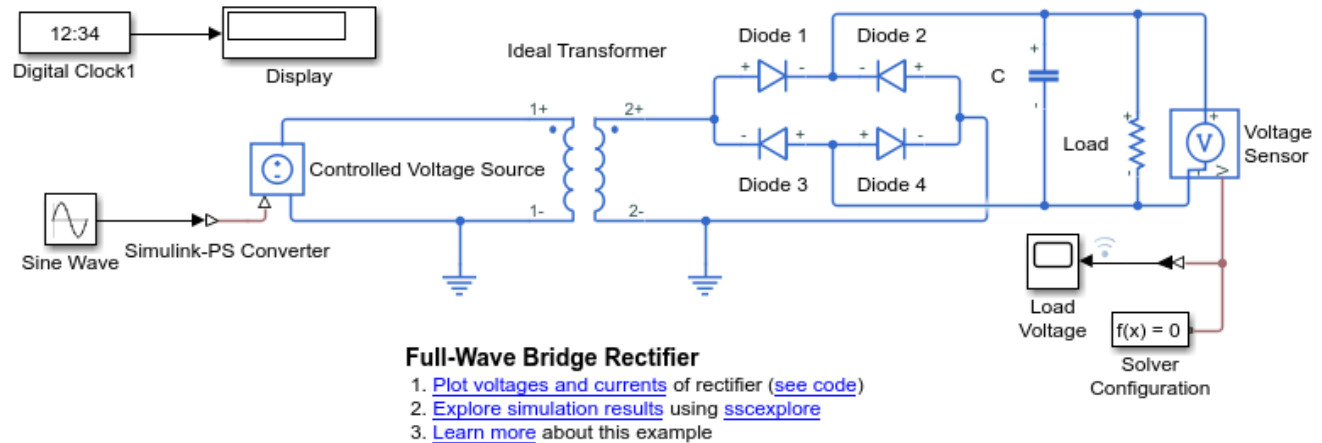
**10** The model is still set to use a variable-step solver. For real time-simulation, you must use a fixed-step solver. Use the sample time colors and annotations to help you to determine if your model contains any continuous settings. To turn on sample time colors and annotations, on the **Debug** tab, click **Information Overlays**, and in the **Sample Time** group, select **Colors** and **Text**.

The model diagram updates and the Timing Legend pane displays.

**11** Configure the model for real-time simulation.

   **a** Configure the Simulink model for fixed-step, fixed-cost simulation. In the Configuration Parameters window, click **Solver** and set:

     • **Type** to `Fixed-step`

     • **Solver** to `Discrete (no continuous states)`

   **b** Configure the Simscape network for fixed-step, fixed-cost simulation. For the Solver Configuration block:

     • Select **Use local solver**.

     • Set **Solver type** to `Backward Euler`.

     • Specify `Ts` for the **Sample time**.

> **Note** You can

**12** Simulate the model and compare the results to the baseline results in the Simulation Data Inspector.

```
% Simulate
sim(baselineModel)

% Get Simulation Data Inspector run IDs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 2);
runRealTime = runIDs(end);

% Open the Simulation Data Inspector
Simulink.sdi.view

compBaseline1 = Simulink.sdi.compareRuns(runBaseline,...
    runRealTime);
```

The results are similar to the baseline results.

**Generate HDL Code by Using the Simscape HDL Workflow Advisor**

Generate HDL code by running the Simscape HDL Workflow Advisor either on the Simscape model that you prepared in the "Model Preparation" on page 11-130 section or by opening the `ssc_bridge_rectifier_hdl` model, which is prepared for code generation.

**1** Rename the model.

- If you prepared the model in the Model Preparation section, rename the model `ssc_model`.
- To open and use a model that is already prepared for HDL code generation, at the MATLAB command prompt, enter

  ```
  open_system('ssc_bridge_rectifier_hdl')
  ```

  Save the model to a local directory as `ssc_model`.

**Full-Wave Bridge Rectifier for Simscape HDL Advisor**

1. Plot voltages and currents of rectifier (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example
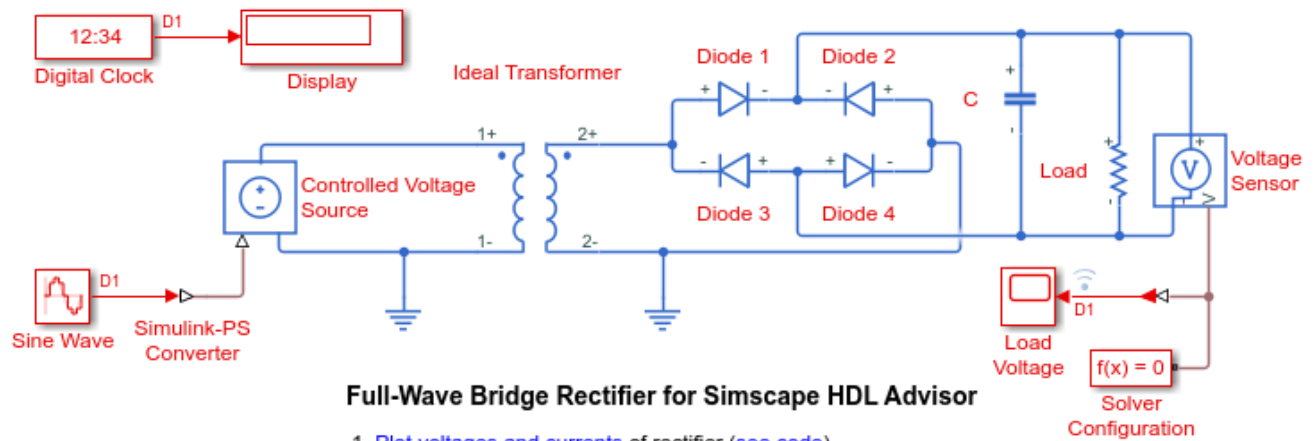
**2**   Run the Simscape HDL Workflow Advisor.

```
sschdladvisor('ssc_model')
```

The Simscape HDL Workflow Advisor opens.

**3**   Run the code generation compatibility checks.

**a**   Select **Code generation compatibility > Check solver configuration** , then click **Run This Task**.

**b**   Select **Check model compatibility**, then click **Run this task**.

The advisor reports when the model passes these checks.

**4**   Extract the state-space coefficients. Select **State-space conversion** and click **Run All**. The conversion can take some time.

After running the task, the advisor displays a summary of the state-space representation and a table of parameters.

- Number of states: 5
- Number of inputs: 1
- Number of outputs: 1
- Number of modes: 7
- Number of differential variables: 1
- Discrete sample time: 1e-05

| Parameter | Parameter size |
|-----------|----------------|
| A | 5 x 5 x 7 |
| B | 5 x 1 x 7 |
| F0 | 5 x 1 x 7 |
| C | 1 x 5 x 1 |

| D | 1 x 1 x 1 |
| --- | --- |
| Y0 | 1 x 1 x 1 |

The size of the state, mode, and parameter data helps you estimate how much of the FPGA resources are required to deploy the model. The higher the values, the more FPGA resources are required. The input and output data indicate the number and type of I/O connections needed for real-time deployment and visualization.

**Note** This table will have different values if you perform this step using the partitioning solver.

5  Generate an HDL implementation of your model. Select **Implementation model generation > Generate implementation model** and click **Run this task**.

When the Simscape HDL Workflow Advisor generates the implementation model, the advisor reports that the task passed and displays a link to the generated implementation model, which is named gmStateSpaceHDL_ssc_model.

6  Open the generated implementation model by clicking **gmStateSpaceHDL_ssc_model**.
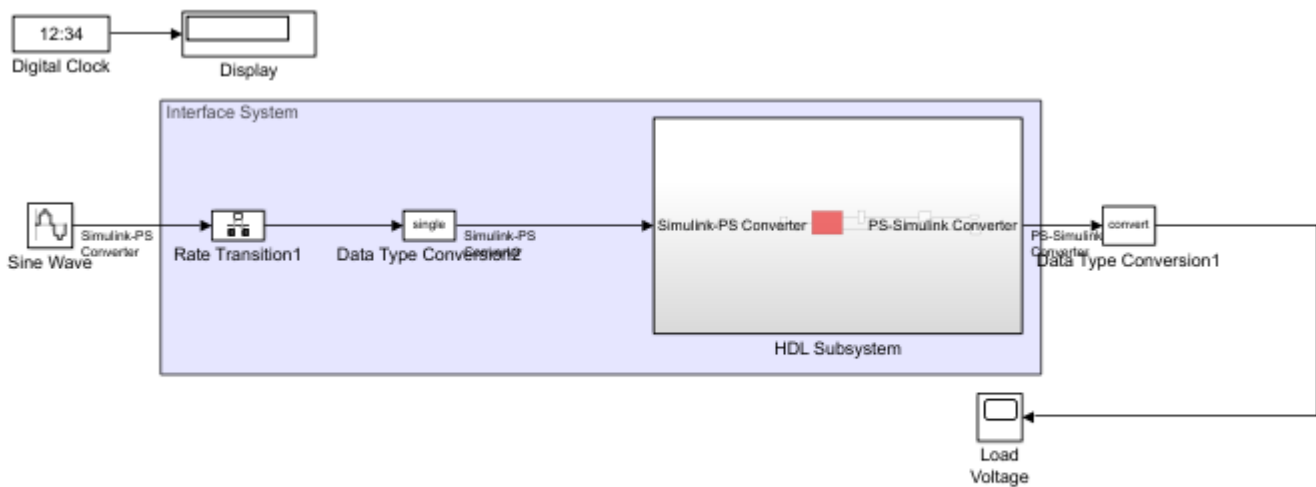


The model contains blocks from the original model as well as new blocks that support the HDL Workflow Advisor:

- Digital Clock, Display, Sine Wave, and Load Voltage—Remnants from your original model
- Rate Transition1 — Handles the transfer of data between blocks operating at different rates.
- Data Type Conversion1, Data Type Conversion2 — Converts between double and single precision data types. HDL code generation requires single-precision data
- HDL Subsystem —Contains an HDL code generation-compatible version of your Simscape network.
- Load Voltage — Scope block that displays the load voltage.

7  Prepare the implementation model for a simulation comparison to the baseline results:

    **a**   You can adjust the automatically generated model and delete vestigial blocks as necessary to improve model cleanliness. The Digial Clock block and the Display block are unnecessary but will not inhibit the simulation results.

    **b**   Right-click the input signal to the Scope block and click **Log Selected Signals**.

The Display block in the model window shows the elapsed simulation time.
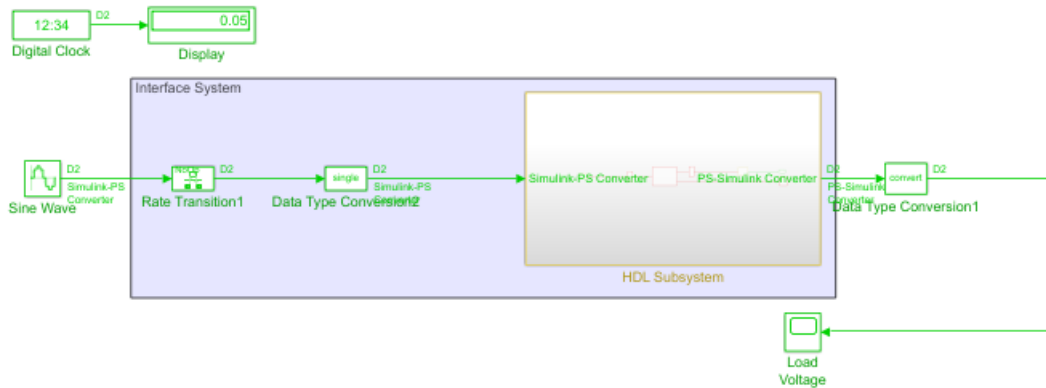


**8**   To ensure that the HDL subsystem corresponds to your original Simscape model, simulate the model and compare the results to the baseline simulation results.

```
% Simulate
sim('gmStateSpaceHDL_ssc_model')

% Get Simulation Data Inspector run IDs
runIDs = Simulink.sdi.getAllRunIDs;
runBaseline = runIDs(end - 3);
runHDLImplementation = runIDs(end);

% Open the Simulation Data Inspector
Simulink.sdi.view

compBaseline1 = Simulink.sdi.compareRuns(runBaseline,...
    runHDLImplementation);
```
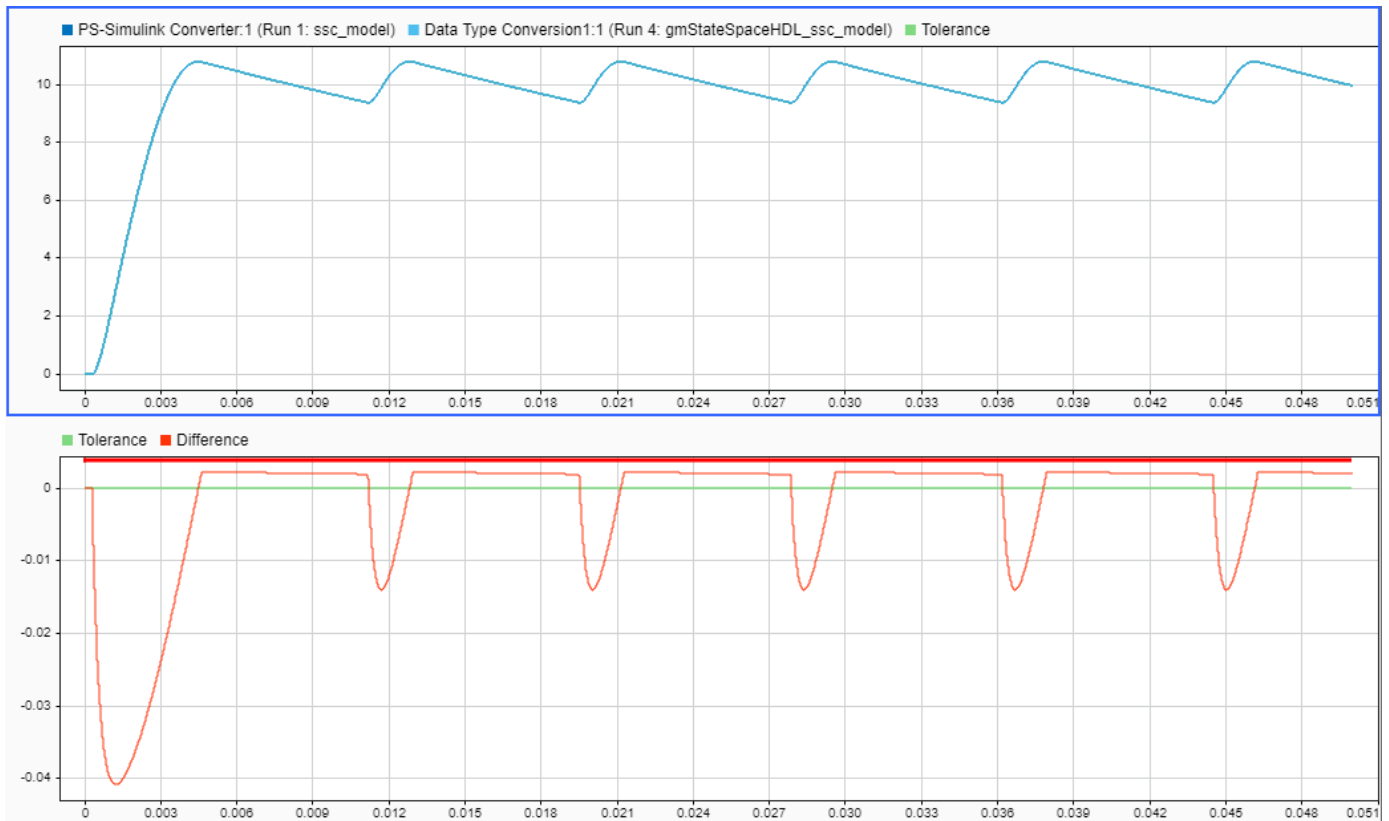
The results are similar to the baseline results. The Simscape model is compatible with HDL code generation.

**9** Generate HDL code from the implementation:

**a** Access the Configuration Parameters window from the HDL implementation model. Expand **HDL Code Generation** and select **Report**. Check the boxes for the **Generate traceability report** and **Generate resource utilization report** options.

**b** Run the `hdlsetup` function.

```
hdlsetup('gmStateSpaceHDL_ssc_model')
```

**c** Save the model and subsystem parameter settings.

```
hdlsaveparams('gmStateSpaceHDL_ssc_model');

%% Set Model 'gmStateSpaceHDL_ssc_model' HDL parameters
hdlset_param('gmStateSpaceHDL_ssc_model', 'FloatingPointTargetConfiguration', hdlcoder.createFloatingPointTarge
, 'LatencyStrategy', 'MIN') ...
);
hdlset_param('gmStateSpaceHDL_ssc_model', 'HDLSubsystem', 'gmStateSpaceHDL_ssc_model/HDL Subsystem');
hdlset_param('gmStateSpaceHDL_ssc_model', 'MaskParameterAsGeneric', 'on');
hdlset_param('gmStateSpaceHDL_ssc_model', 'Oversampling', 49);

% Set SubSystem HDL parameters
hdlset_param('gmStateSpaceHDL_ssc_model/HDL Subsystem', 'FlattenHierarchy', 'on');

hdlset_param('gmStateSpaceHDL_ssc_model/HDL Subsystem/HDL Algorithm/Mode Selection/Generate Mode Vector', 'Arch

% Set SubSystem HDL parameters
hdlset_param('gmStateSpaceHDL_ssc_model/HDL Subsystem/HDL Algorithm/State Update/Multiply State', 'SharingFacto
```

**d** Save the validation model generation settings.

```
HDLmodelname = 'gmStateSpaceHDL_ssc_model';
hdlset_param(HDLmodelname, 'GenerateValidationModel', 'on');
```

**e** Generate HDL code.

```
makehdl('gmStateSpaceHDL_ssc_model/HDL Subsystem')
```

```
### Generating HDL for 'gmStateSpaceHDL_ssc_model/HDL Subsystem'.
### Using the config set for model gmStateSpaceHDL_ssc_model for HDL code generation parameters.
### Running HDL checks on the model 'gmStateSpaceHDL_ssc_model'.
### Begin compilation of the model 'gmStateSpaceHDL_ssc_model'...
### Applying HDL optimizations on the model 'gmStateSpaceHDL_ssc_model'...
### The code generation and optimization options you have chosen have introduced additional pipeline delays.
### The delay balancing feature has automatically inserted matching delays for compensation.
### The DUT requires an initial pipeline setup latency. Each output port experiences these additional delays.
### Output port 1: 1 cycles.
### Begin model generation.
### Model generation complete.
### Clock-rate pipelining results can be diagnosed by running this script: hdlsrc\gmStateSpaceHDL_ssc_model\...
highlightClockRatePipelining.m
### To clear highlighting, click the following MATLAB script: hdlsrc\gmStateSpaceHDL_ssc_model\clearhighlighting
### Generating new validation model: gm_gmStateSpaceHDL_ssc_model_vnl.
### Validation model generation complete.
### Begin VHDL Code Generation for 'gmStateSpaceHDL_ssc_model'.
### MESSAGE: The design requires 49 times faster clock with respect to the base rate = 3.33333e-06.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/HDL Algorithm/Mode Iteration Manager/Compare To Constant
gmStateSpaceHDL_ssc_model\Compare_To_Constant.vhd.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/nfp_mul_single as hdlsrc\gmStateSpaceHDL_ssc_model\nfp_mu
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/nfp_add_single as hdlsrc\gmStateSpaceHDL_ssc_model\nfp_ad
### Working on dot_product_2 as hdlsrc\gmStateSpaceHDL_ssc_model\dot_product_2.vhd.
### Working on dot_product_1 as hdlsrc\gmStateSpaceHDL_ssc_model\dot_product_1.vhd.
### Working on dot_product_1 as hdlsrc\gmStateSpaceHDL_ssc_model\dot_product_1_block.vhd.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/nfp_relop_single as hdlsrc\gmStateSpaceHDL_ssc_model\nfp_
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem/nfp_uminus_single as hdlsrc\gmStateSpaceHDL_ssc_model\nfp
### Working on HDL Subsystem_tc as hdlsrc\gmStateSpaceHDL_ssc_model\HDL_Subsystem_tc.vhd.
### Working on gmStateSpaceHDL_ssc_model/HDL Subsystem as hdlsrc\gmStateSpaceHDL_ssc_model\HDL_Subsystem.vhd.
### Generating package file hdlsrc\gmStateSpaceHDL_ssc_model\HDL_Subsystem_pkg.vhd.
### Code Generation for 'gmStateSpaceHDL_ssc_model' completed.
### Creating HDL Code Generation Check Report HDL_Subsystem_report.html
### HDL check for 'gmStateSpaceHDL_ssc_model' complete with 0 errors, 2 warnings, and 2 messages.
### HDL code generation complete.
```

The HDL code generation report opens and includes any generated errors or warnings. The report includes a link to the resource utilization report, which describes the resource requirements for FPGA deployment.

The generated HDL code and validation model are saved in the `hdlsrc\gmStateSpaceHDL_ssc_model\html` directory. The generated code is saved as `HDL_Subsystem_tc.vhd`.

To generate HDL code for deployment to a specified target, use the HDL Workflow Advisor.

## See Also

**Blocks**
Digital Clock | Display | Sine Wave

**Simscape Blocks**
Controlled Voltage Source | Simulink-PS Converter

**Functions**
hdladvisor | hdlsaveparams | hdlset_param | hdlsetup | makehdl |
simscape.findNonlinearBlocks | sschdladvisor

## Related Examples

- "View Sample Time Information"
- "View Data in the Simulation Data Inspector"

## More About

- "Hardware-In-The-Loop Simulation Workflow" on page 11-106
- "Real-Time Model Preparation Workflow" on page 11-4
- "Real-Time Simulation Workflow" on page 11-72

# Code Generation

# About Code Generation from Simscape Models

You can use Simulink Coder software to generate standalone C or C++ code from your Physical Networks models and enhance simulation speed and portability. Certain features of Simulink software also make use of generated or external code. This section explains code-related tasks you can perform with your Simscape models.

Code versions of Simscape models typically require fixed-step Simulink solvers, which are discussed in the Simulink documentation. Some features of Simscape software are restricted when you translate a model into code. See "How Simscape Code Generation Differs from Simulink" on page 12-5, as well as "Limitations" on page 7-69.

**Note** Code generated from Simscape models is intended for rapid prototyping and hardware-in-the-loop applications. It is not intended for use as production code in embedded controller applications.

Add-on products based on the Simscape platform also support code generation, with some variations and exceptions described in their respective documentation.

# Reasons for Generating Code

Code generation has many purposes and methods. There are two essential rationales:

- Compiled code versions of Simulink and Simscape models run faster than the original block diagram models. The time savings can be dramatic.
- An equally important consideration for Simscape models is the standalone implementation of generated and compiled code. Once you convert part or all of your model to code, you can deploy the standalone executable program on virtually any platform, independently of MATLAB.

  Converting a model or subsystem to code also hides the original model or subsystem.

# Using Code-Related Products and Features

With Simulink, Simulink Coder, and Simulink Real-Time software, using several code-related technologies, you can link existing code to your models and generate code versions of your models.

| Code-Related Task | Component or Feature |
|---|---|
| Link existing code written in C or other supported languages to Simulink models | Simulink S-functions to generate customized blocks |
| Speed up Simulink simulations | Accelerator mode<br>Rapid Accelerator mode |
| Generate standalone fixed-step code from Simulink models | Simulink Coder software |
| Generate variable-step code from Simulink models, well-suited for batch or Monte Carlo simulations | Simulink Coder Rapid Simulation Target (RSim) |
| Convert Simulink model to code and compile and run it on a target PC | Simulink Coder and Simulink Real-Time software |

# How Simscape Code Generation Differs from Simulink

| In this section... |
| --- |
| "Simscape and Simulink Code Generated Separately" on page 12-5 |
| "Compiler and Processor Architecture Requirements" on page 12-5 |
| "Precompiled Libraries Provided for Selected Compilers" on page 12-5 |
| "Simscape Code Reuse Not Supported" on page 12-5 |
| "Tunable Parameters Not Supported" on page 12-6 |
| "Simscape Run-Time Parameter Inlining Override of Global Exceptions" on page 12-6 |

In general, using the code generated from Simscape models is similar to using code generated from regular Simulink models. However, there are certain differences.

## Simscape and Simulink Code Generated Separately

Simulink Coder software generates code from the Simscape blocks separately from the Simulink blocks in your model. The generated Simscape code does not pass through `model.rtw` or the Target Language Compiler. All the code generated from a single model resides in the same directory, however.

## Compiler and Processor Architecture Requirements

To generate and execute Simscape code, you must have a compiler and a processor that support:

- 64-bit precision floating-point arithmetic defined by IEEE® Standard for Floating-Point Arithmetic (IEEE 754)
- 32-bit integer size

For details on supported compiler versions, see

https://www.mathworks.com/support/compilers/current_release

## Precompiled Libraries Provided for Selected Compilers

Simscape software and its add-on products provide static runtime libraries precompiled for compilers supported by Simulink Coder software. For details, see

https://www.mathworks.com/support/compilers/current_release

For all other compilers, the static runtime libraries needed by code generated from Simscape models are compiled once per model during the code generation build process.

## Simscape Code Reuse Not Supported

Reusable subsystems in Simulink reuse code that is generated once from the subsystem. You cannot generate reusable code from subsystems containing Simscape blocks.

## Tunable Parameters Not Supported

A tunable parameter is a Simulink run-time parameter that you can change while the simulation is running. Simscape blocks do not support tunable parameters in either simulations or generated code. However, Simscape run-time parameters, which are parameters you can modify at run time, but not during simulation, are available. For more information, see "Run-Time Parameters".

## Simscape Run-Time Parameter Inlining Override of Global Exceptions

If you choose to enable parameter inlining for code generated from a Simscape model, the software inlines all its run-time parameters. If you choose to make some of the global Simscape block parameters exceptions to inlining, the exceptions are ignored. You can change global tunable parameters only by regenerating code from the model.

**13**

# Data Logging

# About Simulation Data Logging

| **In this section...** |
| --- |
| "Suggested Workflows" on page 13-2 |
| "Limitations" on page 13-3 |

## Suggested Workflows

You can log simulation data to the workspace, or to a temporary file on disk, for debugging and verification. Data logging lets you analyze how internal block variables change with time during simulation. For example, you might want to see that the pressure in a hydraulic cylinder is above some minimum value or compare it against the pump pressure. If you log simulation data, you can later query, plot, and analyze it without rerunning the simulation.

There are two methods of simulation data logging: you can store the data directly in a workspace variable, or you can stream data to a temporary file on disk and have the workspace variable point to that temporary file. For more information on the second method, see "Stream Logging Data to Disk" on page 13-42. In either case, you interact with the logged simulation data through the simulation log variable.

Simulation data logging can replace connecting sensors and scopes to track simulation data. These blocks increase model complexity and slow down simulation. "Log and Plot Simulation Data" on page 13-8 shows how you can log and plot simulation data instead of adding sensors to your model. It also shows how you can print the complete logging tree for a model and plot simulation results for a selected variable.

You can log data either for the whole model, or on a block-by-block basis. In the second case, the workspace variable will contain simulation data for selected blocks only. To log data for selected blocks only, you have to:

- Set the logging configuration parameter
- Select the blocks in your model

You can perform these two steps in any order. For more information, see "Log Data for Selected Blocks Only" on page 13-5.

After running the simulation, you can use the Simscape Results Explorer tool to navigate and plot the data logging results.

For additional information on how you can query, plot, and analyze data by accessing the simulation log variable, see the reference pages for the objects `simscape.logging.Node`, `simscape.logging.Series`, and their associated object functions.

You can also configure your model to automatically record Simscape logging data, along with the rest of the simulation data obtained from a model run, using the Simulation Data Inspector. This way, you can view and analyze the data while the simulation is running. Set up your model to log simulation data, either for the whole model or on a block-by-block basis. Enable data streaming by selecting the **Record data in Simulation Data Inspector** check box on the **Simscape** pane of the Configuration Parameters dialog box. When you simulate the model, as soon as the streamed data becomes available, the Simulation Data Inspector button in the model toolbar highlights. Open the Simulation Data Inspector to view the data during simulation and to compare data for different simulation runs.

For detailed information on how to configure and use the Simulation Data Inspector, see "Analyze Simulation Results".

If you have a Parallel Computing Toolbox™ license, you can make your model simulation and data logging compatible with the `parfor` command by selecting the **Single simulation output** check box on the **Data Import/Export** pane of the Configuration Parameters dialog box. In this case, Simscape log data will be part of the single output object, instead of being a separate workspace variable. For more information, see "Single simulation output". All the other data logging workflows described here assume that you clear the **Single simulation output** check box and that you interact with the logged Simscape data through the simulation log workspace variable.

## Limitations

Simulation data logging is not supported for:

- Model reference
- Generated code
- Accelerator mode
- Rapid accelerator mode
- Partitioning local solver

If you use the `sim` command with a `'StopTime'` name-value pair, the Simscape logging results are not updated.

## See Also

## Related Examples
- "Log, Navigate, and Plot Simulation Data" on page 13-20
- "Log and View Simulation Data for Selected Blocks" on page 13-17
- "Log and Plot Simulation Data" on page 13-8
- "Log Simulation Statistics" on page 13-13

## More About
- "Data Logging Options" on page 13-6
- "Stream Logging Data to Disk" on page 13-42

# Enable Data Logging for the Whole Model

Using data logging is a best practice for Simscape models because it provides access to important simulation and analysis tools. Therefore, when you create a model by using the `ssc_new` function or any of the Simscape model templates, data logging for the whole model is turned on automatically.

However, for models created using other methods, simulation data is not logged by default. To turn on the data logging for a model, use the **Log simulation data** configuration parameter.

**1** In the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

**2** In the Configuration Parameters dialog box, in the left pane, select **Data Import/Export**. Clear the **Single simulation output** check box, which is selected by default. This step enables associating the logged simulation data with a separate workspace variable, instead of it being part of the single output object.

**3** In the Configuration Parameters dialog box, in the left pane, select **Simscape**. The right pane displays the **Log simulation data** option, which is by default set to `None`.

**4** From the drop-down list, select `All`, then click **OK**.

**5** Simulate the model. This creates a workspace variable named `simlog` (as specified by the **Workspace variable name** parameter), which contains the simulation data.

For information on how to access and use the data stored in this variable, see the related examples listed below. For information on additional data logging configuration options, see "Data Logging Options" on page 13-6.

## See Also

## Related Examples
- "Log, Navigate, and Plot Simulation Data" on page 13-20
- "Log and Plot Simulation Data" on page 13-8
- "Log Simulation Statistics" on page 13-13

## More About
- "Data Logging Options" on page 13-6

# Log Data for Selected Blocks Only

Instead of logging the simulation data for the whole model, you can log data just for the selected blocks.

1. In the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

2. In the Configuration Parameters dialog box, in the left pane, select **Data Import/Export**. Clear the **Single simulation output** check box, which is selected by default. This step enables associating the logged simulation data with a separate workspace variable, instead of it being part of the single output object.

3. Set the logging configuration parameter to enable simulation data logging on a block-by-block basis.

   In the Configuration Parameters dialog box, in the left pane, select **Simscape**, then set the **Log simulation data** parameter to `Use local settings`. Click **OK**.

4. Select the blocks in your model. You can do this before or after setting the logging configuration parameter.

   For each block that you want to select for data logging, right-click the block. From the context menu, select **Simscape > Log simulation data**. A check mark appears in front of the **Log simulation data** option.

5. Simulate the model. When the simulation is done, the simulation data log contains only the data from the selected blocks.

To stop logging data for a previously selected block, right-click it and select **Simscape > Log simulation data** again to remove the check mark.

If you set the **Log simulation data** parameter to `All`, the simulation log will contain data from the whole model, regardless of the block selections. Setting the **Log simulation data** parameter to `None` disables data logging for the whole model.

## See Also

## Related Examples

- "Log and View Simulation Data for Selected Blocks" on page 13-17

## More About

- "Data Logging Options" on page 13-6

# Data Logging Options

When you set the **Log simulation data** configuration parameter to `All` or `Use local settings`, other options in the Data Logging group box become available.

- **Log simulation statistics** — Select this check box if you want to access and analyze information on zero crossings during simulation. By default, this check box is not selected and the zero-crossing data is not logged. For more information on using this check box, see "Log Simulation Statistics" on page 13-13.

- **Record data in Simulation Data Inspector** — Use this check box if you want to stream the logged simulation data to Simulation Data Inspector, to be able to view it during simulation. By default, this check box is not selected. For more information on how to configure and use the Simulation Data Inspector, see "Analyze Simulation Results".

- **Open viewer after simulation** — Select this check box if you want to open Simscape Results Explorer, which is an interactive tool that lets you navigate and plot the simulation data logging results. By default, this check box is not selected. For more information, see "About the Simscape Results Explorer" on page 13-24.

  If the **Record data in Simulation Data Inspector** check box is also selected, then the Simulation Data Inspector opens instead of the Simscape Results Explorer.

- **Workspace variable name** — Specifies the name of the workspace variable that stores the simulation data. Subsequent simulations overwrite the data in the simulation log variable. If you want to compare data from two models or two simulation runs, use different names for the respective log variables. The default variable name is `simlog`.

- **Decimation** — Use this parameter to limit the number of data points saved, by outputting data points for every $n$th time step, where $n$ is the decimation factor. The default is 1, which means that all points are logged. Specifying a different value results in the first step, and every $n$th step thereafter, being logged. For example, specifying 2 logs data points for every other time step, while specifying 10 logs data points for just one in ten steps.

- **Limit data points** — Use this check box in conjunction with the **Data history (last N steps)** parameter to limit the number of data points saved. The check box is selected by default. If you clear it, the simulation log variable contains the data points for the whole simulation, at the price of slower simulation speed and heavier memory consumption.

- **Data history (last N steps)** — Specify the number of simulation steps to limit the number of data points output to the workspace. The simulation log variable contains the data points corresponding to the last N steps of the simulation, where N is the value that you specify for the **Data history (last N steps)** parameter. You have to select the **Limit data points** check box to make this parameter available. The default value logs simulation data for the last 5000 steps. You can specify any other positive integer number. If the simulation contains fewer steps than the number specified, the simulation log variable contains the data points for the whole simulation.

Saving data to the workspace can slow down the simulation and consume memory. To avoid this, you can use either the **Decimation** parameter, or **Limit data points** in conjunction with **Data history (last N steps)**, or both methods, to limit the number of data points saved. The two methods work independently from each other and can be used separately or together. For example, if you specify a decimation factor of 2 and keep the default value of 5000 for the **Data history (last N steps)** parameter, your workspace variable will contain downsampled data from the last 10,000 time steps in the simulation.

Another way to reduce memory consumption is to enable data streaming to disk, as described in "Stream Logging Data to Disk" on page 13-42.

**Note** The **Output options** parameter, under **Additional parameters** on the **Data Import/Export** pane of the Configuration Parameters dialog box, also affects which data points are logged. For more information, see "Model Configuration Parameters: Data Import/Export".

After changing your data logging preferences, rerun the simulation to generate a new data log.

## See Also

## Related Examples
- "Enable Data Logging for the Whole Model" on page 13-4
- "Log Data for Selected Blocks Only" on page 13-5

## More About
- "About Simulation Data Logging" on page 13-2

# Log and Plot Simulation Data

This example shows how you can log and plot simulation data instead of adding sensors to your model. This example uses command-line functions, suited for scripting and batch processing. For an interactive example that uses the Simscape Results Explorer to plot logged simulation data, see "Log, Navigate, and Plot Simulation Data" on page 13-20.

The model shown represents a permanent magnet DC motor.
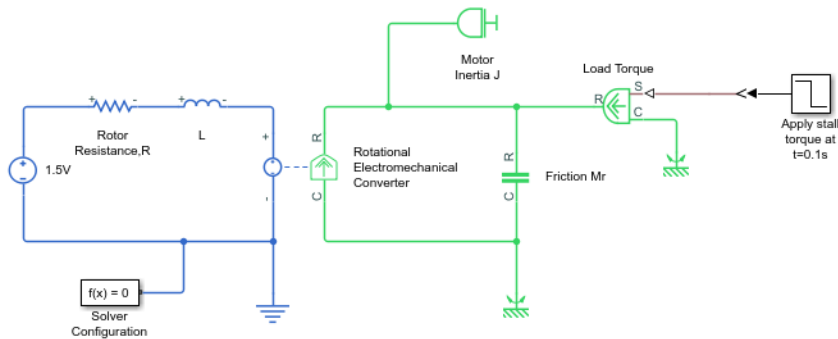


This model is very similar to the "Permanent Magnet DC Motor" on page 20-36 example, but, unlike the example model, it does not include the Sensing unit w (Ideal Rotational Motion Sensor and PS-Simulink Converter block) along with the Motor RPM scope. For a detailed description of the Permanent Magnet DC Motor example, see "Evaluating Performance of a DC Motor".

1   Build the model, as shown in the preceding illustration.

2   In the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

3   In the Configuration Parameters dialog box, in the left pane, select **Data Import/Export**. Clear the **Single simulation output** check box, which is selected by default. This step enables associating the logged simulation data with a separate workspace variable, instead of it being part of the single output object.

4   To enable data logging, in the Configuration Parameters dialog box, in the left pane, select **Simscape**, then set the **Log simulation data** parameter to `All` and click **OK**.
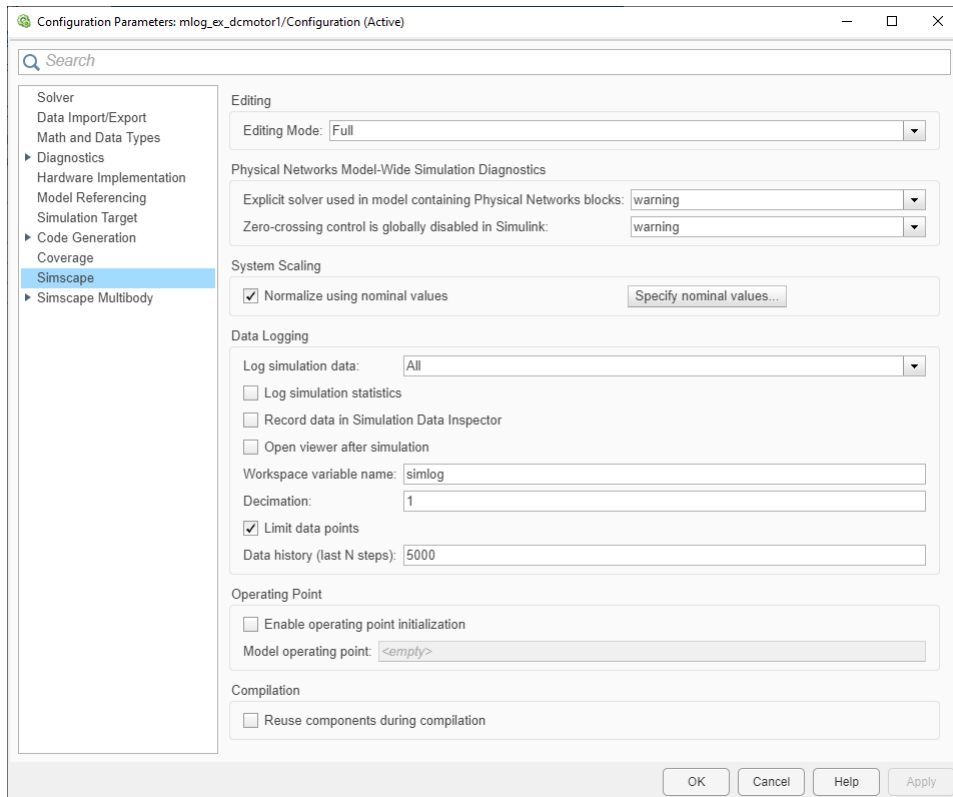
**5** Simulate the model. This creates a workspace variable named `simlog` (as specified by the **Workspace variable name** parameter), which contains the simulation data.

**6** The `simlog` variable has the same hierarchy as the model. To see the whole variable structure, at the command prompt, type:

```
print(simlog)
```

This command prints the whole data tree.

```
mlog_ex_dcmotor1
+-Electrical_Reference2
| +-V
|    +-v
+-Friction_Mr
| +-C
| | +-w
| +-R
| | +-w
| +-t
| +-w
+-L
| +-i
| +-i_L
| +-n
| | +-v
| +-p
| | +-v
| +-v
+-Load_Torque
```

```
                      | +-C
                      | | +-w
                      | +-R
                      | | +-w
                      | +-S
                      | +-t
                      | +-w
                      +-Mechanical_Rotational_Reference
                      | +-W
                      |    +-w
                      +-Mechanical_Rotational_Reference1
                      | +-W
                      |    +-w
                      +-Motor_Inertia_J
                      | +-I
                      | | +-w
                      | +-t
                      | +-w
                      +-Rotational_Electromechanical_Converter
                      | +-C
                      | | +-w
                      | +-R
                      | | +-w
                      | +-i
                      | +-n
                      | | +-v
                      | +-p
                      | | +-v
                      | +-t
                      | +-v
                      | +-w
                      +-Rotor_ResistanceR
                      | +-i
                      | +-n
                      | | +-v
                      | +-p
                      | | +-v
                      | +-v
                      +-x1_5V
                        +-i
                        +-n
                        | +-v
                        +-p
                        | +-v
                        +-v
```

**7** Every node that represents an Across, Through, or internal block variable contains series data. To get to the series, you have to specify the complete path to it through the tree, starting with the top-level variable name. For example, to get a handle on the series representing the angular velocity of the motor, type:

```
s1 = simlog.Rotational_Electromechanical_Converter.R.w.series;
```

From here, you can access the values and time vectors for the series and analyze them.

**8** You do not have to isolate series data to plot its values against time, or against another series. For example, to see how the motor speed (in revolutions per minute) changes with time, type:

```
plot(simlog.Rotational_Electromechanical_Converter.R.w,'units','rpm')
```
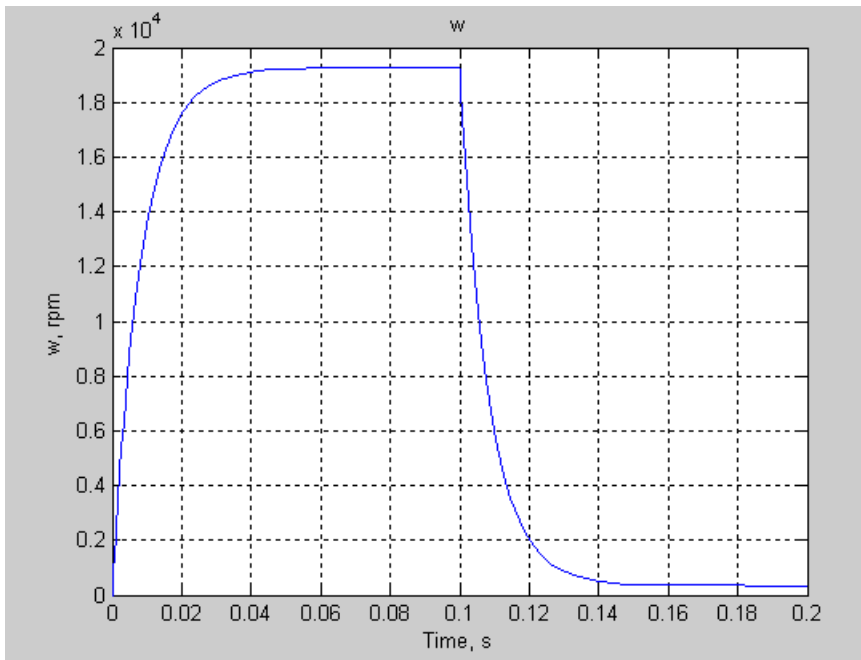
9   Compare this figure to the RPM scope display in the "Permanent Magnet DC Motor" on page 20-36 example. The results are exactly the same.

10  To plot the motor torque against its angular velocity, in `rpm`, and add descriptive axis names, type:

```
plotxy(simlog.Rotational_Electromechanical_Converter.R.w,simlog.Motor_Inertia_J.t,...
    'xunit','rpm','xname','Angular velocity','yname','Torque')
```
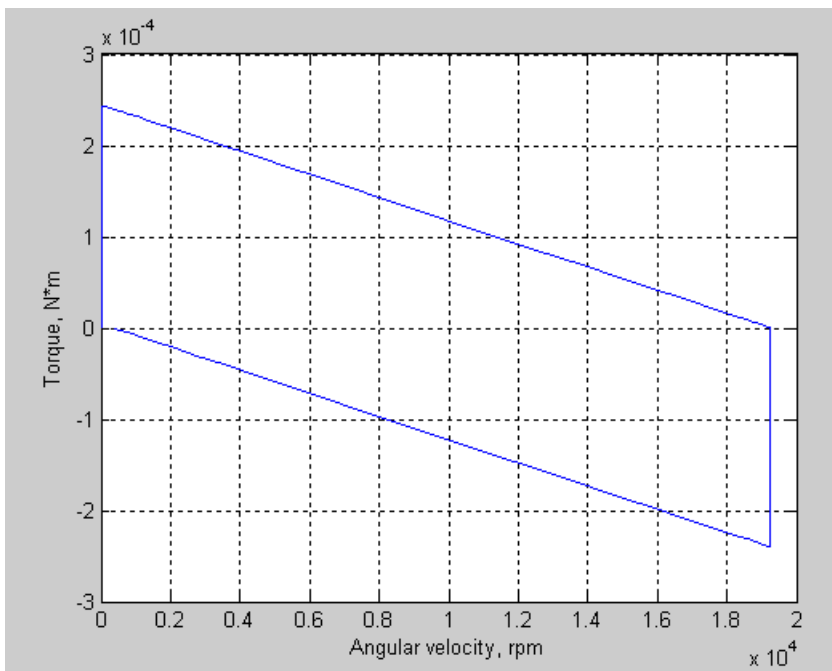
For more information on plotting logged simulation data, see the `simscape.logging.plot` and `simscape.logging.plotxy` reference pages.

# Log Simulation Statistics

This example shows how you can access and analyze information on zero crossings during simulation. By default, the zero-crossing data is not logged. If you select the **Log simulation statistics** check box, the simulation log variable contains an additional `SimulationStatistics` node for each block that can produce zero crossings, at the price of slower simulation speed and heavier memory consumption.

1   Open the Mechanical System with Translational Hard Stop example model by typing `ssc_mechanical_system_translational_hardstop` in the MATLAB Command Window.
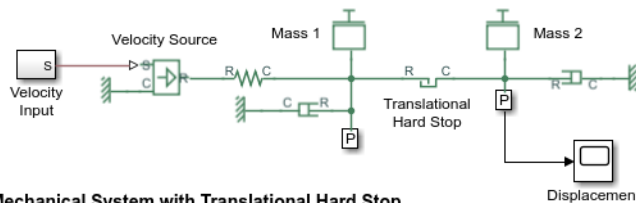


**Mechanical System with Translational Hard Stop**
1. Plot displacement of Mass 1 and Mass 2 (see code)
2. Plot hysteresis curve of mass displacements (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

2   Open the Configuration Parameters dialog box and then, in the left pane, select **Simscape**. You can see that this example model already has data logging for the whole model enabled, as well as simulation statistics, and that the workspace variable name is `simlog_ssc_mechanical_system_translational_hardstop`.
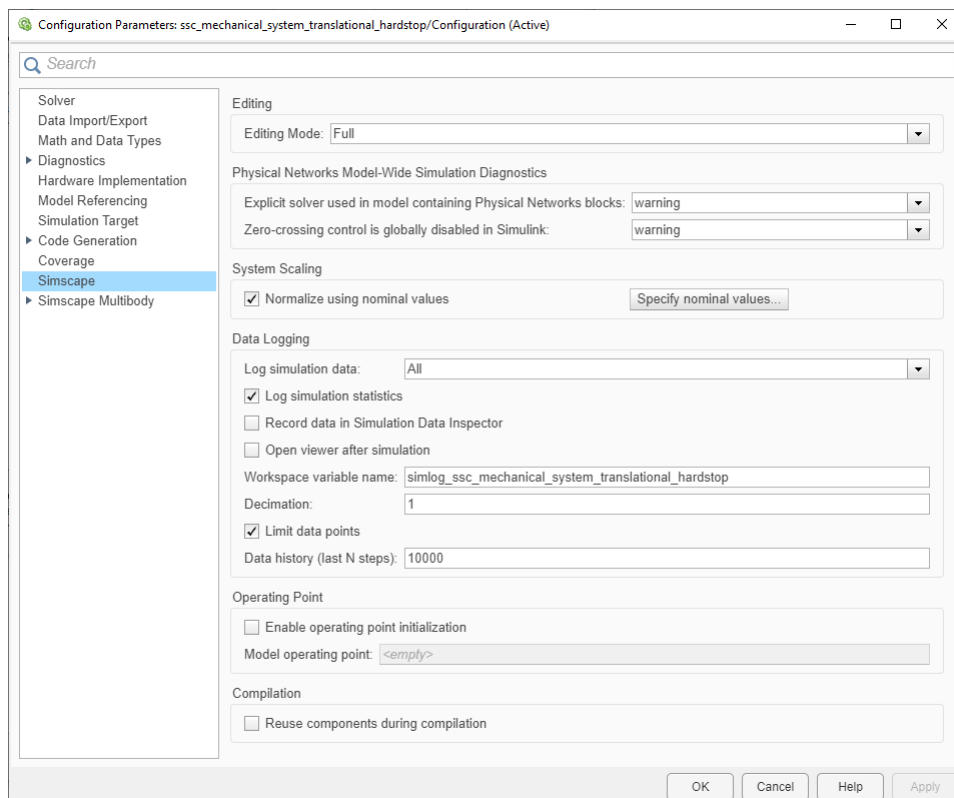
**3** Simulate the model. This creates a workspace variable named `simlog_ssc_mechanical_system_translational_hardstop` (as specified by the **Workspace variable name** parameter), which contains the simulation data. Because you selected the **Log simulation statistics** check box, the workspace variable contains additional nodes that represent zero-crossing data.

**4** The `simlog` variable has the same hierarchy as the model. To see the whole variable structure, at the command prompt, type:

`simlog_ssc_mechanical_system_translational_hardstop.print`

This command prints the whole data tree.

```
ssc_mechanical_system_translational_hardstop
+-Damper_M1
| +-C
| | +-v
| +-R
| | +-v
| +-f
| +-v
+-Damper_M2
| +-C
| | +-v
| +-R
| | +-v
| +-f
| +-v
+-MTRef_DM1
| +-V
|   +-v
+-MTRef_DM2
| +-V
|   +-v
+-MTRef_VS
| +-V
|   +-v
+-Mass_1
| +-M
| | +-v
| +-f
| +-v
+-Mass_2
| +-M
| | +-v
| +-f
| +-v
+-Sensor_M1
| +-Ideal_Translational_Motion_Sensor
| | +-C
| | | +-v
| | +-P
| | +-R
| | | +-v
| | +-V
| | +-f
| | +-v
| | +-x
```

```
        | +-MTRef
        | | +-V
        | |    +-v
        | +-PS_Terminator
        | | +-I
        | +-PS_Terminator1
        |    +-I
+-Sensor_M2
        | +-Ideal_Translational_Motion_Sensor
        | | +-C
        | | | +-v
        | | +-P
        | | +-R
        | | | +-v
        | | +-V
        | | +-f
        | | +-v
        | | +-x
        | +-MTRef
        |    +-V
        |       +-v
+-Spring_M1
        | +-C
        | | +-v
        | +-R
        | | +-v
        | +-f
        | +-v
        | +-x
+-Translational_Hard_Stop
        | +-C
        | | +-v
        | +-R
        | | +-v
        | +-SimulationStatistics
        | | +-zc_1
        | | | +-crossings
        | | | +-values
        | | +-zc_2
        | |    +-crossings
        | |    +-values
        | +-f
        | +-v
        | +-x
+-Velocity_Source
          +-C
          | +-v
          +-R
          | +-v
          +-S
          +-f
          +-v
```

**5** Under the `Translational_Hard_Stop` node, there is a node called `SimulationStatistics`, which contains zero-crossing information. This means that Translational Hard Stop is the only block in the model that can generate zero-crossings during simulation.

**6** You can access and analyze this data similar to other data that is logged to workspace during simulation. For more information, see `simscape.logging.Node` and `simscape.logging.Series` reference pages.

# Log and View Simulation Data for Selected Blocks

This example shows how you can set your model to log simulation data for selected blocks only and how to view simulation data using Simscape Results Explorer.

1 Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. Double-click the DC Motor subsystem to open it.
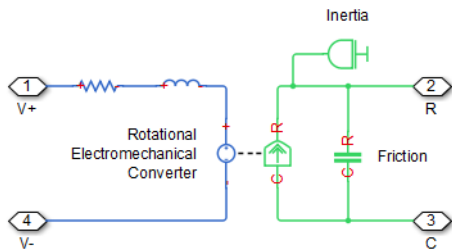


2 Open the Configuration Parameters dialog box and then, in the left pane, select **Simscape**. This example model has data logging for the whole model enabled. To enable data logging on a block-by-block basis, set the **Log simulation data** parameter to `Use local settings` and click **OK**.
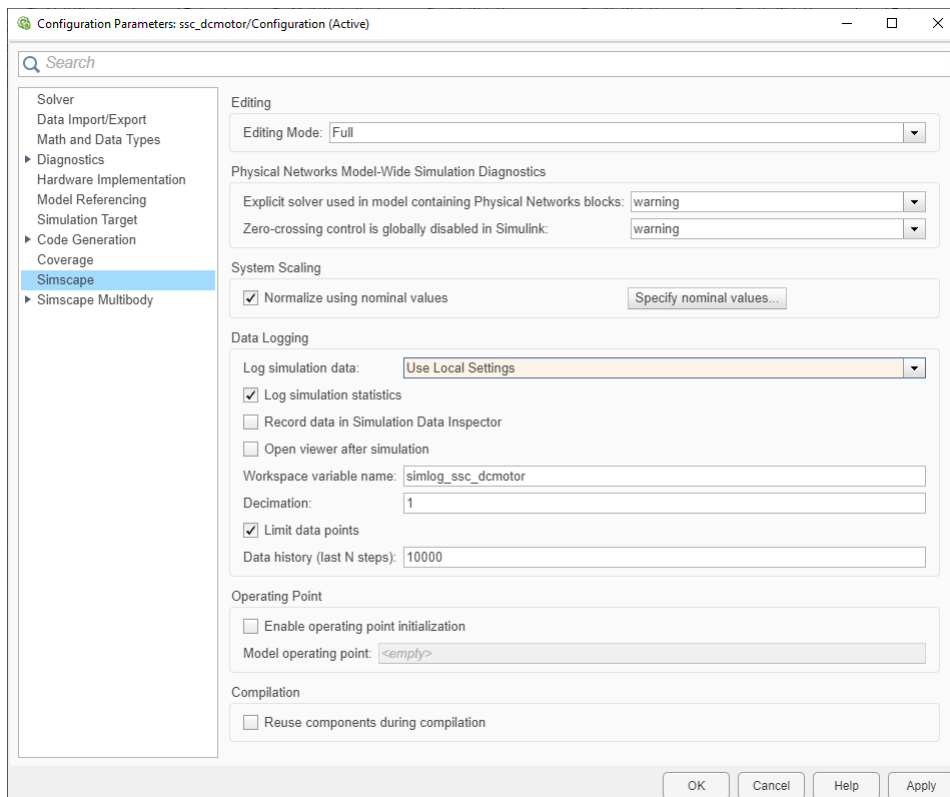


3 Select the blocks for data logging. Right-click the Rotational Electromechanical Converter block. From the context menu, select **Simscape** > **Log simulation data** .
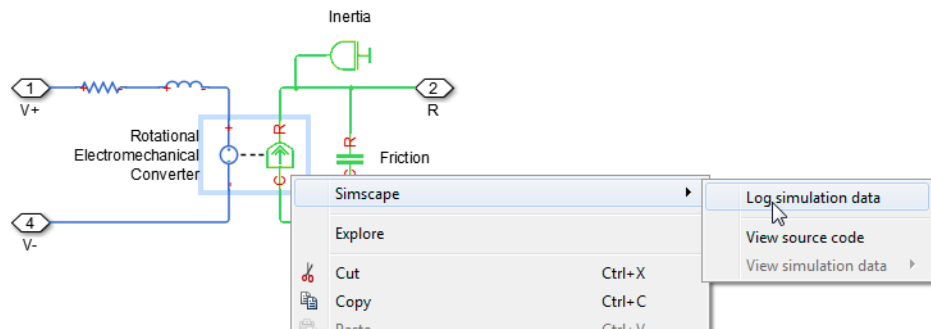
After you select a block for data logging, a check mark appears in front of the **Log simulation data** option in the context menu for that block.

**4** Right-click the Inertia block and select it for data logging, as described in the previous step.

**5** Simulate the model. This creates a workspace variable named `simlog_ssc_dcmotor` (as specified by the **Workspace variable name** parameter), which contains the simulation data for selected blocks only.

**6** To open the Simscape Results Explorer, right-click one of the blocks previously selected for data logging, for example, the Rotational Electromechanical Converter block. From the context menu, select **Simscape > View simulation data > simlog_ssc_dcmotor**.
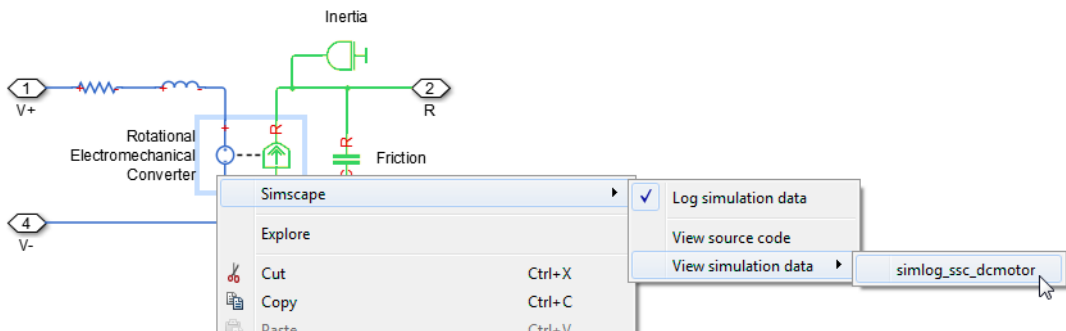


**Note** If you right-click a block that has not been selected for simulation data logging, for example, the Load Torque block, the **View simulation data** option is not available.

If you change the name of the log variable between simulation runs, the context menu lists the names of all the log variables associated with the block. For example, to compare data from two simulation runs, you can use different variable names (such as `simlog1` and `simlog2`). Open a Simscape Results Explorer window with `simlog1` results, then unlink it from the session and open another window with `simlog2` results. For more information, see "About the Simscape Results Explorer" on page 13-24.

The Simscape Results Explorer window opens, with the `Rotational_Electromechanical_Converter` node already selected and expanded in the left pane, and all the node plots for this block displayed in the right pane. You can see that it contains simulation data only for the two selected blocks, Rotational Electromechanical Converter and Inertia.
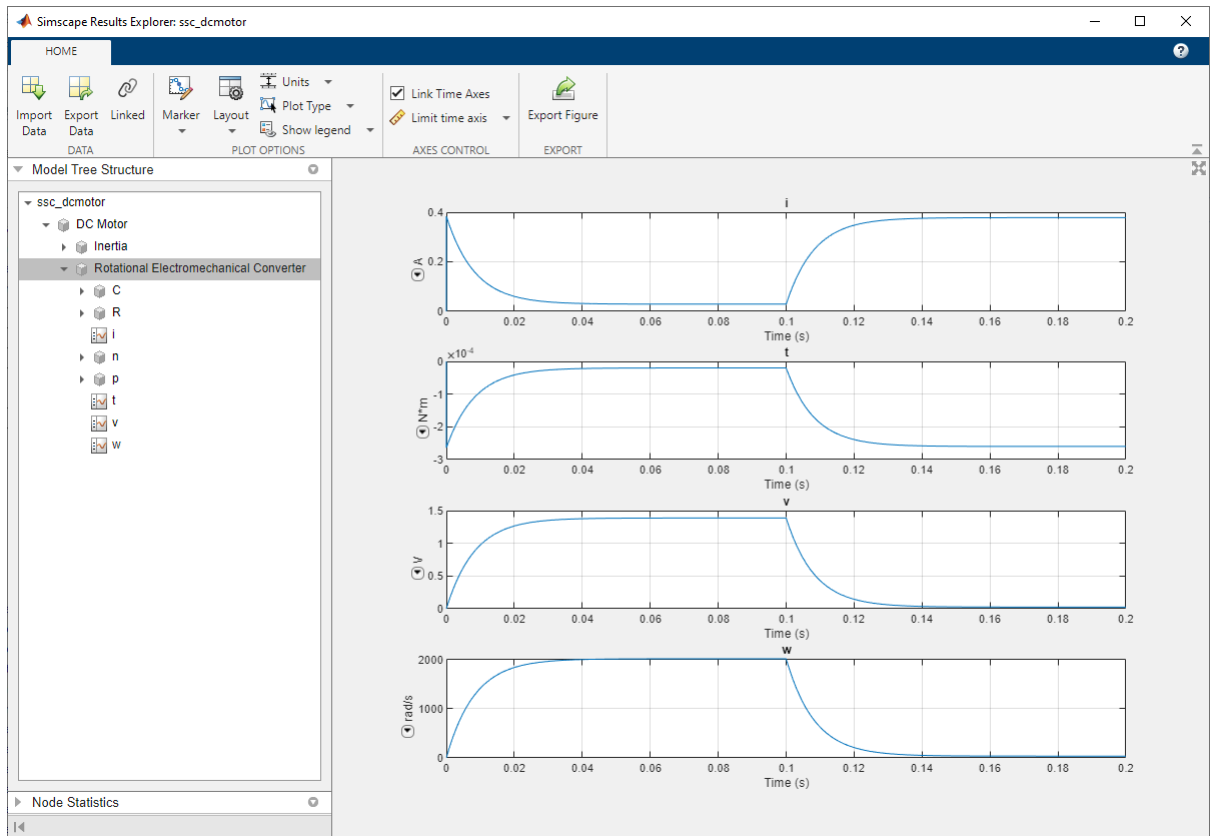
## See Also

## Related Examples

- "Log, Navigate, and Plot Simulation Data" on page 13-20
- "Plot Simulation Data in Different Units" on page 13-32

## More About

- "About Simulation Data Logging" on page 13-2
- "About the Simscape Results Explorer" on page 13-24

# Log, Navigate, and Plot Simulation Data

This example shows the basic workflow for logging simulation data for the whole model and then navigating and plotting the logged data using Simscape Results Explorer.

1 Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window.
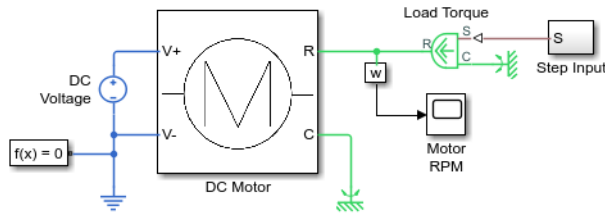


2 Open the Configuration Parameters dialog box and then, in the left pane, select **Simscape**. You can see that this example model already has data logging for the whole model enabled, as well as simulation statistics, and that the workspace variable name is `simlog_ssc_dcmotor`. Select the **Open viewer after simulation** check box and click **OK**.
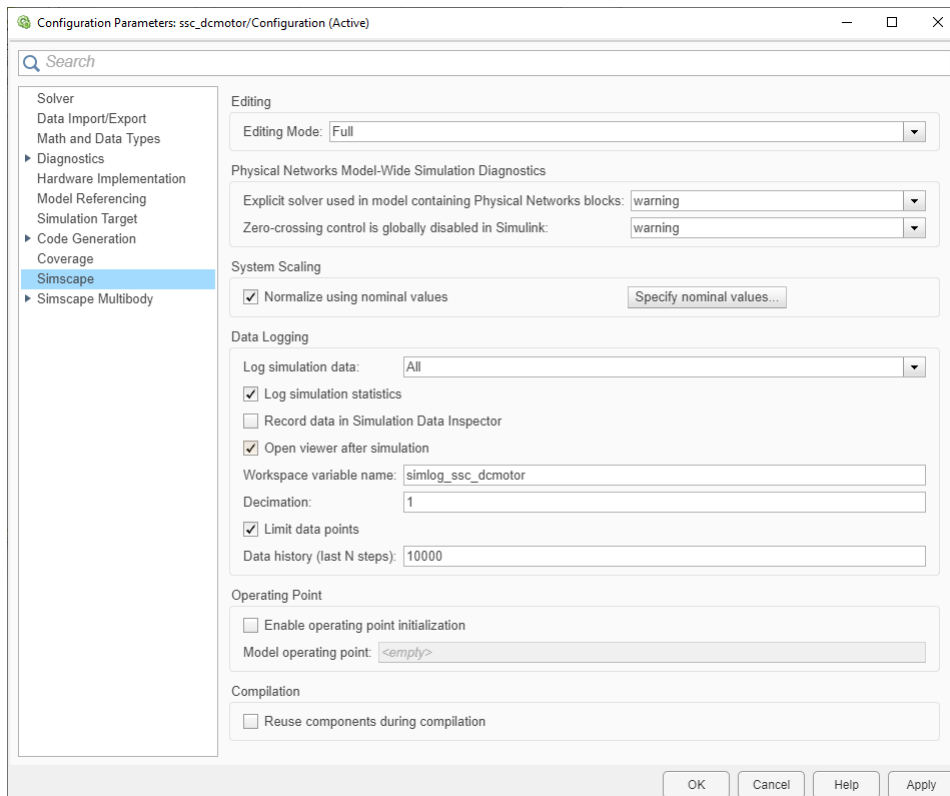


3 Simulate the model. When the simulation is done, the Simscape Results Explorer window opens. In the left pane, it contains the simulation log tree hierarchy, which corresponds to the model hierarchy.
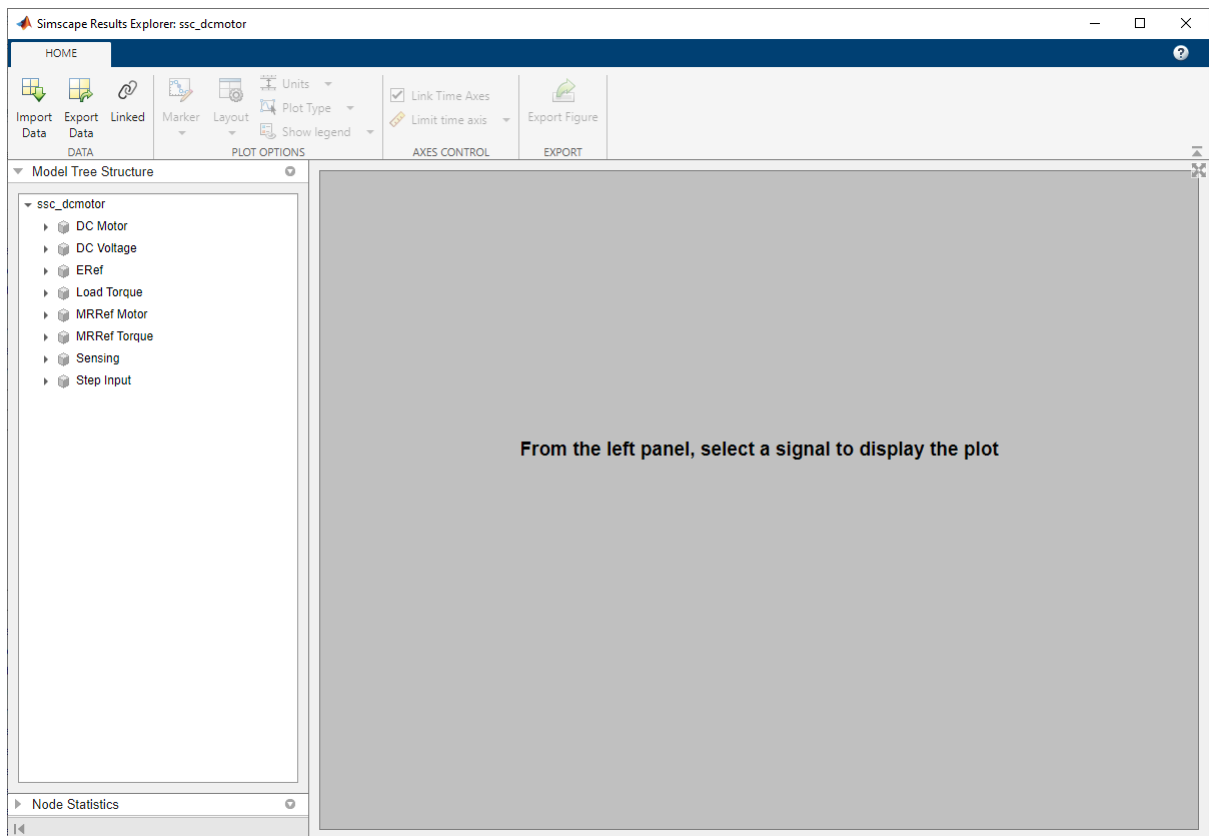
**4**   When you click a node in the left pane, the corresponding plots appear in the right pane. Expand the `DC_Motor` node, and then click the `Rotational_Electromechanical_Converter` node to see all the node plots for this block.
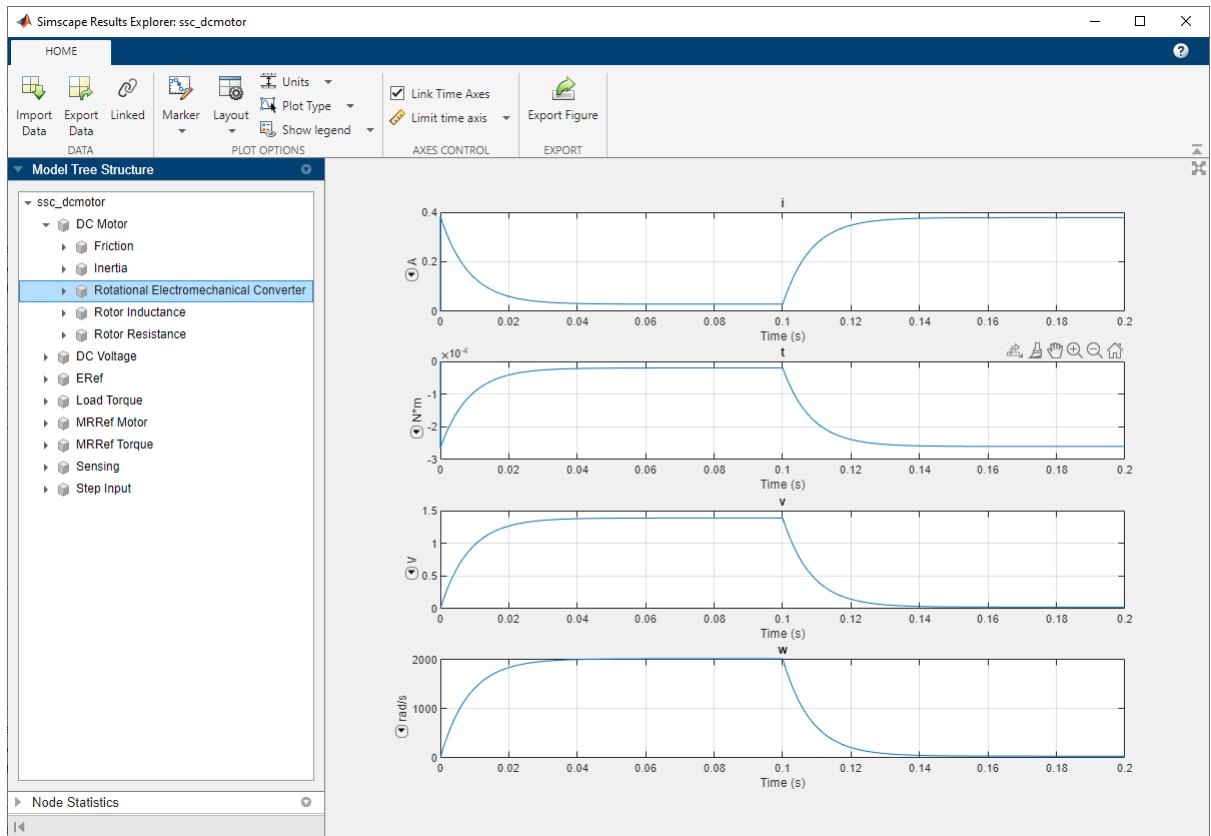
**5** To isolate the plot of the rotor angular velocity series against time, keep expanding the nodes in the left pane until you get to the series data.
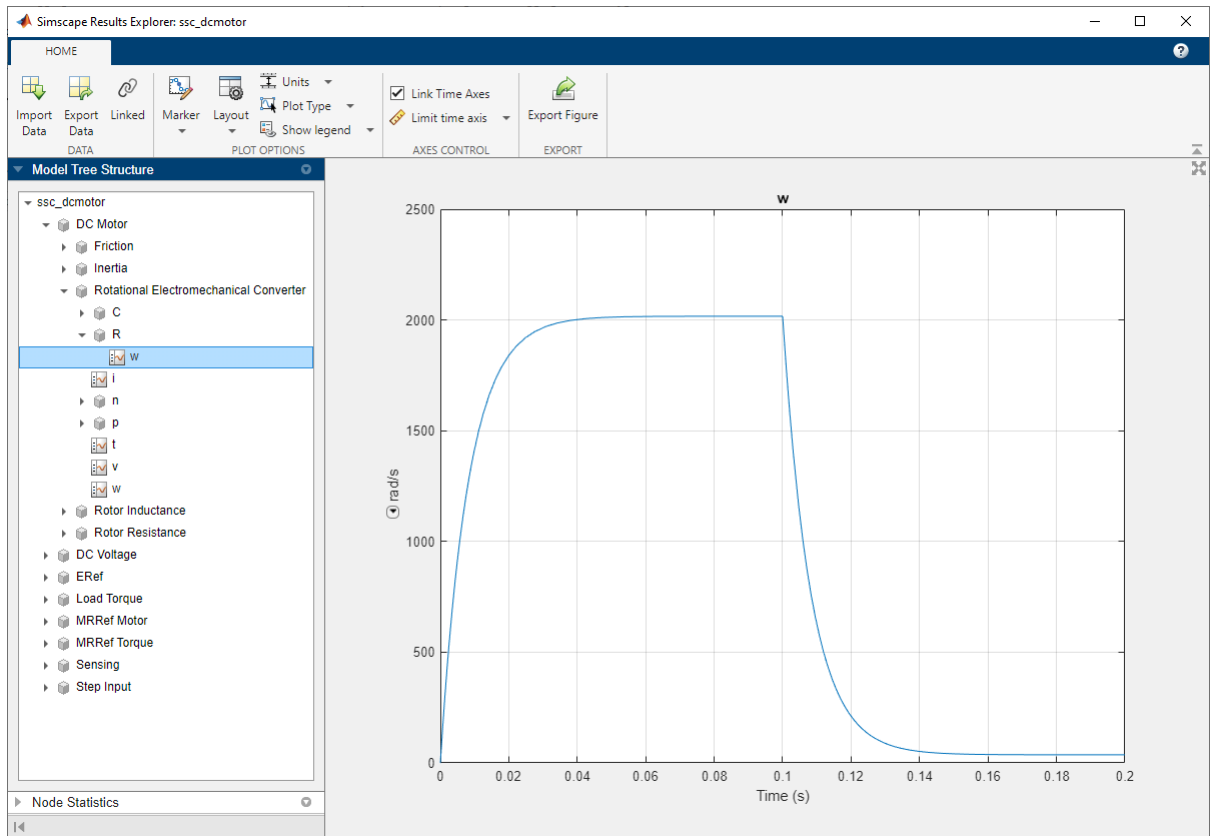
## See Also

sscexplore

## Related Examples

- "Log and View Simulation Data for Selected Blocks" on page 13-17
- "Plot Simulation Data in Different Units" on page 13-32

## More About

- "About Simulation Data Logging" on page 13-2
- "About the Simscape Results Explorer" on page 13-24

# About the Simscape Results Explorer

| In this section... |
| --- |
| "Selecting Nodes to Plot Data" on page 13-24 |
| "Link to MATLAB Session" on page 13-28 |
| "Link to Block Diagram" on page 13-28 |
| "Data Logging for Component Arrays" on page 13-30 |

Simscape Results Explorer is an interactive tool that lets you navigate and plot the simulation data logging results.

When you configure the model to log simulation data (for the whole model or just the selected blocks), you can make the Simscape Results Explorer window open automatically upon completing a simulation run by selecting the **Open viewer after simulation** check box in the Configuration Parameters dialog box. For more information on this workflow, see "Log, Navigate, and Plot Simulation Data" on page 13-20.

Other ways to open the Simscape Results Explorer window are:

- Use the `sscexplore` function.
- In the model window, on the **Simulation** tab, click **Review Results > Results Explorer**.
- Select a Simscape block in the model, and then, on the **Simscape Block** tab, click **Review Results > Results Explorer**.
- Right-click a block and, from the context menu, select **Simscape > View simulation data**. For more information, see "Log and View Simulation Data for Selected Blocks" on page 13-17.

## Selecting Nodes to Plot Data

When you click a node in the left pane of the Simscape Results Explorer, the corresponding plots appear in the right pane:

- Clicking a node that represents a block displays plots of all the variables in this block.
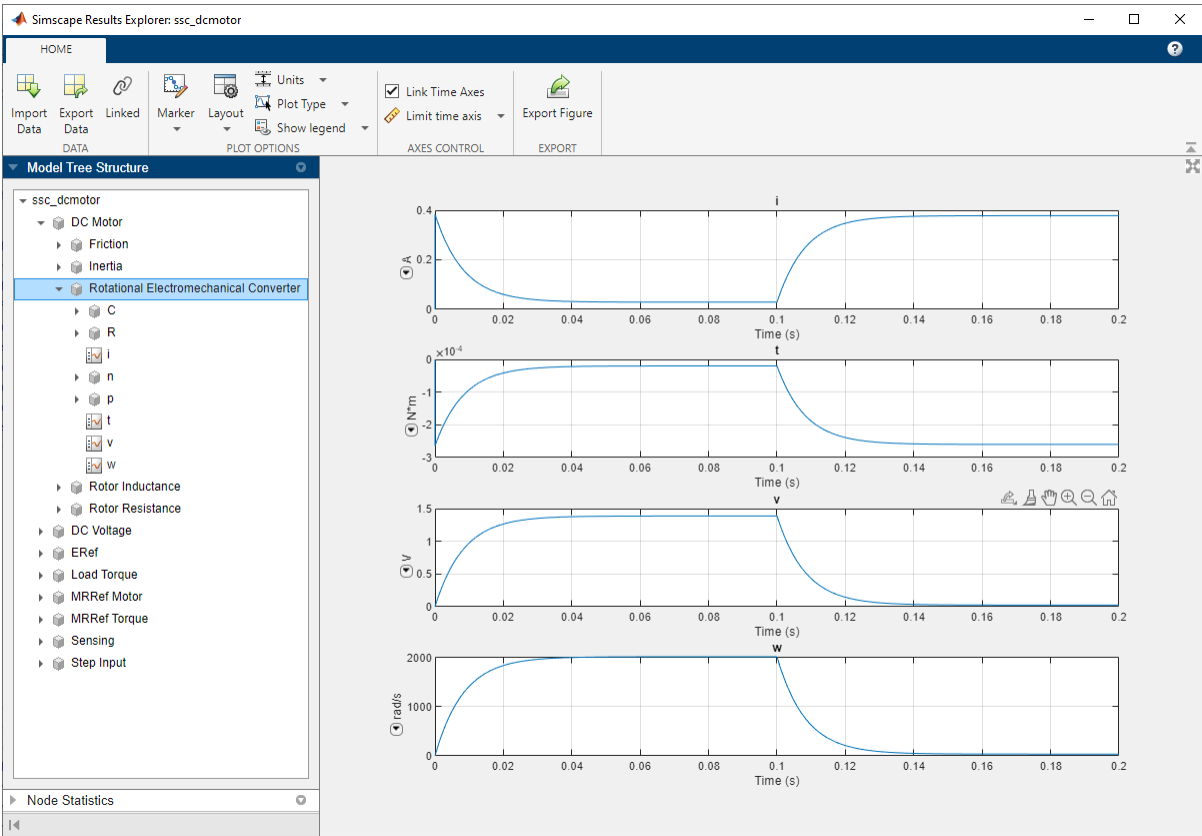
- Clicking a node that represents a variable displays the plot for this variable only.
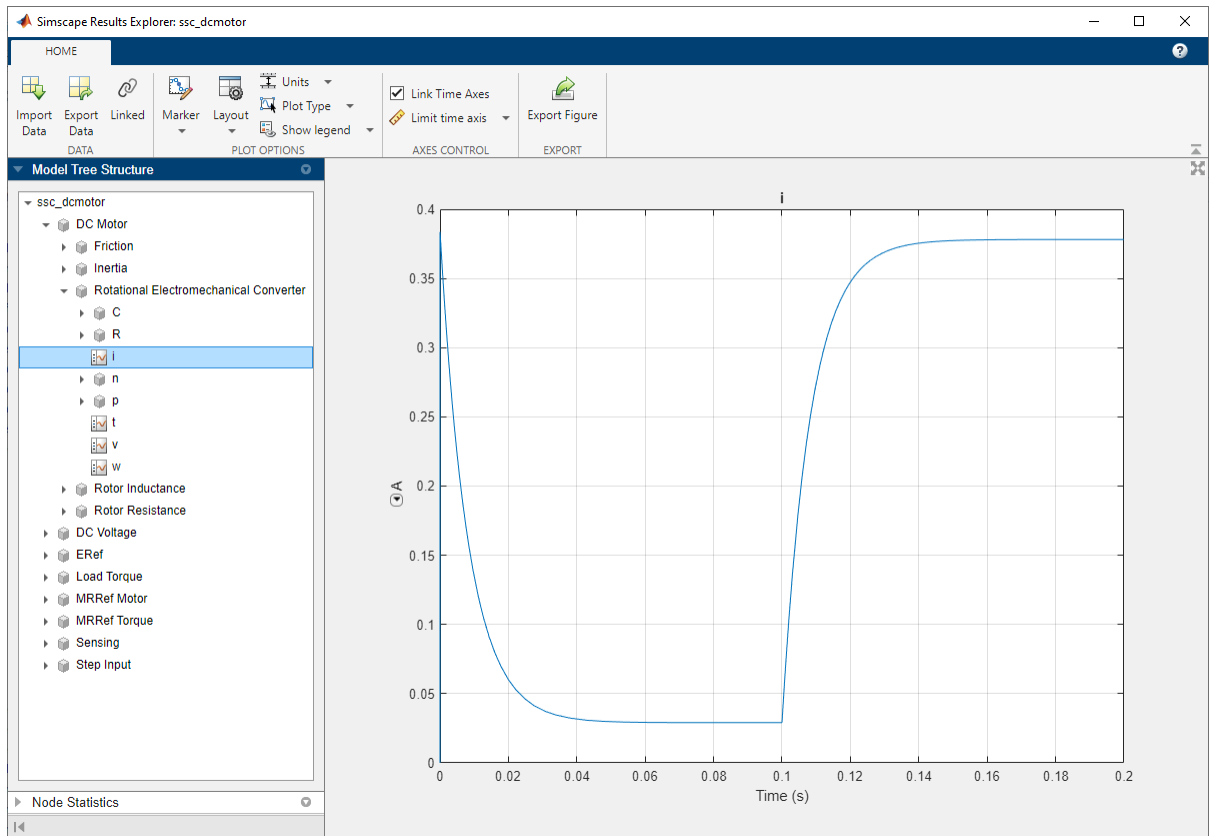
- In frequency-and-time simulation mode, clicking a node that represents a frequency variable displays the plots for this variable's instantaneous value, amplitude, offset, and phase. You can click each of the subnodes to see that data separately. For more information, see "Frequency and Time Simulation Mode" on page 7-55.
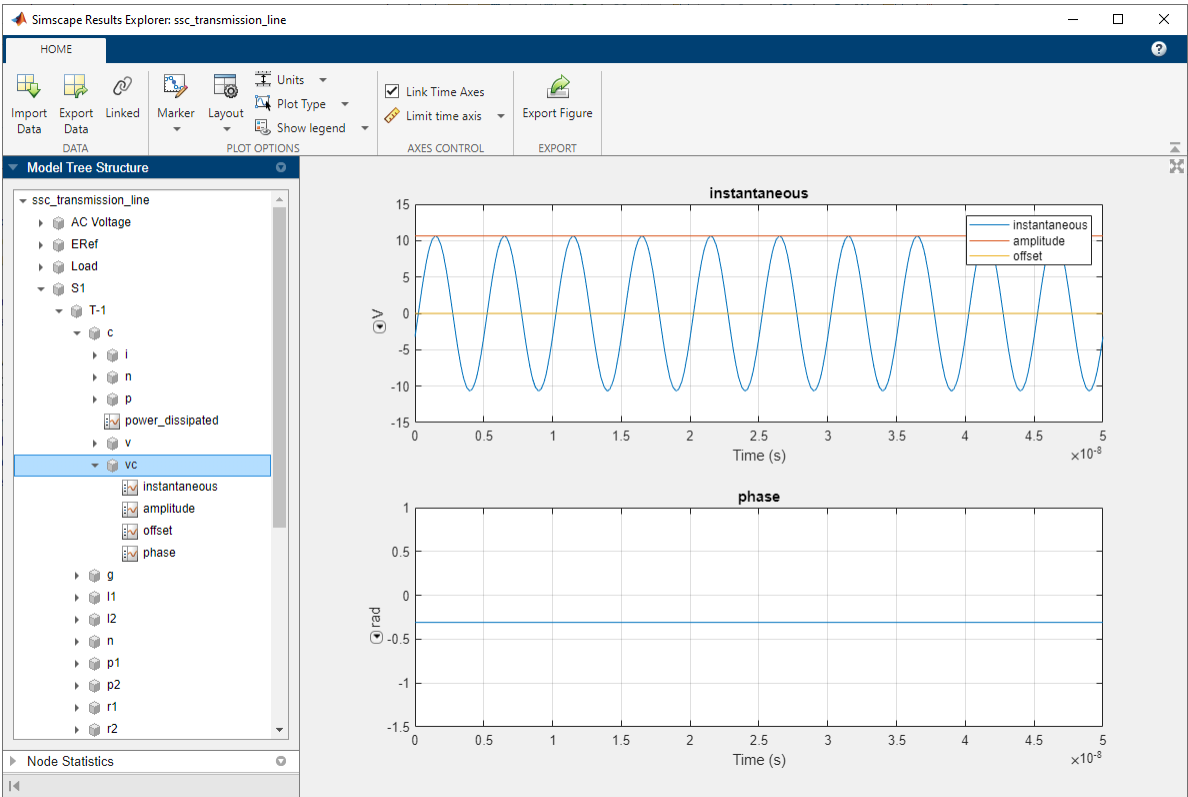
- To select several variables for side-by-side plot comparison, press CTRL and click multiple nodes.
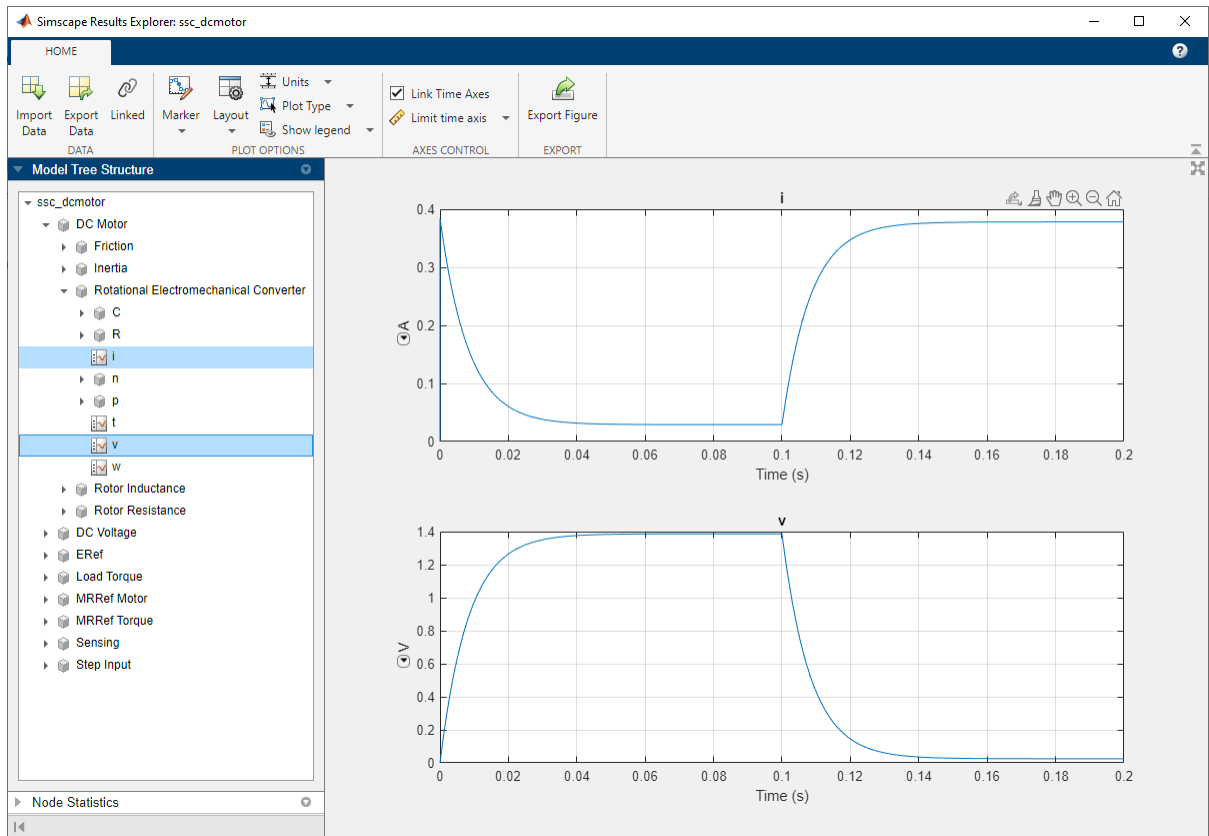
## Link to MATLAB Session

You can control whether the Simscape Results Explorer window is reused when you rerun the simulation, or a new window is opened after the next simulation run, by linking and unlinking the window.

When you first open the Simscape Results Explorer window, it is linked to the current MATLAB session. This means that when you run a new simulation, the results in the window will be overwritten. To retain the current results and open a new window after the next simulation, click the **Linked** button in the Simscape Results Explorer window toolbar. The button changes to **Unlinked**, meaning that this window is now unlinked from simulation. Next time you rerun a simulation or use an option to view the simulation data in the Simscape Results Explorer, a new window opens, and that window is now linked to the session.

Only one window can be linked to the session, so if you have multiple windows open, linking one of them (by clicking its **Unlinked** button) unlinks the previous one.

## Link to Block Diagram

The Simscape Results Explorer tool provides direct linking to the block diagram. These links let you highlight the appropriate block or open the block dialog box, to easily go from a variable listed in the Simscape Results Explorer tree to the **Initial Targets** section in the corresponding block dialog box.
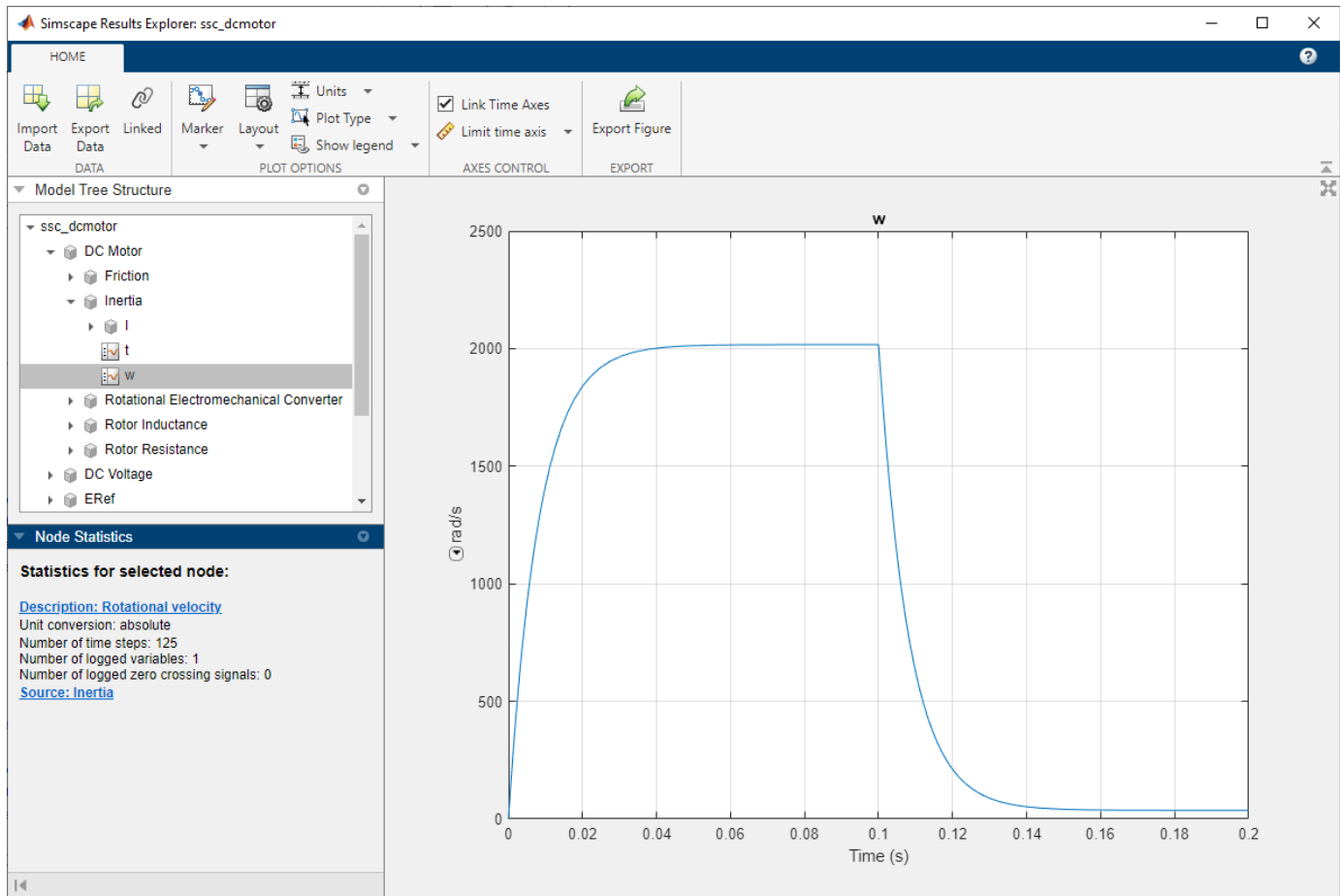
When you select a node in the Simscape Results Explorer tree, the **Node Statistics** panel in the bottom-left corner of the window is collapsed by default. When you expand this panel, it contains the following links:

- **Description** — If the node represents a block or subsystem, displays the block or subsystem name. If the node represents a variable, displays the descriptive variable name, which is the same name that appears in the **Initial Targets** section of the block dialog box. Clicking the link opens the dialog box for the corresponding block.

  For example, in the illustration above, the selected node w represents a variable called `Rotational velocity`. Clicking the **Description** link opens the Inertia block dialog box, which is the parent block for this variable. In the block dialog box, expand the **Initial Targets** section to see the **Rotational velocity** variable.
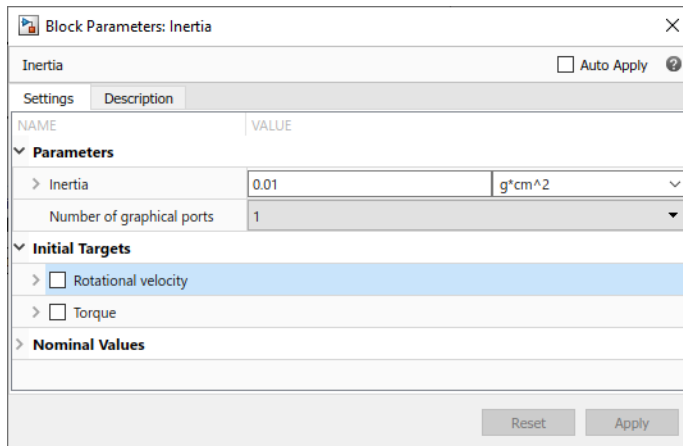
- **Source** — If the node represents a variable, displays the name of the parent block for this variable. Clicking the link highlights the corresponding block in the block diagram, opening the appropriate subsystem if needed.

  In the same example, clicking the **Source** link opens the DC Motor subsystem and highlights the Inertia block, which is the parent block for the selected node w.
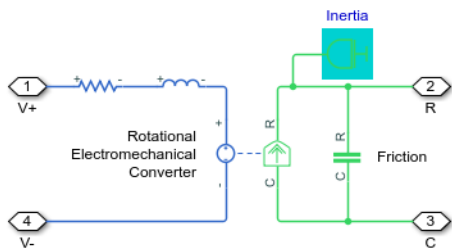


**Tip** If the descriptive name of a variable or block is too long to fit into the status panel, it is truncated with an ellipsis (...). If you hover over the truncated name, the tooltip for the status panel displays the entire descriptive name.

## Data Logging for Component Arrays

If your model contains blocks with underlying arrays of components, the Simscape Results Explorer includes logged simulation data for the array members.

For example, in this model, the Resistor Array block contains an underlying array of resistors.
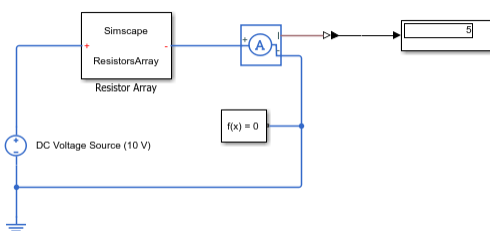
When you simulate this model, the Simscape Results Explorer tree includes nodes for the numbered array members, such as `resistor(1)`, `resistor(2)`, and so on. If the component array size is `1xN`, the members are numbered `comp(1)`, ..., `comp(N)`. If the array size is `NxM`, the members are numbered `comp(1,1)`, `comp(1,2)`, ..., `comp(NxM)`.
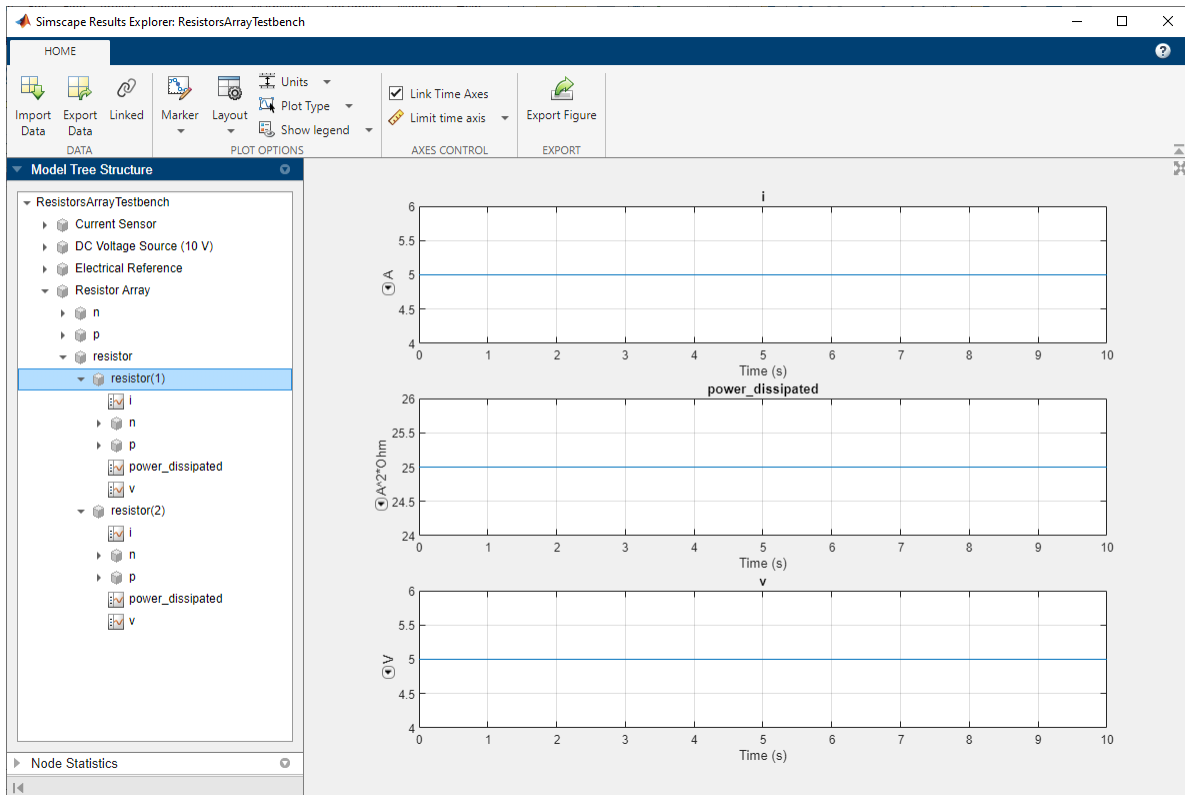


Unlike regular blocks, clicking a node that represents a block with an underlying array of components does not display any plots. Click a node that represents an individual array member to see plots of all of its variables.

## See Also
sscexplore

## Related Examples

- "Log, Navigate, and Plot Simulation Data" on page 13-20
- "Log and View Simulation Data for Selected Blocks" on page 13-17
- "Plot Simulation Data in Different Units" on page 13-32
- "Use Custom Units to Plot Simulation Data" on page 13-36

# Plot Simulation Data in Different Units

When you display logged simulation data in Simscape Results Explorer, the data along the *x*-axis is always time, in seconds. However, you can change the *y*-axis units directly on the plot.

Each of the plots has a drop-down arrow next to the unit name for the *y*-axis. When you click this arrow, a context menu appears containing names of all the units in the unit registry that are commensurate with the current plot unit, as well as two other options:

- `Default` — Use the default unit.
- `Specify` — Type the unit name or expression in a pop-up window and click **OK**. The specified unit name or expression must be commensurate with the current plot unit.

Once you select the option you want, the drop-down menu collapses and the plot is redrawn in specified units.

This example shows how you can plot the data in different units by selecting the unit interactively in the plot pane.

1. Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. This example model has data logging enabled for the whole model, with the **Workspace variable name** parameter set to `simlog_ssc_dcmotor`.
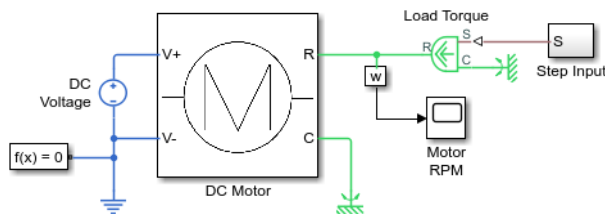


2. Simulate the model to log the simulation data.
3. Open the Simscape Results Explorer window and plot the rotational velocity of the Inertia block:

   `sscexplore(simlog_ssc_dcmotor,'DC_Motor.Inertia.w')`
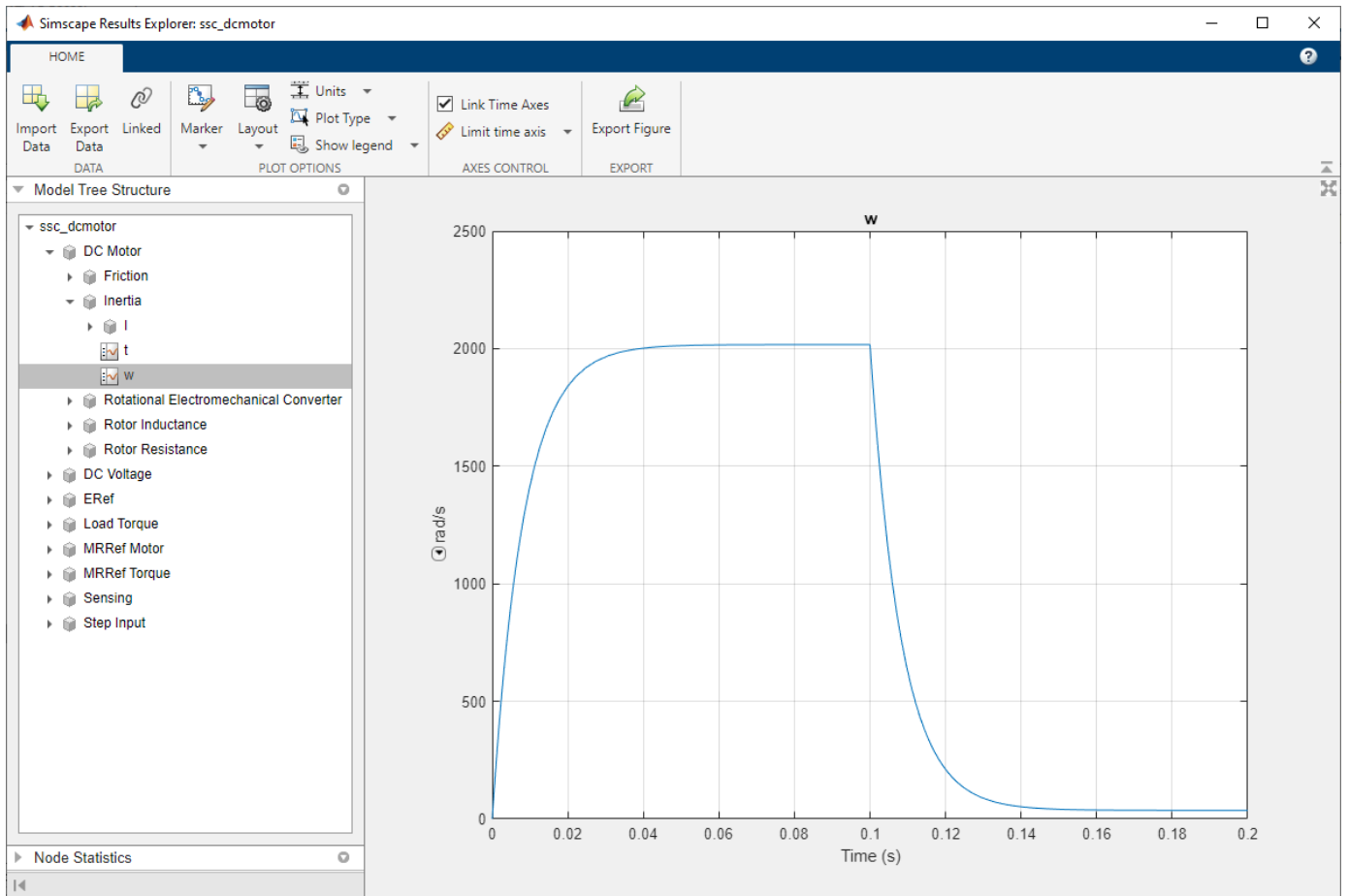
By default, Simscape Results Explorer plots rotational velocity in rad/s.

**4** To plot data in different units, click the arrow under the unit name and then, from the context menu, select `rpm`.
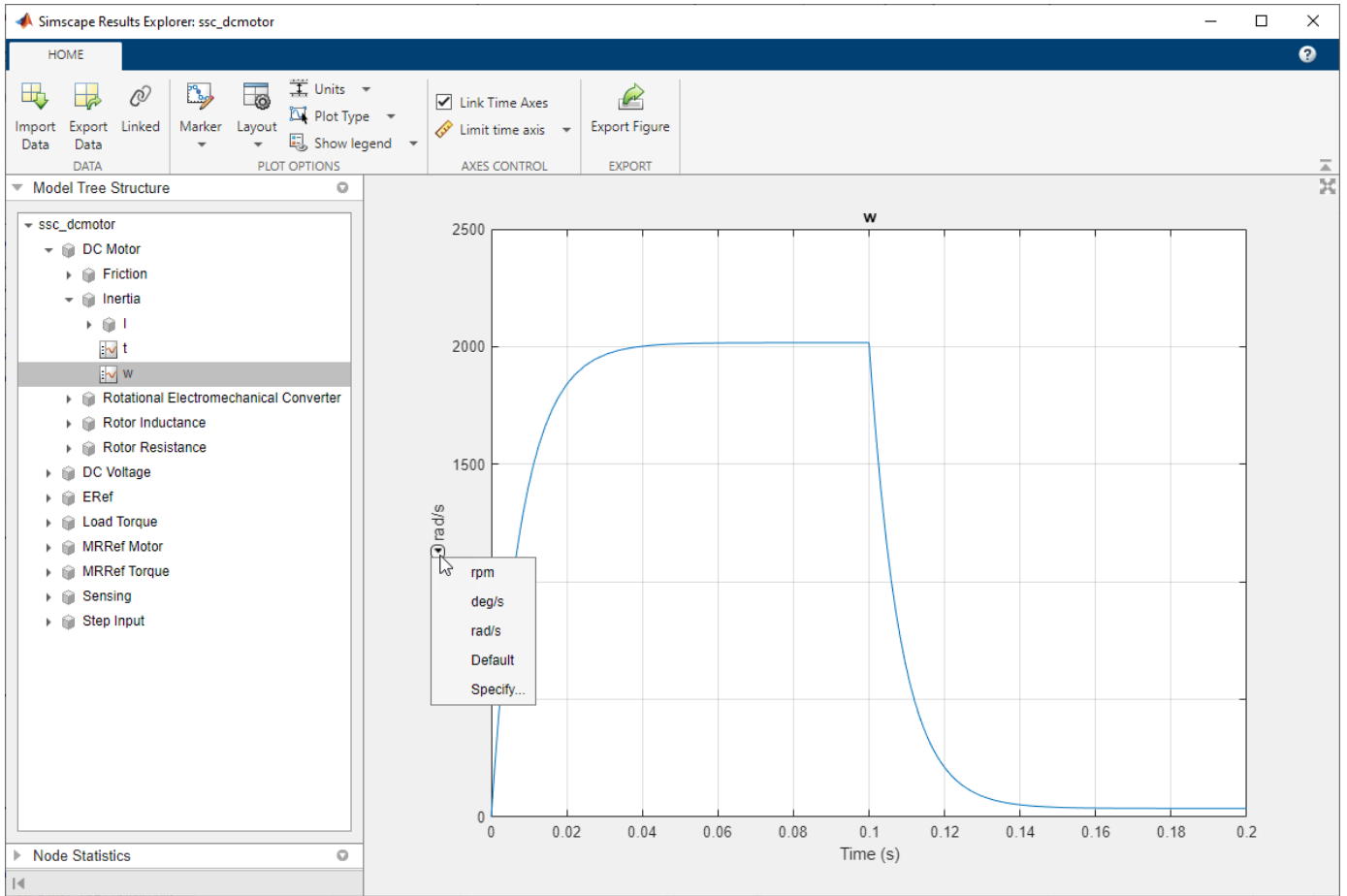
The rotational velocity plot is redrawn in rpm.
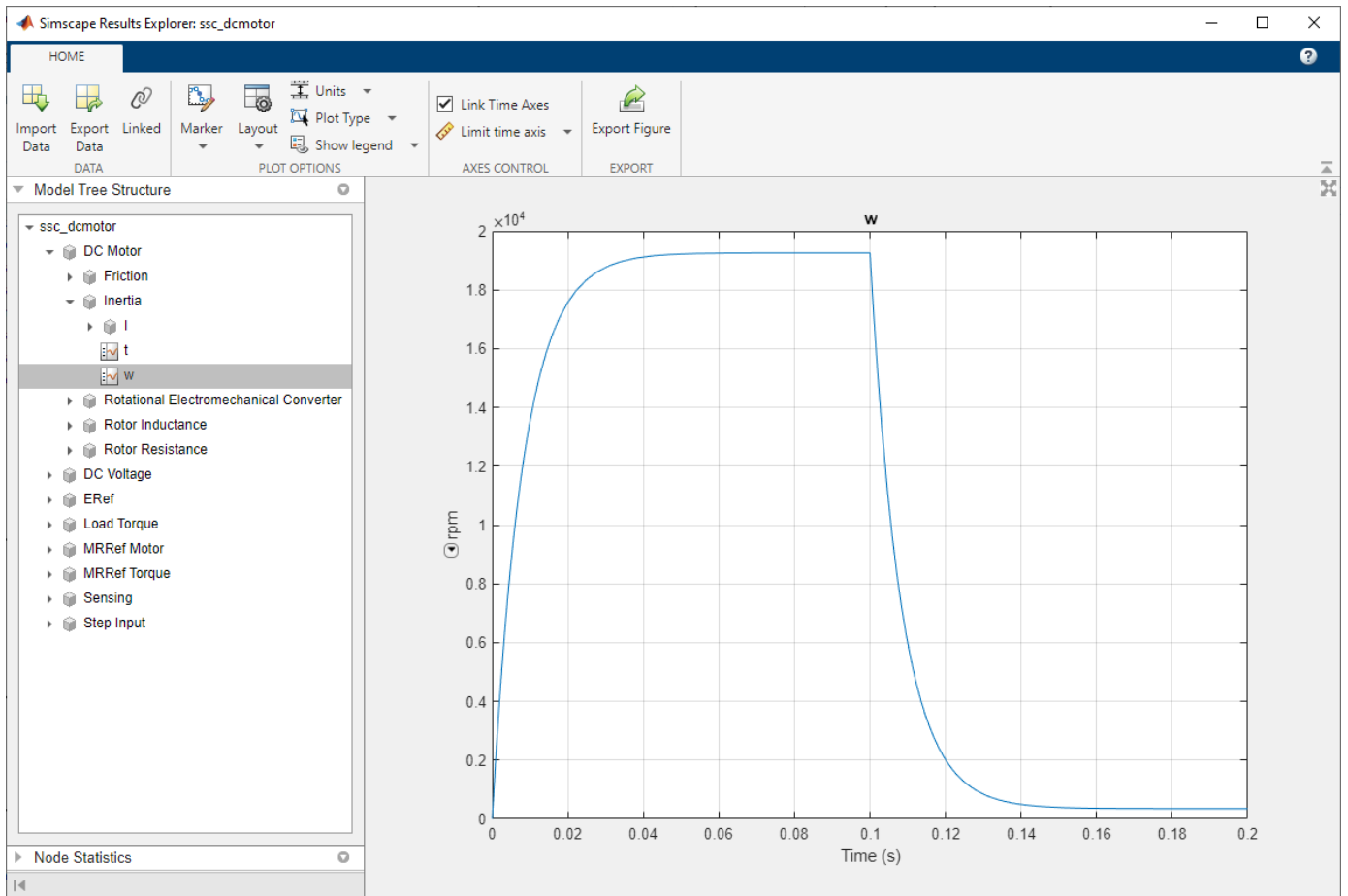
## See Also

### Related Examples

*   "Log, Navigate, and Plot Simulation Data" on page 13-20
*   "Log and View Simulation Data for Selected Blocks" on page 13-17
*   "Use Custom Units to Plot Simulation Data" on page 13-36

### More About

*   "About Simulation Data Logging" on page 13-2
*   "About the Simscape Results Explorer" on page 13-24

# Use Custom Units to Plot Simulation Data

Simscape Results Explorer has a set of default units for plotting the logged data. This example shows how you can change to a custom unit; for example, to plot rotations in degrees rather than radians.

1   Create a file named `ssc_customlogunits.m` and save it anywhere on the MATLAB path. The file should contain a function called `ssc_customlogunits`, which returns a cell array of the units to be used:

```
function customUnits = ssc_customlogunits()
    customUnits = {'deg/s','deg'};
end
```

Include only the units you want to customize. For everything else, Simscape Results Explorer will use the default units.

2   Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. This example model has data logging enabled for the whole model, with the **Workspace variable name** parameter set to `simlog_ssc_dcmotor`.
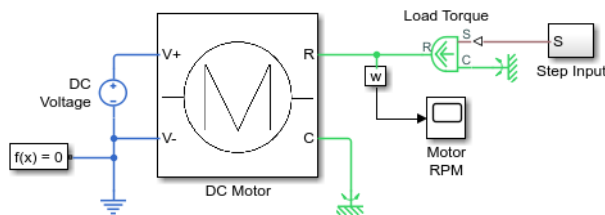


3   Simulate the model to log the simulation data.

4   Open the Simscape Results Explorer window and plot the rotational velocity of the Inertia block:

```
sscexplore(simlog_ssc_dcmotor,'DC_Motor.Inertia.w')
```
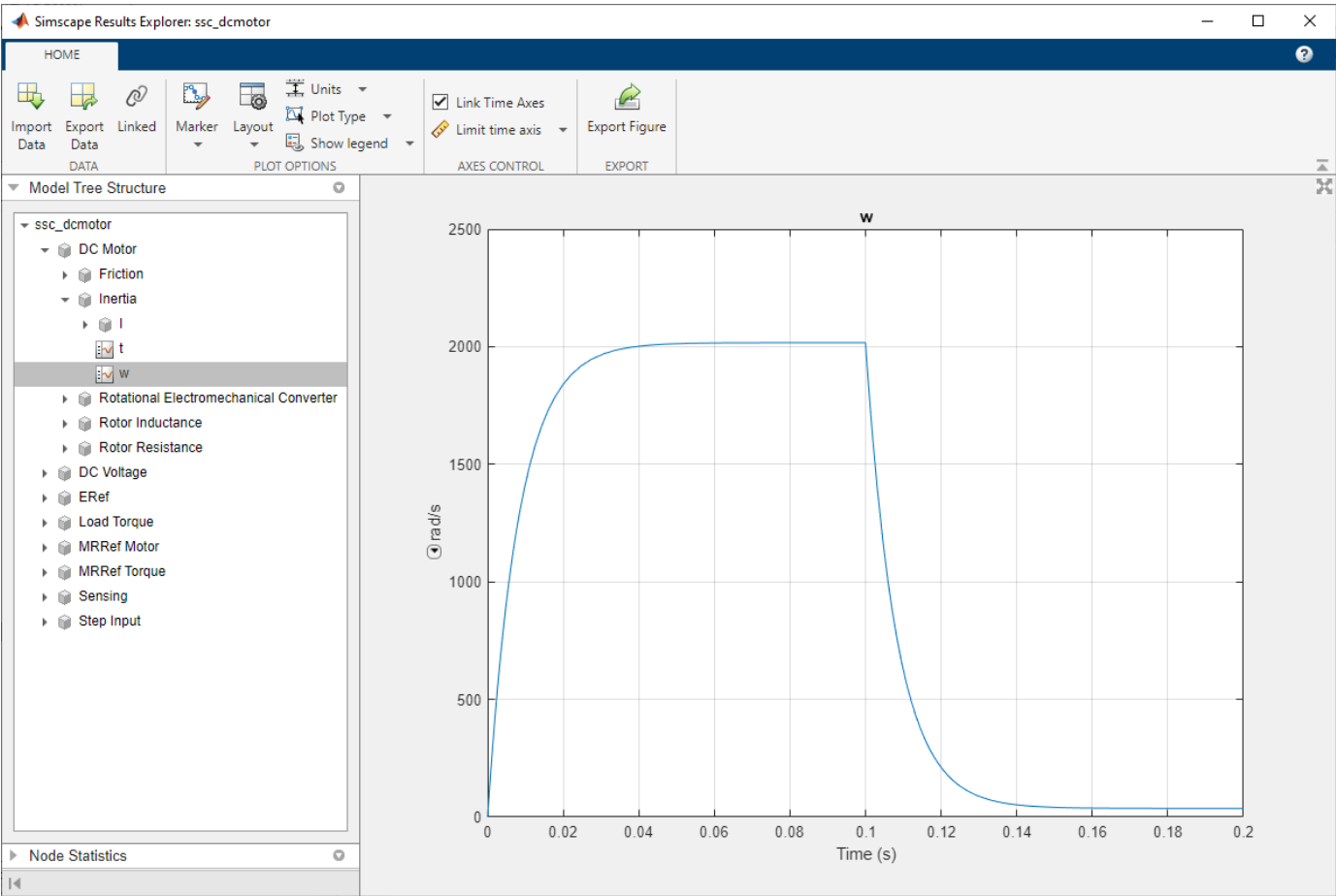
By default, Simscape Results Explorer plots rotational velocity in rad/s.

**5** To switch to custom units, in the Simscape Results Explorer toolbar, in the **Units** drop-down, select `Custom`. The rotational velocity plot is redrawn in deg/s.
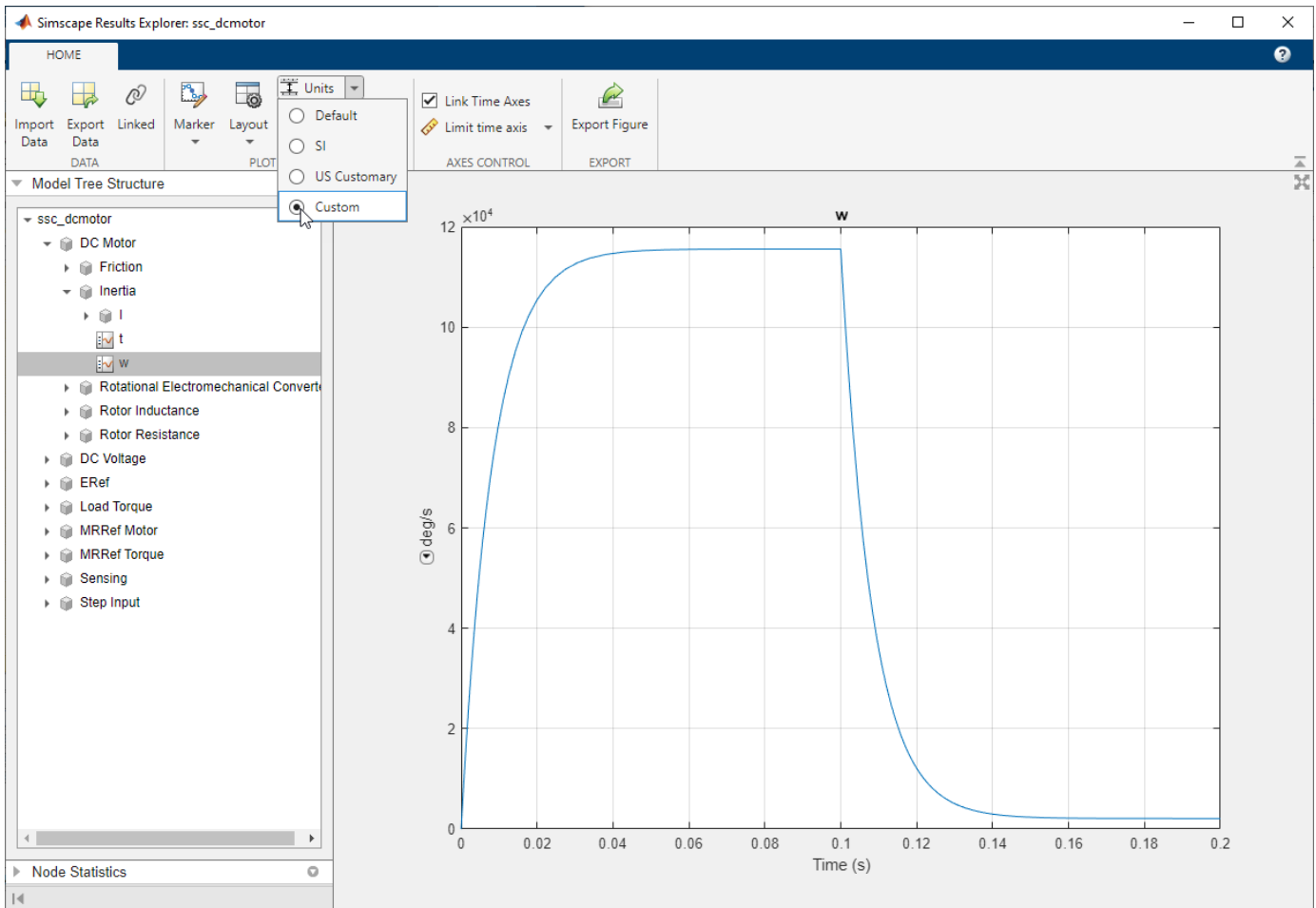
**Tip**    Use the function `pm_getunits` to get the full list of available units.

## See Also

## Related Examples

- "Log, Navigate, and Plot Simulation Data" on page 13-20
- "Log and View Simulation Data for Selected Blocks" on page 13-17
- "Plot Simulation Data in Different Units" on page 13-32

## More About

- "About Simulation Data Logging" on page 13-2
- "About the Simscape Results Explorer" on page 13-24

# View Sparkline Plots of Simulation Data

This example shows the basic workflow for viewing sparkline plots of logged simulation data for selected blocks and variables directly on the model canvas. Before viewing sparkline plots, you must enable data logging for the whole model, or at least for those blocks where you want to display the data, and run the simulation.

---

**Note** If you log Simscape data as part of the single simulation output, the sparkline plots functionality is not available. For more information, see "Single simulation output".

---

**1** Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. This example model has data logging for the whole model enabled.
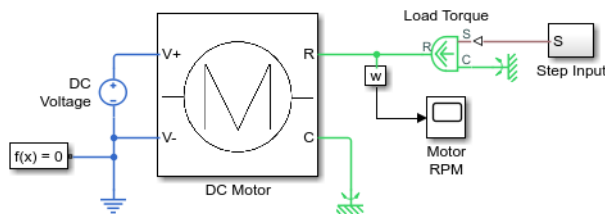


**2** Double-click the DC Motor subsystem to open it.

**3** Simulate the model.

**4** To enable display of the sparkline plots on the model canvas, select any Simscape block in the model, for example, the Rotational Electromechanical Converter block. On the **Simscape Block** tab at the top of the model window, under **Review Results**, click **Sparkline Plot > Enable Sparkline Plots**. This action adds the check mark next to the **Enable Sparkline Plots** option, and you can start selecting blocks to display sparkline plots of logged data for their variables. Repeatedly clicking a block toggles the display of its sparkline plots on and off.

**5** Click the Rotational Electromechanical Converter block again, to toggle its sparkline plots on. Sparkline plots of the first three variables available for this block are displayed on the canvas, and the field below the plots shows that 5 more variables are available.
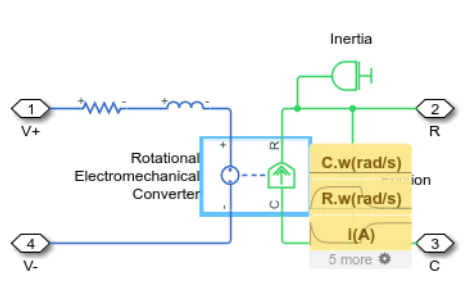


**6** To customize which plots are shown on the canvas, click the little wheel symbol in the field below the plots.
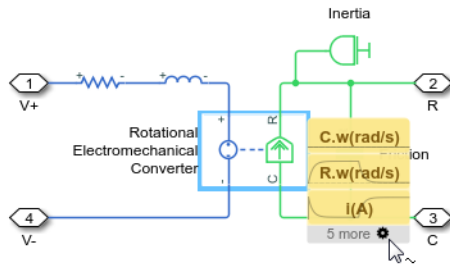
This action displays a list of all the block variables available, with check marks next to the one currently plotted.
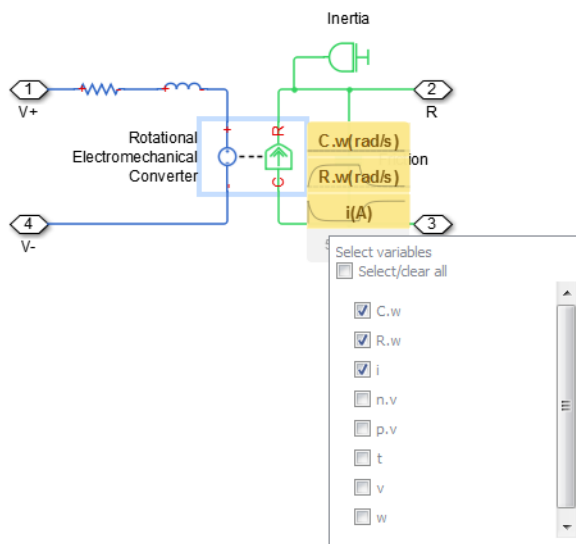


7   Clear all the check marks and select the last variable, w, instead. Then click anywhere on the model canvas to close the variable selection box.

As you hover over the field with the variable name on the canvas, it expands into a sparkline plot of logged simulation data for that variable.
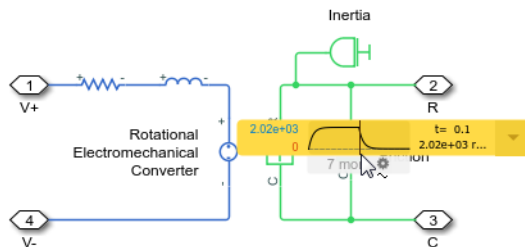


The plot display includes the minimum and maximum values, as well as time and value for the current cursor position.

8   As you move your cursor past the right edge of the plot, the current value is replaced with the last value of the variable. Clicking or hovering over the arrow to the right of the sparkline plot

opens an additional field underneath, which contains a link to the Simscape Results Explorer. When you click the icon, the Simscape Results Explorer window opens, displaying the corresponding plot in the right pane, with the appropriate node selected in the left pane.
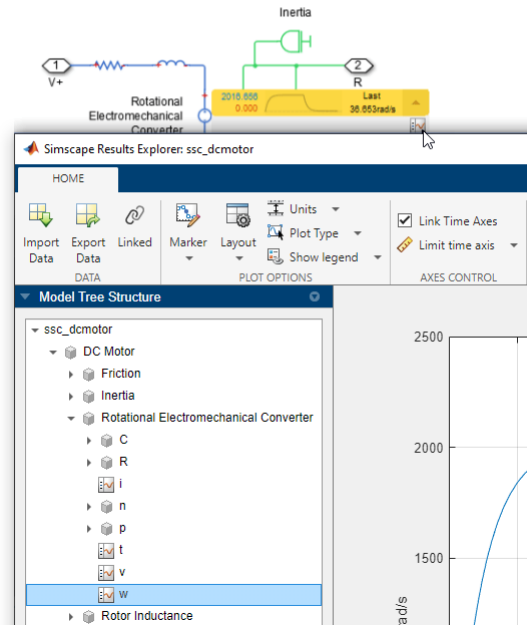


**Tips**

- If you select a block for which simulation data is not being logged, it displays `No variables` instead of the sparkline plots. Right-click the block, select **Simscape** > **Log simulation data**, and rerun the simulation.

- To clear all plots and start again with a clean canvas, on the **Simscape Block** tab, click **Sparkline Plot** > **Remove Sparklines**. Then you can select more blocks and variables to display their sparkline plots.

- Repeatedly selecting the **Enable Sparkline Plots** option toggles the ability to view the sparkline plots for the model on or off, as indicated by the check mark. When the check mark is on, repeatedly clicking a block toggles the display of its sparkline plots on and off.

# See Also

# Related Examples

- "Log, Navigate, and Plot Simulation Data" on page 13-20
- "Log and View Simulation Data for Selected Blocks" on page 13-17

# More About

- "About Simulation Data Logging" on page 13-2
- "About the Simscape Results Explorer" on page 13-24

# Stream Logging Data to Disk

| **In this section...** |
| --- |
| |

When you log simulation data, you can either store the data in a workspace variable, or stream the data to a temporary file on disk and have the workspace variable point to that temporary file. In either case, you interact with the logged simulation data through the simulation log variable.

Saving data to the workspace consumes memory. Streaming logged data to disk significantly increases the data logging capacity, because you are no longer limited by the system memory.

To enable streaming data to disk for all models, on the MATLAB Toolstrip, click **Preferences**. In the left pane of the Preferences dialog box, select **Simscape**, then select the **Stream data to temporary disk directory** check box.
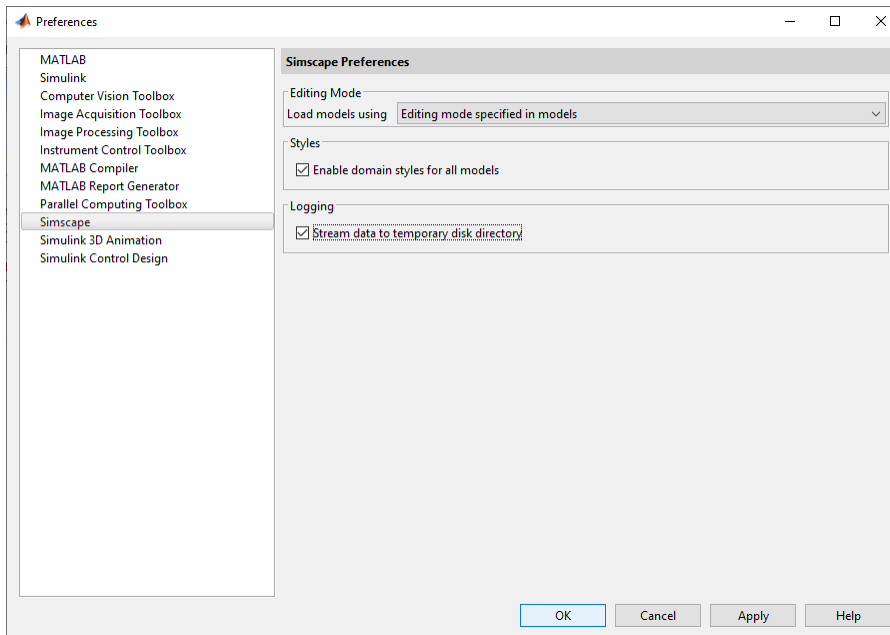


When this preference is turned on, the simulation data, in the form of a `simlog` object generated during simulation, is stored in an MLDATX file in a temporary folder under your user name. The workspace variable of type `simscape.logging.Node`, named as specified by the **Workspace variable name** configuration parameter, gets created, but instead of storing all the simulation data it references the `simlog` object in the temporary file. The temporary file persists as long as there is a logging variable name in the workspace that references it.

You view and analyze logged simulation data by accessing the simulation log variable, exactly the same way as if the simulation data was stored inside it. All the interaction between the workspace variable and the stored object happens behind the scenes. Therefore, you can use the Simscape Results Explorer, as well as all the methods associated with the `simscape.logging.Node` and `simscape.logging.Series` classes to query, plot, and analyze logged simulation data.

The following limitations apply when streaming data to disk:

- The **Limit data points** and **Data history (last N steps)** configuration parameters are ignored. However, you can use the **Decimation** parameter to limit the number of logged data points. For more information, see "Data Logging Options" on page 13-6.

- When you pause model simulation and step back and then forward, all the time points are logged on disk. This is different from storing the data directly in the workspace variable, where the log data is rolled back in this case.

## Streaming to Disk and parfor Loop

If you have a Parallel Computing Toolbox license, then, when you simulate a model inside the `parfor` loop, the temporary MLDATX file is generated within the address space of the worker threads. To access the simulation data outside the `parfor` loop, export the data and then import the exported file outside the `parfor` loop.

```
parfor i=1:2
    model = 'ssc_dcmotor'
    load_system(model);
    set_param(model, 'SimulationMode', 'normal');
    set_param(model, 'SimscapeLogType', 'all', 'SimscapeLogName', 'simlog');
    simOut = sim(model, 'ReturnWorkspaceOutputs', 'on');

    % save to a different file by appending the index
    file = ['fileName_' num2str(i) '.mldatx'];
    simscape.logging.export(simOut.get('simlog'), file);
end

% import the exported files
var = simscape.logging.import('fileName_1.mldatx');
...
```

## Streaming to Disk with parsim

If you have a Parallel Computing Toolbox license, simulating a model with the `parsim` command provides additional functionality, compared to using the `parfor` loop. The following example shows how you can use the `parsim` command when streaming logged simulation data to disk.

Before running the script, make sure that streaming to disk is enabled: open the Preferences dialog box, select **Simscape**, then select the **Stream data to temporary disk directory** check box.

```
model = 'ssc_dcmotor';
% Create array of inputs to run multiple simulations
num = 10;
in(1:num) = Simulink.SimulationInput(model);

% Specify any directory where the MLDATX files will be created for every run
logDir=fullfile(cd, 'tmp'); % current directory
mkdir(logDir)
for i = 1:num
    % This will only work with local pools
    in(i).PostSimFcn = @(x) locHandleSimscapeLTF(model, x, logDir, i);
end

out = parsim(in);

for idx = 1:numel(out)
    simlog = simscape.logging.import(out(idx).SimscapeFileName);
    sscexplore(simlog);
end

function newOut = locHandleSimscapeLTF(model, out, dirName, runId)
    % All the logged variables along with simlog should be part of 'newOut' object
    loggedVars = out.who;
    newOut = struct;
    for i = 1 : numel(loggedVars)
        loggedData = out.(loggedVars{i});
        if isa(loggedData, 'simscape.logging.Node')
            % Specify any file name with .mldatx extension
```

```
            filename = [model '_simlog_file_' num2str(runId) '.mldatx'];
            simscapeFileName = fullfile(dirName, filename);

            % Export simlog to MLDATX file
            simscape.logging.export(loggedData, simscapeFileName);
            newOut.SimscapeFileName = simscapeFileName;
        else
            newOut.(loggedVars{i}) = out.(loggedVars{i});
        end
    end
end
```

## See Also

## More About

- "About Simulation Data Logging" on page 13-2
- "Data Logging Options" on page 13-6
- "About the Simscape Results Explorer" on page 13-24

# Saving and Retrieving Logged Simulation Data

| In this section... |
|---|
| "Data Logged to Memory" on page 13-45 |
| "Data Logged to Temporary File on Disk" on page 13-45 |
| "Data Recorded in Simulation Data Inspector" on page 13-46 |

Ways of saving and retrieving logged simulation data depend on the logging method. Each simulation log has two properties, `savable` and `exportable`, that indicate how the data was logged and therefore, how to save and retrieve it.

| Logging Method | Enabled By | Savable | Exportable | Use |
|---|---|---|---|---|
| Memory | **Stream data to temporary disk directory** preference is off, **Record data in Simulation Data Inspector** option is off | 1 | 0 | MATLAB `save` and `load` functions, "Workspace Variables and MAT-Files" |
| Temporary File on Disk | **Stream data to temporary disk directory** preference is on, **Record data in Simulation Data Inspector** option is off | 0 | 1 | `simscape.logging.export` and `simscape.logging.import` |
| Simulation Data Inspector | **Record data in Simulation Data Inspector** option is on | 0 | 0 | Simulation Data Inspector data management functions, "Analyze Simulation Results" |

## Data Logged to Memory

When you log simulation data to workspace (**Stream data to temporary disk directory** check box on the **Simscape** pane of the Preferences dialog box is off), all the data is stored in the workspace variable. You can use the regular MATLAB interface to save the workspace variable as a MAT-file and load a MAT-file into a variable. For more information, see "Workspace Variables and MAT-Files".

## Data Logged to Temporary File on Disk

When you stream simulation data to disk (**Stream data to temporary disk directory** check box on the **Simscape** pane of the Preferences dialog box is on), the data is stored as a `simlog` object in a temporary file, and the workspace variable references the `simlog` object. The temporary file persists as long as there is a logging variable name in the workspace that references it. The `simscape.logging.export` function saves the logged simulation data to an MLDATX file, other than the temporary file, for use in a later session. To load the MLDATX file containing the stored `simlog` object back into workspace and associate it with a workspace variable, use the `simscape.logging.import` function.

The following restrictions apply:

- When streaming data to disk, you can export and import only the entire `simlog` object. This is different from logging data to workspace, where you can save part of the workspace variable (such as a node in the `simlog` tree) as a separate MAT-file.
- `simscape.logging.export` function does not support cross-locale characters in file name.

## Data Recorded in Simulation Data Inspector

If you select the **Record data in Simulation Data Inspector** check box on the **Simscape** pane of the Configuration Parameters dialog box for a particular model, this choice overrides the logging preference. The `simlog` object has both `savable` and `exportable` properties set to `0`. You must use the Simulation Data Inspector data management functions to save and retrieve this data. For more information, see "Analyze Simulation Results".

## See Also

## More About
- "About Simulation Data Logging" on page 13-2
- "Data Logging Options" on page 13-6
- "Stream Logging Data to Disk" on page 13-42

# Model Statistics

# Simscape Model Statistics

Viewing Simscape model statistics is a good way to evaluate the model prior to simulation. Model statistics provide feedback on the model complexity, so that you can make informed choices about whether you want to simulate the model in its current configuration or make changes to it. This approach helps you achieve the desired simulation performance and goals.

Unlike other derived data (such as data logging or simulation statistics), which is generated during simulation, model statistics is compile-time data that is generated before the model is simulated. When you generate model statistics, the model must be in a compilable state, that is, it must satisfy the requirements described in "Model Validation" on page 7-7.

Use model statistics as part of the iterative model building process. For example, after you make a change to the model, you can view model statistics to answer the following questions:

- Did the change increase the number of variables?
- Does the model have redundant constraints or have I resolved them?
- How many potential zero-crossing signals does the model have?
- Is the circuit high-index, and therefore hard to solve? Did my change have any effect on the index?

The Statistics Viewer analysis tool is available for models containing Simscape blocks and blocks from add-on products. Depending on the types of blocks in the model, the analysis can produce any or all of the following statistics categories:

- **1-D Physical System** — This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape, Simscape Driveline, Simscape Fluids, and Simscape Electrical (except the Specialized Power Systems) libraries.
- **3-D Multibody System** — This node represents aggregate statistics generated from all physical networks that are associated with blocks from the Simscape Multibody library.
- **1-D/3-D Interface** — This node lists the connections between the two types of physical networks. It appears only for models that connect blocks from the Simscape Multibody library to Simscape blocks, or blocks from other add-on products.

Each statistic is generated separately from each topologically distinct physical network of these blocks and then aggregated to appear as a single statistic in the Statistics Viewer.

The **Sources** section of the Statistics Viewer window lists variable sources for the selected statistic:

- If you select a connection under the **1-D/3-D Interface** statistic category, the **Sources** section lists the source and destination for this connection, with links to relevant blocks.
- If you select a statistic with a nonzero value under the **1-D Physical System** category, the **Sources** section lists all the variables that fall under this statistic.

  For each variable, the **Source** column contains the full path to the variable, starting from the top-level model, with a link to the relevant block. If you click the link in the **Source** column, the corresponding block is highlighted in the block diagram. The **Value** column contains the name of the variable, as it would appear in the **Initial Targets** section of the block dialog box.

## Updating the Statistics Viewer

Opening the Statistics Viewer does not trigger an automatic block diagram update. For complex models, a diagram update can last several minutes, and unnecessary diagram updates could lead to loss of productivity.

When you open the Statistics Viewer, it gets populated with the data from the last simulation or

diagram update. You have to update the data explicitly by clicking the **Refresh** button (  ).

The status at the bottom of the viewer window displays the timestamp of its last update. If you have modified the model since the viewer has last been updated, the **Refresh** button displays a warning

symbol (  ), and the timestamp at the bottom of the viewer window turns red to indicate that the data in the viewer might not reflect the latest model changes.

If you open a model, and then open the Statistics Viewer before simulating the model, then the viewer does not contain any data. The **Refresh** button displays a warning symbol (yellow triangle), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.

## See Also

## Related Examples

## More About

# 1-D Physical System Statistics

This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape, Simscape Driveline, Simscape Fluids, and Simscape Electrical libraries, with the exception of Specialized Power Systems blocks.

Each statistic is generated separately from each topologically distinct physical network of these blocks and then aggregated to appear as a single statistic.

The individual statistics are:

- **Number of variables** — This statistic represents the number of variables associated with all 1-D physical systems in the model. Variables are categorized further as continuous, eliminated, secondary, and discrete variables.

- **Number of continuous variables (retained)** — This statistic represents the number of continuous variables associated with all 1-D physical systems in the model. Continuous variables are those variables whose values vary continuously with time, although some continuous variables can change values discontinuously after events. Continuous variables are categorized further as algebraic and differential variables.

  This statistic represents the number of continuous variables in the system after variable elimination. If a system is truly input-output with no dynamics, it is possible to completely eliminate all variables and, in that case, the number of variables is zero.

  - **Number of differential variables** — This statistic represents the number of differential variables associated with all 1-D physical systems in the model. Differential variables are continuous variables whose time derivative appears in one or more system equations. These variables add dynamics to the system and require the solver to use numerical integration to compute their values.

    This statistic represents the number of differential variables in the model after variable elimination.

  - **Number of algebraic variables** — This statistic represents the number of algebraic variables associated with all 1-D physical systems in the model. Algebraic variables are continuous system variables whose time derivative does not appear in any system equations. These variables appear in algebraic equations but add no dynamics, and this typically occurs in physical systems due to conservation laws, such as conservation of mass and energy.

    This statistic represents the number of algebraic variables in the model after variable elimination.

- **Number of continuous variables (eliminated)** — This statistic represents the number of eliminated variables associated with all 1-D physical systems in the model. Eliminated variables are continuous variables that are eliminated by the software and are not used in solving the system. Eliminated variables are categorized further as algebraic and differential variables.

  - **Number of differential variables** — This statistic represents the number of eliminated differential variables associated with all 1-D physical systems in the model. Differential variables are continuous variables whose time derivative appears in one or more system equations. These variables add dynamics to the system and require the solver to use numerical integration to compute their values.

    This statistic represents the number of differential variables in the model that have been eliminated.

- **Number of algebraic variables** — This statistic represents the number of eliminated algebraic variables associated with all 1-D physical systems in the model. Algebraic variables are continuous system variables whose time derivative does not appear in any system equations. These variables appear in algebraic equations but add no dynamics, and this typically occurs in physical systems due to conservation laws, such as conservation of mass and energy.

  This statistic represents the number of algebraic variables in the model that have been eliminated.

- **Number of continuous variables (secondary)** — This statistic represents the number of secondary variables associated with all 1-D physical systems in the model. Secondary variables are variables that the software has added to the system to make it solvable. Secondary variables are categorized further as algebraic and differential variables.

  - **Number of differential variables** — This statistic represents the number of secondary differential variables associated with all 1-D physical systems in the model. Differential variables are continuous variables whose time derivative appears in one or more system equations. These variables add dynamics to the system and require the solver to use numerical integration to compute their values.

    This statistic represents the number of differential variables in the model that have been added to the system to make it solvable.

  - **Number of algebraic variables** — This statistic represents the number of secondary algebraic variables associated with all 1-D physical systems in the model. Algebraic variables are continuous system variables whose time derivative does not appear in any system equations. These variables appear in algebraic equations but add no dynamics, and this typically occurs in physical systems due to conservation laws, such as conservation of mass and energy.

    This statistic represents the number of algebraic variables in the model that have been added to the system to make it solvable.

- **Number of discrete variables** — This statistic represents the number of discrete, or event, variables associated with all 1-D physical systems in the model. Discrete variables are those variables whose values can change only at specific events. Discrete variables are categorized further as integer-valued and real-valued discrete variables.

  - **Number of integer-valued variables** — This statistic represents the number of integer-valued discrete variables associated with all 1-D physical systems in the model. Integer-valued discrete variables are system variables that take on integer values only and can change their values only at specific events, such as sample time hits. These variables are typically generated from blocks that are sampled and run at specified sample times.

  - **Number of real-valued variables** — This statistic represents the number of real-valued discrete variables associated with all 1-D physical systems in the model. Real-valued discrete variables are system variables that take on real values and can change their values only at specific events.

    If you select a local solver in the Solver Configuration block, then all continuous variables associated with that system are discretized and represented as real-valued discrete variables.

- **Number of discrete variables (secondary)** — This statistic represents the number of secondary discrete variables associated with all 1-D physical systems in the model. Discrete secondary variables represent those discrete variables in the model that have been added to the system to make it solvable.

- **Number of zero-crossing signals** — This statistic represents the number of scalar signals that are monitored by the Simulink zero-crossing detection algorithm. Zero-crossing signals are scalar functions of states, inputs, and time whose crossing zero indicates discontinuity in the system. These signals are typically generated from operators and functions that contain discontinuities, such as comparison operators, `abs`, `sqrt` functions, and so on. Times when these signals cross zero are reported as zero-crossing events. During simulation it is possible for none of these signals to produce a zero-crossing event or for one or more of these signals to have multiple zero-crossing events.
- **Number of dynamic variable constraints** — This statistic represents the number of constraints involving only dynamic variables and inputs. Such constraints result in high-index differential algebraic equations (DAEs) and therefore can cause numerical difficulties or slow down your simulation. These are the state constraints that exist at run time, they do not include state constraints that have been eliminated by the compile-time index reduction.

If you select a statistic with a nonzero value, the **Sources** section lists all the variables that fall under this statistic. For each variable:

- The **Source** column contains the full path to the variable, starting from the top-level model, with a link to the relevant block. If you click the link in the **Source** column, the corresponding block is highlighted in the block diagram.

  If your model contains blocks with underlying arrays of components, the full path to the variable in the **Source** column contains the numbered name of the array member. For more information, see "Model Statistics for Component Arrays" on page 14-17.
- The **Value** column contains the name of the variable, as it would appear in the **Initial Targets** section of the block dialog box.

If your model uses a Partitioning local solver, the Statistics Viewer contains additional statistics specific to this solver type. For more information, see "Model Statistics Available when Using the Partitioning Solver" on page 14-19.

## See Also

### Related Examples
- "View Model Statistics" on page 14-10
- "Access Block Variables Using Statistics Viewer" on page 14-14

### More About
- "Simscape Model Statistics" on page 14-2

# 3-D Multibody System Statistics

This node represents aggregate statistics generated from all physical networks that are associated with blocks from the Simscape Multibody library.

Each statistic is generated separately from each topologically distinct physical network of these blocks and then aggregated to appear as a single statistic.

The individual statistics are:

- **Number of bodies (total, excluding ground)** — This statistic provides the total number of bodies present in a mechanical system. This number equals the sum of two types of bodies: rigid components and flexible bodies. For more information, see the statistic descriptions for these types of bodies.

- **Number of rigidly connected components (excluding ground)** — This statistic provides the number of rigid components present in a mechanical system. Rigid components are subsets of rigidly connected blocks that represent rigid bodies or rigid frame networks in a model. These subsets include blocks from the Body Elements library, including the Variable Mass sublibrary, as well as Rigid Transform blocks.

  Rigid connections within a rigid component can include Rigid Transform blocks, but not Weld Joint blocks. Rigid Transform blocks provide rigid connections between blocks in the same rigid component. Weld Joint blocks, like all joint blocks, provide connections between blocks in different rigid components.

  This statistic excludes from the count any rigid component that rigidly connects to the World Frame blocks.

- **Number of flexible bodies** — This statistic provides the number of flexible bodies present in a mechanical system. These correspond to individual blocks from the Flexible Bodies sublibrary.

  Each individual flexible body block counts as one in this statistic. For example, two flexible body blocks directly connected through a frame line count as two separate flexible bodies.

  Flexible bodies do not combine with any rigid component blocks. For example, if two flexible body blocks are connected through a Rigid Transform block, then the Rigid Transform block makes up a rigid component wedged between two separate flexible bodies.

- **Number of joints (total)** — This statistic provides the total number of joints present in a mechanical system. This number equals the sum of three types of joints: explicit tree, cut, and implicit 6-DOF joints. For more information, see the statistic descriptions for these types of joints.

- **Number of explicit tree joints** — This statistic provides the number of joints in a mechanical system that correspond to explicit joint blocks. This number excludes joints that are cut to open kinematic loops.

- **Number of implicit 6-DOF tree joints** — This statistic provides the number of 6-DOF joints in a mechanical system that do not correspond to explicit joint blocks. Simscape Multibody adds one implicit 6-DOF joint for each portion of the system that is disconnected from the ground body. Such implicit joints never create kinematic loops and are therefore never cut.

- **Number of cut joints** — This statistic provides the number of joints that are cut from a mechanical system in order to open all of its closed kinematic loops. The number of cut joints equals the number of closed loops present in the system.

- **Number of constraints** — This statistic provides the total number of constraints in a mechanical system. This number includes constraint elements stemming from explicit constraint blocks as

well as those generated from belt-cable networks. It does not include constraints stemming from cut joints.

- **Number of tree degrees of freedom (total)** — This statistic provides the total number of degrees of freedom in a mechanical system, before the application of any constraint equations. This number equals the sum of the total number of uncut joint degrees of freedom and the total number of body degrees of freedom. For more information, see the statistic descriptions for **Number of tree joint degrees of freedom** and **Number of flexible body degrees of freedom**.

- **Number of tree joint degrees of freedom** — This statistic provides the number of uncut joint degrees of freedom in a mechanical system. This number equals the sum of all degrees of freedom that the uncut joints provide. It excludes degrees of freedom associated with cut joints.

- **Number of flexible body degrees of freedom** — This statistic provides the number of body degrees of freedom in a mechanical system. This number equals the sum of all degrees of freedom that the flexible bodies provide. Rigid components do not have any degrees of freedom and do not contribute to this statistic.

- **Number of position constraint equations (total)** — This statistic provides the number of scalar equations that impose position constraints on a mechanical system. Constraint equations arise from three types of blocks: constraints, joints, and belt-cable blocks. Joint blocks contribute constraint equations only if the joints are cut. The number of position constraint equations that a cut joint contributes equals six minus the number of degrees of freedom that joint provides.

- **Number of position constraint equations (non-redundant)** — This statistic provides the number of unique position constraint equations associated with a model. This number is smaller than or equal to the total number of position constraint equations. The difference between the two is the number of redundant position constraint equations, which are satisfied whenever the unique position constraint equations are satisfied. Simscape Multibody attempts to remove redundant equations to improve simulation performance.

- **Number of mechanism degrees of freedom (minimum)** — This statistic provides a lower bound on the number of degrees of freedom in a mechanical system. It equals the difference between the total number of (unconstrained) degrees of freedom and the number of non-redundant position constraint equations. The actual number of degrees of freedom can exceed this lower bound if Simscape Multibody fails to detect a position constraint equation.

  Some position constraint equations become redundant only in certain configurations. If an equation becomes redundant during simulation, the actual number of degrees of freedom in a model can change. However, that number must still equal or exceed the lower bound that this statistic provides.

- **State vector size** — This statistic provides the number of scalar values in the state vector of a mechanical system.

- **Average number of degrees of freedom in kinematic loops** — This statistic provides the average number of degrees of freedom in the closed kinematic loops of a mechanical system. The average number is taken over all loops in the system. If the system has no kinematic loops, this number equals zero.

## See Also

## More About

- "Simscape Model Statistics" on page 14-2

# 1-D/3-D Interface Statistics

This node lists statistics related to the interface between all 1-D physical and 3-D multibody systems present in the model. It appears only for models that connect blocks from the Simscape Multibody library to blocks from Simscape, Simscape Driveline, Simscape Fluids, and Simscape Electrical libraries, except the Specialized Power Systems blocks.

All connections are listed individually as `Connection 1`, `Connection 2`, and so on. If you select an individual connection, the **Sources** section lists the source and destination ports for this connection:

- The **Source** column contains the full path to the interface port, starting from the top-level model, with a link to the relevant block. If you click the link in the **Source** column, the corresponding block is highlighted in the block diagram.
- The **Value** column specifies whether the port is the source or destination.

If you expand a connection node, the Statistics Viewer provides the filtering information: whether a filter is used, and, if yes, the filter order and time constant.
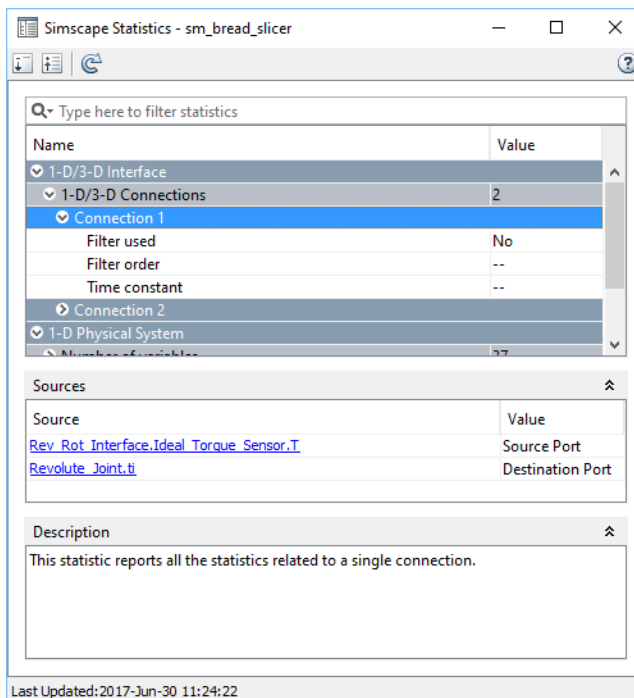


## See Also

## More About

- "Simscape Model Statistics" on page 14-2

# View Model Statistics

This example shows how you can use model statistics to determine the effect of a change on model complexity.

1  Open the Simple Mechanical System example model.
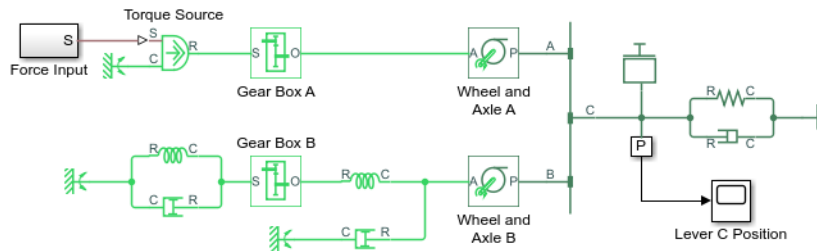


**Simple Mechanical System**
1. Explore simulation results using sscexplore
2. Learn more about this example

2  To view model statistics, in the model window, on the **Debug** tab, click **Simscape > Statistics Viewer**.

The Simscape Statistics window opens, but it does not contain any data. If you open a model, and then open the Statistics Viewer before running the simulation, the statistics data is not available. The **Refresh** button in the toolbar of the viewer window displays a warning symbol (yellow triangle), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.



3  Click the **Refresh** button. The Simscape Statistics window now displays an overview of the models statistics in a collapsed state.

**4** Click ⬇ to expand all nodes.



You can see that, after variable elimination, the model contains five continuous differential variables, no algebraic variables, no discrete variables, and no zero-crossing signals.

**5** To limit the range of motion, add a Translational Hard Stop block to the model diagram, in parallel with the Translational Inertia and the Translational Damper blocks, as shown in the following figure.
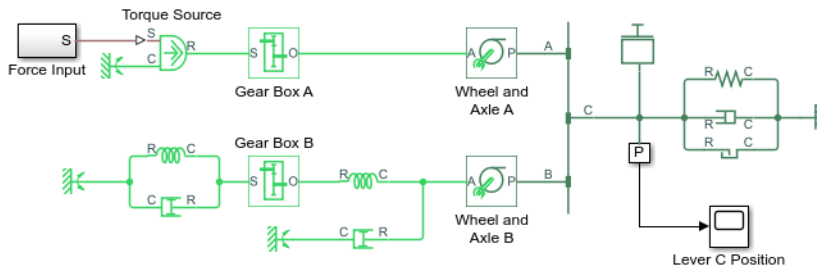
**Simple Mechanical System**
1. Explore simulation results using sscexplore
2. Learn more about this example

**6** Refresh the model statistics. Again, expand all the nodes.



The revised model, after variable elimination, contains six differential variables and five algebraic variables. This happened because you added a nonlinear block (Translational Hard Stop).

Therefore, the linear optimization that the solver initially performed on the model no longer applies.

## See Also

## Related Examples
- "Access Block Variables Using Statistics Viewer" on page 14-14

## More About
- "1-D Physical System Statistics" on page 14-4

# Access Block Variables Using Statistics Viewer

This example shows how you can use the **Sources** section of the Statistics Viewer to access a block variable of interest, to verify (or change) its initialization priority and target value.

**1**    Open the Permanent Magnet DC Motor example model.
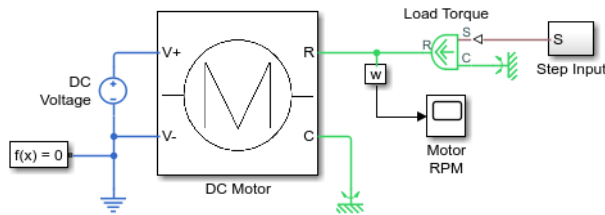


**2**    To view model statistics, in the model window, on the **Debug** tab, click **Simscape > Statistics Viewer**.

The Simscape Statistics window opens, but it does not contain any data. If you open a model, and then open the Statistics Viewer before running the simulation, the statistics data is not available. The **Refresh** button in the toolbar of the viewer window displays a warning symbol (yellow triangle), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.



**3**    Click the **Refresh** button. The Simscape Statistics window now displays an overview of the models statistics in a collapsed state. The status at the bottom of the window displays the timestamp of its last update.

**4**    Expand the **Number of variables** node, then **Number of continuous variables (retained)**, and then click **Number of differential variables**.

You can see that, after variable elimination, the model contains three continuous differential variables, and the **Sources** section of the Statistics Viewer lists these three variables. For each variable:

- The **Source** column contains the full path to the variable, starting from the top-level model, with a link to the relevant block.

- The **Value** column contains the name of the variable, as it would appear in the **Initial Targets** section of the block dialog box.

**5** Click the first link in the **Source** column. The full path indicates that the source variable w belongs to the Inertia block in the DC Motor subsystem of the top-level example model, therefore the DC Motor subsystem opens and the corresponding block is highlighted in the block diagram, as shown in the following figure.
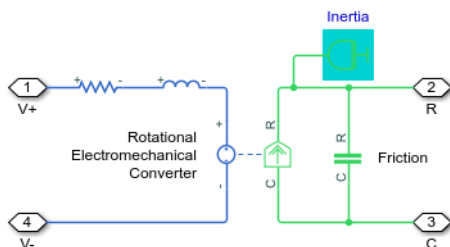


**6** Double-click the highlighted block.

**7** In the block dialog box, open the **Initial Targets** section.
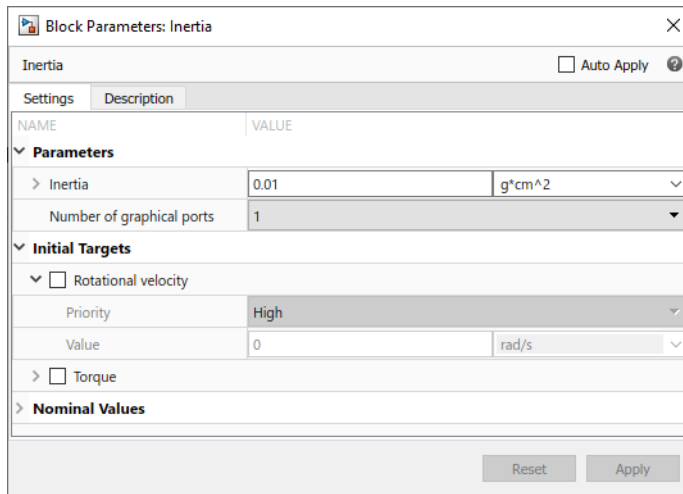
According to the **Value** column in the Statistics Viewer, the name of the variable in the block dialog box is **Rotational velocity**. The dialog box shows that this variable has high initialization priority and a target value of 0 rad/s. You can modify the priority and value, if needed, and then open the Variable Viewer to see the model initialization results.

## See Also

## Related Examples

- "Set Priority and Initial Target for Block Variables" on page 8-4

## More About

- "1-D Physical System Statistics" on page 14-4
- "Block-Level Variable Initialization" on page 8-2
- "Variable Viewer" on page 8-18

# Model Statistics for Component Arrays

If your model contains blocks with underlying arrays of components, the Statistics Viewer includes variables that belong to the array members.

For example, in this model, the Simscape Component block contains an underlying array of resistors.
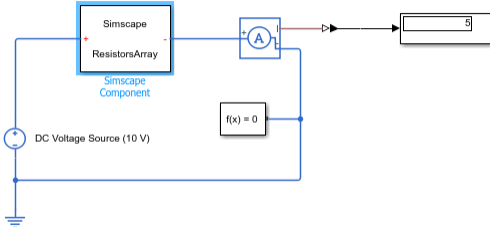


In the Statistics Viewer, if you select a statistic with a nonzero value, the **Sources** section lists all the variables that fall under this statistic, including the array member variables. For variables that belong to the array members, the full path to the variable in the **Source** column contains the numbered name of the array member, such as `resistor(1)`, `resistor(2)`, and so on. If the component array size is `1xN`, the members are numbered `comp(1)`, ..., `comp(N)`. If the array size is `NxM`, the members are numbered `comp(1,1)`, `comp(1,2)`, ..., `comp(NxM)`.
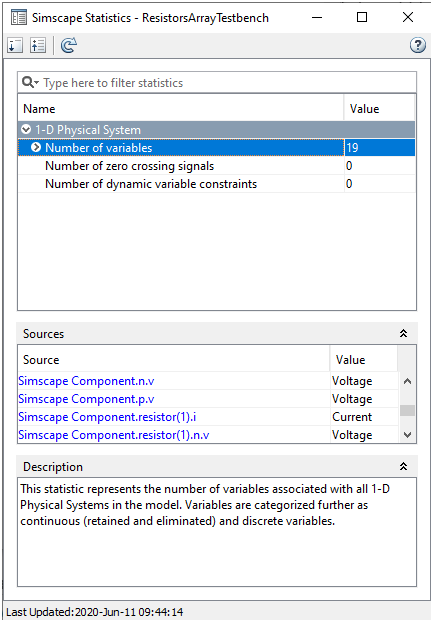


If you click an array member variable link in the **Source** column, the parent block containing the component array is highlighted in the block diagram.
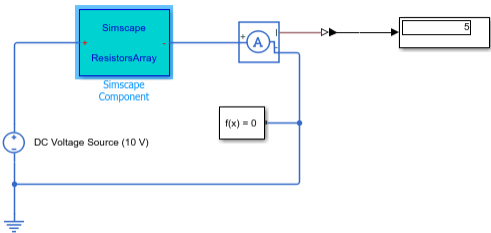
# Model Statistics Available when Using the Partitioning Solver

If your model uses a Partitioning local solver, the Statistics Viewer contains additional statistics specific to this solver type:

- **Number of partitions** — This statistic represents the total number of partitions in the model. When you expand this node, you can view additional statistics, listed below.

- **Total memory estimate** — This statistic represents the estimate for memory usage, in kB, for this model when using the exhaustive partition storage method.

- Additional nodes, named **Partition1**, **Partition2**, and so on, up to the total number of partitions in the model. For each of these nodes, the **Value** column lists the integration method applied to this partition. Possible methods are forward Euler (explicit) and backward Euler (implicit). For each partition node, you can also see:

  - **Equation type** — The **Value** column for this statistic lists the type of the equations in the partition. Possible types are linear time-invariant, switched linear, and linear time-varying.

  - **Number of variables** — This statistic represents the number of scalar variables in the partition. When you select this node, the **Sources** section of the Statistics Viewer lists all the variables that fall under this statistic. For each variable, the **Source** column contains the full path to the variable, starting from the top-level model, with a link to the relevant block. If you click the link in the **Source** column, the corresponding block is highlighted in the block diagram. The **Value** column contains the name of the variable, as it would appear in the **Initial Targets** section of the block dialog box.

  - **Number of equations** — This statistic represents the number of scalar equations in the partition. Sources of the equations are provided, if they are available. When you select this node, the **Sources** section of the Statistics Viewer lists all the blocks that provide the equations. If the block source code is available (that is, not protected), clicking the link in the **Source** column opens the Simscape source file for this block in the MATLAB Editor, pointing to the appropriate equation.

  - **Number of modes** — If the **Equation type** is linear time-invariant or switched linear, this statistic represents the number of modes in the partition. Every `if` and `elseif` statement in Simscape source code corresponds to a mode. In general, when an implicit integration method is applied to a partition, the more modes there are, the more iterations are possibly needed to solve the equations for this partition.

  - **Number of configurations** — This statistic represents the total number of different systems of linear equations that need to be solved when simulating the partition. In the linear time-invariant or switched linear cases, this is $2^n$, where $n$ is the number of modes in the partition. To accelerate computation, decompositions of some systems are cached for each set of modes (the **Partition storage method** parameter in the Solver Configuration block defines how the decompositions are cached). If this number exceeds the greatest possible supported unsigned integer value, the **Value** column for this statistic displays `Overflow`.

  - **Memory estimate** — This statistic represents the estimate for memory usage, in kB, for this partition when using the exhaustive partition storage method.

## Estimate the Memory Budget for Exhaustive Partitioning Storage

This example shows how you can use the Statistics Viewer to estimate the memory budget needed for simulating a model that uses the Partitioning solver.

1   Open the Permanent Magnet DC Motor example model.
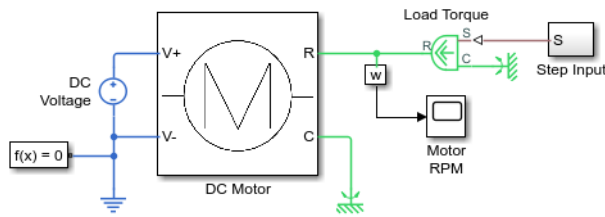


2   Double-click the Solver Configuration block, select the **Use local solver** check box, and then set the **Solver type** to `Partitioning`.

3   To view model statistics, in the model window, on the **Debug** tab, click **Simscape > Statistics Viewer**. Click the **Refresh** button in the toolbar of the viewer window, if necessary, to populate the viewer with data.

4   Expand the **Number of partitions** node.



The **Total memory estimate** statistic indicates that the estimate for memory usage for this model is 3 kB. When you use the exhaustive partition storage method, the default memory budget

allocated for partition storage is 1024 kB. Therefore, the default memory budget value is sufficient for simulating this model, and you can even reduce it, if necessary.

If the memory estimate is unexpectedly large, you can investigate further by expanding each of the individual partition nodes and checking the **Memory estimate** value for that partition.

## See Also

Solver Configuration

## More About

- "Understanding How the Partitioning Solver Works" on page 7-25
- "Increase Simulation Speed Using the Partitioning Solver" on page 11-19
- "1-D Physical System Statistics" on page 14-4
- "Access Block Variables Using Statistics Viewer" on page 14-14

# Physical Units

# How to Work with Physical Units

Parameters, variables, and physical signals can have units associated with them. You specify the units along with the parameter values in the block dialogs and Property Inspector, and Simscape unit manager performs the necessary unit conversion operations when solving a physical network. Simscape blocks support standard measurement systems. The default block units are meter-kilogram-second or MKS (SI).

Simscape software comes with a library of standard units, and you can define additional units as needed (see "Unit Definitions" on page 15-3). You can use these units in your block diagrams:

- To specify the units of an input physical signal, type a unit name, or a mathematical expression with unit names, in the **Input signal unit** field of the Simulink-PS Converter block dialog. You can also select a unit from a drop-down list, which is prepopulated with some common input units. Signal units that you specify in a Simulink-PS Converter block must match the input type expected by the Simscape block connected to it. For example, when you provide the input signal for an Ideal Angular Velocity Source block, specify angular velocity units, such as `rad/s` or `rpm`, in the Simulink-PS Converter block, or leave it unitless. If you leave the block unitless, with the **Input signal unit** parameter set to `1`, then the physical signal units are inferred from the destination block.

- Simscape block dialogs and Property Inspector have drop-down combo boxes of units next to a parameter value, letting you either select a unit from the drop-down list, or type a unit name (or a mathematical expression with unit names) directly into the box. These drop-down lists are automatically populated by those units that are commensurate with the unit of the parameter, based on the current list of unit definitions. For example, if a parameter is specified, by default, with the units of meters per second, `m/s`, the drop-down list of units contains commensurate units, such as `mm/s`, `in/s`, `fps` (feet per second), `fpm` (feet per minute), and so on, including any other linear velocity units currently defined in your unit registry.

- To specify the units of an output physical signal, type a unit name, or a mathematical expression with unit names, in the **Output signal unit** field of the PS-Simulink Converter block dialog. You can also select a unit from a drop-down list, which is prepopulated with some common output units. The system compares the units you specified with the actual units of the input physical signal coming into the converter block and applies a gain equal to the conversion factor before outputting the Simulink signal. The default value is `1`, which means that the unit is not specified. If you do not specify a unit, or if the unit matches the actual units of the input physical signal, no gain is applied.

For more information, see "How to Specify Units in Block Dialogs" on page 15-9.

# Unit Definitions

Simscape unit names are defined in the `pm_units.m` file, which is shipped with the product. You can open this file to see how the physical units are defined in the product, and also as an example when adding your own units. This file is located in the directory *matlabroot*`\toolbox\physmod\common\units\mli\m`.

Default registered units and their abbreviations are listed in the following table. Use the `pm_getunits` command to get an up-to-date list of units currently defined in your unit registry. Use the `pm_adddimension` and `pm_addunit` commands to define additional units.

**Physical Unit Abbreviations Defined by Default in the Simscape Unit Registry**

| Quantity | Abbreviation | Unit |
|---|---|---|
| Acceleration | gee | Earth gravitational acceleration (9.80665 m/s^2) |
| Amount of substance | mol | Mole |
| Angle | rad | Radian |
|  | deg | Degree |
|  | rev | Revolution |
| Angular velocity | rpm | Revolutions/minute |
| Capacitance | F | Farad |
|  | pF | Picofarad |
|  | nF | Nanofarad |
|  | uF | Microfarad |
| Charge | C | Coulomb |
|  | nC | Nanocoulomb |
|  | uC | Microcoulomb |
|  | mC | Millicoulomb |
| Conductance | S | Siemens |
|  | nS | Nanosiemens |
|  | uS | Microsiemens |
|  | mS | Millisiemens |
| Current | A | Ampere |
|  | pA | Picoampere |
|  | nA | Nanoampere |
|  | uA | Microampere |
|  | mA | Milliampere |
|  | kA | Kiloampere |

| Quantity | Abbreviation | Unit |
|---|---|---|
| Energy | J | Joule |
| | kJ | Kilojoule |
| | MJ | Megajoule |
| | Btu_IT | British thermal unit |
| | eV | Electronvolt |
| Flow rate | lpm | Liter/minute |
| | gpm | Gallon/minute |
| Force | N | Newton |
| | dyn | Dyne |
| | lbf | Pound-force |
| | mN | Millinewton |
| Frequency | Hz | Hertz |
| | kHz | Kilohertz |
| | MHz | Megahertz |
| | GHz | Gigahertz |
| Inductance | H | Henry |
| | uH | Microhenry |
| | mH | Millihenry |
| Length | m | Meter |
| | cm | Centimeter |
| | mm | Millimeter |
| | km | Kilometer |
| | um | Micrometer |
| | in | Inch |
| | ft | Foot |
| | mi | Mile |
| | yd | Yard |
| Magnetic flux | Wb | Weber |

| Quantity | Abbreviation | Unit |
|---|---|---|
| Magnetic flux density | T | Tesla |
| | G | Gauss |
| Mass | kg | Kilogram |
| | g | Gram |
| | mg | Milligram |
| | t | Tonne (metric ton) |
| | lbm | Pound mass |
| | oz | Ounce |
| | slug | Slug |
| Pressure | Pa | Pascal |
| | uPa | Micropascal |
| | kPa | Kilopascal |
| | MPa | Megapascal |
| | GPa | Gigapascal |
| | bar | Bar |
| | kbar | Kilobar |
| | atm | Atmosphere |
| | psi | Pound/inch^2 |
| | ksi | Kilopound/inch^2 |
| Power | W | Watt |
| | uW | Microwatt |
| | mW | Milliwatt |
| | kW | Kilowatt |
| | MW | Megawatt |
| | HP_DIN | Horsepower |

| Quantity | Abbreviation | Unit |
|---|---|---|
| Resistance | Ohm | Ohm |
| | kOhm | Kiloohm |
| | MOhm | Megaohm |
| | GOhm | Gigaohm |
| Temperature | K | Kelvin |
| | degC | Celsius |
| | degF | Fahrenheit |
| | degR | Rankine |
| | deltaK, deltadegC, deltadegF, deltadegR | Relative temperature units (see "Thermal Unit Conversions" on page 15-11) |
| Time | s | Second |
| | min | Minute |
| | hr | Hour |
| | d | Day |
| | ms | Millisecond |
| | us | Microsecond |
| | ns | Nanosecond |
| Velocity | mph | Miles/hour |
| | fpm | Feet/minute |
| | fps | Feet/second |
| Viscosity absolute | P | Poise |
| | cP | Centipoise |
| | reyn | Reyn |
| Viscosity kinematic | St | Stokes |
| | cSt | Centistokes |
| | newt | Newt |
| Volume | l | Liter |
| | gal | US liquid gallon |
| | igal | Imperial (UK) gallon |

| Quantity | Abbreviation | Unit |
|---|---|---|
| Voltage | V | Volt |
| | mV | Millivolt |
| | kV | Kilovolt |

**Note** This table lists the unit abbreviations defined in the product. For information on how to use the abbreviations above, or mathematical expressions with these abbreviations, to specify units for the parameter values in the block dialogs, see "How to Specify Units in Block Dialogs" on page 15-9.

# How to Specify Units in Block Dialogs

Simscape block dialogs and Property Inspector have drop-down combo boxes for units next to a parameter value. For example, if you open the Translational Friction block dialog box, the drop-down list for the **Breakaway friction force** parameter contains N, kN, lb, mN, dyn, and lbf. Likewise, the drop-down list for the **Breakaway friction velocity** parameter, besides the default m/s, contains other unit expressions commensurate with distance over time, such as mm/s, fpm, mph, and so on.

You can either select a unit from the drop-down list, or type a commensurate unit name (or a mathematical expression with unit names) directly into the unit combo box. You can use the abbreviations for the units defined in your registry, or any valid mathematical expressions with these abbreviations. For example, you can specify torque in newton-meters (N*m) or pound-feet (lbf*ft). To specify velocity, you can use one of the defined unit abbreviations (mph, fpm, fps), or an expression based on any combination of the defined units of length and time, such as meters/second (m/s), millimeters/second (mm/s), inches/minute (in/min), and so on.

> **Note** Affine units (such as Celsius or Fahrenheit) are not allowed in unit expressions. For more information, see "About Affine Units" on page 15-11.

The following operators are supported in the unit mathematical expressions:

| * | Multiplication |
|---|---|
| / | Division |
| ^ | Power |
| + | Plus — for exponents only |
| - | Minus — for exponents only |
| ( ) | Brackets to specify evaluation order |

Metric unit prefixes, such as *kilo*, *milli*, or *micro*, are not supported. For example, if you want to use milliliter as a unit of volume, you have to add it to the unit registry:

```
pm_addunit('ml', 0.001, 'l');
```

The drop-down lists next to parameter names are automatically populated by those units that are commensurate with the unit of the parameter. If you specify the units by typing, it is your responsibility to enter units that are commensurate with the unit of the parameter. The unit manager performs error checking when you click **Apply** in the block dialog box and issues an error if you type an incorrect unit.

> **Note** By default, changing a value in the block dialog box immediately applies the new value. For better control, it is recommended that you clear the **Auto Apply** check box in the upper-right corner of the block dialog box to enable the **Reset** and **Apply** buttons.

In the Simulink-PS Converter and the PS-Simulink Converter block dialog boxes, the drop-down lists are prepopulated with some common input and output units, and it is your responsibility to select or type a unit expression commensurate with the expected input or output units. The error checking for

the converter blocks is performed at the time of simulation. See "Model Validation" on page 7-7 for details.

# Thermal Unit Conversions

| In this section... |
|---|
| "About Affine Units" on page 15-11 |
| "When to Apply Affine Conversion" on page 15-11 |

## About Affine Units

Thermal units often require an affine conversion, that is, a conversion that performs both multiplication and addition. To convert from the old value $T_{old}$ to the new value $T_{new}$, we need a linear conversion coefficient $L$ and an offset $O$:

$T_{new} = L * T_{old} + O$

For example, to convert a temperature reading from degrees Celsius into degrees Fahrenheit, the linear term equals 9/5, and the offset equals 32:

$T_{Fahr} = 9 / 5 * T_{Cels} + 32$

Simscape unit manager defines kelvin (K) as the fundamental temperature unit. This makes Celsius (degC) and Fahrenheit (degF) affine units because they are both related to kelvin with an affine conversion. Rankine (degR) is defined in terms of kelvin with a zero linear offset and, therefore, is not an affine unit.

The following are the default Simscape unit registry definitions for temperature units:

```
pm_adddimension('temperature', 'K');        % defines kelvin as fundamental temperature unit
pm_addunit('degC', [1 273.15], 'K');         % defines Celsius in terms of kelvin
pm_addunit('degF', [5/9 -32*5/9], 'degC');   % defines Fahrenheit in terms of Celsius
pm_addunit('degR', [5/9 0], 'K');            % defines rankine in terms of kelvin
```

## When to Apply Affine Conversion

In dealing with affine units, sometimes you need to convert them using just the linear term. Usually, this happens when the value you convert represents relative, rather than absolute, temperature, $\Delta T = T_1 - T_2$.

$\Delta T_{new} = L * \Delta T_{old}$

In this case, adding the affine offset would yield incorrect conversion results.

For example, the outdoor temperature rose by 18 degrees Fahrenheit, and you need to input this value into your model. When converting this value into kelvin, use linear conversion

$\Delta T_{kelvin} = 5 / 9 * \Delta T_{Fahr}$

and you get 10 K, that is, the outdoor temperature changed by 10 kelvin. If you apply affine conversion, you will get a temperature change of approximately 265 kelvin, which is incorrect.

This is even better illustrated if you use degrees Celsius for the input units because the linear term for conversion between Celsius and kelvin is 1:

- If the outdoor temperature *changed* by 10 degrees Celsius (relative temperature value), then it changed by 10 kelvin (do not apply affine conversion).

- If the outdoor temperature *is* 10 degrees Celsius (absolute temperature value), then it is 283 kelvin (apply affine conversion).

For relative temperatures, you can also use the relative temperature units: `deltaK`, `deltadegC`, `deltadegF`, `deltadegR`. These units are consistent with the Simulink unit database (see "Units in Simulink"). If you use these units, affine conversion does not apply.

## See Also

## Related Examples
- "How to Apply Affine Conversion" on page 15-13

# How to Apply Affine Conversion

When you specify affine units for an input temperature signal, it is important to consider whether you need to apply affine conversion. Usually this decision depends on whether the signal represents absolute or relative temperature (see "When to Apply Affine Conversion" on page 15-11).

For example, you model a house-heating system, and you need to input the outdoor temperature. In the following diagram, the Constant source block represents the average outdoor temperature, in degrees Celsius, and the Sine source block adds the daily temperature variation. The average outdoor temperature, in this case, is 12 degrees Celsius. Daily variation with an amplitude of 8 makes the input outdoor temperature vary between 4 and 20 degrees Celsius.
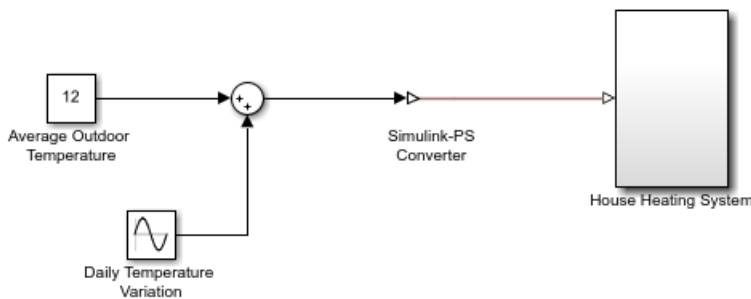


This signal is an absolute temperature reading. Therefore, when the signal converts into kelvin for further computations, you need to specify that it should use affine conversion. Double-click the Simulink-PS Converter block, type `degC` in the **Input signal unit** field, and select the **Apply affine conversion** check box.
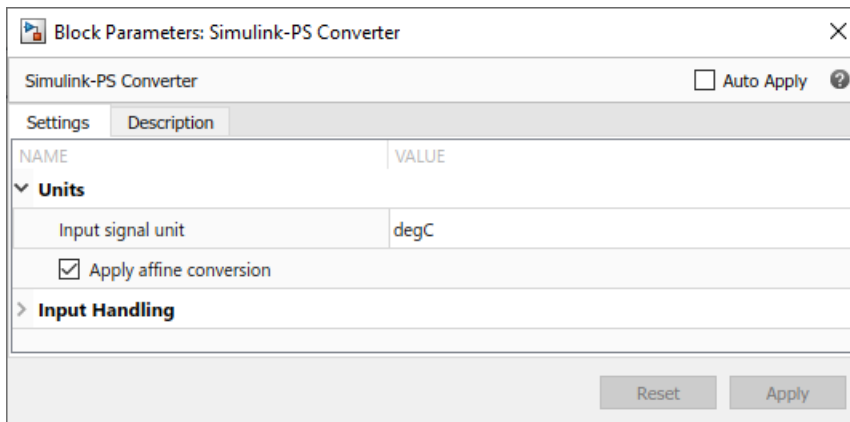


As a result, the Simulink-PS Converter block outputs a value varying between 277 K and 293 K.

## See Also

## More About

- "Thermal Unit Conversions" on page 15-11

# Angular Units

Simscape implementation of angular units relies on the concept of angular units, specifically radians, being a unit but dimensionless. The notion of angular units being dimensionless is widely held in the metrology community. The fundamental angular unit, radian, is defined in the Simscape unit registry as:

```
pm_addunit('rad', 1, 'm/m');
```

which corresponds to the SI and NIST definition [1 on page 15-14]. In other words, Simscape unit manager does not introduce a separate dimension, `'angle'`, with a fundamental unit of `'rad'` (similar to dimensions for length or mass), but rather defines the fundamental angular unit in terms of meter over meter or, in effect, `1`.

The additional angular units, degree and revolution, are defined respectively as:

```
pm_addunit('deg', pi/180, 'rad');
pm_addunit('rev', 2*pi, 'rad');
```

As a result, forward trigonometric functions, such as `sin`, `cos`, and `tan`, work directly with arguments expressed in angular units. For example, cosine of 90 degrees equals the cosine of (pi/2) radians and equals the cosine of (pi/2). Expansion of forward trigonometric functions works in a similar manner.

Another effect of dimensionless implementation of angular units is the convenience of the work-energy conversion. For example, torque (in N*m) multiplied by angle (in rad) can be added directly to energy (in J, or N*m). If you specify other commensurate units for the components of this equation, Simscape unit manager performs the necessary unit conversion operations and the result is the same.

## References

[1] *The NIST Reference on Constants, Units, and Uncertainty*, https://physics.nist.gov/cuu/Units/units.html

# Units for Angular Velocity and Frequency

Angular velocity units, such as `rad/s`, `deg/s`, and `rpm`, can also be used to measure frequency for cyclical processes. This is consistent with frequency defined as revolutions per second in a mechanical context, or cycles per second in an electrical context, and lets you write frequency-dependent equations without requiring the `2*pi` conversion factor. In the SI unit system, however, the unit of frequency is hertz (Hz), defined as `1/s`.

Simscape software defines the unit hertz (`Hz`) as `1/s`, in compliance with the SI unit system. This definition works well when frequency refers to a nonrotational periodic signal such as the frequency of a PWM source. For cyclical processes, however, the block equations have to contain the `2*pi` conversion factor, to convert the numerical value specified in `Hz`, or $s^{-1}$, to angular frequency.

As a result, frequency units (based on `Hz`) and angular velocity units (based on `rpm`) are not directly convertible, and using one instead of the other may result in unexpected conversion factors applied to the numerical values by the block equations. For example, the AC Voltage Source block explicitly multiplies the value you specify for its **Frequency** parameter by `2*pi`, to convert it to angular frequency before calculating the sine function.

Drop-down lists of suggested units in block dialogs and Property Inspector reflect this distinction. For example, if a block has a **Frequency** parameter with the default unit of `Hz`, the drop-down list for this parameter contains only units directly convertible to Hz (such as `kHz`, `MHz`, and `GHz`) and does not contain the angular velocity units. Conversely, if you define a custom block where the **Frequency** parameter has the default unit of `rpm`, its drop-down list of suggested units will include `deg/s` and `rad/s`, but will not contain Hz, `kHz`, `MHz`, or `GHz`.

When you type a unit expression in the parameter units combo box (instead of selecting a value from the drop-down list), the Simscape unit manager considers the units of frequency and angular velocity to be commensurate. For example, when the default parameter unit is `Hz`, you are able to type not only `1/s`, but also expressions such as `deg/s` and `rad/s`. This behavior is consistent with the Simscape implementation of angular units (see "Angular Units" on page 15-14). It is your responsibility to verify that the unit expression you typed works correctly with the block equations and reflects your design intent.

**Note** Prior to Release R2013a, the unit definition for Hz was `rev/s`. For information on how to update legacy models and custom Simscape libraries written in R2012b or earlier, see Compatibility Considerations under "Unit definition of Hz now consistent with SI", in the R2013a Release Notes.

# Working with Simulink Units

You can specify physical units on Simulink signals. For details, see "Units in Simulink".

Interface blocks, such as Simulink-PS Converter and PS-Simulink Converter, handle the boundary between the Simscape physical network and the Simulink blocks connected to it. These converter blocks also handle the physical signal unit conversions:

- On a Simulink-PS Converter block, you specify the unit using the **Input signal unit** parameter. This unit must be commensurate with the unit of the physical signal at the **PS** output port of the block, which is inferred from the signal destination inside the physical network. The **Input signal unit** parameter provides information to the Simscape unit manager, which performs the necessary unit conversion and scales the signal value accordingly.

- On a PS-Simulink Converter block, you specify the unit using the **Output signal unit** parameter. This unit must be commensurate with the unit of the input physical signal coming into the block. The block applies a gain equal to the conversion factor before outputting the Simulink signal.

If you specify a physical unit on a Simulink signal connected to a Simulink-PS Converter or a PS-Simulink Converter block, the software compares this unit with the unit specified inside the block. If the parameter value does not match the physical unit of the Simulink signal connected to the block, you get a warning.

When you add a new unit to your Simscape unit registry, by using the `pm_addunit` function, and use this unit inside a Simulink-PS Converter or PS-Simulink Converter block:

- If your unit definition conflicts with the one in the Simulink database, you get a warning about incompatible unit.

- If you add a unit that does not exist in the Simulink database, you get a warning about undefined unit.

Note that these warnings apply only to the Simulink database; the Simscape physical network works as expected.

For example, you want to view the motor speed in revolutions per second, rather than revolutions per minute (`rpm`):

**1** Add a new unit `rps`, defined in terms or `rpm`:

    pm_addunit('rps', 1/60, 'rpm');

**2** To open the Permanent Magnet DC Motor example model, in the MATLAB Command Window, type:

    ssc_dcmotor

**3** Simulate the model. Examine the simulation results in the Motor RPM scope window.
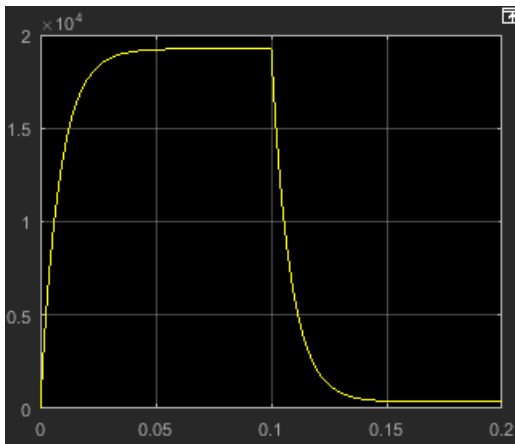
**4** Open the Sensing subsystem (designated as w in the block diagram), double-click the PS-Simulink Converter block and type `rps` as the **Output signal unit** parameter value.
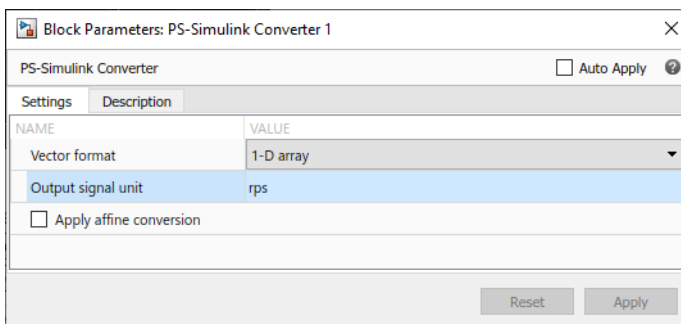


**5** Rerun the simulation.

The model works correctly, with the scope displaying the results in revolutions per second.
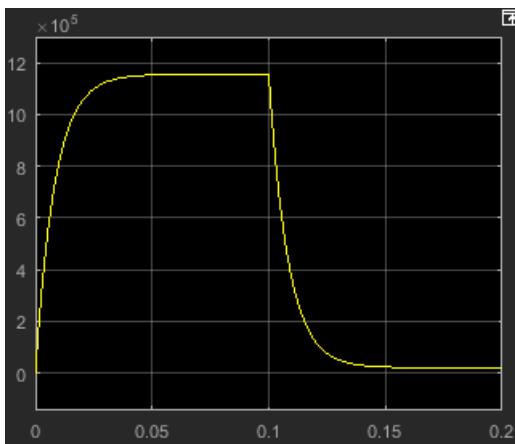


However, the output Simulink signal of the PS-Simulink Converter block now displays a warning badge, with a message `The units 'rps' are undefined`. The detailed message explains that the units are not defined in the Simulink unit database.
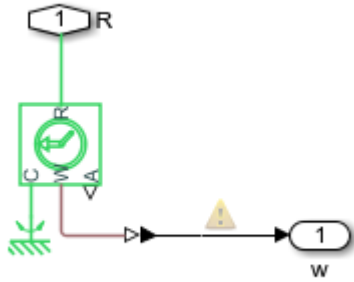
If you issue a `pm_getunits` command, you can see `rps` in the return unit list, which means that the unit is successfully defined in the Simscape unit registry. In other words, the warning applies only to Simulink unit checking.

**6**   To turn off the unit inconsistency warnings, in the MATLAB Command Window, type:

```
set_param('ssc_dcmotor','UnitsInconsistencyMsg','none');
```

Another way to avoid the unit inconsistency warnings is to add the same units to your Simulink unit database. For information on how to create and load a custom Simulink unit database, see "Working with Custom Unit Databases".

# Working with simscape.Value and simscape.Unit Objects

| **In this section...** |
| --- |
| |
| |
| |
| |
| |
| |

In physical modeling, block parameters, variables, and physical signals are represented as a value with associated unit. Simscape unit manager automatically performs the necessary unit conversion operations when solving a physical network. `simscape.Value` and `simscape.Unit` objects essentially implement a MATLAB interface that replicates the unit manager functionality outside of Simscape:

- `simscape.Value` binds arrays of arithmetic values to units and propagates those units through mathematical operations. All members of an array must have the same unit.
- `simscape.Unit` represents units of measure without an associated value, and therefore lets you write MATLAB functions that emulate the unit propagation behavior.

Use `simscape.Value` and `simscape.Unit` to:

- Preprocess or postprocess simulation data with units in MATLAB, for example, calculate total fluid mass or plot vehicle dynamics.
- Create value with unit objects in MATLAB and manipulate them during programmatic model construction.
- Write MATLAB functions that operate on values with units.

For example, this function computes page-wise matrix multiplication while propagating units:

```
function out = mypagemtimes(a, b)
  out = simscape.Value(pagemtimes(value(a),value(b)),unit(a)*unit(b));
end
```

The function:

- Takes two `simscape.Value` objects, `a` and `b`, as arguments.
- Extracts values from `a` and `b`.
- Uses the MATLAB `pagemtimes` function on these extracted values.
- Uses the `simscape.Unit` objects to compute the derived unit.
- Reattaches the new unit to the computed array of values and returns the resulting `simscape.Value` object.

## Core MATLAB Functions Supporting simscape.Value Arrays

You can use core MATLAB array functions when working with `simscape.Value` arrays. The tables list additional restrictions that pertain to using `simscape.Value` arguments.

**`simscape.Value` Math Functions**

| Name | Description |
|---|---|
| abs | Absolute value |
| acos | Inverse cosine |
| asin | Inverse sine |
| atan | Inverse tangent |
| atan2 | Four quadrant inverse tangent |
| cos | Cosine |
| cosh | Hyperbolic cosine |
| cross | Cross product |
| diff | Difference |
| dot | Dot product |
| exp | Exponential |
| log | Natural logarithm |
| log10 | Common (base 10) logarithm |
| max | Maximum elements of an array |
| min | Minimum elements of an array |
| mod | Modulus after division |
| prod | Product of elements |
| sign | Signum function |
| sin | Sine |
| sinh | Hyperbolic sine |
| sqrt | Square root |
| sum | Sum of elements |
| tan | Tangent |
| tanh | Hyperbolic tangent |

**`simscape.Value` Operations**

| Name | Description |
|---|---|
| + (plus), - (minus) | Unary and binary plus and minus |
| * (mtimes), .* (times) | Multiplication |
| / (mrdivide), \ (mldivide), ./ (rdivide), .\ (ldivide) | Division |
| == (eq), ~= (ne), < (lt), <= (le), > (gt), >= (ge) | Relational |
| ^ (mpower), .^ (power) | Power |

**`simscape.Value` Inspection**

| Name | Description |
|---|---|
| isequal | Compares two `simscape.Value` objects. Returns 1 if both arrays are the same size, contain the same values, and have the same unit, and 0 otherwise. |
| isequaln | Compares two `simscape.Value` objects, treating NaN elements as equal to each other. Returns true if both arrays are the same size, contain the same values, and have the same unit, and 0 otherwise. |
| isfinite | Returns 1 for finite array elements, and 0 otherwise. |
| isinf | Returns 1 for infinite array elements, and 0 otherwise. |
| isnan | Returns 1 for NaN array elements, and 0 otherwise. |

**`simscape.Value` Type Conversions**

| Name | Description |
|---|---|
| double | For a dimensionless argument only, converts array elements to double precision. Dimensionless quantities are implicitly converted to unitless before conversion to double. For example:<br><br>`double(simscape.Value(1,'m')/simscape.Value(2,'cm'))`<br><br>`ans =`<br><br>   `50` |
| logical | Converts numeric values to logicals. Supported for `simscape.Value` objects with nonaffine units. Zero values, regardless of linear unit, are treated as false and nonzero values are treated as true. |

## Core MATLAB Functions Supporting simscape.Unit Arrays

The basic multiplication, division, power, and relational operations are defined on `simscape.Unit`. The table lists additional restrictions that pertain to using `simscape.Unit` arguments.

**`simscape.Unit` Arithmetic and Relational Operations**

| Name | Description |
|---|---|
| times | Elementwise unit products |
| mtimes | Matrix unit products |
| rdivide, ldivide | Elementwise unit quotients |
| mrdivide, mldivide | Matrix unit quotients. Supported, but denominator must be a scalar `simscape.Unit` object. |
| power | Elementwise power. Base must be a `simscape.Unit` object. Exponent must be a compatibly sized numeric array. See also "Fractional and Rational Exponents" on page 15-23. |
| mpower | Matrix power. Supported, but requires base and exponent to be scalar. |
| eq, ne | Determine equality or inequality. Compares units in canonical form. |
| isequal | Determine array equality. True if the unit arrays are the same size and all units are equivalent. |

**`simscape.Unit` Type Conversions**

| Name | Description |
|---|---|
| string | Converts each element of a unit array into a string. Returns a string array. |
| cellstr | Converts a unit array into a cell array of character vectors. |
| char | Converts a unit array into a character array. |

For example, create a 2x3 unit array:

```
u1 = simscape.Unit(["kg", "g", "lb"; "km", "m", "mm"])

u1 =

    2×3 unit array

    kg    g    lb
    km    m    mm
```

Convert unit array into a string array:

```
string(u1)

ans =

  2×3 string array

    "kg"    "g"    "lb"
    "km"    "m"    "mm"
```

Convert the same unit array into a cell array of character vectors:

```
cellstr(u1)

ans =

  2×3 cell array

    {'kg'}    {'g'}    {'lb'}
    {'km'}    {'m'}    {'mm'}
```

Convert the same unit array into a character array:

```
char(u1)

ans =

  6×2 char array

    'kg'
    'km'
    'g '
    'm '
    'lb'
    'mm'
```

## Computational Units

When a math operation, such as addition, requires a computational unit, the return unit is the operand unit whose conversion factor to the fundamental unit is the largest. For example:

```
simscape.Value(2.2,'cm') + simscape.Value(2,'mm')

ans =

    2.4000 : cm
```

The fundamental unit for length is m. The conversion factor of cm into m is larger than that of mm into m, therefore, the return unit when adding cm and mm is cm.

Multiplication and division operations do not require a computational unit. These operations are performed by multiplying (dividing) the values and multiplying (dividing) the units. For example:

```
simscape.Value(2.2,'N') * simscape.Value(2,'m')

ans =

    4.4000 : N*m
```

## Fractional and Rational Exponents

Fractional exponents are supported for simscape.Value objects. The numeric value is raised to the double power specified by the function and a rational approximation of the fractional exponent is applied to the units.

For simscape.Unit objects, exponents must be rational numbers. When computing unit powers, simscape.Unit uses the same rational fraction approximation as the MATLAB rat function.

## Operations with Affine Units

`simscape.Value` objects can have affine units. For more information, see "About Affine Units" on page 15-11.

The following operations are prohibited on values with affine unit:

- Forming compound units that include affine units.
- Basic array operations that require unit conversion.
- Implicit or explicit comparison to 0.

## Limitations

- Direct block parameterization is not supported, that is, you cannot use `simscape.Value` or `simscape.Unit` objects directly to specify block parameters. You can use these objects only during programmatic model construction.

  For example, to programmatically set a block parameter based on a `simscape.Value` object:

  ```
  x = simscape.Value(V, U);
  set_param(block, 'x', mat2str(value(x)));
  set_param(block, 'x_unit', char(unit(x)));
  ```

- You cannot use `simscape.Value` or `simscape.Unit` objects to specify values with units or perform unit computations in Symbolic Math Toolbox™.
- MATLAB Coder does not support `simscape.Value` and `simscape.Unit` objects.

# Scalable Compilation

# About Scalable Compilation

| **In this section...** |
| --- |
| |
| |

## What Is Scalable Compilation?

Large models often take a long time to compile and simulate. Scalable compilation helps reduce compilation time for models that consist of a pattern of repeated components by compiling a repeated component once and then reusing these compilation artifacts for other instances of the same component. Scalable compilation improves compilation performance, it does not reduce the simulation time of the model.

Scalable compilation supports these types of reusable components:

- Referenced subsystems — Subsystem instances using a subsystem reference, where you save the contents of a subsystem in a separate file and reference it using a Subsystem Reference block. You can create multiple instances referencing the same subsystem file. For more information, see "Subsystem Reference".

- Linked subsystems — Subsystem instances created from Simulink library Subsystem blocks with link to the source. Note that if you create copies from Subsystem blocks, with `LinkStatus` set to `none`, they are not linked subsystems.

- Individual blocks where `simscape.scalable.getBlockConfig` returns `on`. For more information, see "Reuse Compilation Artifacts of Individual Simscape Blocks" on page 16-16.

- Textual components with the `CompileReuse` attribute set to `true`. For more information, see "Reuse Compilation Artifacts of Textual Components" on page 16-18.

If you have a slow-to-compile model, determine whether it can benefit from scalable compilation by asking yourself the following questions:

- Does the model consist of a pattern of repeated components, such as a transmission line or a battery pack? Can you easily turn these components into reusable components, such as referenced subsystems or linked subsystems?

- How much does the Simscape part of the model contribute to total model compilation time? Some models have complicated controllers that take a long time to compile. In such cases, even if scalable compilation can significantly reduce the compilation time of the Simscape part of the model, the impact on the entire model could be less noticeable.

- Does the model use unsupported patterns, optimizations, or simulation modes? For example, scalable compilation does not support code generation, including rapid accelerator mode, or the Partitioning local solver. For more information, see "Scalable Compilation Limitations" on page 16-10.

The Advisory tool can provide information about unsupported patterns or workflows in your model, as well as guidance regarding subsystem reusability and compilation statistics. For more information, see "Analyze the Model Using the Advisory Tool" on page 16-6.

The complexity level of the repeated components has significant impact on the scalable compilation results. You can try different ways of restructuring your model into repeated components to

determine the optimal configuration. For more information, see "Determining Optimal Complexity Level for Reusable Components" on page 16-11.

## Scalable Compilation Workflow

The flowchart represents the scalable compilation workflow.
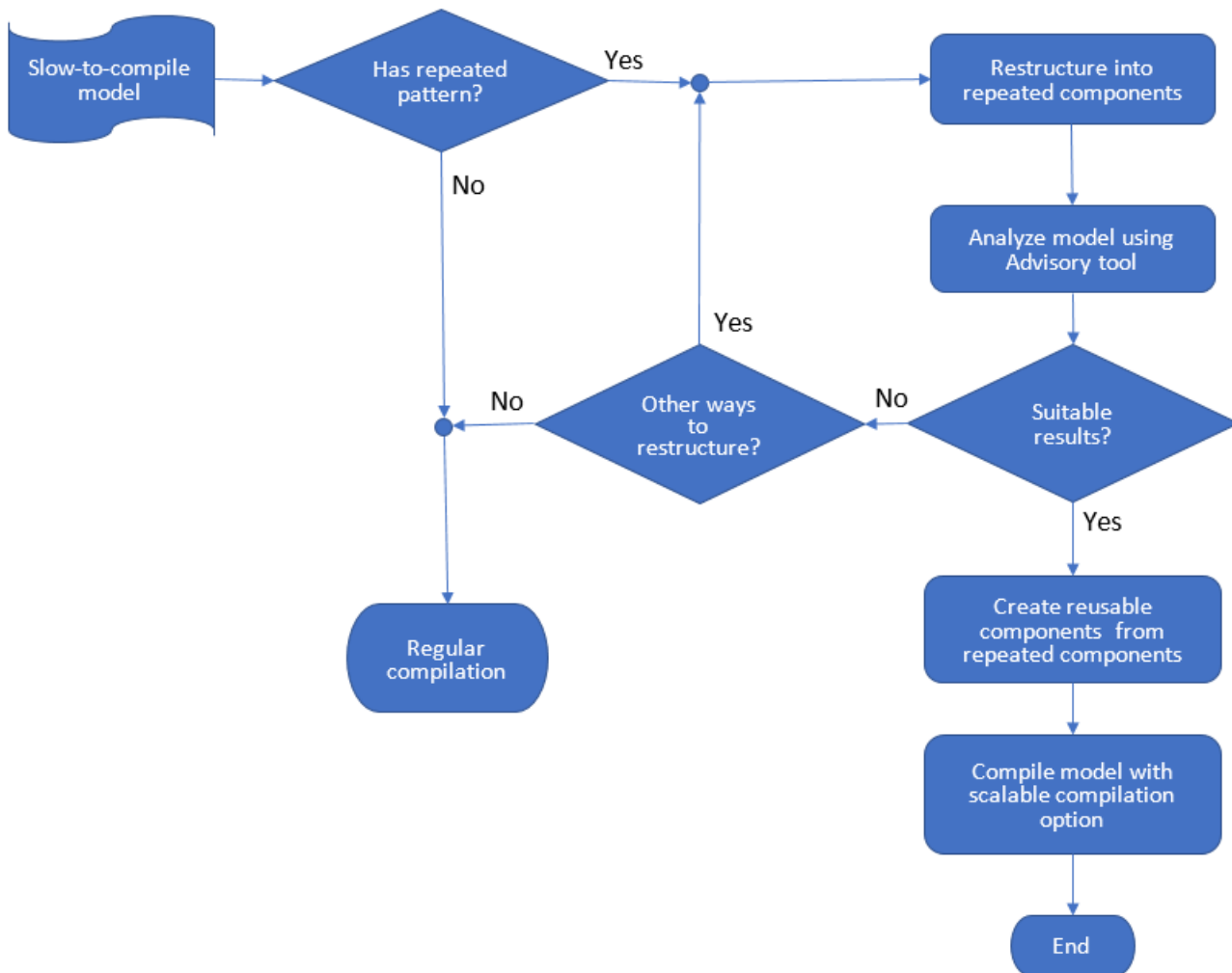


If you have a slow-to-compile model:

**1**  Determine whether the model is a suitable candidate for scalable compilation. Does it have a repeated pattern of components consisting of Simscape blocks?

**2**  If yes, identify a way to restructure the model into repeated components. Some models do not need restructuring because they already consist of a pattern of repeating subsystems. Other models may need reorganization into repeating subsystems. You do not have to create referenced subsystems or linked subsystems at this point. The Advisory tool lets you analyze your model as if it already contained reusable components.

**3**  Analyze the model using the Advisory tool and review the results.

If the performance improvement from scalable compilation is not satisfactory, consider whether there are other possible ways to restructure the model. The complexity level of the repeated components has significant impact on scalable compilation results. You can try different ways of restructuring your model into repeated components and use the Advisory tool to determine the optimal configuration.

4   Once you are satisfied with the results, turn the repeated components in your model into reusable components. For example, convert one of the repeated components into a referenced subsystem, and then replace all instances of this component in the model with Subsystem Reference blocks. For more information, see "Subsystem Reference".

## See Also

## More About

- "Preparing Your Model for Scalable Compilation" on page 16-5

# Preparing Your Model for Scalable Compilation

| In this section... |
| --- |
| "Explore and Restructure the Model" on page 16-5 |
| "Analyze the Model Using the Advisory Tool" on page 16-6 |
| "Create Reusable Components" on page 16-7 |
| "Turn On Scalable Compilation" on page 16-8 |

This example shows how to analyze and restructure a model to prepare it for scalable compilation.

## Explore and Restructure the Model

To open the Transmission Line example model, type `ssc_transmission_line` in the MATLAB Command Window. Then, save the model under a new name (for example, `transmission_line`) in your work folder, to be able to make modifications to it.
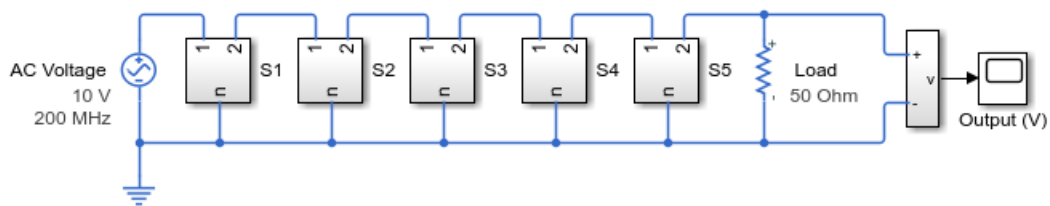


**Transmission Line**
1. Plot voltages along line (see code)
2. Explore simulation results using sscexplore
3. Open transmission line component library
4. Learn more about this example

This example models a transmission line by concatenating 50 identical T-section segments. A custom Simscape block, T-Section Transmission Line, represents a single T-section segment. Each T-section segment is 0.1 m long, therefore the transmission line represents a 5 m length of coaxial cable. The model contains five identical subsystems, named S1 through S5, with each subsystem consisting of ten T-section segment blocks, named T-1 through T-10.
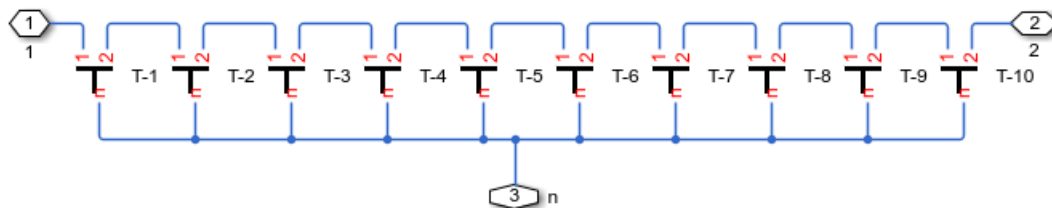


The preliminary analysis shows that the model:

• Has a repeating pattern of Simscape blocks

• Contains very few Simulink blocks, which means that the Simscape part of the model accounts for most of the compilation time

- Uses a variable-step solver and does not require code generation

Based on this analysis, the model is a suitable candidate for scalable compilation. Moreover, the model already consists of a pattern of repeating subsystems and does not need any additional restructuring at this point. The Advisory tool lets you analyze your model as if it already contained reusable components in place of repeating subsystems.

## Analyze the Model Using the Advisory Tool

The next step is to analyze the model using the Advisory tool. You can call the `sscScalableAdvisor` function using the following syntax:

```
sscScalableAdvisor(modelname,subsyspaths)
```

where:

- `modelname` is the name of the model to analyze, specified as a character vector, string, or a handle to the model. This argument is required.
- `subsyspaths` is an optional argument that identifies the repeating subsystems in the model. Specify this argument as a cell array of subsystem names, including the path to each subsystem from the root of the model, to have the compilation artifacts be reused among subsystems with identical contents.

  If the model contains reusable components, the tool recognizes them automatically. Reusable components are individual Simscape blocks or textual components designated as reusable, referenced subsystems, or linked subsystems. If the model contains other types of subsystems, you must provide their names with the `subsyspaths` argument for the Advisory tool to consider them as repeating components.

In this example, you provide the names of all five subsystems, S1 through S5, in a cell array as the second input argument:

```
r = sscScalableAdvisor('transmission_line', ...
{'transmission_line/S1', 'transmission_line/S2', 'transmission_line/S3', ...
'transmission_line/S4', 'transmission_line/S5',})

r =

ScalableReport for model 'transmission_line'

          TotalModelCompilationTime: 7.2313
            SimscapeCompilationTime: 6.7703
                         PeakMemory: '27 MB'

    ScalableSimscapeCompilationTime: 3.5232
                 ScalablePeakMemory: '19 MB'

                         Subsystems: [1×4 table]
                         Components: [0×4 table]
```

The Advisory tool returns the compilation statistics. If the model contains unsupported patterns, workflows, or optimizations, the Advisory tool recommends to not enable scalable compilation and lists the applicable limitations. In this case, however, none of the limitations apply, and the compilation time is significantly reduced. The compilation time for the Simscape part of the model is

3.5232 s, instead of 6.7703 s, which brings the total compilation time down by about 45% (7.2313 - 6.7703 + 3.5232 = 3.9842 s).

---

**Note**

- Model compilation time depends on multiple factors, such as the processor used, other processes running at the same time, memory caching, and so on. The exact numbers may vary slightly between successive runs of the Advisory tool, but the ratio between different parts of the compilation process remains the same. Use this data to estimate the potential gain from scalable compilation.

- If the compilation time for a model is reduced by more than 5 seconds, the Advisory tool recommends to enable scalable compilation, opens the Configuration Parameters dialog box, and highlights **Reuse components during compilation** to simplify your workflow. However, 5 seconds is an arbitrary number. Use the Advisory tool to estimate the potential gain from scalable compilation, and if you decide that the results are satisfactory, follow the rest of the steps in this example and enable scalable compilation as described in "Turn On Scalable Compilation" on page 16-8.

---

You can also view the data on subsystem reuse:

```
r.Subsystems

ans =

  1×4 table

                          Total Instances    Compiled Instances    Reuse      Details
                          _____    _____    _____      _____

    transmission_line/S1          5                  1            "100%"    {1×5 table}
```

In this model, all five subsystems are identical, and therefore the reuse is 100% (the five instances are replaced with one compiled instance). In other models, the reuse statistics might be different. For example, if the subsystems have different configurations or parameterizations, the Advisory tool report indicates how many compiled instances are needed and calculates the reuse statistics accordingly. In that case, the `Details` table provides more information.

---

**Note** The `Components` field in the Advisory tool report provides reuse data for individual Simscape blocks or textual components designated as reusable. This model does not contain reusable Simscape blocks or textual components, therefore, the `Components` table is empty.

---

## Create Reusable Components

Because the compilation results provided by the Advisory tool in the previous step are satisfactory, you do not need to consider other ways to restructure the model. You now need to prepare the model for scalable compilation by actually replacing the repeated subsystems in the Advisory tool analysis with reusable components, in this case, with reference subsystems:

**1**  Right-click subsystem S1.

**2**  From the context menu, select **Subsystem & Model Reference > Convert to > Referenced Subsystem**.

**3**  Keep **Subsystem file name** as S1. Use the **Browse** button if you want to save the subsystem in a different folder.

**4**  Click **Convert**. As a result, the contents of subsystem S1 are saved as a separate block diagram file S1.slx in the folder you specified, and the subsystem S1 in the model is replaced with the referenced subsystem.
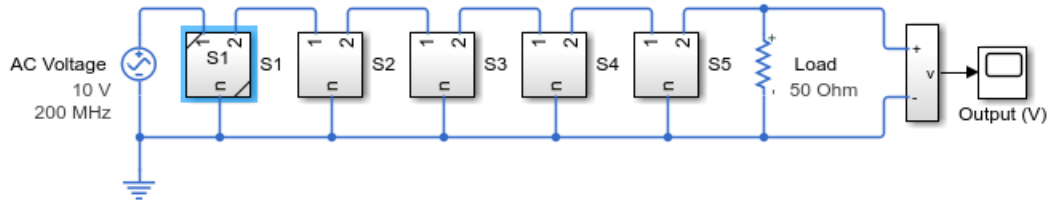


**5**  Delete subsystems S2, S3, S4, and S5.

**6**  Right-click subsystem S1 and drag it in place of subsystem S2.
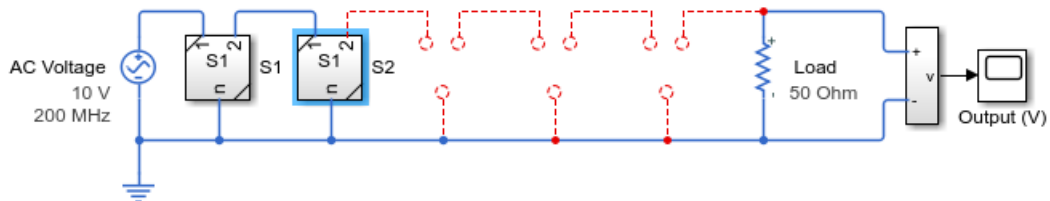


**7**  Repeat for subsystems S3, S4, and S5.
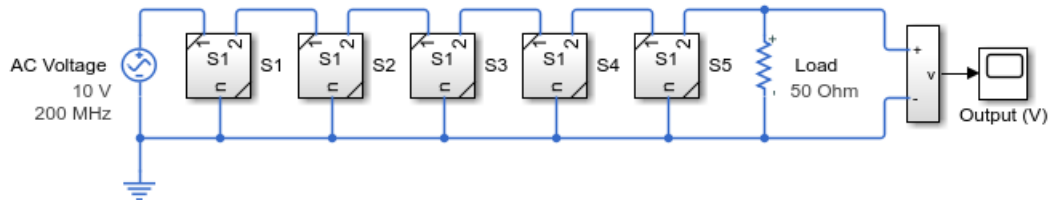


Notice that your model still contains subsystems named S1, S2, S3, S4, and S5, but they now all reference the same subsystem S1, saved as a separate block diagram file.

## Turn On Scalable Compilation

To turn on scalable compilation for a model:

**1**  Open the Configuration Parameters dialog box.

**2**  On the **Simscape** pane, select the **Reuse components during compilation** check box.
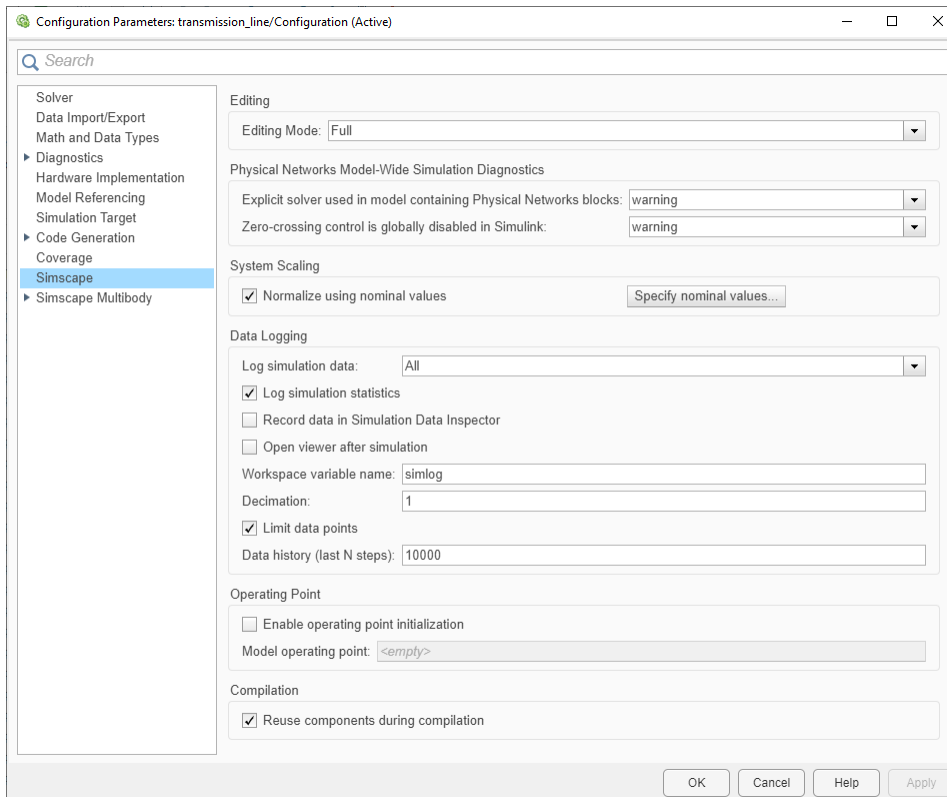
You can also use the equivalent command-line interface to set the model configuration parameter:

```
set_param(bdroot,'SimscapeCompileComponentReuse','on')
```

# See Also

# More About

- "About Scalable Compilation" on page 16-2

# Scalable Compilation Limitations

Scalable compilation reduces compilation time for models that consist of a pattern of repeated components by compiling a repeated component once, and then reusing these compilation artifacts for other instances of the same component. Besides individual Simscape blocks and textual components, scalable compilation supports only two types of reusable subsystems: referenced subsystems and linked subsystems. Other types of graphical subsystems must be converted to one of these types to take advantage of scalable compilation.

Different instances of reusable subsystems can have different parameter values, by utilizing system masks. For more information, see "Reference a Subsystem File in a Model". You can also parameterize a variable initialization target value within a reusable subsystem and supply different values of that initialization target to different subsystem instances, for example, from a mask parameter or from the base workspace. However, other properties of a variable, such as the initialization priority or nominal value, have to be the same in all the reusable subsystem instances.

Additionally, scalable compilation does not support the following simulation tools and workflows:

- The Partitioning local solver
- Nonlinear index reduction, which is a global transformation used for solving high-index nonlinear equations
- Stiffness impact analysis
- Rapid accelerator mode
- Code generation

The Advisory tool can provide information about unsupported patterns or workflows in your model. If the model contains unsupported patterns, workflows, or optimizations, the Advisory tool lists the applicable limitations and returns the recommendation not to enable scalable compilation.

## See Also

## More About
- "About Scalable Compilation" on page 16-2

# Determining Optimal Complexity Level for Reusable Components

The complexity level of the repeated components has significant impact on the scalable compilation results. You can try different ways of restructuring your model into repeated components to determine the optimal configuration.

For example, consider this battery pack model.
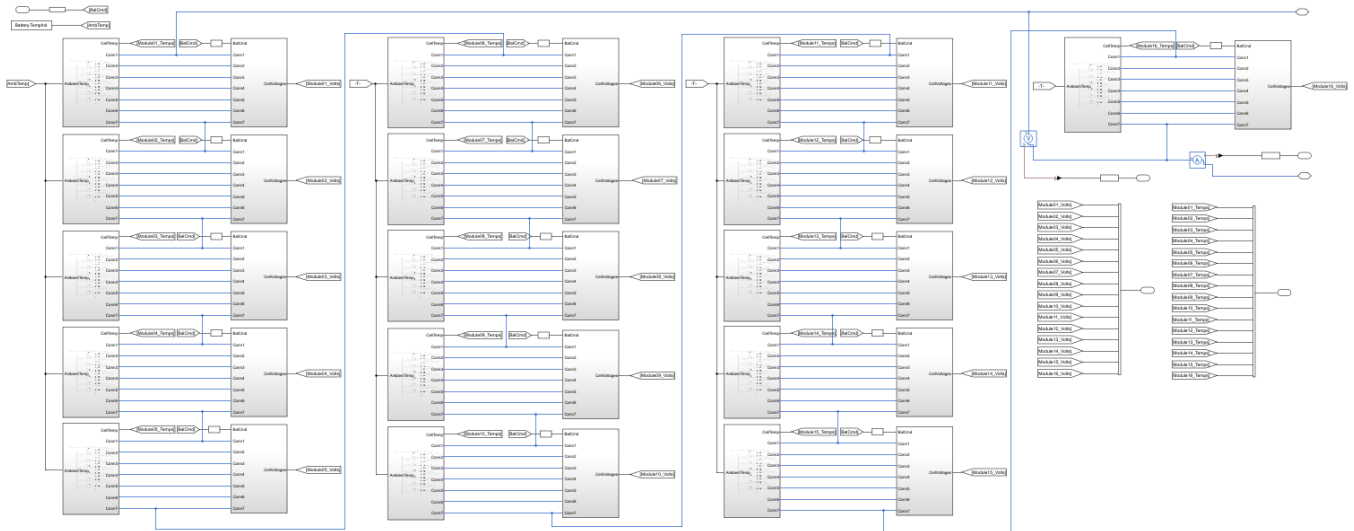


The entire model has a controller and a plant. The plant model, shown in the previous figure, is a battery pack that consists of 16 identical battery modules. Each of these battery modules, in turn, consists of two interconnected subsystems: a battery cell and a monitoring unit, as shown in the next figure.
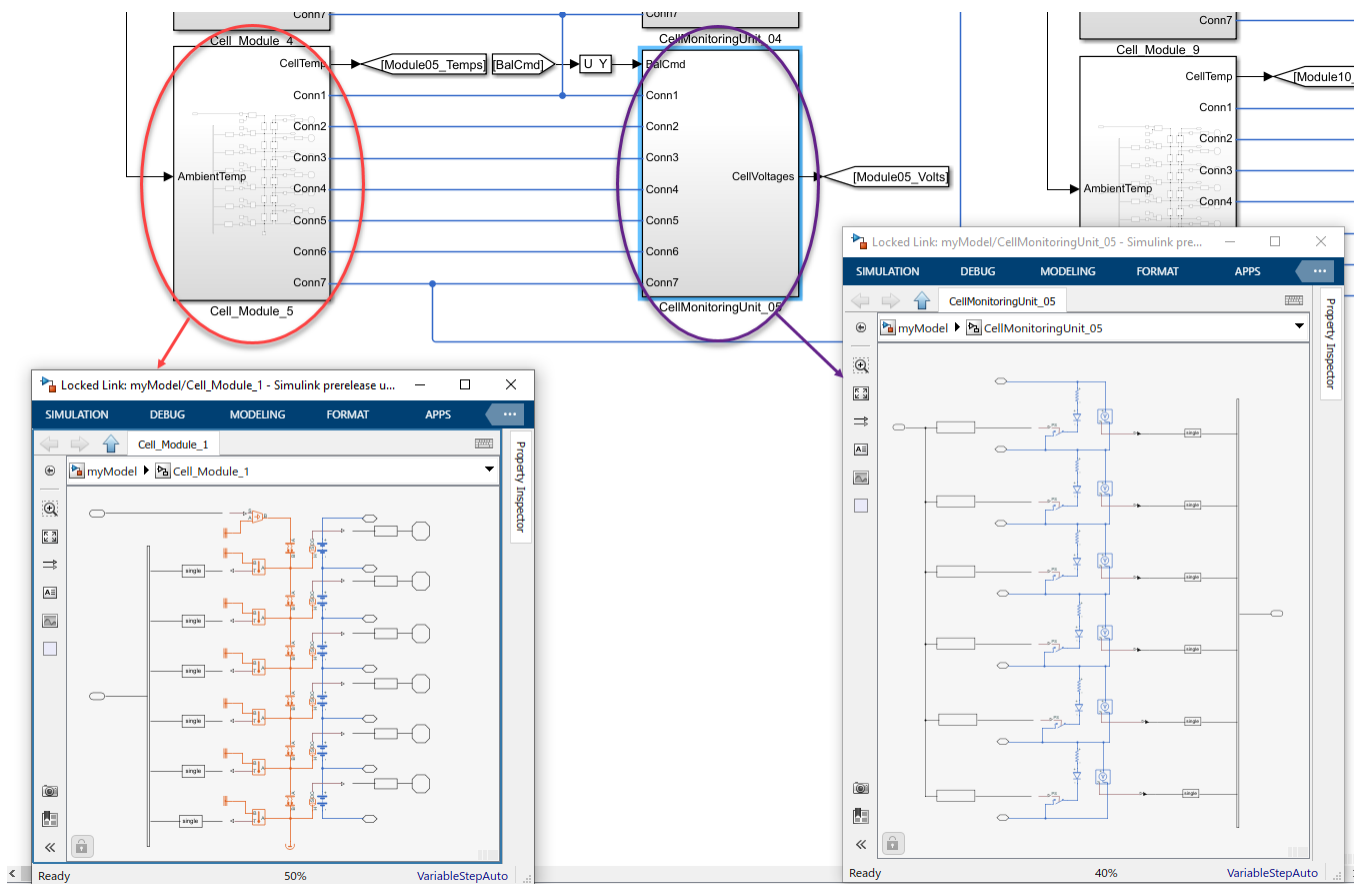
Because the complexity of a reusable component has significant impact on scalable compilation performance, the model might need to be restructured into a different set of reusable components to improve performance. One important factor that has a negative impact on performance is the number of the interface variables (external connections) of a reusable component. Because of that, scalable compilation is likely to be more efficient if you base the reusable components on a module consisting of the two facing subsystems, rather than on each of the individual battery and monitoring unit subsystems.

Use the Advisory tool to analyze the different restructuring options and determine the optimal configuration:

**1** Run `sscScalableAdvisor` on the model and supply the names of the 32 individual subsystems (16 battery cells and 16 monitoring units), in a cell array, as the second input argument. Review the compilation statistics and component reusability results.

**2** Try a different configuration. Using the **Create Subsystem from Selection** option, create a virtual subsystem from each pair of the facing subsystems, so that each battery module is represented by one subsystem.
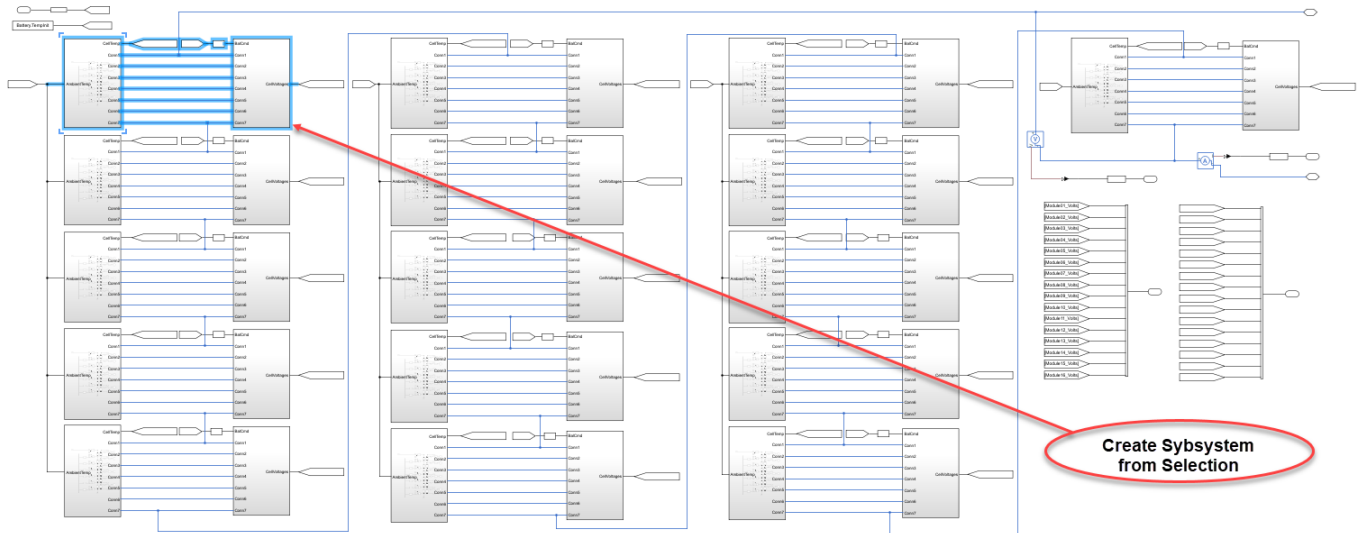
The restructured model may look like this.
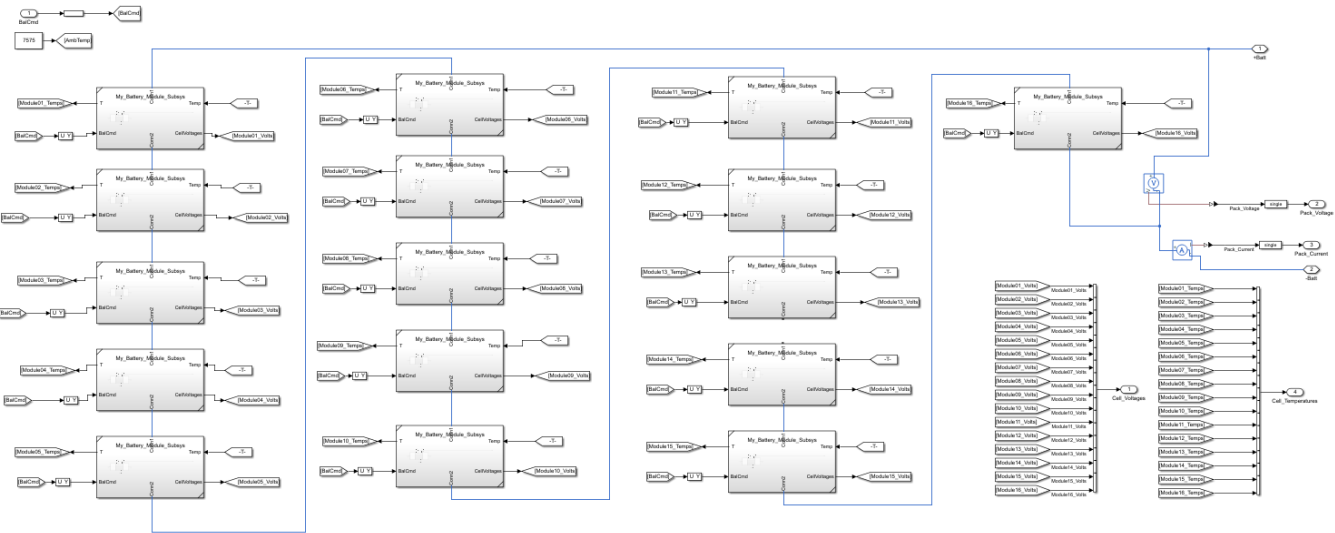


Run `sscScalableAdvisor` again, and this time supply a cell array that contains the 16 module subsystem names as the second argument. Each of the module subsystems is more complex and has fewer external connections than the individual batteries and monitoring units, and therefore the scalable compilation performance is likely to be much better.

**3**  You can probably improve the compilation performance even more by restructuring the model into eight repeated subsystems instead of 16. Analyze the performance results and make a decision based on both compilation performance and maintaining the modeling hierarchy. Even if restructuring the model into eight reusable components compiles slightly faster, you might want to maintain the structure where one battery module corresponds to one subsystem.

Once you have determined the optimal model configuration, replace the repeated subsystems with reusable components and enable scalable compilation, as described in the last two steps of "Preparing Your Model for Scalable Compilation" on page 16-5.

## Selectively Disable Scalable Compilation of Certain Subsystems

Scalable compilation is hierarchical. When you enable scalable compilation of a model, the compiler tries to reuse compilation artifacts of every referenced subsystem and linked subsystem in the model hierarchy. However, for models with a deep hierarchy of referenced or linked subsystems this default behavior might not be optimal.

For example, in the battery model discussed earlier, the individual battery cells and monitoring units are virtual subsystems. Consider instead that these components are represented by referenced subsystems or linked subsystems. In this case, restructuring the model into 16 modules would not improve the scalable compilation results because the compiler would treat each of the 16 modules as consisting of two reusable components. To get the benefit of fewer external connections and increased complexity of each of the reusable components, in addition to restructuring the model into 16 modules you must disable scalable compilation of the underlying subsystems in each module.

Similarly, if your model contains linked subsystems with deep hierarchy, such as linked libraries, disable scalable compilation inside the library subsystem. For example, if you have a refrigerator model with several identical tanks and each tank, in turn, uses a linked library of water pipes, you might want to disable scalable compilation of these linked libraries. This way, the compiler treats the whole tank as one reusable component, rather than trying to reuse compilation artifacts from within the water pipe library.

To disable scalable compilation for a specific referenced subsystem or linked subsystem within a model, type :

```
simscape.scalable.setSubsystemConfig(blockPath,'off')
```

where `blockPath` is the path to an instance of a Subsystem Reference block, or a Subsystem block with a library link, from the root of the model. This setting affects all instances of this subsystem in the model.

For example, typing:

```
simscape.scalable.setSubsystemConfig('Refrigerator/Tank1/WaterPipeLibrary','off')
```

disables scalable compilation for all instances of `WaterPipeLibrary` in all the tanks in the `Refrigerator` model.

To restore the default setting, type:

```
simscape.scalable.setSubsystemConfig(blockPath,'auto')
```

To query the subsystem setting, type:

```
setting = simscape.scalable.getSubsystemConfig(blockPath)
```

This subsystem parameter setting takes effect only if scalable compilation is enabled for the whole model. Then, the compiler tries to reuse compilation artifacts of subsystems with the `auto` setting but does not consider subsystems with the `off` setting reusable.

## See Also

## More About

*   "About Scalable Compilation" on page 16-2

# Reuse Compilation Artifacts of Individual Simscape Blocks

If your model contains multiple instances of the same Simscape block, you can use the `simscape.scalable.setBlockConfig` function to make all or some of these instances reusable.

When you enable scalable compilation for a model, all referenced subsystems or linked subsystems included in the model are by default treated by the compiler as reusable components. You can selectively disable scalable compilation of these subsystems, as described in "Determining Optimal Complexity Level for Reusable Components" on page 16-11.

In contrast, with library and custom Simscape blocks, you must enable them for scalable compilation individually. Most library blocks do not have enough complexity for a model to benefit from reusing their artifacts in scalable compilation. Use `sscScalableAdvisor` to see whether enabling scalable compilation for particular blocks reduces the overall compilation time.

To enable scalable compilation for a specific block within a model, type :

```
simscape.scalable.setBlockConfig(blockPath,'on')
```

where `blockPath` is the path to a block from the root of the model.

This setting affects only the specified instance of the block. To enable scalable compilation for all instances of a particular block in the model, use the `find_system` function in conjunction with `simscape.scalable.setBlockConfig`, for example:

```
load_system('ssc_fluid_vaporization_in_pipe')
all_pipes = find_system('ssc_fluid_vaporization_in_pipe','MaskType','Pipe (2P)')

all_pipes =

  5×1 cell array

    {'ssc_fluid_vaporization_in_pipe/Pipe Segment 1/Pipe (2P)'}
    {'ssc_fluid_vaporization_in_pipe/Pipe Segment 2/Pipe (2P)'}
    {'ssc_fluid_vaporization_in_pipe/Pipe Segment 3/Pipe (2P)'}
    {'ssc_fluid_vaporization_in_pipe/Pipe Segment 4/Pipe (2P)'}
    {'ssc_fluid_vaporization_in_pipe/Pipe Segment 5/Pipe (2P)'}

for i=1:length(all_pipes)
simscape.scalable.setBlockConfig(all_pipes{i},'on')
end
```

This series of commands enables scalable compilation for all the Pipe (2P) blocks in the model.

To restore the default setting, type:

```
simscape.scalable.setBlockConfig(blockPath,'off')
```

To query the block setting, type:

```
setting = simscape.scalable.getBlockConfig(blockPath)
```

This block parameter setting takes effect only if scalable compilation is enabled for the whole model. Then, the compiler tries to reuse compilation artifacts of individual blocks with the `on` setting, but does not consider blocks with the `off` setting reusable.

The `Components` field in the Advisory tool report provides the reuse statistics. For example:

```
r = sscScalableAdvisor('ssc_fluid_vaporization_in_pipe')

r =

ScalableReport for model 'ssc_fluid_vaporization_in_pipe'

        TotalModelCompilationTime: 9.9268
          SimscapeCompilationTime: 8.6089
                       PeakMemory: '29 MB'

  ScalableSimscapeCompilationTime: 9.0798
                ScalablePeakMemory: '17 MB'

                       Subsystems: [0×4 table]
                       Components: [1×4 table]
```

**r.Components**

```
ans =

  1×4 table
```

|  | Total Instances | Compiled Instances | Reuse | Details |
| --- | --- | --- | --- | --- |
| foundation.two_phase_fluid.elements.pipe | 5 | 1 | "100%" | {1×5 table} |

## See Also

## More About

- "About Scalable Compilation" on page 16-2

# Reuse Compilation Artifacts of Textual Components

Scalable compilation also lets you reuse compilation artifacts of repeated textual components. The CompileReuse attribute lets you specify whether the components are reusable or not. If set to true, the compilation artifacts for these components are reusable. The default is false. This attribute is available for components only.

For example, consider a model similar to the "Lithium-Ion Battery Pack with Fault Using Arrays" on page 20-49 example. (For a detailed description of this model, see "Case Study — Battery Pack with Fault Using Arrays".) The battery_pack component models the battery pack as an array of battery_cell components connected to an array of thermal conduction components from the Foundation library. In the battery_pack component file, you can use the CompileReuse attribute to specify which component arrays are reusable:

```
component battery_pack
...
for i =1:Ncells
    components(ExternalAccess=none,CompileReuse=true)
        battery_cell(i) = BatteryPack.battery_cell(cell_mass=cell_mass);
    end
    ...
end
...
for i=1:Ncells-1
    components(ExternalAccess=none)
        conduction(i) = foundation.thermal.elements.conduction(area={1e-3,'m^2'},...
            th_cond={200,'W/(m*K)'});
    end
    ...
end
...
```

If scalable compilation is enabled for a model containing this battery pack component, then the members of the first component array, battery_cell, are reusable. Members of the second array, conduction, are not designated as reusable because the conduction element in the Foundation library is not complex enough to benefit from scalable compilation.

Similar to reusable subsystems, members of reusable component arrays can also have different parameterizations. The Components field in the Advisory tool report indicates how many compiled instances are needed and calculates the reuse statistics accordingly.

## See Also

## More About

- "About Scalable Compilation" on page 16-2

# Add-On Product License Management

# About the Simscape Editing Mode

## Suggested Workflows

The Simscape Editing Mode functionality is implemented for customers who perform physical modeling and simulation using Simscape platform and its add-on products: Simscape Driveline, Simscape Electrical, Simscape Fluids, and Simscape Multibody. It allows you to open, simulate, and save models that contain blocks from add-on products in Restricted mode, without checking out add-on product licenses, as long as the products are installed on your machine. It is intended to provide an economical way to distribute simulation models throughout a team or organization.

**Note** Unless your organization uses concurrent licenses, see the Simscape product page on the MathWorks Web site for specific information on how to install add-on products on your machine, to be able to work in Restricted mode.

The Editing Mode functionality supports widespread use of Physical Modeling products throughout an engineering organization by making it economical for one user to develop a model and provide it to many other users.

Specifically, this feature allows a user, model developer, to build a model that uses Simscape platform and one or more add-on products and share that model with other users, model users. When building the model in Full mode, the model developer must have a Simscape license and the add-on product licenses for all the blocks in the model. For example, if a model combines Simscape, Simscape Fluids, and Simscape Driveline blocks, the model developer needs to check out licenses for all three products to work with it in Full mode. Once the model is built, model users need only to check out a Simscape license to simulate the model and fine-tune its parameters in Restricted mode. As long as no structural changes are made to the model, model users can work in Restricted mode and do not need to check out add-on product licenses.

Another workflow, available with concurrent licenses only, lets multiple users, who all have Simscape licenses, share a small number of add-on product licenses by working mostly in Restricted mode, and temporarily switching models to Full mode only when they need to perform a specific design task that requires being in Full mode.

**Note** It is recommended that you save all the models in Full mode before upgrading to a new version of Simulink or Simscape software.

If you have saved a model in Restricted mode and, upon upgrading to a new product version, open the model and it does not run, switch it to Full mode and save. You can then again switch to Restricted mode and work without problem.

## What You Can Do in Restricted Mode

When your model is open in Restricted mode, you can:

- Simulate the model.
- Inspect parameters.
- Change certain block parameters. In general, you can change numerical parameter values, but cannot change the block parameterization options. See the block reference pages for specifics.
- Generate code.
- Make data logging or visualization changes.
- Add or delete regular Simulink blocks (such as sources or scopes) and appropriate connections.

For other types of changes, listed in the following section on page 17-3, your model has to be in Full mode. Some of these disallowed changes are impossible to make in Restricted mode (for example, Restricted parameters are grayed out in block dialog boxes). Other changes, like changing the physical topology of a model, are not explicitly disallowed, but if you make these changes in Restricted mode, the software will issue an error message when you try to run, compile, or save such a model.

## What You Can Do in Full Mode

You need to open a model in Full mode if you need to do any of the following:

- Add or delete Physical Modeling blocks (that is, Simscape blocks or blocks from the add-on product libraries).
- Make or break Physical connections (between Conserving or Physical Signal ports).
- Change the types of signals going into actuators or out of sensors (for example, from velocity to torque).
- Change configuration parameters.
- Change block parameterization options and other restricted parameters.
- Change physical units of parameters.
- Protect a referenced model containing Physical Modeling blocks.

## Switching Between Modes

The following flow chart shows what happens when you switch between modes.
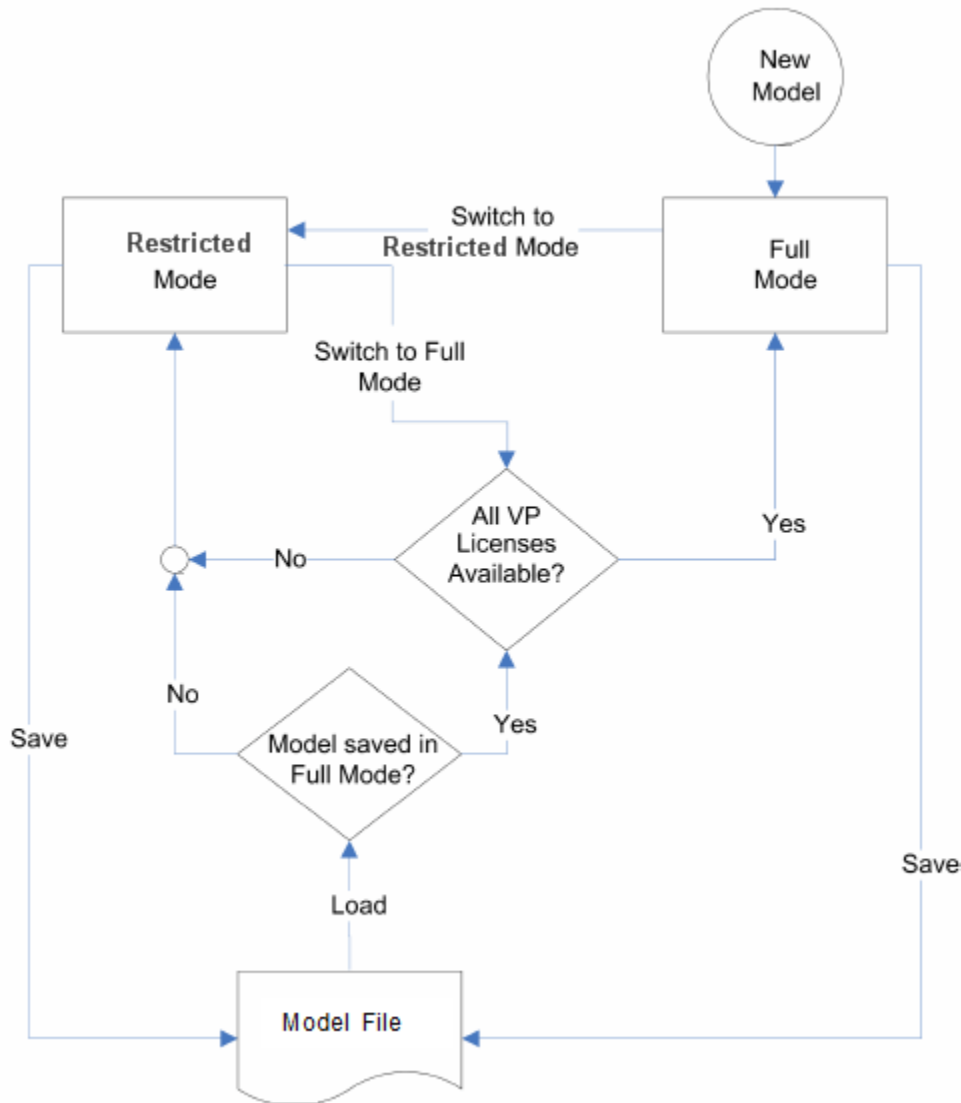
New models are always created in Full mode. You can then either save the model in Full mode, or switch to Restricted mode and save the model in Restricted mode.

When you load an existing model, the license manager checks whether it has been saved in Full or Restricted mode.

- If the model has been saved in Restricted mode, it opens in Restricted mode.
- If the model has been saved in Full mode, the license manager checks whether all the add-on product licenses for this model are available and, if so, opens it in Full mode. If a add-on product license is not available, the license manager issues an error message and opens the model in Restricted mode. See also "Example with Multiple Add-On Products" on page 17-5.

**Note** You can set a Simulink preference to specify that the models are always to open in Restricted mode, regardless of the way they have been saved.

When a model is open, you can transition it between Full and Restricted modes at any time, in either direction:

- When you try to switch from Restricted to Full mode, the license manager checks whether all the add-on product licenses for this model are available. If a add-on product license is not available, the license manager issues an error message and the model stays in Restricted mode. See also "Example with Multiple Add-On Products" on page 17-5.

- No checks are performed when switching from Full to Restricted mode.

**Note** If a add-on product license has been checked out to open a model in Full mode, it remains checked out for the remainder of the MATLAB session. Switching to Restricted mode does not immediately return the license.

**Example with Multiple Add-On Products**

When you try to open a model in Full mode or to switch from Restricted to Full mode, the license manager scans the model and attempts to check out the required add-on product licenses as it encounters them in the model. If a license is not available, the license manager issues an error message and the model stays in Restricted mode. The licenses are checked out sequentially. As a result, if a model uses blocks from multiple add-on products, some of the add-on product licenses may have already been checked out by the time the license manager encounters an unavailable license. In this case, these add-on product licenses stay checked out until you quit the MATLAB session, even though the model is in Restricted mode.

For example, consider a model that uses blocks from Simscape Fluids and Simscape Driveline libraries, but the user who tries to open it has only the Simscape Driveline license available. It may happen that the license manager checks out a Simscape Driveline license first, and then tries to check out a Simscape Fluids license, which is not available. The license manager then issues an error message and opens the model in Restricted mode, but the Simscape Driveline license stays checked out until the end of the MATLAB session.

## Working with Block Libraries

This section describes the specifics of working with block libraries while using the Editing Mode functionality. These rules are applicable to any physical modeling blocks, that is, blocks from all Simscape libraries, including the add-on products. In general, you need to work in Full mode when you modify a library block. However, when you open a model that references the modified block, you may work in Restricted mode, under certain conditions. The following summary details the Editing Mode rules for modifying and using library blocks:

- To add Simscape blocks to a library block, you need to work in Full mode.

  - If this library block had not previously contained Simscape blocks, you need to work in Full mode to load a preexisting model that uses this library block or to drag this block to a model.

  - If this library block had previously contained Simscape blocks, you can work in Restricted mode when loading a preexisting model that uses this library block. However, you have to work in Full mode to drag this block from the library to a model.

- To add external physical ports to a library block, you need to work in Full mode.

  - You can work in Restricted mode when loading a preexisting model that uses this library block.

- However, to connect these additional ports, you need to work in Full mode because you are changing the model topology.
- To delete external physical ports from a library block, you need to work in Full mode. If these ports were connected in a model saved in Restricted mode, loading the model causes the topology to change, so you need to switch to Full mode to save or compile the model.

**Resolving Block Library Links**

All Simscape blocks in your models, including the add-on products' blocks, must have resolved block library links. You can neither disable nor break these library links. This is a global requirement of Simscape platform, which is necessary to enforce the Editing Mode rules for modifying and using library blocks, listed above. A model with broken library links will neither compile nor save. You must restore all the broken block library links for your model to be valid.

If you want to customize certain blocks and use them in your models, you must add these modified blocks to your own custom library, then copy the block instances that you need to your model.

## See Also

## Related Examples

- "Set the Model Loading Preference" on page 17-7
- "Save a Model in Restricted Mode" on page 17-9
- "Work with a Model in Restricted Mode" on page 17-11
- "Switch from Restricted to Full Mode" on page 17-18

## More About

- "Get Editing Mode Information" on page 17-20

# Set the Model Loading Preference

By default, when you load an existing model, the license manager checks whether it has been saved in Full or Restricted mode and tries to open it in this mode. However, you can set your preferences so that the models are always open in Restricted mode, regardless of the way they have been saved.

**1** On the MATLAB Toolstrip, click **Preferences**. The Preferences dialog box opens.

**2** In the left pane of the Preferences dialog box, select **Simscape**. The right pane displays the **Editing Mode** group box. By default, the **Load models using** option is set to `Editing mode specified in models`.

**3** Select `Restricted mode always` from the drop-down list, as shown, and click **OK**.
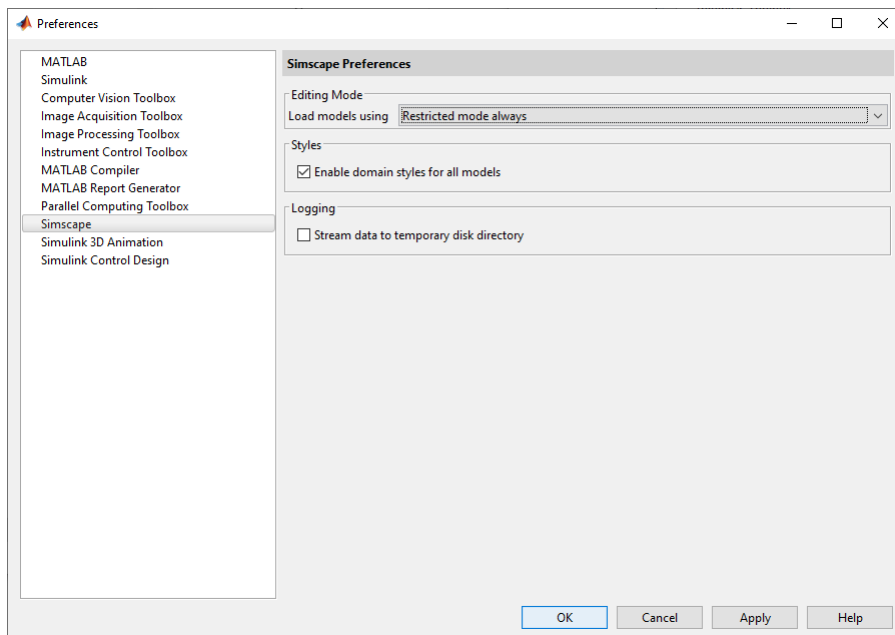


Now, when you open a model, the license manager does not attempt to check out add-on product licenses and always opens the model in Restricted mode.

## See Also

## Related Examples

- "Save a Model in Restricted Mode" on page 17-9
- "Work with a Model in Restricted Mode" on page 17-11
- "Switch from Restricted to Full Mode" on page 17-18

## More About

- "About the Simscape Editing Mode" on page 17-2
- "Get Editing Mode Information" on page 17-20
- "Domain-Specific Line Styles" on page 1-34

- "About Simscape Run-Time Parameters" on page 10-2
- "Stream Logging Data to Disk" on page 13-42

# Save a Model in Restricted Mode

Rather than setting your preferences so that all the models always open in Restricted mode, you can switch an individual model to Restricted mode before saving it. Such a model will then, by default, open in Restricted mode.

1   In the Simulink Toolstrip at the top of the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

2   In the left pane of the Configuration Parameters dialog box, select **Simscape**. The right pane displays the **Editing Mode** option, which is by default set to `Full`.

3   Select `Restricted` from the drop-down list, as shown, and click **OK**.
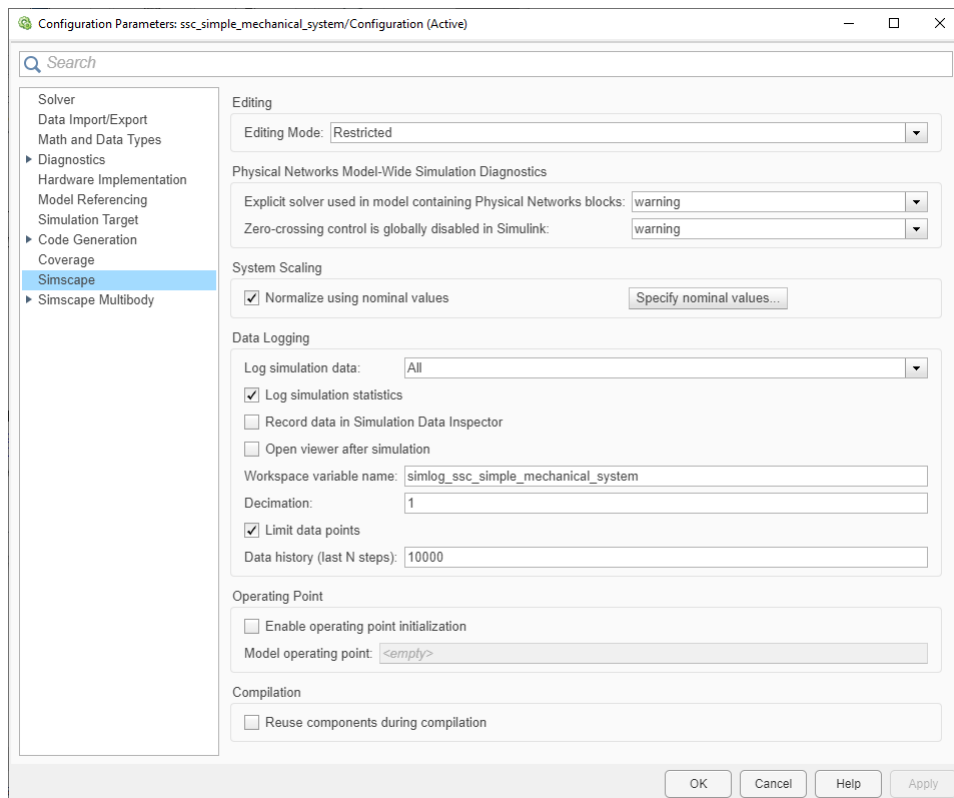


4   Save the model.

**Note** The **Simscape** entry does not appear in the left pane of the Configuration Parameters dialog box until you add at least one Physical Modeling block to your model. If you create an additional configuration set for a model, the **Simscape** entry does not appear in it until you either activate it or perform a Physical Modeling operation, such as adding or deleting a Physical Modeling block or connection, opening a Physical Modeling block dialog box, and so on.

Once you have switched a model to Restricted mode, working with it follows the rules described in "Work with a Model in Restricted Mode" on page 17-11. Note, however, that the add-on product licenses for this model stay checked out until you quit the MATLAB session.

When you open a model that has been saved in Restricted mode, the license manager opens it in Restricted mode and does not check out the add-on product licenses.

## Example of Saving a Model in Restricted Mode

In this example, you switch a model to Restricted mode and save it.

**1**  Open the Simple Mechanical System example model (`ssc_simple_mechanical_system`).
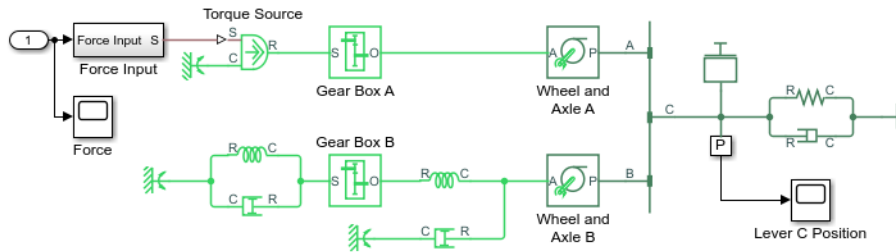


**2**  In the Simulink Toolstrip at the top of the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

**3**  In the left pane of the Configuration Parameters dialog box, select **Simscape**. The right pane displays the **Editing Mode** option, which is set to `Full` by default.

**4**  Select `Restricted` from the drop-down list and click **OK**.

**5**  Save the model as `model_test_edit_mode`.

## See Also

### Related Examples

### More About

# Work with a Model in Restricted Mode

When you open a model in Restricted mode, you can perform a variety of tasks: simulate the model, inspect and fine-tune block parameters, add and delete basic Simulink blocks, and so on. For a complete list of allowed operations, see "What You Can Do in Restricted Mode" on page 17-3.

When you open a block dialog box in Restricted mode, some of the block parameters may be grayed out. These are the so-called restricted parameters that can be modified only in Full mode. In general, you can change numerical parameter values in Restricted mode, but you cannot change the block parameterization options. See the block reference pages for specifics. Note also that when a restricted parameter defines the block parameterization schema, nonrestricted parameters available for fine-tuning in Restricted mode depend on the value of this restricted parameter. For example, in a Laminar Leakage (IL) block, the **Cross-sectional geometry** parameter is restricted. If, at the time the model entered Restricted mode, this parameter was set to `Circular`, then the nonrestricted parameters available for fine-tuning would be **Diameter** and **Longitudinal length**. If, however, **Cross-sectional geometry** was set to another option, you would have a different set of parameters available in Restricted mode.

You cannot change physical units in Restricted mode. When you open a block dialog box in Restricted mode, the drop-down lists of units next to a parameter name and value are grayed out. When you open a PS-Simulink Converter or Simulink-PS Converter block dialog box, the **Unit** parameter is grayed out.

The following examples illustrate operations allowed and disallowed in Restricted mode.

## How to Simulate and Fine-Tune a Model in Restricted Mode

This example shows how you can work with a model in Restricted mode by changing certain parameter values and observing the simulation results.

1   Open the `model_test_edit_mode` model, which you saved in Restricted mode in "Example of Saving a Model in Restricted Mode" on page 17-10. The model opens in Restricted mode.
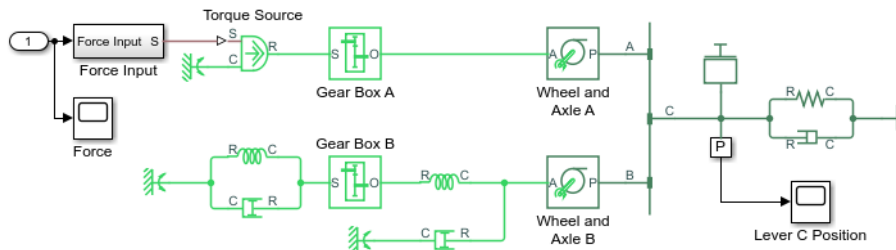


2   Open the Lever C Position scope and simulate the model. The models runs and simulates in Restricted mode.
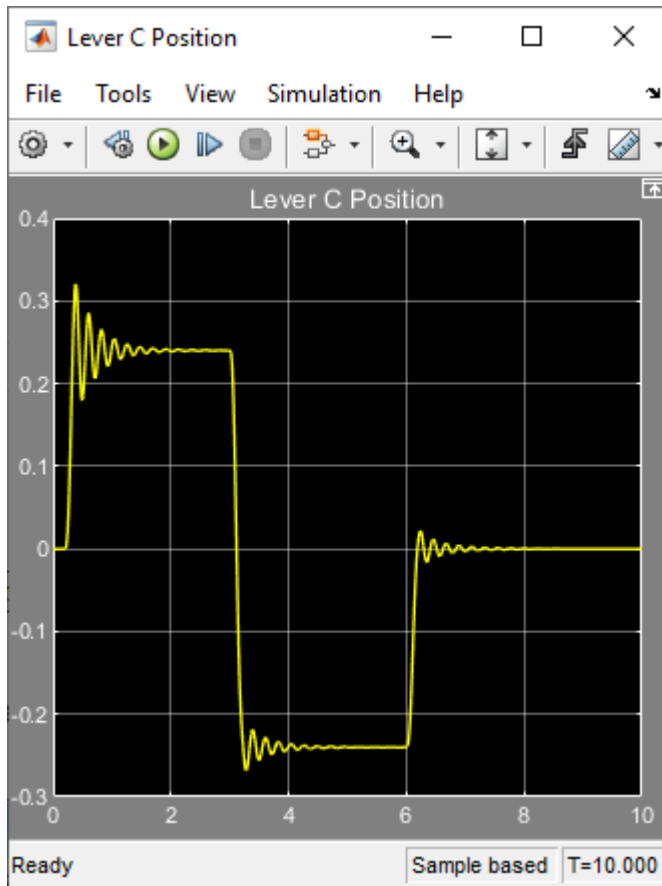
**3** Double-click the Wheel and Axle A block to open its dialog box. Notice that the **Mechanism orientation** parameter is grayed out, because you cannot modify the block driving direction in Restricted mode.
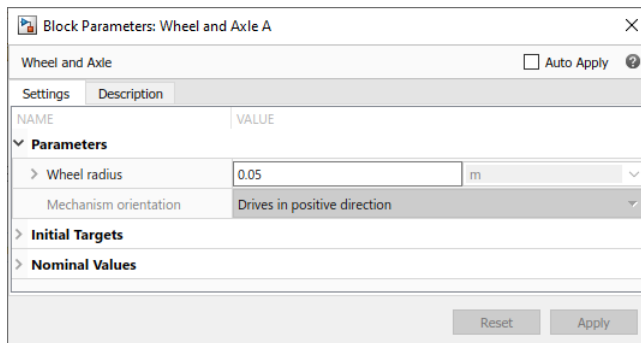


**4** Change the **Wheel radius** parameter value to `0.1`.

**5** Simulate the model again. Notice that the motion amplitude of node C became smaller as a result of the wheel radius change.
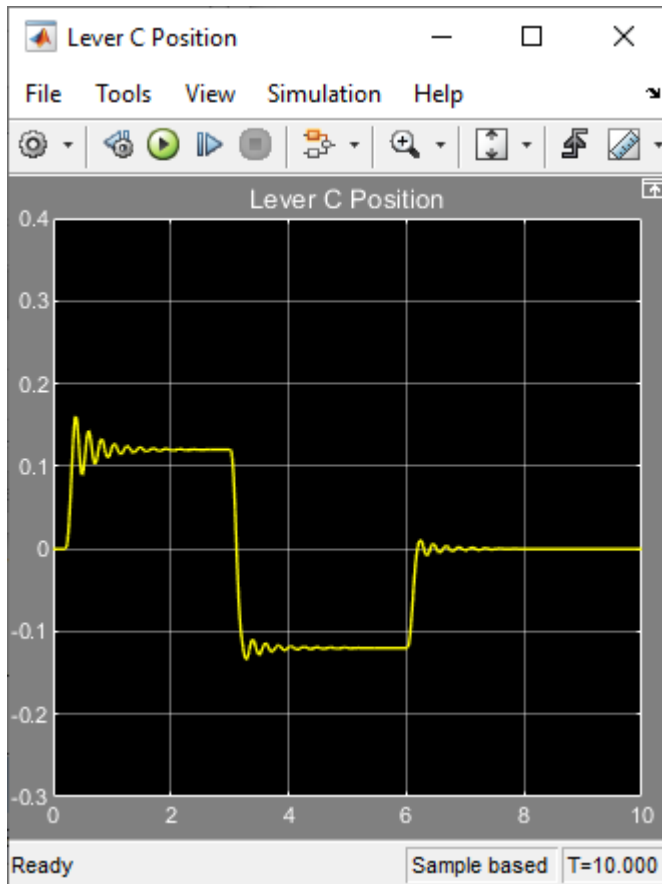
**6**    Double-click the Mass block and change the **Mass** parameter value to 24.

**7**    Simulate the model. Notice that doubling the mass resulted in increased vibrations.
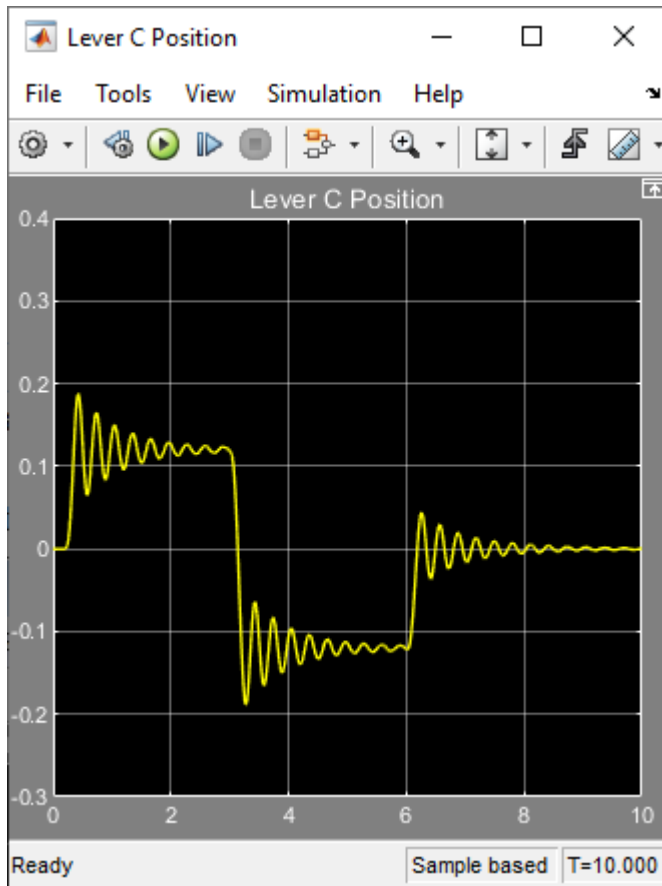
## How to Add and Delete Simulink Blocks in Restricted Mode

This example shows how you can change the model input signal in Restricted mode by adding and deleting basic Simulink blocks.

1   Open the `model_test_edit_mode` model, which you saved in Restricted mode in "Example of Saving a Model in Restricted Mode" on page 17-10. The model opens in Restricted mode.
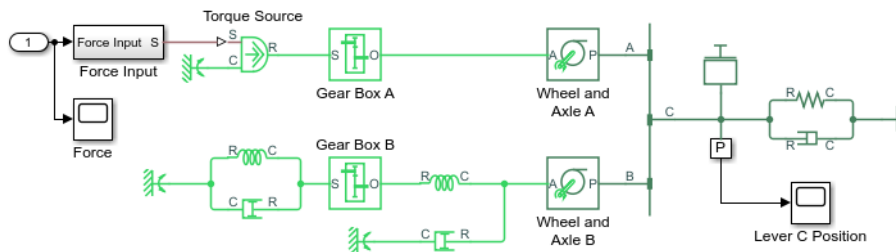


2   Open the Lever C Position scope and simulate the model.
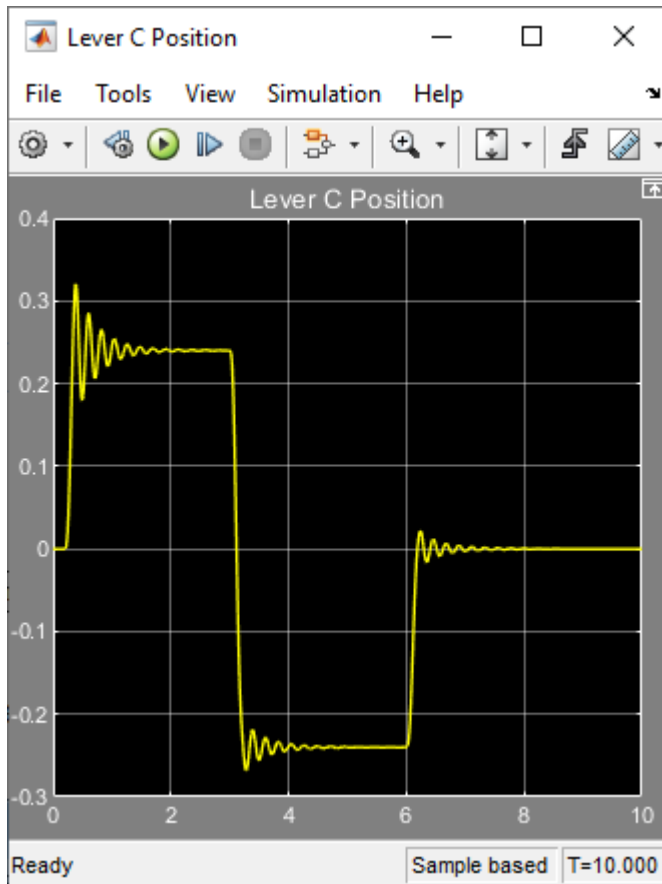
**3**   Replace the external input with a sine wave. On the **Data Import/Export** pane of the
Configuration Parameters dialog box, under **Load from workspace**, clear the **Input** check box.
Delete the Simulink Inport block, the Force scope, and the signal lines connecting the Inport
block to the scope and going into the Force Input subsystem.
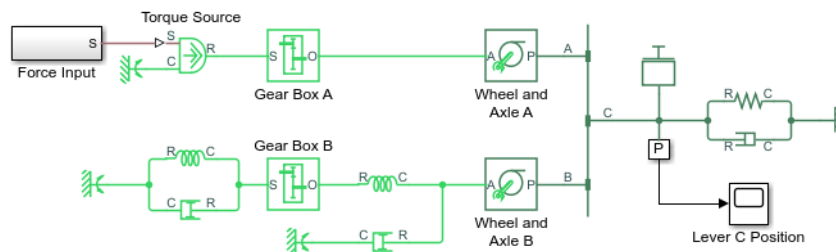


Double-click the Force Input subsystem to open it. Inside the subsystem, delete the Inport block.
Replace it with a Sine Wave block from the Simulink Sources library, as shown below.
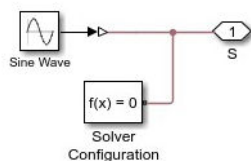
4 Simulate the model again. The model successfully compiles and simulates in Restricted mode because all the model topology changes have been performed in the Simulink part of the model.
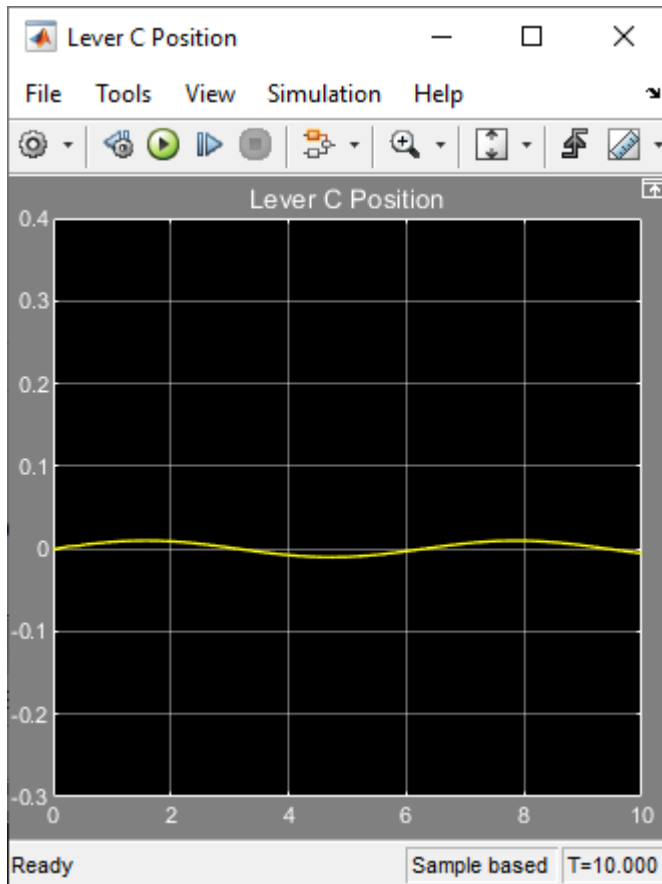


## Performing an Operation Disallowed in Restricted Mode

This example shows what happens when you perform an operation that is disallowed in Restricted mode.

1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in "Example of Saving a Model in Restricted Mode" on page 17-10. The model opens in Restricted mode.
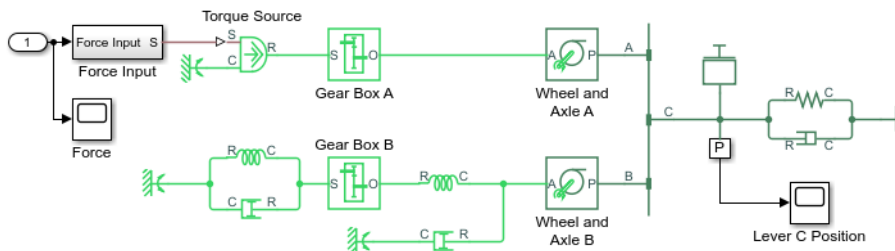


2 Double-click the P subsystem to open it.
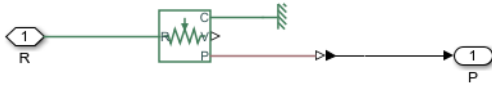
**3** Delete the connection line between port **P** of the Ideal Translational Motion Sensor block and the PS-Simulink Converter block. Instead, connect port **V** of the Ideal Translational Motion Sensor block to the input port of the PS-Simulink Converter block, to measure the velocity on node **C** of the lever.
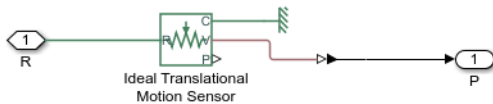


**4** Try to simulate the model. An error message appears saying that the model cannot be compiled because its topology has been changed while in Restricted mode. You can either undo the changes, or switch to Full mode, as described in "Switch from Restricted to Full Mode" on page 17-18.

## See Also

## Related Examples

- "Set the Model Loading Preference" on page 17-7
- "Save a Model in Restricted Mode" on page 17-9
- "Switch from Restricted to Full Mode" on page 17-18

## More About

- "About the Simscape Editing Mode" on page 17-2
- "Get Editing Mode Information" on page 17-20

# Switch from Restricted to Full Mode

If you need to perform a task that is disallowed in Restricted mode, you can try to switch the model to Full mode.

1   In the Simulink Toolstrip at the top of the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

2   In the left pane of the Configuration Parameters dialog box, select **Simscape**. The right pane displays the **Editing Mode** option.

3   Select Full from the drop-down list, as shown, and click **OK**.
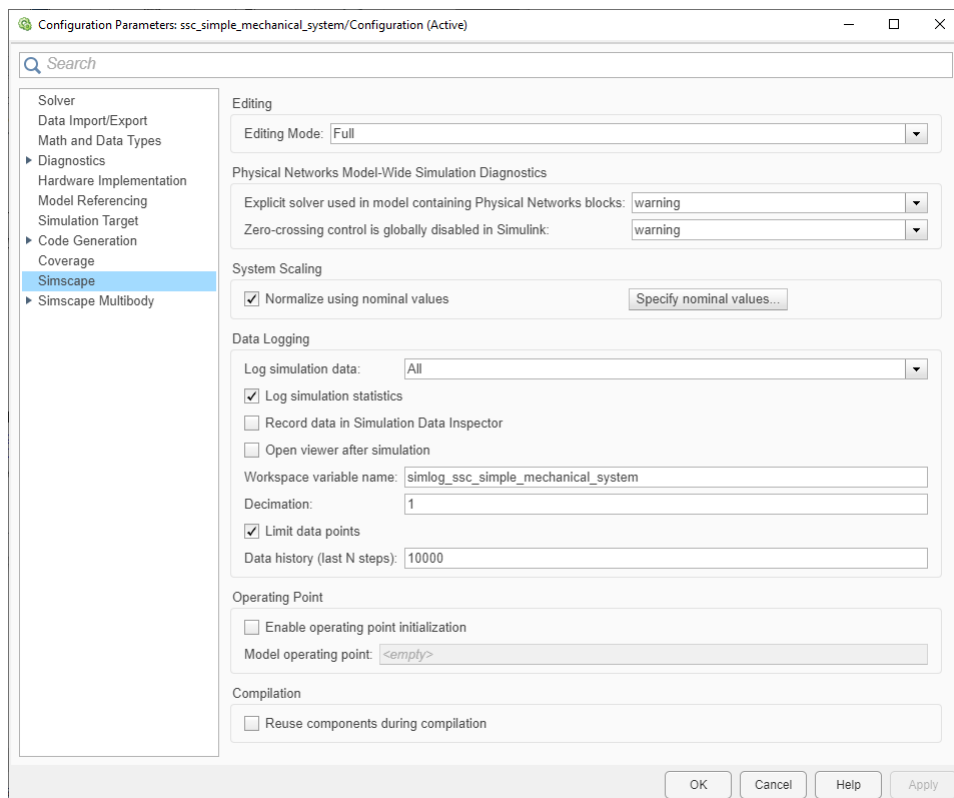


The license manager checks whether all the add-on product licenses for this model are available. If yes, it checks out the add-on product licenses and switches the model to Full mode. If a add-on product license is not available, the license manager issues an error message and the model stays in Restricted mode.

**Note**  If the switch to Full mode fails but some of the add-on product licenses have already been checked out, they stay checked out until you quit the MATLAB session. For more information, see "Example with Multiple Add-On Products" on page 17-5.

Once the model is switched to Full mode, you can perform the needed design and simulation tasks, and then either save it in Full mode, or switch back to Restricted mode and save it in Restricted mode.

## See Also

### Related Examples

- "Set the Model Loading Preference" on page 17-7
- "Save a Model in Restricted Mode" on page 17-9
- "Work with a Model in Restricted Mode" on page 17-11

### More About

- "About the Simscape Editing Mode" on page 17-2
- "Get Editing Mode Information" on page 17-20

# Get Editing Mode Information

| In this section... |
| --- |
| "What Is the Current Mode?" on page 17-20 |
| "Which Licenses Are Checked Out?" on page 17-20 |

## What Is the Current Mode?

If you are unsure whether the model is currently open in Restricted or Full mode, you can check by following these steps.

1  In the Simulink Toolstrip at the top of the model window, open the **Modeling** tab and click **Model Settings**. The Configuration Parameters dialog box opens.

2  In the left pane of the Configuration Parameters dialog box, select **Simscape**. The right pane displays the **Editing Mode** option, which is either `Full` or `Restricted`.

3  At this point, you can either try switching the mode by selecting a different option from the drop-down list, or click **Cancel** to stay in the current mode.

## Which Licenses Are Checked Out?

Use the MATLAB `license` command to get a list of all the licenses currently in use. In the MATLAB Command Window, type

```
license('inuse')
```

This command returns a list of licenses checked out in the current MATLAB session. In the list, products are listed alphabetically by their license feature names.

## See Also

## Related Examples
- "Set the Model Loading Preference" on page 17-7
- "Save a Model in Restricted Mode" on page 17-9
- "Work with a Model in Restricted Mode" on page 17-11
- "Switch from Restricted to Full Mode" on page 17-18

## More About
- "About the Simscape Editing Mode" on page 17-2

# Network Couplers

# Using Network Couplers to Split Physical Networks

| **In this section...** |
| --- |
| |
| |
| |
| |
| |
| |

## Why Use Network Couplers?

When modeling a complete physical system, the models can become quite large, resulting in longer simulation times. This can become problematic when you need to deploy the physical model to real time for hardware-in-the-loop studies. It can also be an issue for desktop simulation when you need to explore multiple scenarios or design options.

In such cases, the best first step is always to reduce modeling fidelity to the minimum level needed. For example, if modeling an electric vehicle drive cycle, there is no need to model the switching of power electronics because average value or energy-balancing methods should be sufficient. Use the detailed switching power electronics models in separate test harnesses to determine average losses to be used at the system level.

For some systems, however, this is not enough. Even when modeled at a requisite level of detail, system complexity is sometimes too high to run the complete model on a single microprocessor. In this case, you need to split the model into smaller pieces that can be run by multiple processors. The blocks in the Network Couplers library help you do that.

A Simscape network cannot be directly divided into separate parts for simulation on different processors. This is because the network is parsed, simplified, and presented to the solver as a single entity. Therefore, you need to split your Simscape network into separate smaller networks interfaced to each other through Simulink connections. The Simscape Network Couplers library provides the starting point for you to do this. The blocks in the library look like standard Simscape blocks (for example, Inductor), but are actually subsystems that implement the network separation. The design of the coupler blocks enables you to iterate on where to divide your model without initially changing its architecture.

## Naming Conventions

Once you use the network coupler blocks to split a Simscape network in your model into multiple coupled networks, each of these networks can have its own solver settings. For example, you can use a variable solver for one of the coupled networks and a fixed-step solver for another, or use two fixed-step solvers with different step sizes. This makes it easier to deploy your system for hardware-in-the-loop simulation.

For clarity, the block interface and the documentation use these naming conventions for coupled networks:

| Network Coupling Scenario | Naming Convention |
|---|---|
| One network is variable-step and the other is fixed-step | • Network 1 is the variable-step network<br>• Network 2 is the fixed-step network |
| Both networks are fixed-step, with different step sizes | • Network 1 is the fixed-step network with the higher sampling rate (smaller step size)<br>• Network 2 is the fixed-step network with the lower sampling rate (larger step size) |

When using the network coupler blocks, connect port **1** of the block to Network 1 and port **2** to Network 2.

If both networks are variable-step, or both networks are fixed-step with the same step size, then the block orientation does not matter.

## Best Practices for Splitting Physical Networks

Deciding where to split a network requires some thought. Ultimately, you are looking to balance the computational burden evenly across two or more microprocessors. However, to minimize loss of modeling fidelity and to reduce likelihood of numerical instability, you also want to split the network at places where there are no fast dynamics.

When interfacing the separated networks, it is necessary to introduce some dynamics or delays to remove any algebraic constraints, which may lead to the loss of fidelity or instability issues. One approach to mitigate these issues is to select a dynamic component in the system as a splitting point and use its time constant to remove the algebraic loop. Suitable components include inductors, flexible shafts, and compressible volumes of fluid.

Also, avoid splitting the network at a point where there is high-frequency activity. For example, in a power network, split at a DC link rather than an AC link if you have the choice.

## About the Network Couplers Library
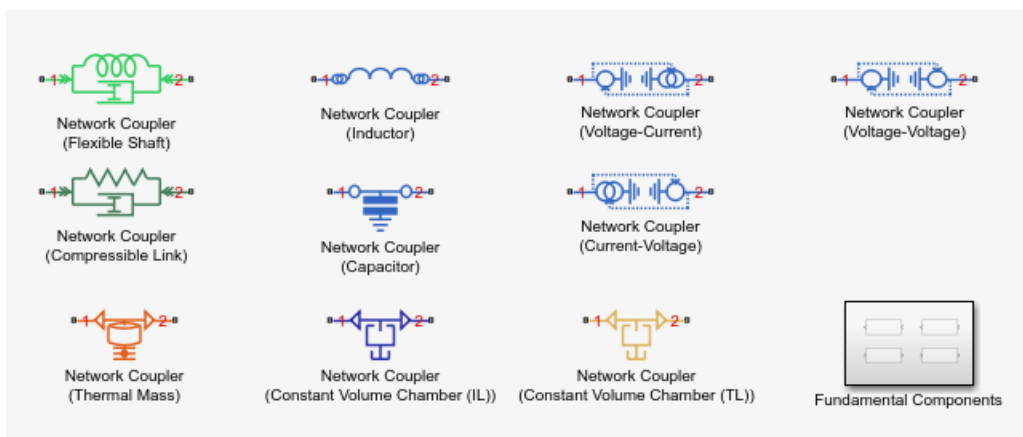


The Network Couplers library, available as a sublibrary in the Utilities library, contains network coupler blocks for mechanical, electrical, thermal, and fluid domains:

| Domain | Network Coupler Blocks |
|---|---|
| Mechanical Rotational | Network Coupler (Flexible Shaft) |
| Mechanical Translational | Network Coupler (Compressible Link) |
| Electrical | Network Coupler (Inductor) |
| | Network Coupler (Capacitor) |
| | Network Coupler (Voltage-Current) |
| | Network Coupler (Current-Voltage) |
| | Network Coupler (Voltage-Voltage) |
| Thermal | Network Coupler (Thermal Mass) |
| Isothermal Liquid | Network Coupler (Constant Volume Chamber (IL)) |
| Thermal Liquid | Network Coupler (Constant Volume Chamber (TL)) |

For the electrical domain, the library provides several blocks for use in a variety of scenarios:

| Block Name | When To Use | Limitations |
|---|---|---|
| Network Coupler (Inductor) | Circuit has an inductor at the point where you want to split the network. | • You cannot have other inductors or current sources in series with the block because this would cause an Index-2 topology (for more information, see "Avoiding Numerical Simulation Issues" on page 1-31). If you can, lump all series inductances into one block and split the network using this block.<br>• The two resulting networks must both have an electrical reference. |
| Network Coupler (Capacitor) | Circuit has a capacitor (connected to an electrical reference) at the point where you want to split the network. This block can be useful for connecting ground planes or return paths. | Make sure that the RC time constant, which results from the capacitor and the connected network series resistances, is not faster than can be tracked by the simulation step size. |
| Network Coupler (Voltage-Current) | Network 1 has series-connected inductors or current sources, while Network 2 models something like a capacitive load or battery. | Changes the behavior of the system because it adds a first-order lag to break the algebraic loop. |

| Block Name | When To Use | Limitations |
|---|---|---|
| Network Coupler (Current-Voltage) | Network 2 has series-connected inductors or current sources, while Network 1 models something like a capacitive load or battery. | Changes the behavior of the system because it adds a first-order lag to break the algebraic loop. |
| Network Coupler (Voltage-Voltage) | Both networks have series-connected inductors or current sources. | Some experimentation may be required to find good values for the internal source resistances. Accuracy is not as good as when splitting the network using an inductor or capacitor. |

The block icons are similar to the equivalent Foundation library blocks, but have small annotations to indicate whether they act internally as Through or Across sources. These annotations help when deciding which coupler block to use. For example, the Network Coupler (Inductor) block icon shows the standard inductor symbol with two overlapping circles adjacent to each port, to indicate two internal current sources. Therefore, you should never connect the Network Coupler (Inductor) block directly to a current source, or to another inductor. For more information, see "Avoiding Numerical Simulation Issues" on page 1-31.

Additionally, the Network Couplers library contains a Fundamental Components sublibrary with prediction and smoothing blocks. For more information, see "Prediction and Smoothing" on page 18-6.

## General Principles of Using the Network Couplers Blocks

The blocks in the Network Couplers library look like standard Simscape blocks, but are actually subsystems that implement the network separation.

When you add a Network Couplers library block to your model, the library links are broken. This means that you can customize the coupler blocks and associated subsystem contents, if needed.

When you double-click a coupler block in your model, the underlying subsystem opens. It generally contains two masked subsystems, Port 1 Interface and Port 2 Interface, connected to each other through two rate transition blocks. By default, the rate transition blocks are commented through. Uncomment them if at least one of the coupled networks is running fixed step.

Masking the Port 1 Interface and Port 2 Interface subsystems, rather than the top-level network coupler subsystem, helps with restructuring the model into different task subsystems for real-time deployment.

### Sampling Types

All of the coupler blocks have some common options and parameters to specify the sampling types of the two connected networks. The table shows the possible scenarios.

| Case | Network 1 | Network 2 |
|---|---|---|
| 1 | Variable-step | Variable-step |
| 2 | Variable-step | Fixed-step |

| Case | Network 1 | Network 2 |
|------|-----------|-----------|
| 3 | Fixed-step, step size *h* | Fixed-step, step size *h* |
| 4 | Fixed-step, step size *h1* | Fixed-step, step size *h2*, where *h2>h1* |

In cases 1 and 3 both networks have the same sampling type. However, for cases 2 and 4, the sampling types differ, and Network 1 is the faster network (as described in "Naming Conventions" on page 18-2), that is, its solver takes smaller steps. The Port 1 Interface block implements the dynamic breaking of the Simulink algebraic loop because the smaller time step supports a faster time constant and less loss of accuracy.

In the Port 1 Interface block dialog, set the **Sampling type** parameter to match your network sampling scenario:

- `Variable step at ports 1 and 2` — Both networks are variable-step.
- `Variable step at port 1 and fixed step at port 2` — Network 1 is variable-step and Network 2 is fixed-step.
- `Fixed step both ports with common sample time` — Both networks are fixed-step, with the same step size.
- `Fixed step both ports with faster sampling at port 1` — Both networks are fixed-step, with different step sizes.

The parameter list changes, based on the selected option. Set the other parameter values, as needed. Use the **Analysis > Derived values** section of the block dialog to look up recommended values.

**Prediction and Smoothing**

Connecting a network with faster sampling (Network 1) to a network with slower sampling (Network 2) may result in accuracy loss in the Network 1 simulation when its sample times are between the sample times of Network 2. The Prediction (slow->fast) block helps prevent the accuracy loss. It predicts the output of Network 2 between its sample times by using the last computed derivative of the output of Network 2. The block implementation is a masked subsystem.
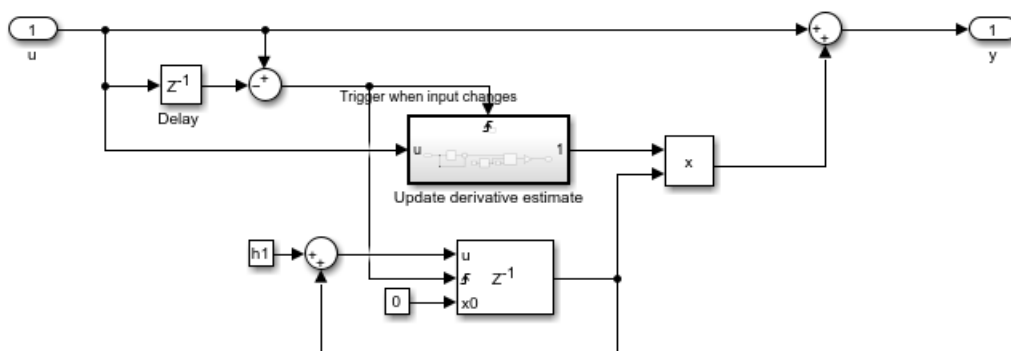


The subsystem contains blocks that run at both the slower and the faster rate. The Delay block is driven by the slower rate, and changes in its output are used as a trigger, so that the triggered subsystem captures the latest derivative of input *u*. The discrete-time integrator runs at the faster rate and uses this derivative to predict the output *y* until the next trigger event happens.

The Prediction (discrete->continuous) block works in a similar fashion to provide estimates for the case when Network 1 is variable-step and Network 2 is fixed-step.

When Network 1 has high-frequency activity above what Network 2 can sample, applying smoothing to the output of Network 1 can sometimes be advantageous because it helps you avoid picking a local peak or trough of the Network 1 activity. The Smoothing (fast->slow) block provides an averaged value of the Network 1 output to Network 2. The block implementation is also a masked subsystem.
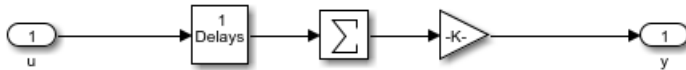


The smoothing works by averaging the last $N$ samples. You specify $N$ by using the **Number of samples over which to smooth** parameter.

The Smoothing (continuous->discrete) block is intended for when Network 1 is variable-step and Network 2 is fixed-step. Instead of averaging, the block uses a first-order filter to remove unwanted high-frequency information.

These prediction and smoothing algorithms are built into the Network Coupler blocks, and you can enable them by using the check boxes in the block dialogs. The prediction and smoothing blocks in the Fundamental Components sublibrary are provided for your reference.

## Typical Workflow

This is the recommended workflow for dividing a network:

1   Create a baseline simulation and save associated results, to later validate the divided network against these results. Usually this step is done with the Simscape network simulated using a variable-step solver, for best accuracy.

2   Identify a suitable place to split the network. If possible, select an existing system component for which there is an equivalent in the Network Couplers library. Also try to avoid splitting the network at places where there is high-frequency activity. For more information, see "Best Practices for Splitting Physical Networks" on page 18-3.

Replace the existing component with the equivalent from the Network Couplers library and set parameters accordingly, or add a new component. Also remember to add a Solver Configuration block to the part of the network that does not have one after splitting.

Pay particular attention to setting of correct initial conditions for the coupler blocks. One way to get initial conditions is to extract them from the baseline in Step 1.

3   Run the simulation again, still using variable-step solver, and validate the results against the baseline. If you replaced an existing component, results should be the same. If you added a component, the simulation results will differ, in which case use this step to determine acceptable parameter values for the added component.

4   Change the sampling types for the two connected networks to the values you expect to use. To switch a Simscape network to using a fixed-step solver, open its Solver Configuration block, select the **Use local solver** check box, and use the **Sample time** parameter to specify the sample time.

Change the **Sampling type** parameter value of the coupler block to match the sampling types of the two networks. Uncomment the rate transition blocks in the top-level subsystem of the coupler block.

**5** Run the simulation and compare to the baseline, and make adjustments to the sample times if necessary. Use the **Analysis** tab of the coupler block to get information about suggested maximum sampling times.

**6** Check to see if prediction or smoothing can help in your application by selecting and clearing the corresponding check boxes in the coupler block dialog.

**7** When satisfied with the simulation results, redraw the block diagram by expanding the Network Coupler subsystems and moving them up one level, to make the Simulink connections visible at the top level of the block diagram, along with the rate transition blocks. You can then group the blocks and subsystems by sampling times and perform other steps needed for deployment to a multirate HIL platform, the same way as for any other Simulink model.

For a detailed example of splitting an electrical vehicle model into separate networks and then deploying it to a multirate HIL, see "Configuring an EV Simulation for Multirate HIL" on page 20-340.

## See Also

Network Coupler (Compressible Link) | Network Coupler (Flexible Shaft) | Network Coupler (Inductor) | Network Coupler (Capacitor) | Network Coupler (Current-Voltage) | Network Coupler (Voltage-Current) | Network Coupler (Voltage-Voltage) | Network Coupler (Thermal Mass) | Network Coupler (Constant Volume Chamber (IL)) | Network Coupler (Constant Volume Chamber (TL))

# Modeling Variants in a Physical Network Using Connector Block

# Model Variants in an Electrical Circuit Using Variant Connector Blocks

| In this section... |
| --- |
| "Explore the Model" on page 19-2 |
| "Simulate the Flow of a Current for Different Variant Configurations" on page 19-4 |

This example shows how to simulate the flow of a current in an electrical circuit for different variant configurations using primary and nonprimary type Variant Connector blocks. Variant Connector blocks allow you to activate or deactivate a set of components in the network during simulation without having to physically remove the components or exclude them from simulation.

## Explore the Model

To open the Variant Bounded Region in Electrical Circuit example model, type ssc_variant_connector_bounded_region in the MATLAB Command Window.

**Variant Bounded Region in Electrical Circuit**

1. Plot current for different variant configuration (see code)

|  | Bounded Region 1 (A==1) | Bounded Region 2 (B==1) |
|---|---|---|
| Plot | True | True |
| Plot | True | False |
| Plot | False | True |
| Plot | False | False |

2. Learn more about this example
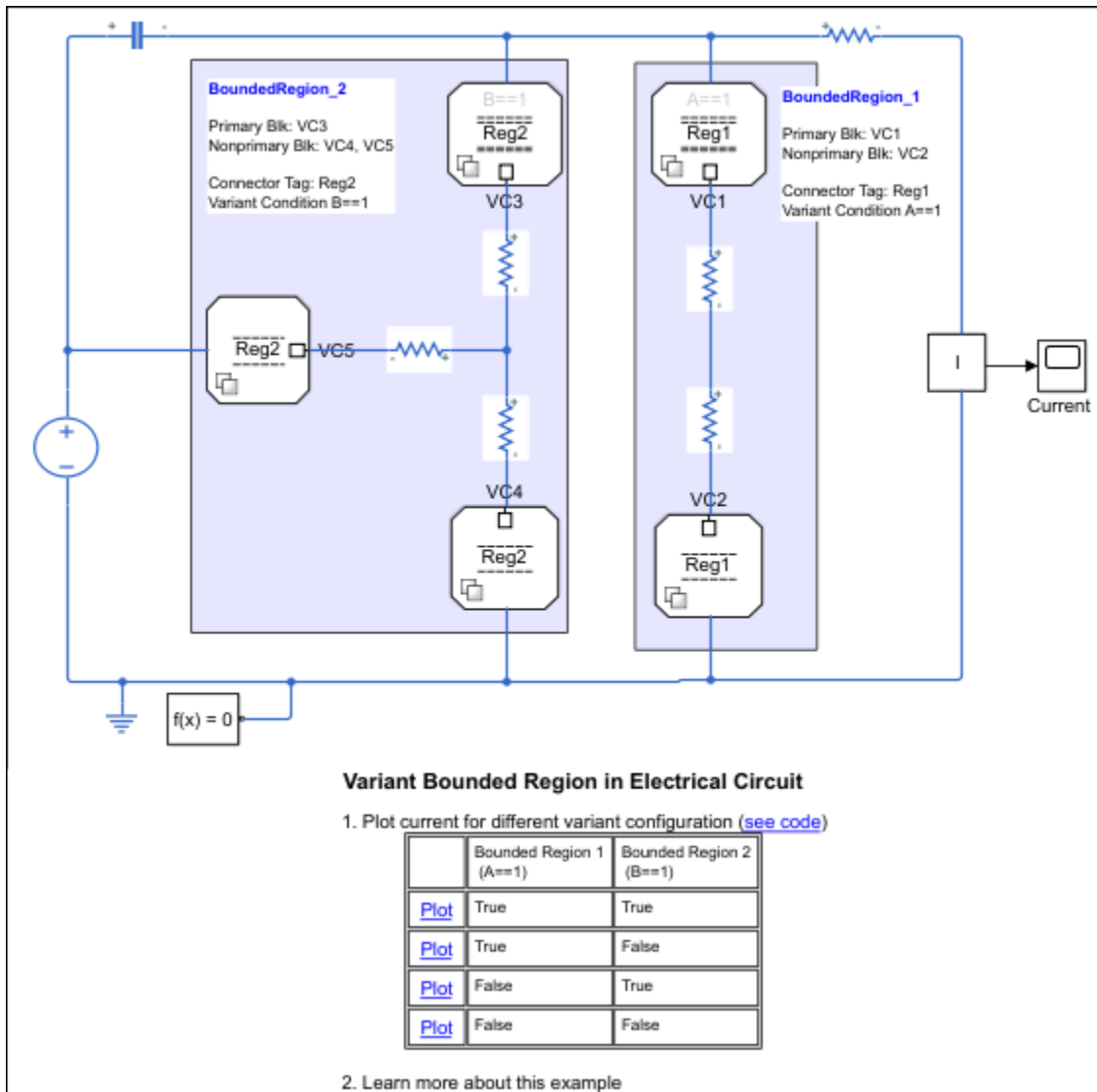
This model has two bounded regions, BoundedRegion_1 and BoundedRegion_2. In BoundedRegion_1, the **Connector tag** parameters of the Variant Connector blocks are set to `Reg1`, and in BoundedRegion_2, the **Connector tag** parameters are set to `Reg2`. BoundedRegion_1 has one primary Variant Connector block, VC_1, and an associated nonprimary Variant Connector block, VC_2. The variant condition of BoundedRegion_1 is $A == 1$. BoundedRegion_2 has one primary Variant Connector block, VC_3, and two associated nonprimary Variant Connector blocks, VC_4 and VC_5. The variant condition of BoundedRegion_2 is $B == 1$.

During simulation, Simulink computes the variant conditions associated with each bounded region. If the variant condition of a region evaluates to `true`, all the physical components located inside the region become active. For example, if $A == 1$ evaluates to `true` during simulation, the components of BoundedRegion_1, Resistor3 and Resistor4, become active. If $A == 1$ evaluates to `false`, the components of BoundedRegion_1 are inactive.

## Simulate the Flow of a Current for Different Variant Configurations

The variant condition variables, A and B, are defined in the `PostLoadFcn` callback. To view or modify the value of these variables, on the **Modeling** tab, select **Model Settings > Model Properties**. On the **Callbacks** tab, in the **Model callbacks** pane, click **PostLoadFcn**. In this example, A = 1 and B = 2. The associated bounded region activates based on these variables..

**Case 1: BoundedRegion_1 Is Active and BoundedRegion_2 Is Inactive**

1   In the **Model Properties** window, set the value of A to 1 and B to 2.

2   Click **Run** and see the variant conditions propagate from the Variant Connector blocks to the connected components.

3   To analyze the propagated variant conditions and the block activation state, on the **Debug** tab, select **Information Overlays > Variant Legend**. For more information on Variant Condition Legend, see "Visualize Propagated Variant Conditions Using Variant Conditions Legend".

   - A == 1 evaluates to `true`. The components inside BoundedRegion_1 become active.

   - B == 1 evaluates to `false`. The component inside BoundedRegion_2 become inactive.



4   To view the flow of the current in this scenario, double-click the Scope block named `Current`. Alternatively, on the model, click the **Plot** link in the **Variant Bounded Region in Electrical Circuit** table that corresponds to the condition, A == 1 is `true` and B == 1 is `false`.
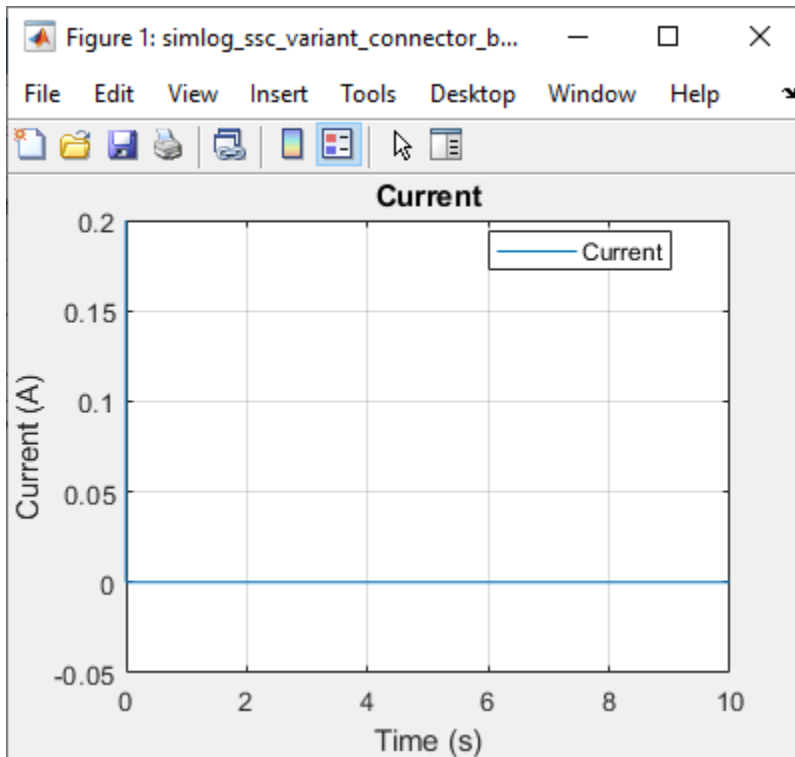
**Case 2: BoundedRegion_1 Is Inactive and BoundedRegion_2 Is Active**

1   In the **Model Properties** window, set the value of A to 2 and B to 1, and then simulate the model..

2   Analyze the variant conditions and the block activation state.

   - A == 1 evaluates to false. The components inside BoundedRegion_1 become inactive.
   - B == 1 evaluates to true. The component inside BoundedRegion_2 become active.

3   View the flow of the current in Current, or click the **Plot** link in the **Variant Bounded Region in Electrical Circuit** table that corresponds to the condition, A == 1 is false and B == 1 is true.
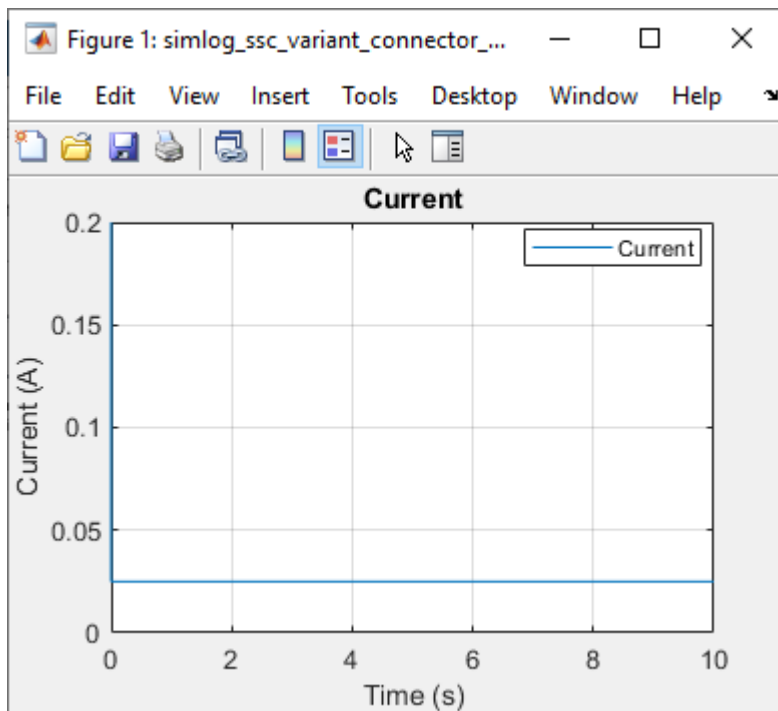
Similarly, you can set the value of A and B to 0 and analyze how both the regions become inactive during simulation.

## See Also

## Related Examples

- "Model Variants in a Mechanical System Using Variant Connector Blocks" on page 19-7

# Model Variants in a Mechanical System Using Variant Connector Blocks

| In this section... |
|---|
| "Explore the Model" on page 19-7 |
| "Simulate the Mechanical System for Different Variant Configurations" on page 19-8 |

This example shows how to simulate the displacement of velocity sources and mass for different variant configurations using leaf type Variant Connector blocks. Variant Connector blocks allow you to activate or deactivate a set of components in the network during simulation without having to physically remove the components or exclude them from simulation.

## Explore the Model

To open the Variant Leaf Region in Mechanical System example model, type ssc_variant_connector_leaf_region in the MATLAB Command Window.
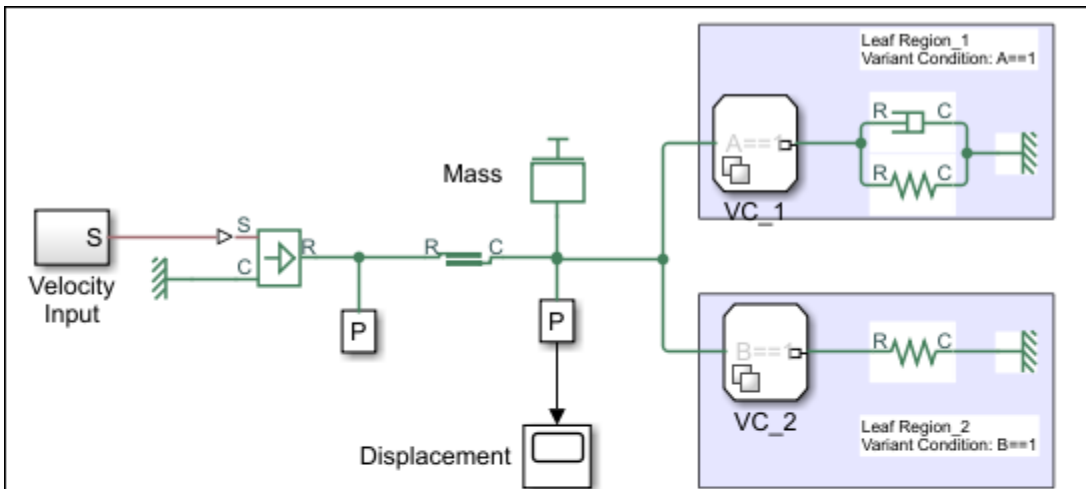


Copyright 2020 The MathWorks, Inc.

### Variant Leaf Region in Mechanical System

1. Plot displacement of source and mass for different variant configuration (see code)

| | Leaf Region 1 (A==1) | Leaf Region 2 (B==1) |
|---|---|---|
| Plot | True | True |
| Plot | True | False |
| Plot | False | True |
| Plot | False | False |

This model has two Variant Connector blocks, VC_1 and VC_2. These are leaf type Variant Connector blocks. VC_1 has the variant condition `A == 1` and VC_2 has the variant condition `B == 1`.

During simulation, Simulink computes the variant conditions associated with each leaf type Variant Connector block. If the variant condition of a Variant Connector block evaluates to `true`, all the physical components that are inside the leaf region of that block become active. For example, if `A == 1` evaluates to `true`, the components inside LeafRegion_1 become active. If `A == 1` evaluates to `false`, the components inside LeafRegion_1 remain inactive.

## Simulate the Mechanical System for Different Variant Configurations

The variant condition variables, A and B, are defined in the `PostLoadFcn` callback. To view or modify the value of these variables on the **Modeling** tab, select **Model Settings > Model Properties**. On the **Callbacks** tab, in the **Model callbacks** pane, click **PostLoadFcn**. In this example, the value of A = 1 and B = 2. The associated leaf region activates based on these variables.

**Case 1: LeafRegion_1 Is Active and LeafRegion_2 Is Inactive**

1  In the **Model Properties** window, set the value of A to 1 and B to 2.
2  Click **Run** and see the variant conditions propagate from the Variant Connector blocks to the connected components.
3  To analyze the propagated variant conditions and the block activation state, on the **Debug** tab, select **Information Overlays > Variant Legend**. For more information on Variant Condition Legend, see "Visualize Propagated Variant Conditions Using Variant Conditions Legend".

   - `A == 1` evaluates to `true`. The components inside LeafRegion_1 become active.
   - `B == 1` evaluates to `false`. The components inside LeafRegion_2 become inactive.



4  View the displacement of mass and the velocity source by clicking the **Plot** link in the **Variant Leaf Region in Mechanical System** table that corresponds to the condition, `A == 1` is `true` and `B == 1` is `false`.
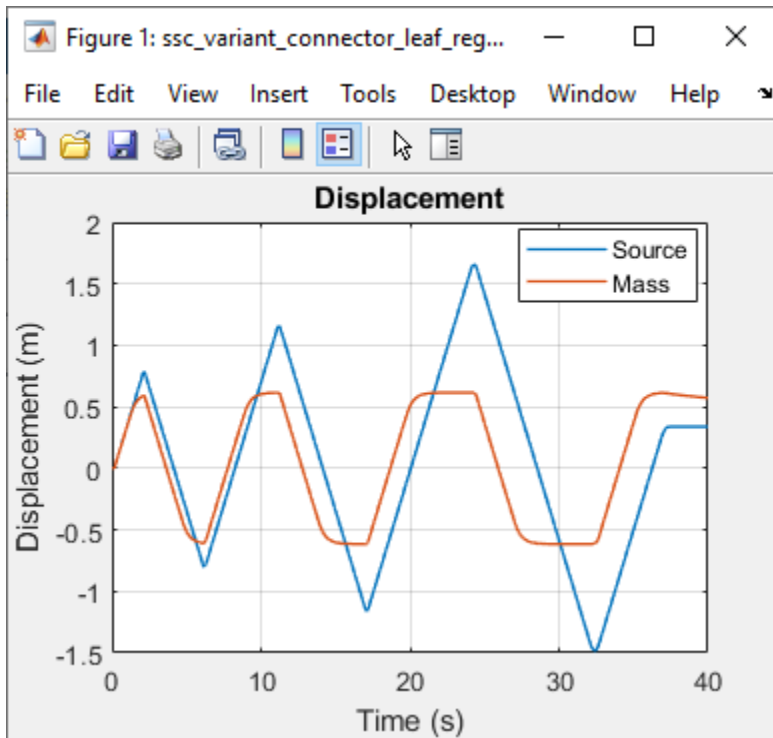
**Case 2: LeafRegion_1 Is Inactive and LeafRegion_2 Is Active**

**1**   In the **Model Properties** window, set the value of A to 1 and B to 2, and then simulate the model.

**2**   Analyze the variant conditions and the block activation state.

- A == 1 evaluates to false. The components inside LeafRegion_1 become inactive.
- B == 1 evaluates to true. The components inside LeafRegion_2 become active

**3**   View the displacement of mass and the velocity source by clicking the **Plot** link in the **Variant Leaf Region in Mechanical System** table that corresponds to the condition, A==1 is false and B==1 is true.
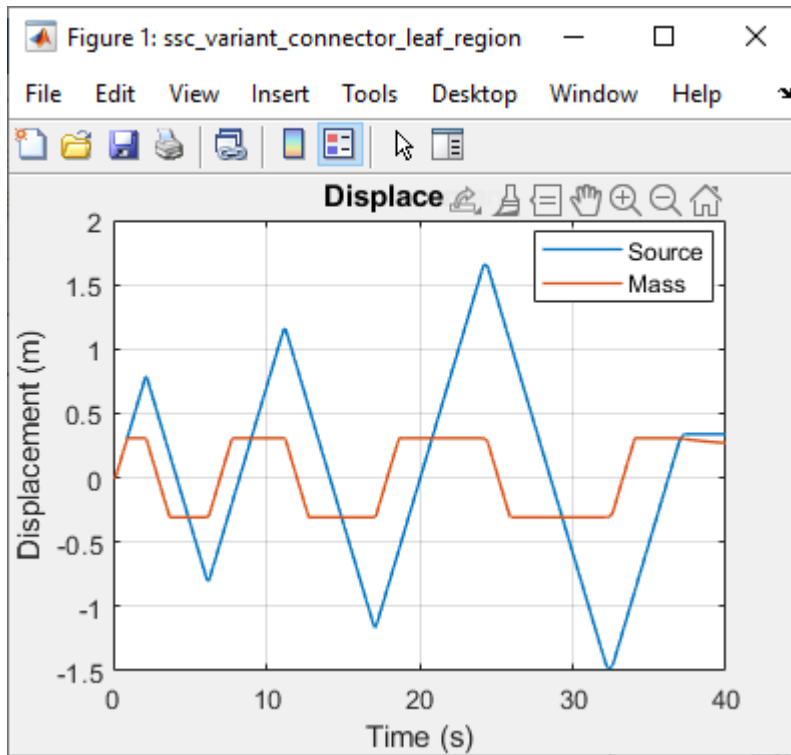
Similarly, you can set the value of A and B to 0 and analyze how both the regions become inactive during simulation.

## See Also

## Related Examples

- "Model Variants in an Electrical Circuit Using Variant Connector Blocks" on page 19-2

# Simscape Examples

# Mass-Spring-Damper with Controller

This example shows a controlled mass-spring-damper. A controller adjusts the force on the mass to have its position track a command signal. The initial velocity for the mass is 10 meters per second. The controller adjusts the force applied by the Force Source to track the step changes to the input signal.

**Model**
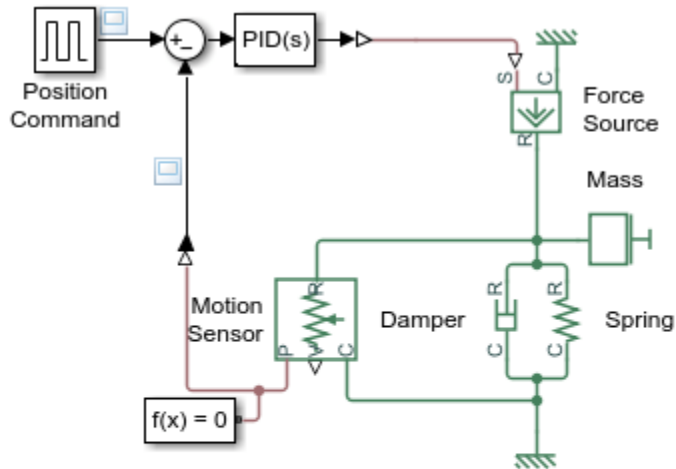


**Mass-Spring-Damper with Controller**
1. Plot forces in system and mass position (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2014-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**

The plots below show the position of the mass and the forces acting on the mass.
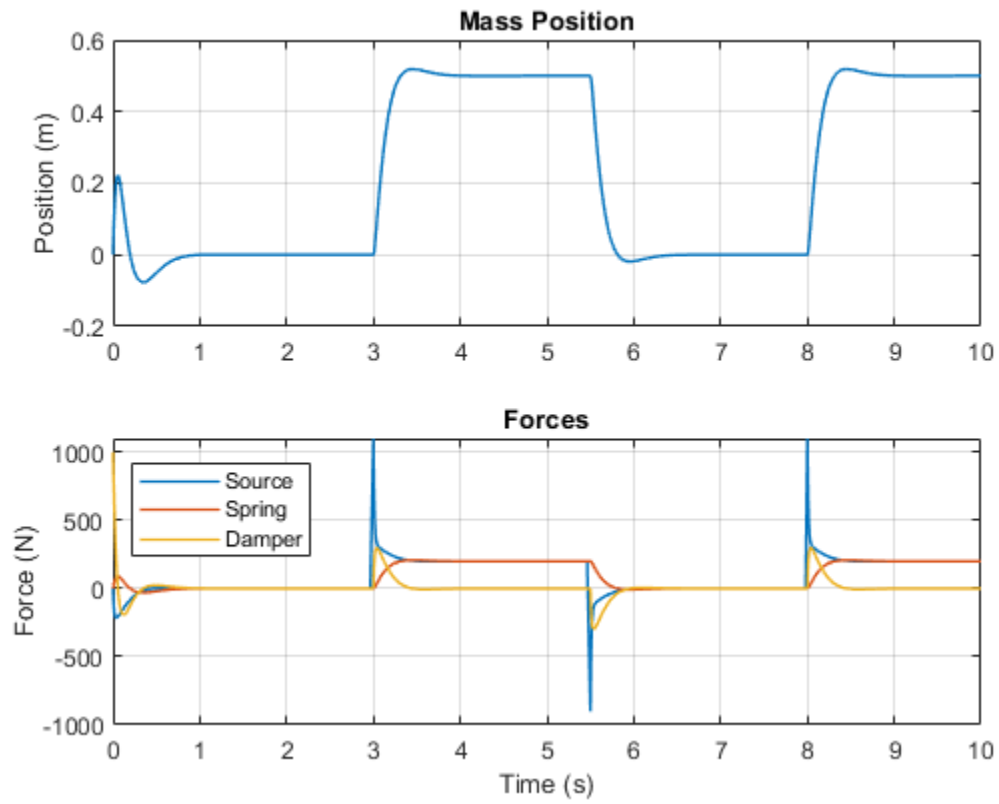
# Mass-Spring-Damper in Simulink and Simscape

This example shows two models of a mass-spring-damper, one using Simulink® input/output blocks and one using Simscape™ physical networks.

The Simulink model uses signal connections, which define how data flows from one block to another. The Simscape model uses physical connections, which permit a bidirectional flow of energy between components. Physical connections make it possible to add further stages to the mass-spring-damper simply by using copy and paste. Input/output connections require rederiving and reimplementing the equations.

The initial deflection for the spring is 1 meter. This is shown in the block annotations for the Spring and one of the Integrator blocks.

**Model**
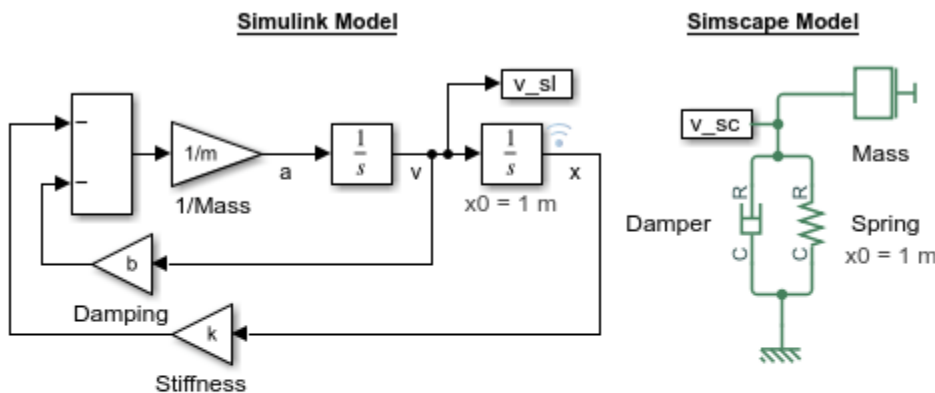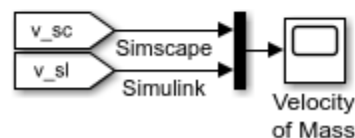


Copyright 2014-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
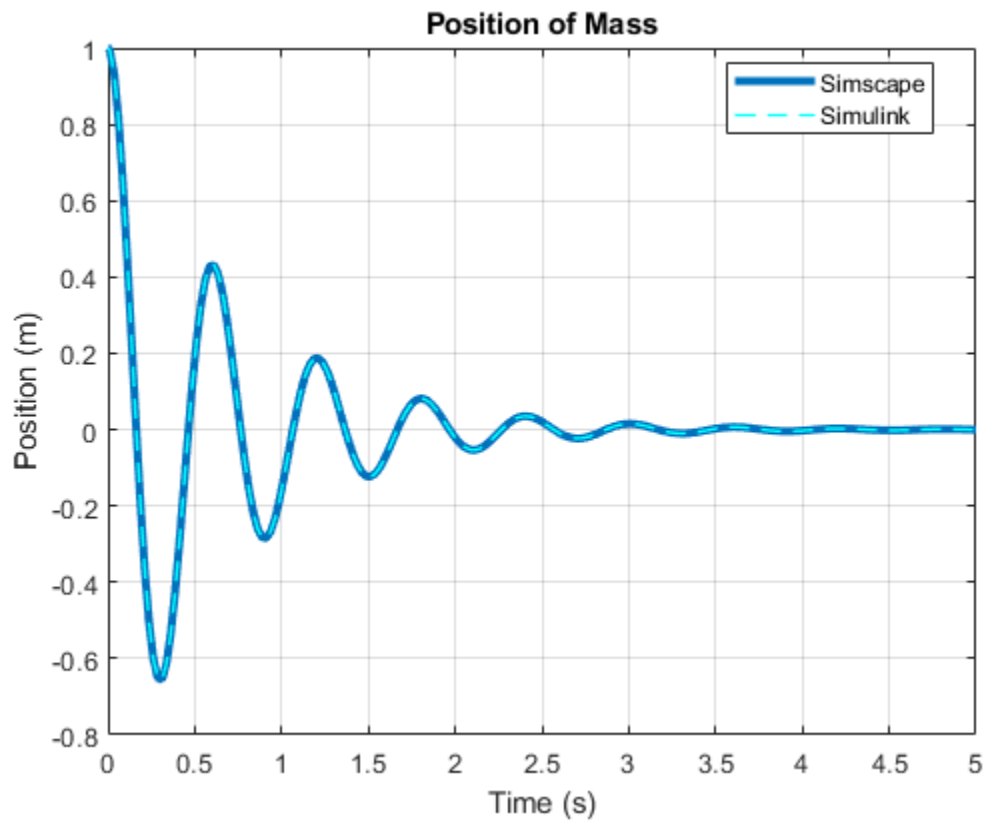
# Double Mass-Spring-Damper in Simulink and Simscape

This example shows two models of a double mass-spring-damper, one using Simulink® input/output blocks and one using Simscape™ physical networks.

The Simulink model uses signal connections, which define how data flows from one block to another. The Simscape model uses physical connections, which permit a bidirectional flow of energy between components. Physical connections make it possible to add further stages to the mass-spring-damper simply by using copy and paste. Input/output connections require rederiving and reimplementing the equations.

The initial deflection for each spring is 1 meter. This is shown in the block annotations for Spring1 and Spring2. The annotations on the Integrator blocks show the initial positions of the masses relative to a single fixed point in space.

**Model**
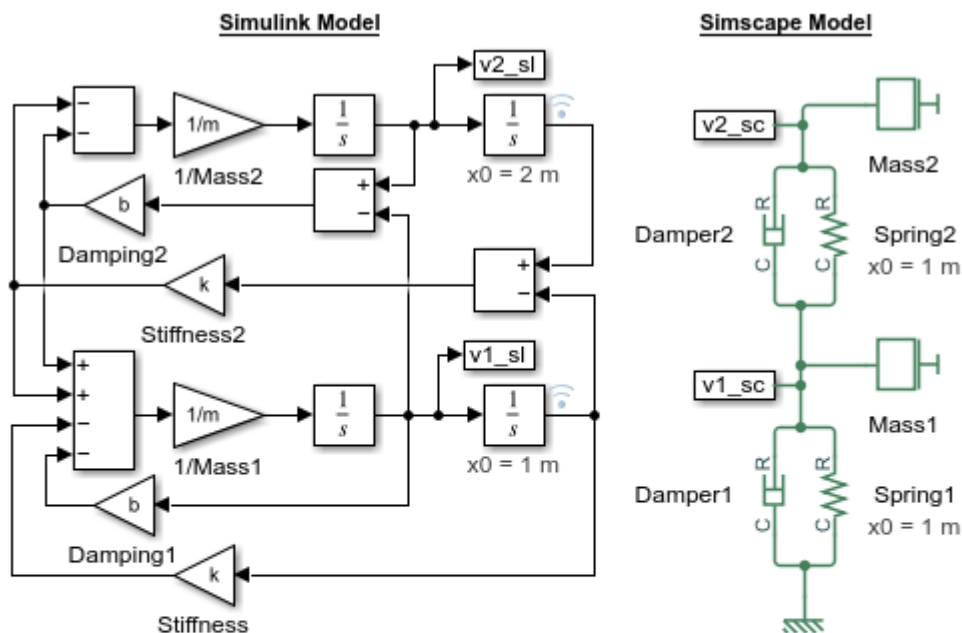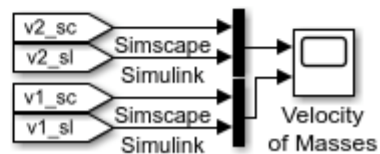
**Simulation Results from Simscape Logging**
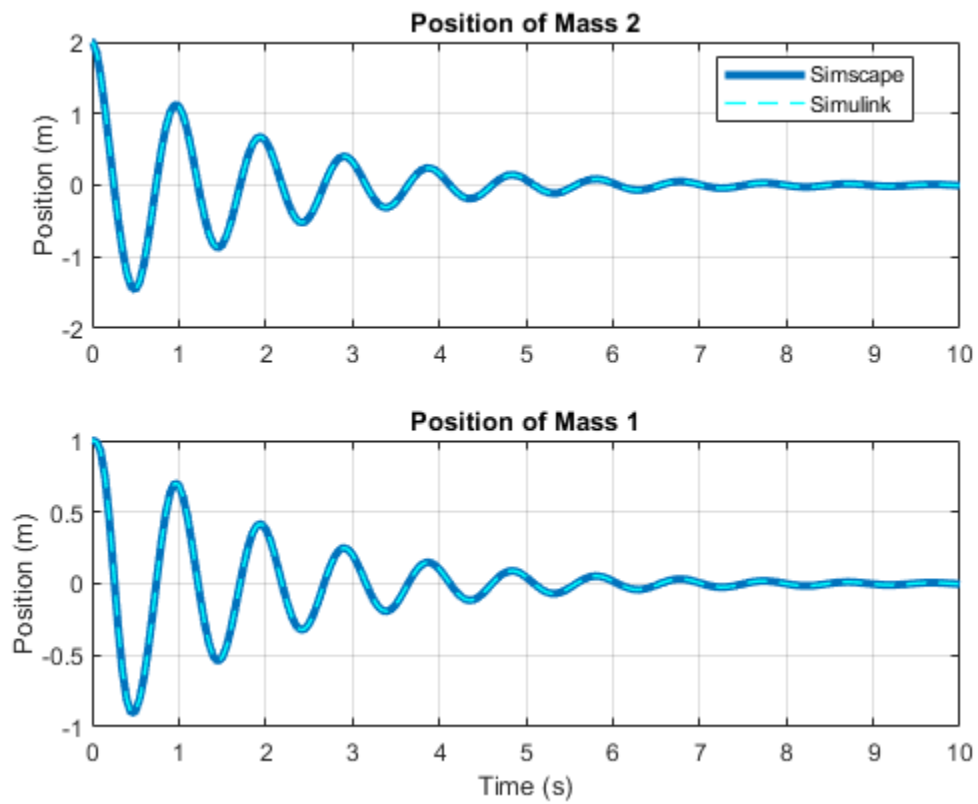
# Simple Mechanical System

This example shows a model of a system that connects rotational and translational motion. A summing lever drives a load consisting of a mass, viscous friction, and a spring connected to its joint C. Joint B is suspended on two rotational springs connected to reference point through a wheel and axle and a gear box. Joint A is connected to a torque source through a gear box and a wheel and axle mechanism.

**Model**
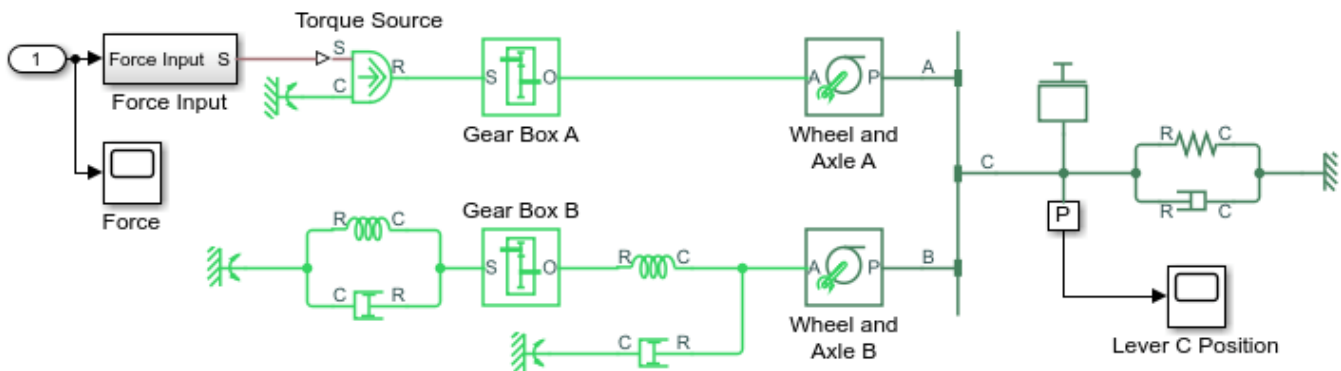


### Simple Mechanical System

1. Explore simulation results using sscexplore
2. Learn more about this example

Copyright 2005-2021 The MathWorks, Inc.

**Simulation Results from Scopes**
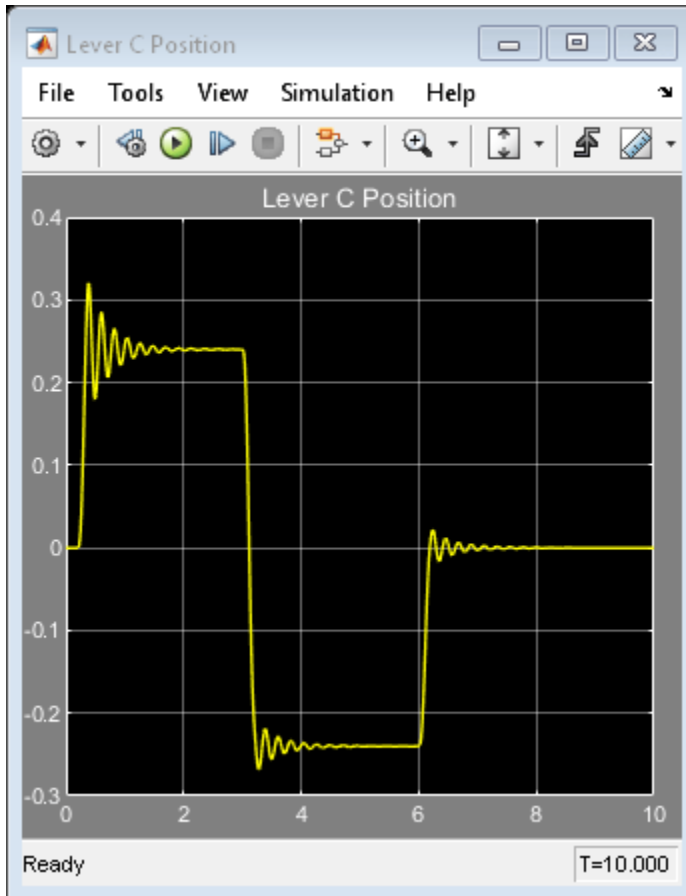
# Mechanical System with Translational Friction

This example shows a mass attached to a spring and a viscous damper. The mass is driven by an ideal velocity source through a friction element. The motion profile of the source is selected in such a way that plotting the displacement of the mass against the displacement provided by the source produces a typical hysteresis curve.

**Model**
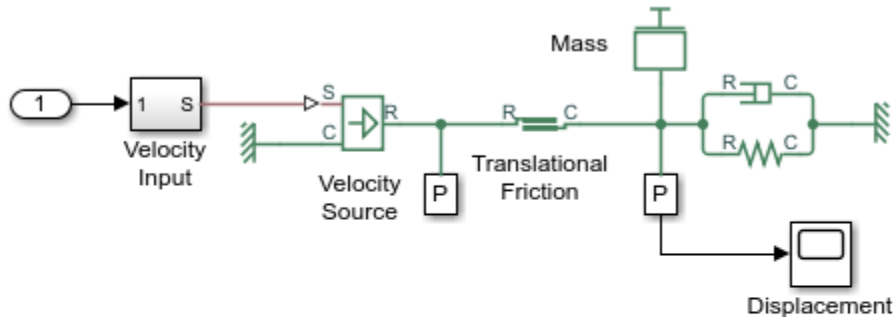


### Mechanical System with Translational Friction

1. Plot displacement of source and mass (see code)
2. Plot hysteresis curve due to friction effect (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**

Displacement Hysteresis Curve
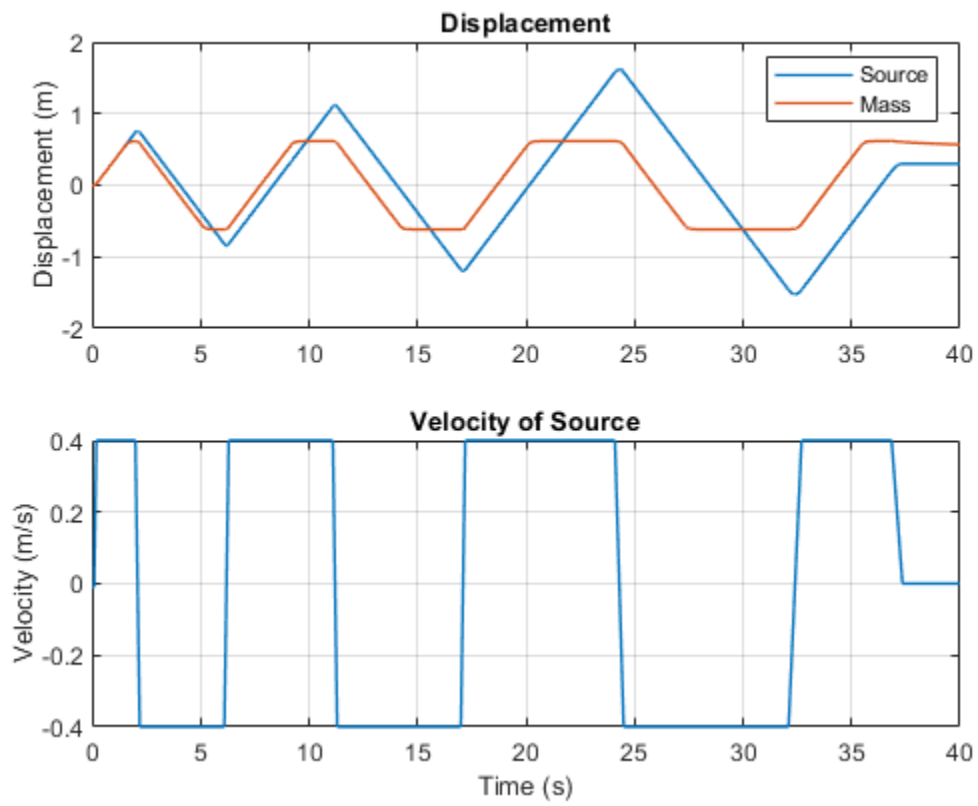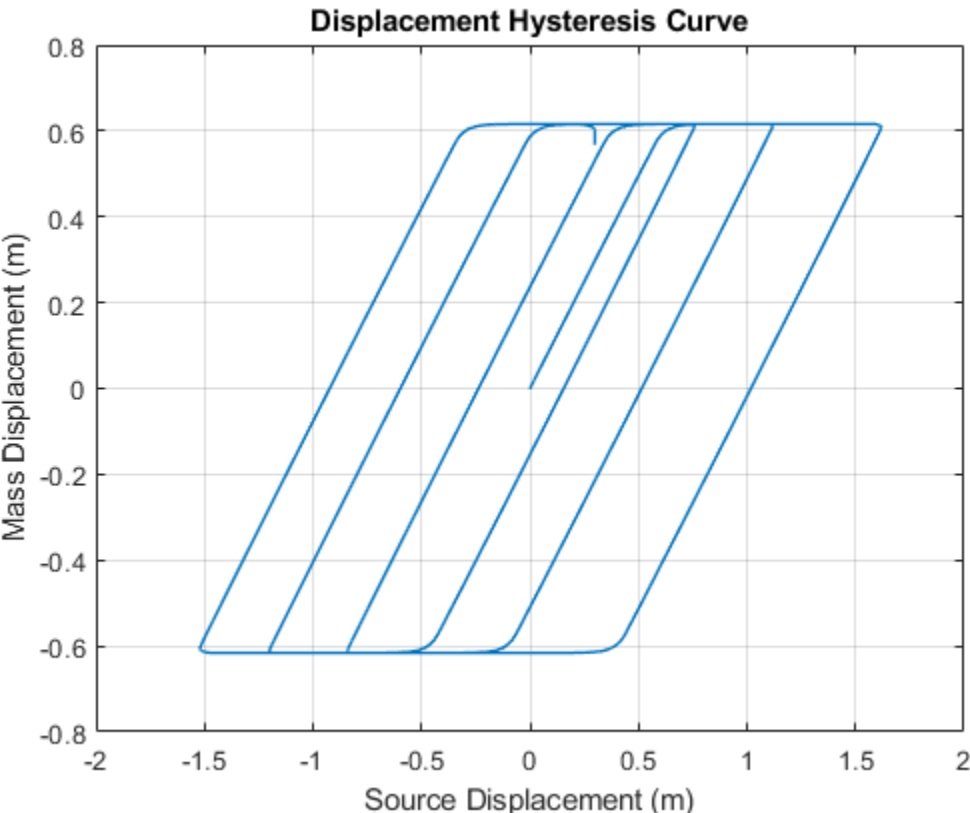
# Mechanical System with Translational Hard Stop

This example shows two masses connected by a hard stop. Mass 1 is driven by an Ideal Velocity Source. As the velocity input changes direction, Mass 2 will stay at rest until Mass 1 reaches the other end of the backlash modeled by the Translational Hard Stop. Plotting the displacement of Mass 2 against the displacement of the Mass 1 produces a typical hysteresis curve.
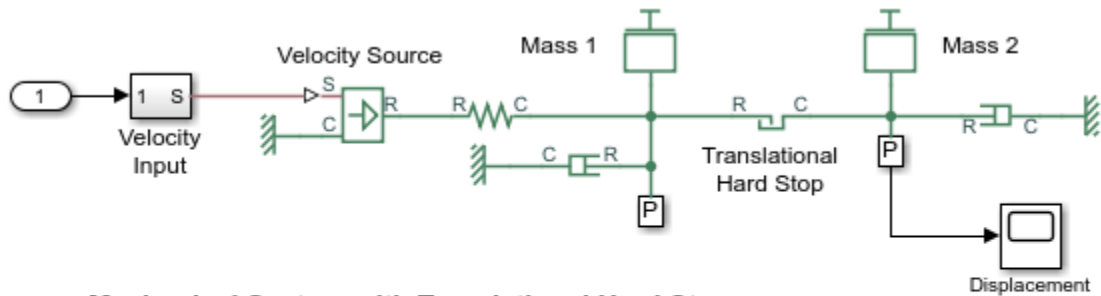
**Model**



**Mechanical System with Translational Hard Stop**

1. Plot displacement of Mass 1 and Mass 2 (see code)
2. Plot hysteresis curve of mass displacements (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2005-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**

Displacement Hysteresis Curve
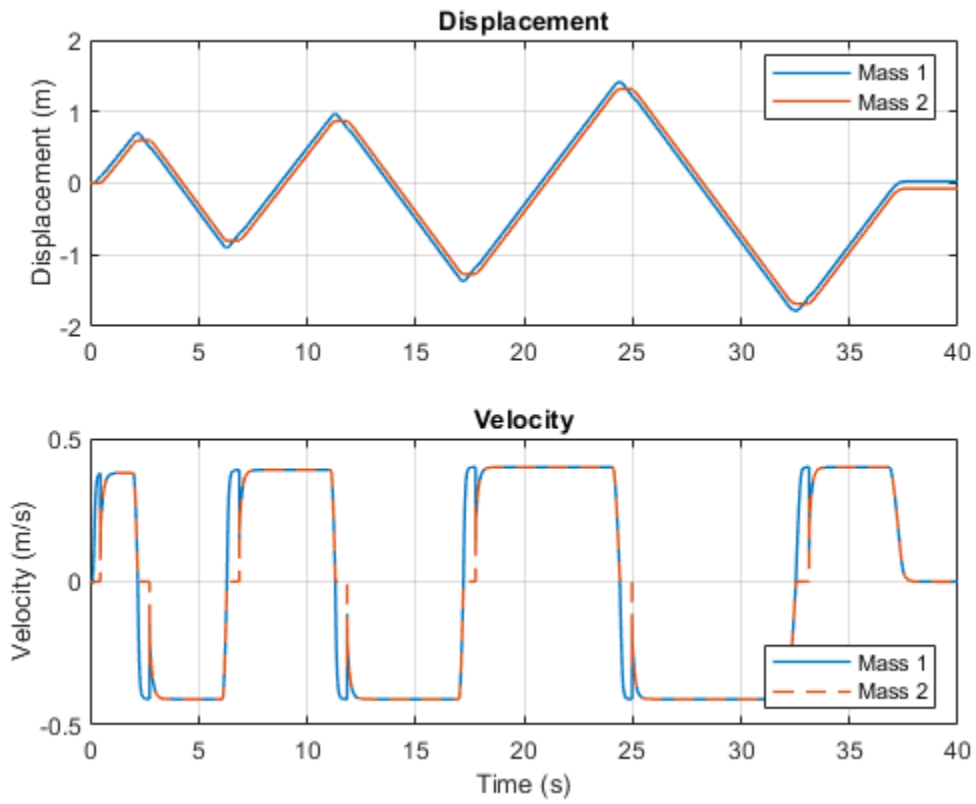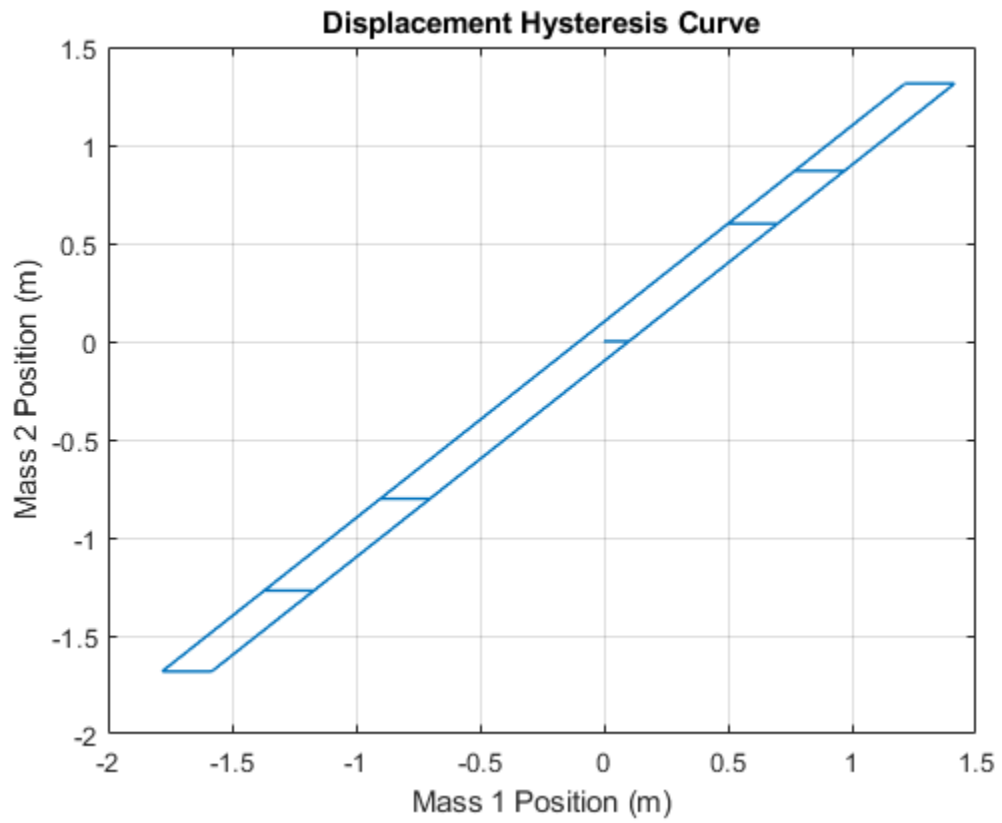
# Mechanical Rotational System with Stick-Slip Motion

This model shows a mechanical rotational system with stick-slip friction. An inertia is connected to a fixed point by spring and damper. The inertia is driven by a velocity source via a stick-slip friction element. The friction element has a difference between the breakaway and the Coulomb frictions, resulting in stick-slip motion of the inertia.

**Model**
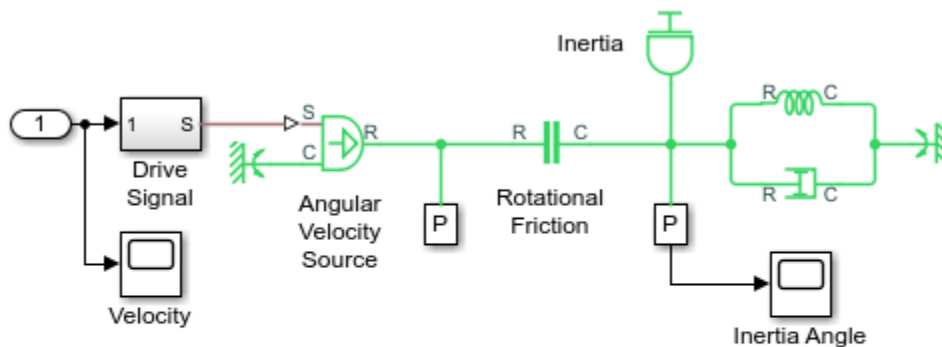


## Mechanical Rotational System with Stick-Slip Motion

1. Plot angle of source and inertia (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
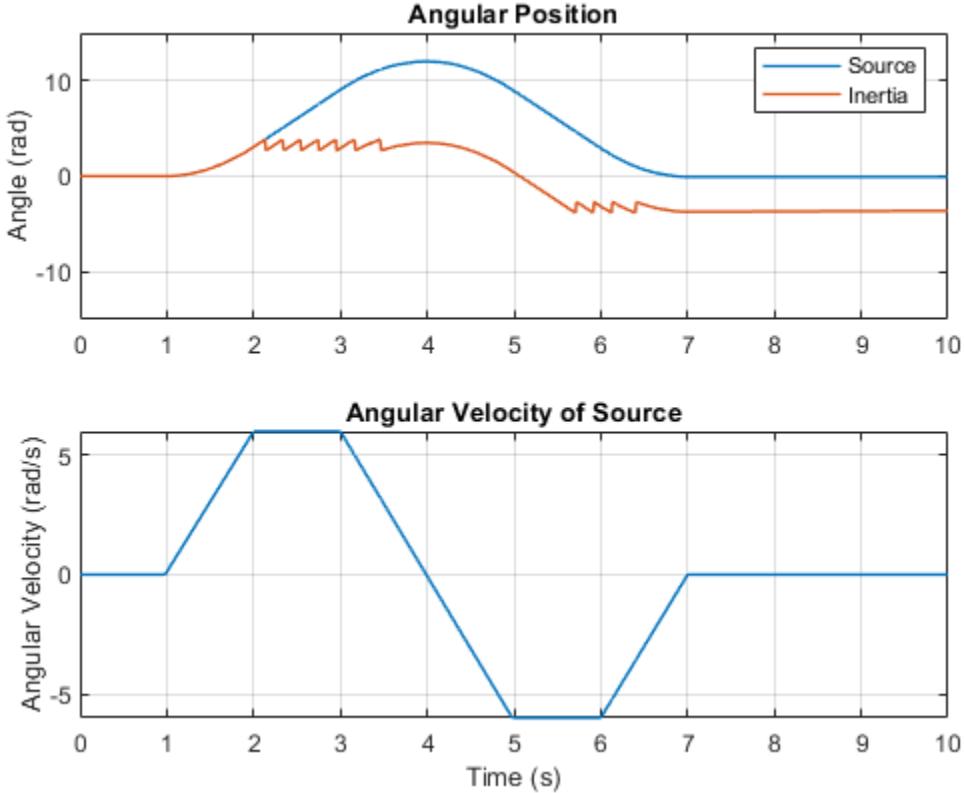
# Linkage Mechanism

This example shows the use of the Simscape™ Lever block in a linkage mechanism. Lever 1 and Lever 4 are first class levers with the fulcrum at the end. Lever 3 is a second class lever with the fulcrum in the middle. Lever 2 is a summing lever driven by the first and the third levers.

The mechanism is excited by two force sources. One source abruptly applies force at 1 second to Lever 1, then the other source abruptly applies force at 2 seconds to Lever 3.

**Model**
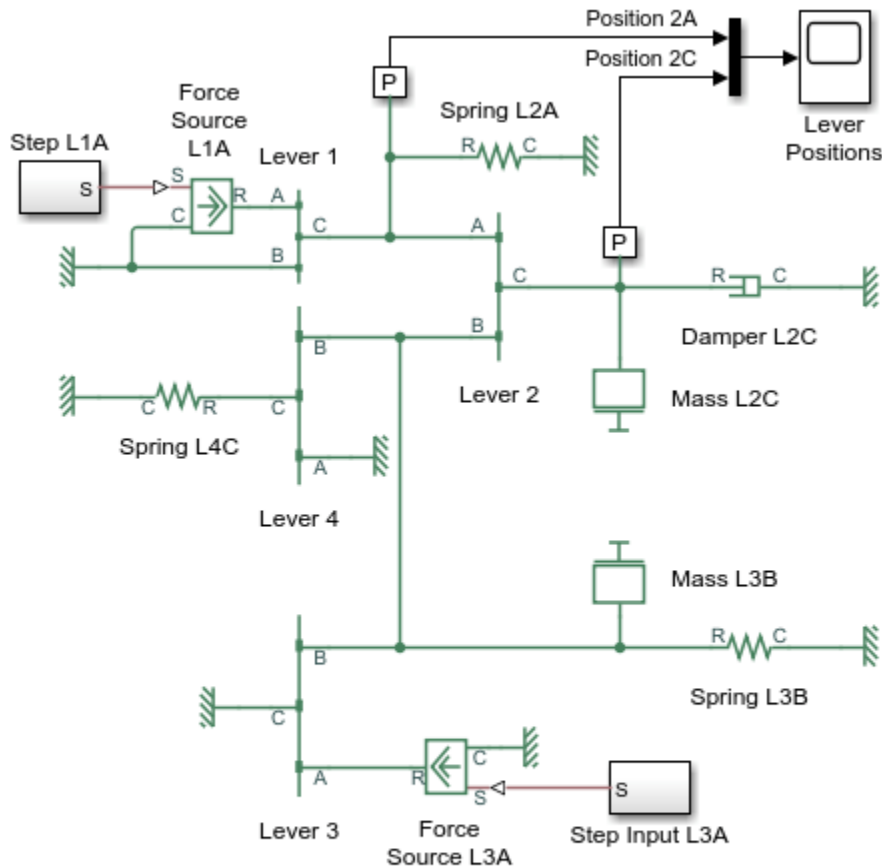


## Linkage Mechanism
1. Explore simulation results using sscexplore
2. Learn more about this example

Copyright 2005-2021 The MathWorks, Inc.

**Simulation Results from Scopes**
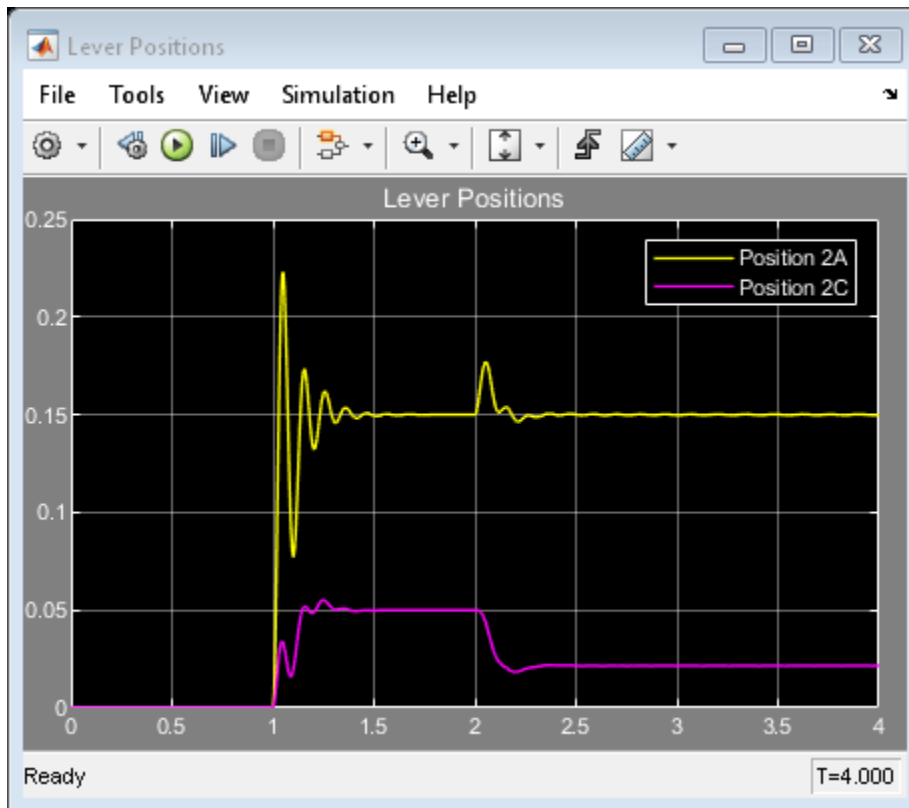
# Pendulum in Cartesian and Polar Coordinates

This example shows two different implementations of a planar pendulum.

The implementation in Cartesian coordinates has a high differential index, which means that nonlinear index reduction using the Projection method is required in order to simulate this model.

The implementation in Polar coordinates does not have a high differential index, which means that nonlinear index reduction is not required.

**Model**
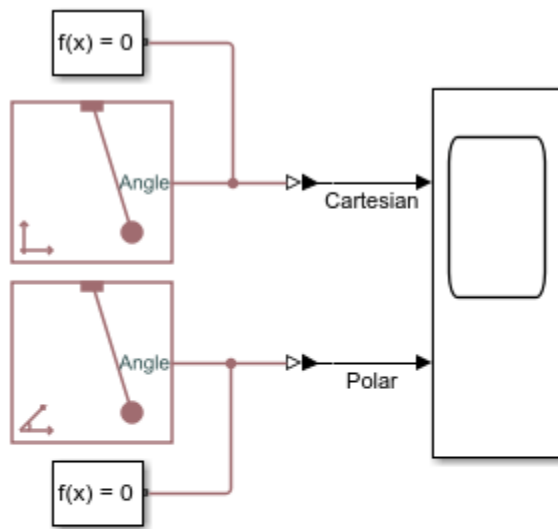


**Pendulum in Cartesian and Polar Coordinates**

1. Set parameters (see code)
2. Plot comparison of pendulum angle (see code)
3. Plot comparison of pendulum energy (see code)
4. Explore simulation results using sscexplore
5. Learn more about this example

Copyright 2021 The MathWorks, Inc.

**Model Settings**

The Cartesian implementation of a planar pendulum has a differential index of three, and is not compatible with the Derivative Replacement method. The "Projection" method must be used in order to simulate this implementation, and neither "None" nor "Derivative replacement" will simulate successfully.

**Simulation Results from Simscape Logging**

The figure shows the pendulum angle as a function of time. The amplitude decreases due to the damping in the model.
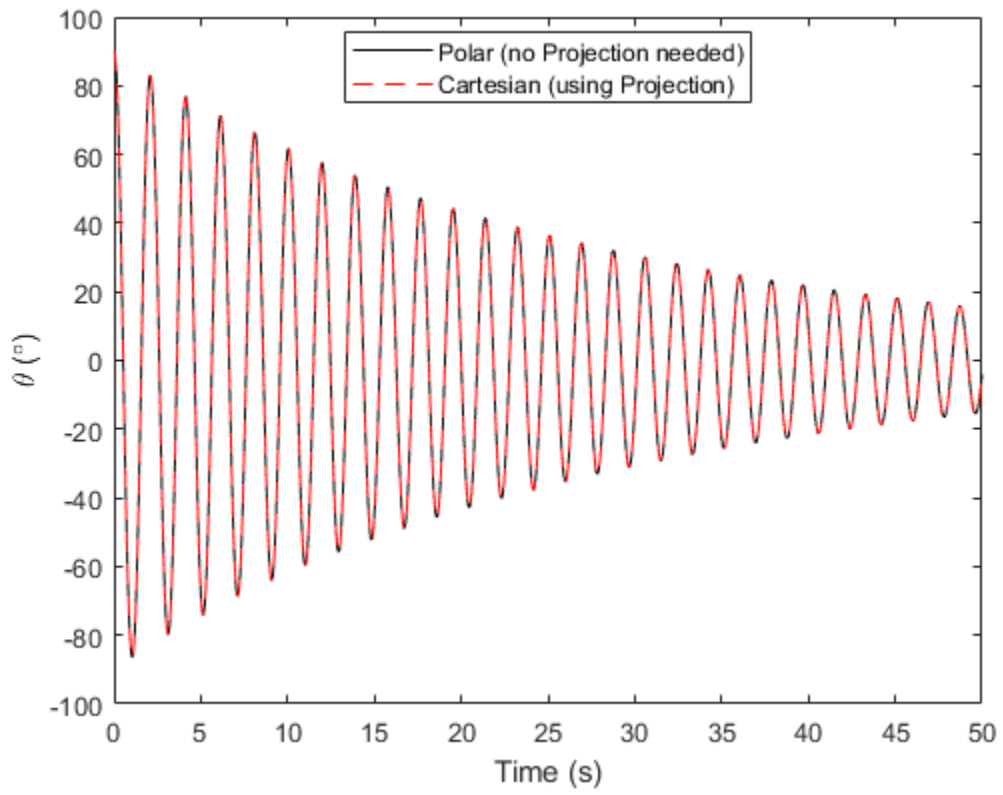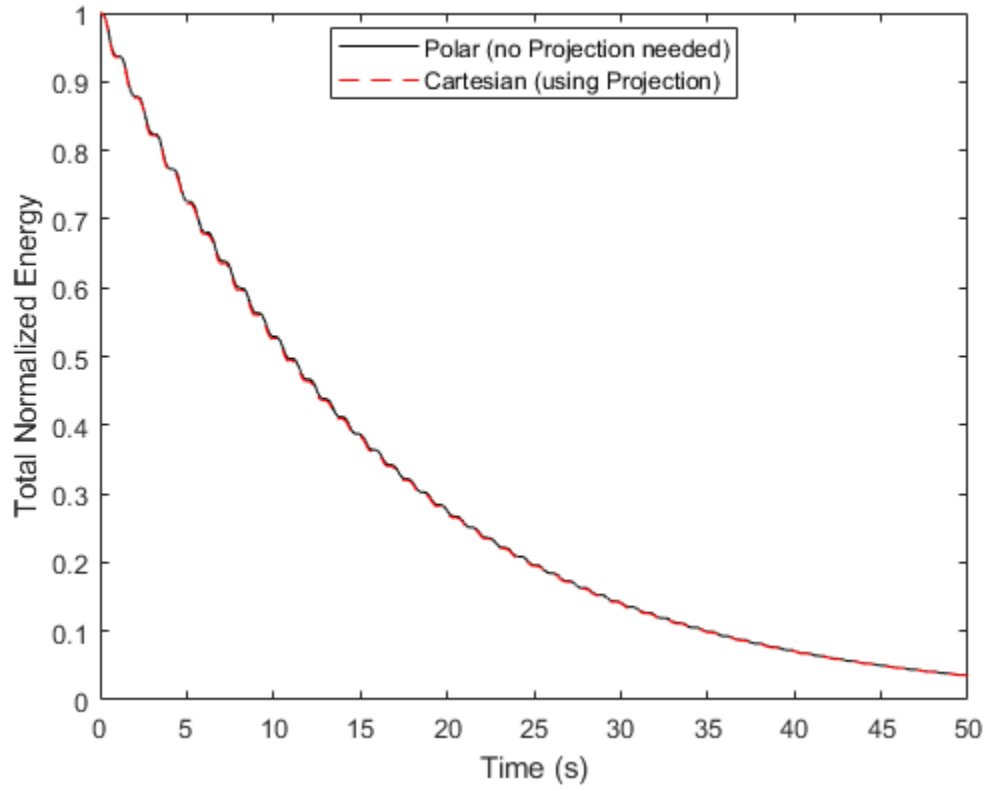
# Creating a New Circuit

This example shows a starting point for creation of a new electrical model. The model also opens an Electrical Starter Palette that shows how you can create your own customized library that also provides links to Foundation Library components.

Another way to create a new model with Solver Configuration block, Reference block and solver type predefined is to use ssc_new.m e.g. ssc_new('MyModel','electrical','ode23t').

**Model**

## Creating a New Circuit

1. Open electrical starter palette library
2. Plot current in resistor (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example
5. Learn more about creating new Simscape models with ssc_new

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
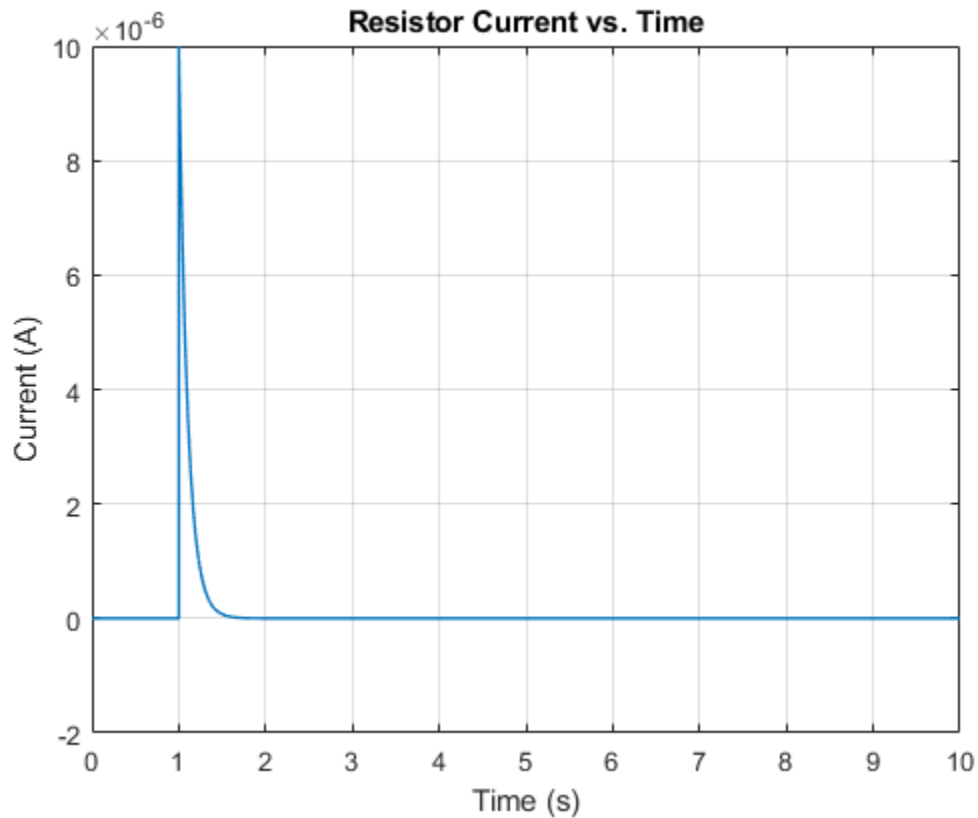
# RC Circuit in Simulink and Simscape

This example shows two models of an RC circuit, one using Simulink® input/output blocks and one using Simscape™ physical networks.

The Simulink uses signal connections, which define how data flows from one block to another. The Simscape model uses physical connections, which permit a bidirectional flow of energy between components. Physical connections make it possible to add further stages to the RC circuit simply by using copy and paste. Input/output connections require rederiving and reimplementing the circuit equations.

The circuit is driven by a voltage square wave. Resistance and capacitance values are defined using MATLAB® variables.

**Model**



## RC Circuit in Simulink and Simscape

1. Plot resistor current (see code)
2. Explore simulation results using sscexplore
3. Open model of cascaded RC circuit
4. Learn more about this example
5. Learn more about modeling physical networks

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Scopes**
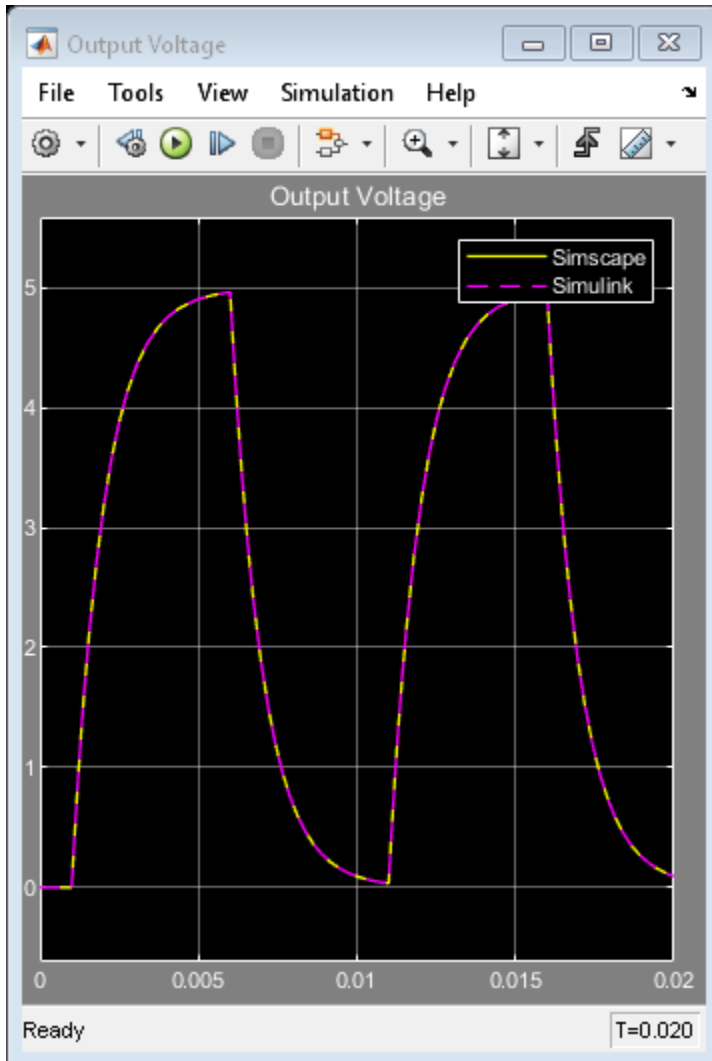
**Simulation Results from Simscape Logging**
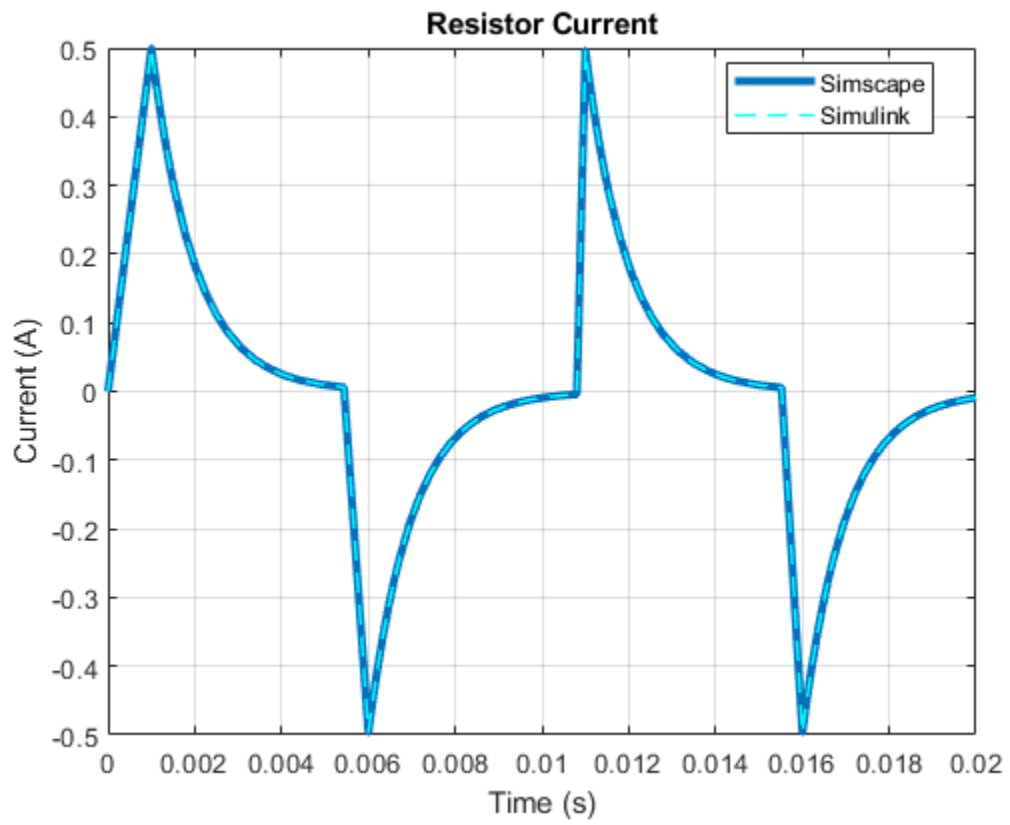
# Cascaded RC Circuit in Simulink and Simscape

This example shows two models of a cascaded RC circuit, one using Simulink® input/output blocks and one using Simscape™ physical networks.

The Simulink model is built using signal connections, which define how data flows from one block to another. The Simscape model is built using physical connections, which permit a bidirectional flow of energy between components. Physical connections make it possible to add further stages to the RC circuit simply by using copy and paste. Input/output connections require rederiving and reimplementing the circuit equations.

The circuit is driven by a voltage square wave. Resistance and capacitance values are defined using MATLAB® variables.

**Model**



### Cascaded RC Circuit in Simulink and Simscape

1. Plot resistor current (see code)
2. Explore simulation results using sscexplore
3. Open model of single RC circuit
4. Learn more about this example
5. Learn more about modeling physical networks

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Scopes**
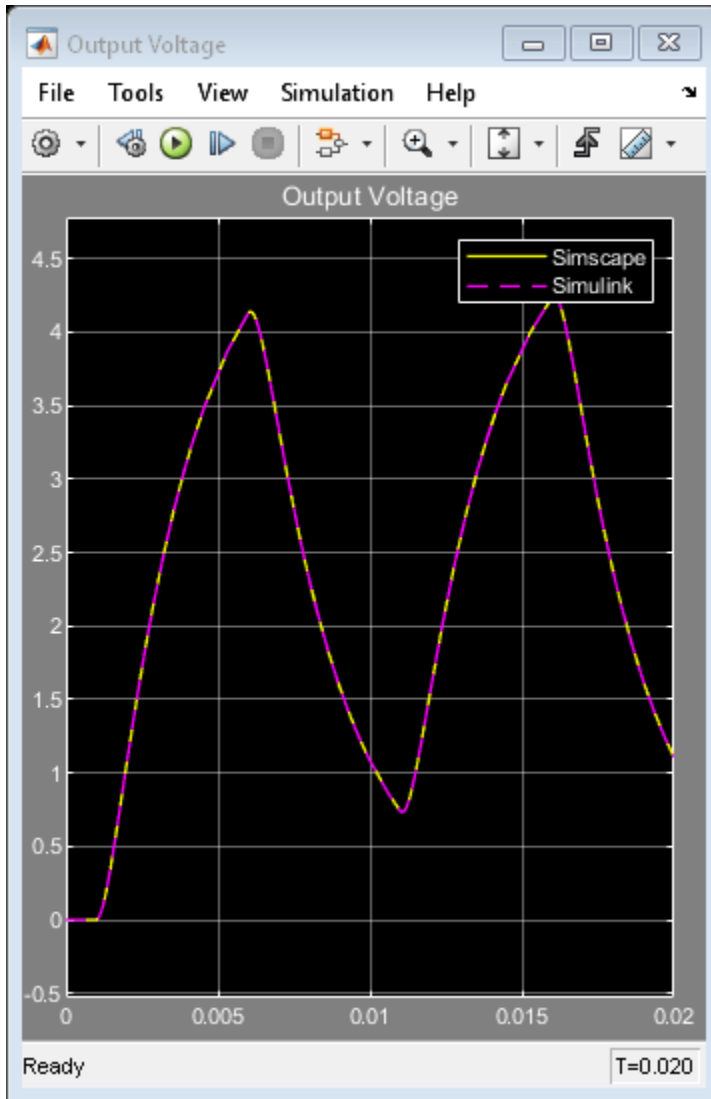
**Simulation Results from Simscape Logging**
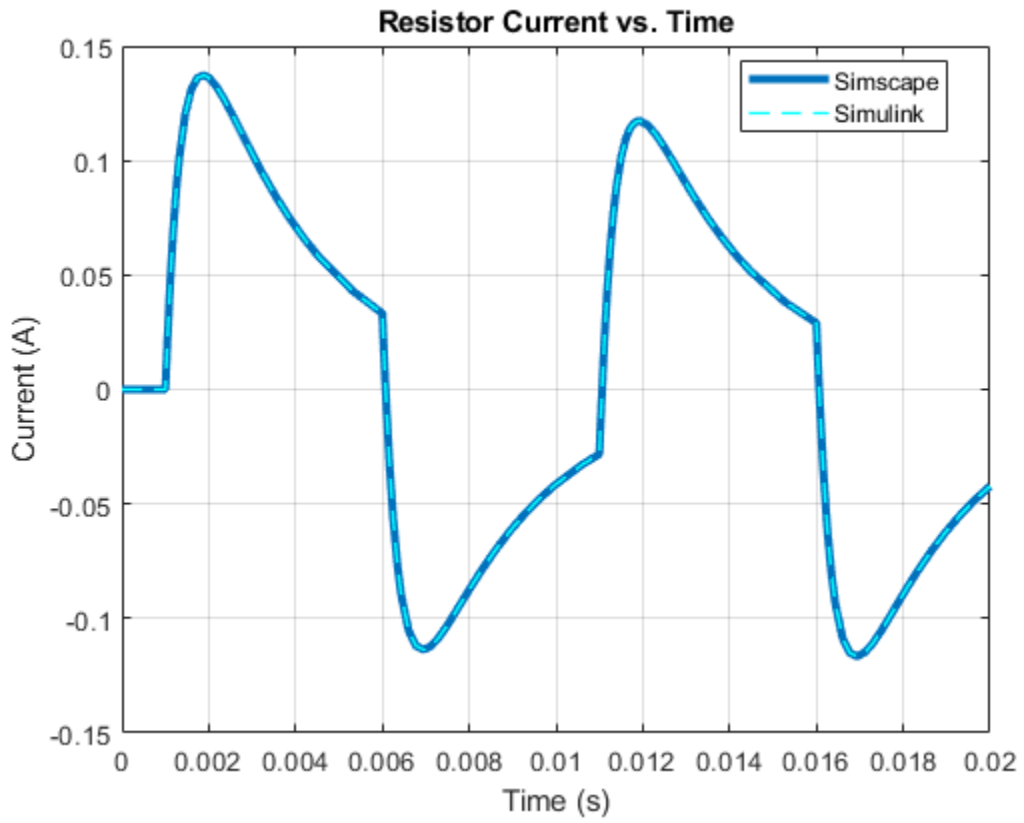
# Shunt Motor

This example shows a model of a shunt motor. In a shunt motor, the field and armature windings are connected in parallel. Equivalent circuit parameters are armature resistance Ra = 110 Ohms, field resistance Rf = 2.46KOhms, and back emf coefficient Laf = 5.11. The back-emf is given by Laf*If*Ia*w, where If is the field current, Ia is the armature current, and w is the rotor speed in radians/s. The rotor inertia J is 2.2e-4kgm^2, and rotor damping B is 2.8e-6Nm/(radian/s).

Manufacturer data for this model gives the no-load speed as 4600rpm, and speed at rated load as 4000rpm. Simulating the model confirms these values and correct calculation of equivalent circuit values.

**Model**
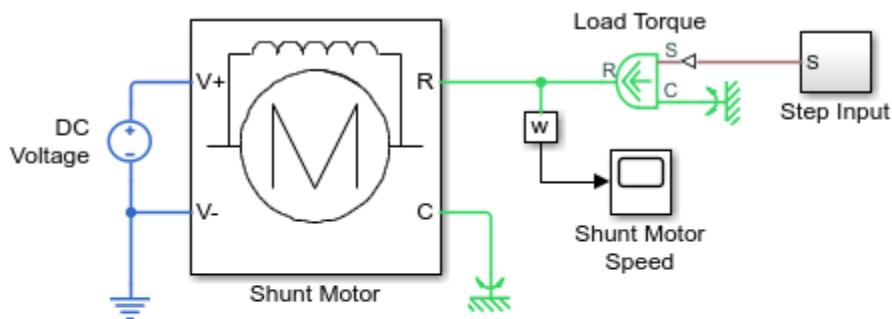


## Shunt Motor

1. Plot current and load torque (see code)
2. Explore simulation results using sscexplore
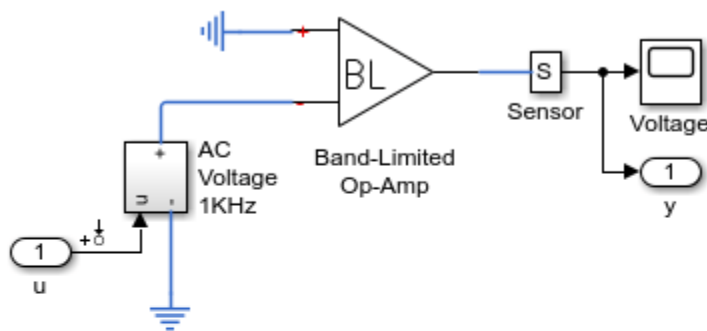3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Shunt Motor Subsystem**
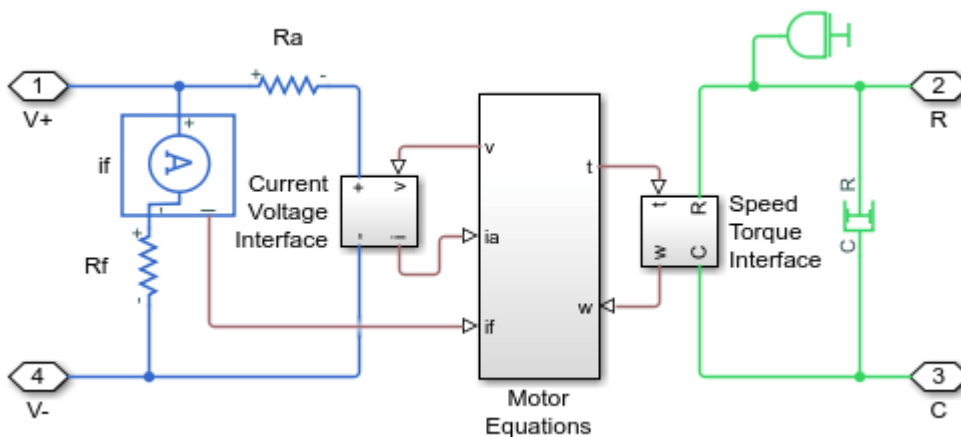
**Simulation Results from Scopes**
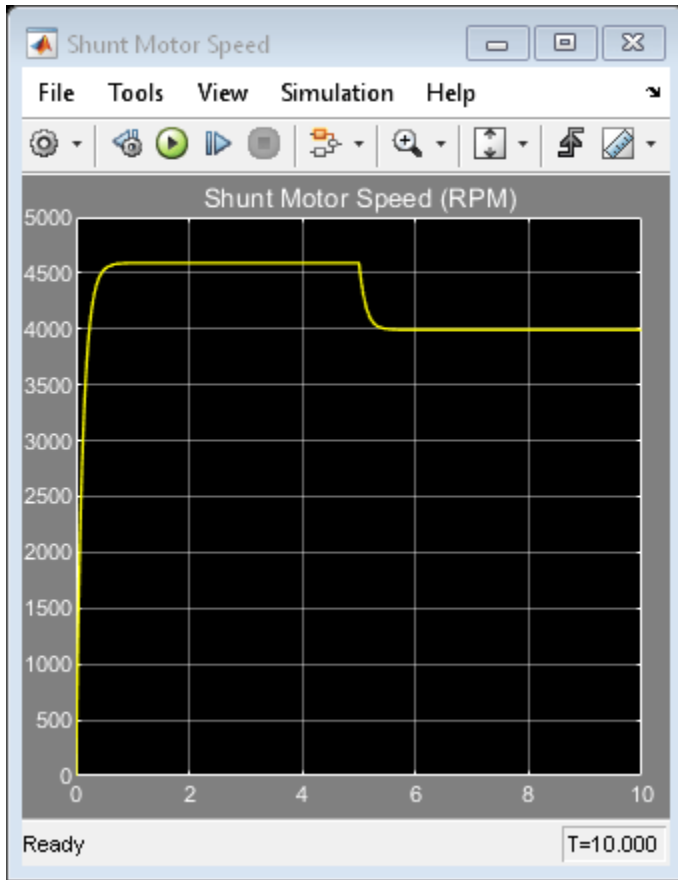
## Simulation Results from Simscape Logging
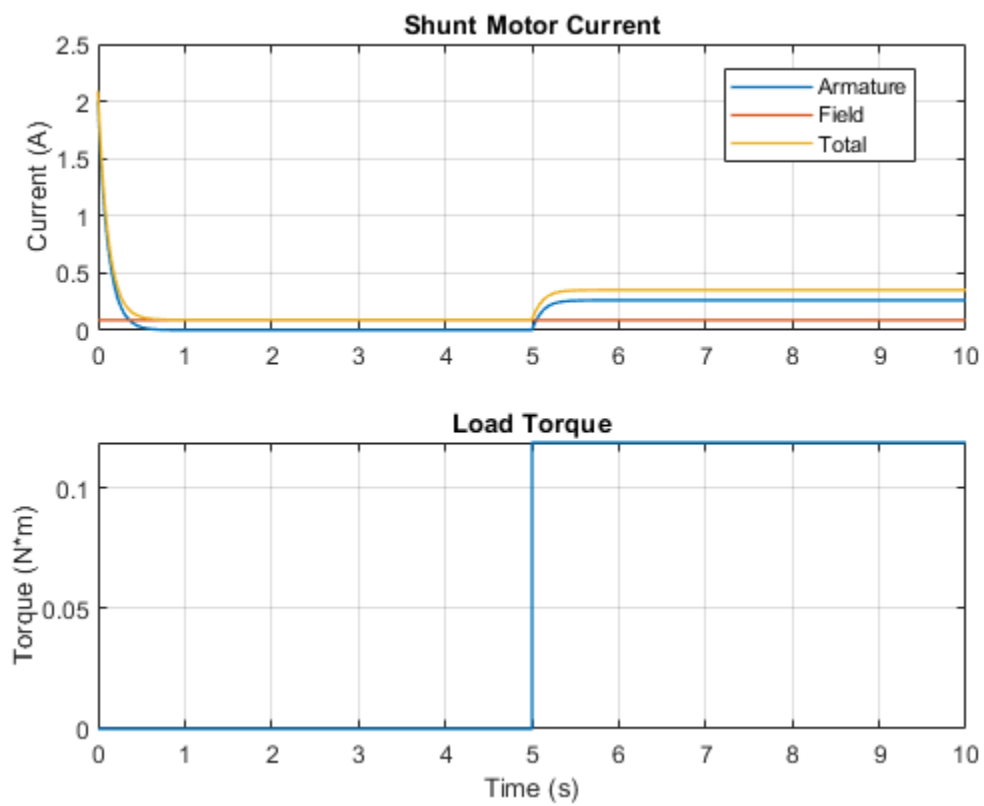
# Permanent Magnet DC Motor

This model is based on a Faulhaber Series 0615 DC-Micromotor. The parameters values are set to match the 1.5V variant of this motor. The model uses these parameters to verify manufacturer-quoted no-load speed, no-load current, and stall torque.

When running the simulation, for the first 0.1 seconds the motor has no external load, and the speed builds up to the no-load value. Then at 0.1 seconds the stall torque is applied as a load to the motor shaft. The simulation results shows a good level of agreement with manufacturer data.

Often manufacturers do not quote the equivalent circuit parameters, and they must be estimated from information such as no-load speed, stall torque, and efficiency. A test harness such as this model can then be used to test the estimated equivalent circuit prior to using the motor model in a complete system simulation.

**Model**
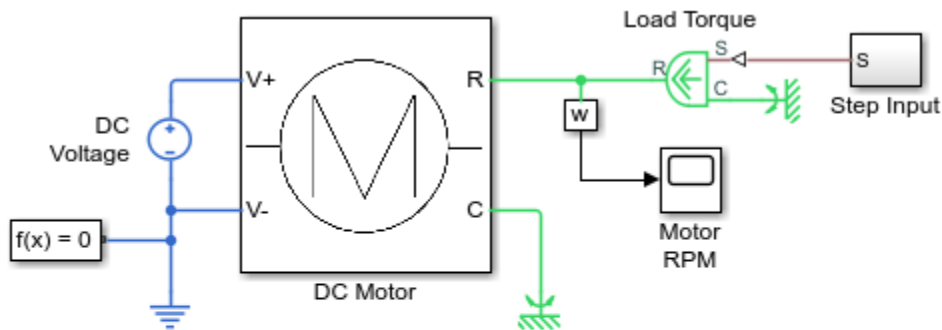


**Permanent Magnet DC Motor**

1. Plot current and load torque (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**DC Motor Subsystem**
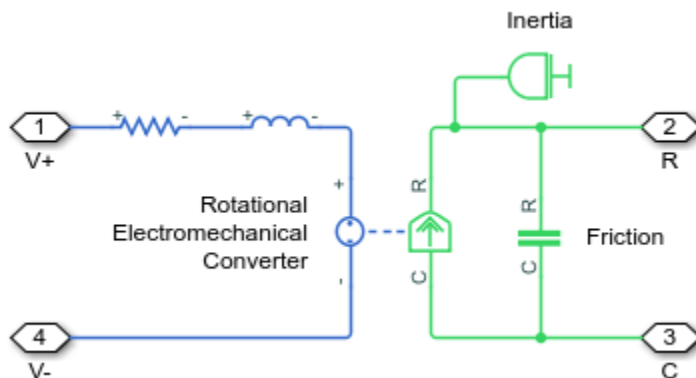
**Simulation Results from Simscape Logging**
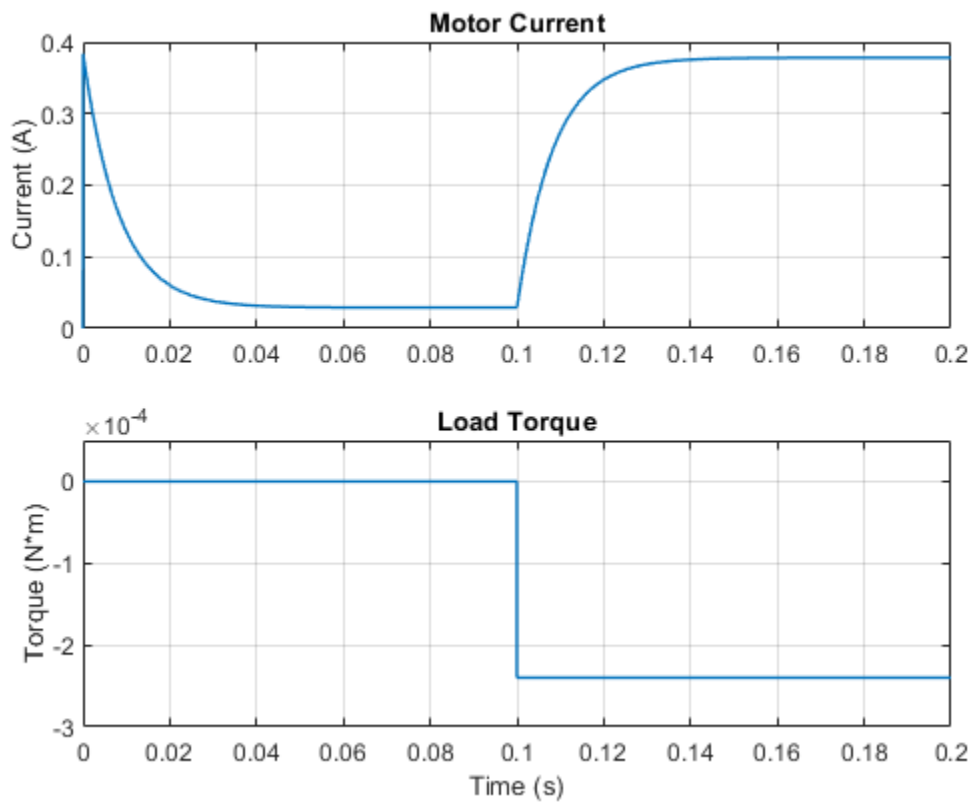
# Lead-Acid Battery

This example shows how to model a lead-acid battery cell using the Simscape™ language to implement the nonlinear equations of the equivalent circuit components. In this way, as opposed to modeling entirely in Simulink®, the connection between model components and the defining physical equations is more easily understood. For the defining equations and their validation, see Jackey, R. "A Simple, Effective Lead-Acid Battery Modeling Process for Electrical System Component Selection", SAE World Congress & Exhibition, April 2007, ref. 2007-01-0778.

In this simulation, initially the battery is discharged at a constant current of 10A. The battery is then recharged at a constant 10A back to the initial state of charge. The battery is then discharged and recharged again. A simple thermal model is used to model battery temperature. It is assumed that cooling is primarily via convection, and that heating is primarily from battery internal resistance, R2. A standard 12 V lead-acid battery can be modeled by connecting six copies of the 2V battery cell block in series.

This model is constructed using the Simscape example library LeadAcidBattery_lib. The library comes built and on your path so that it is readily executable. However, it is recommended that you copy the source files to a new directory, for which you have write permission, and add that directory to your MATLAB® path. This will allow you to make changes and rebuild the library for yourself. The source files for the example library are in the following package directory: matlabroot/toolbox/physmod/ simscape/simscapedemos/+LeadAcidBattery where matlabroot is the MATLAB root directory on your machine, as returned by entering matlabroot in the MATLAB Command Window.

**Model**



## Lead-Acid Battery

1. Plot battery current and SOC in a figure (see code)
2. View battery current and SOC in Simulation Data Inspector
3. Browse simulation results in Simulation Data Inspector
4. Explore simulation results using sscexplore
5. Open custom library used by Battery Cell block
6. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Battery Cell Subsystem**



**Battery Thermal Model Subsystem**

**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

The figure below shows the battery current and state of charge in a MATLAB figure. You can also view the data in the Simscape Results Explorer and the Simulation Data Inspector.

**Battery Current (Lead Acid)**

**Battery SOC (Lead Acid)**

# Lead-Acid Battery with Dashboard Blocks

This example shows how to model a lead-acid battery cell using the Simscape™ language.

Implementing the nonlinear equations of the equivalent circuit components in Simscape as opposed to modeling entirely in Simulink® more clearly relates the model components to the defining physical equations. For the defining equations and their validation, see Jackey, R. "A Simple, Effective Lead-Acid Battery Modeling Process for Electrical System Component Selection", SAE World Congress & Exhibition, April 2007, ref. 2007-01-0778.

In this simulation, initially the battery is discharged at a constant current of 10A. The battery is then recharged at a constant 10A back to the initial state of charge. The battery is then discharged and recharged again. A simple thermal model is used to model battery temperature. It is assumed that cooling is primarily via convection, and that heating is primarily from battery internal resistance, R2. A standard 12 V lead-acid battery can be modeled by connecting six copies of the 2V battery cell block in series.

This model is constructed using the Simscape example library LeadAcidBattery_lib. The library comes built and on your path so that it is readily executable. However, it is recommended that you copy the source files to a new directory, for which you have write permission, and add that directory to your MATLAB® path. This will allow you to make changes and rebuild the library for yourself. The source files for the example library are in the following package directory: matlabroot/toolbox/physmod/simscape/simscapedemos/+LeadAcidBattery where matlabroot is the MATLAB root directory on your machine, as returned by entering matlabroot in the MATLAB Command Window.

This example includes animations of the current, state of charge, and temperature created using a Circular Gauge and two Vertical Gauge blocks from the Simulink / Dashboard / Customizable Blocks library.

**Model**



**Lead-Acid Battery with Dashboard Blocks**

1. Plot battery current and SOC in a figure (see code)
2. View battery current and SOC in Simulation Data Inspector
3. Browse simulation results in Simulation Data Inspector
4. Explore simulation results using sscexplore
5. Open custom library used by Battery Cell block
6. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Battery Cell Subsystem**



**Battery Thermal Model Subsystem**

**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

The figure below shows the battery current and state of charge in a MATLAB figure. You can also view this data in the Simulation Data Inspector by clicking the **View battery current SOC** hyperlink in the model canvas or by clicking the **Data Inspector** button on the model **Simulation** tab.
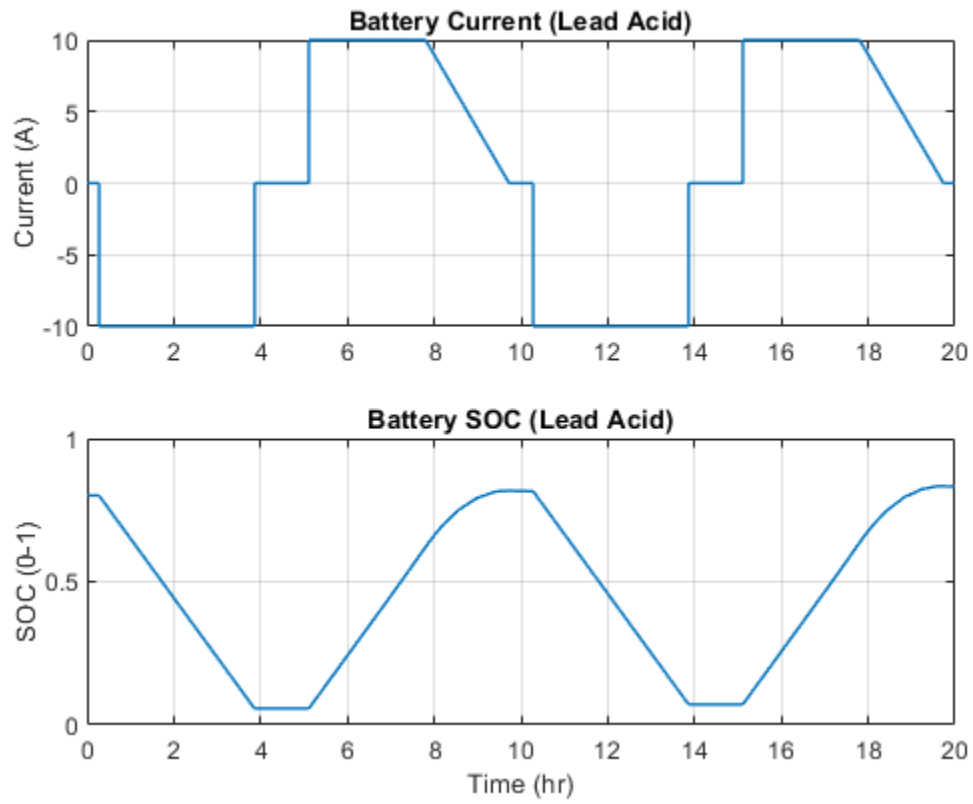
# Lithium-Ion Battery Pack with Fault

This example shows how to simulate a battery pack consisting of multiple series-connected cells in an efficient manner. It also shows how a fault can be introduced into one of the cells to see the impact on battery performance and cell temperatures. For efficiency, identical series-connected cells are not just simply modeled by connecting cell models in series. Instead a single cell is used, and the terminal voltage scaled up by the number of cells. The fault is represented by changing the parameters for the Cell 10 Fault subsystem, reducing both capacity and open-circuit voltage, and increasing the resistance values.

**Model**
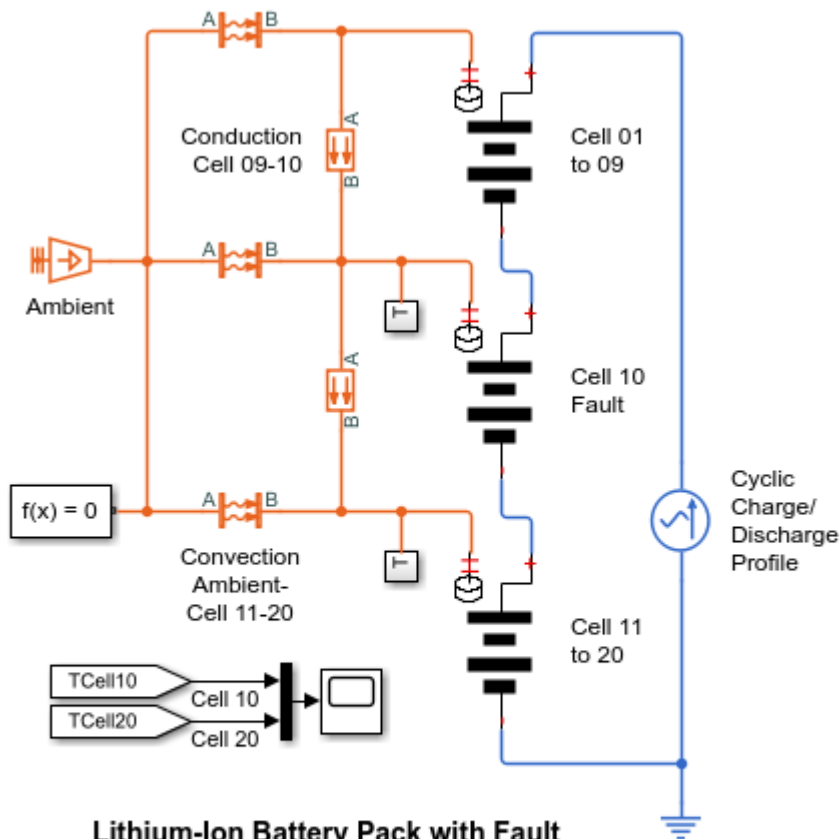


**Lithium-Ion Battery Pack with Fault**

1. Plot temperature & SOC for different cells (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Cell 01 to 09 Subsystem**
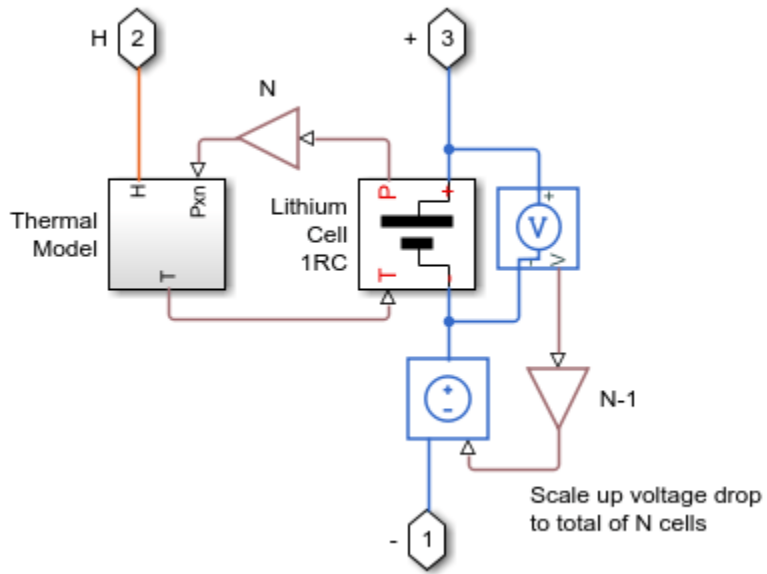


**Lithium Cell 1RC Subsystem**
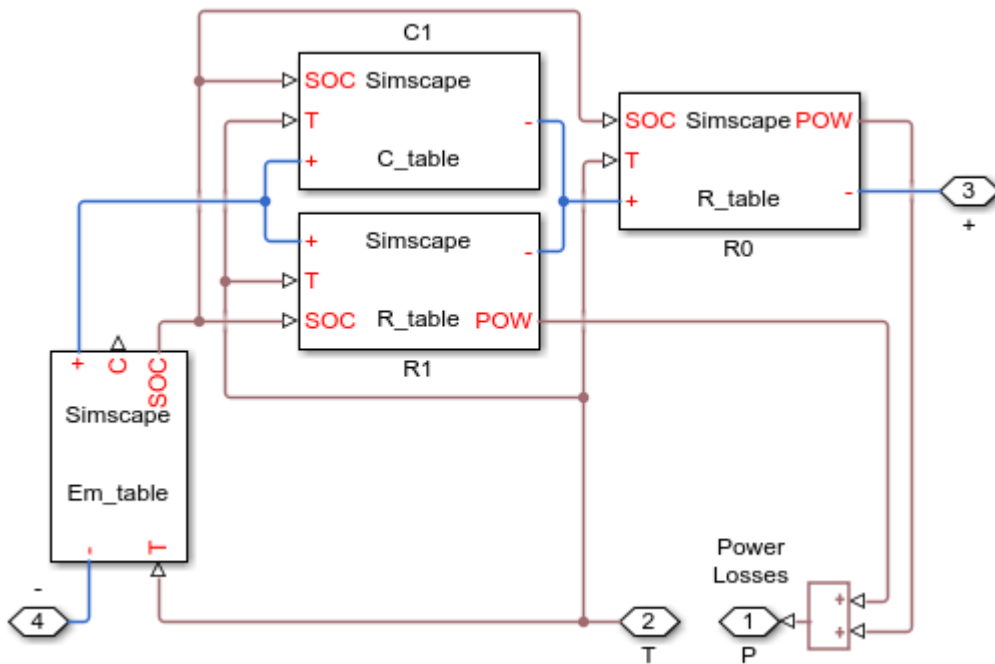
**Simulation Results from Simscape Logging**
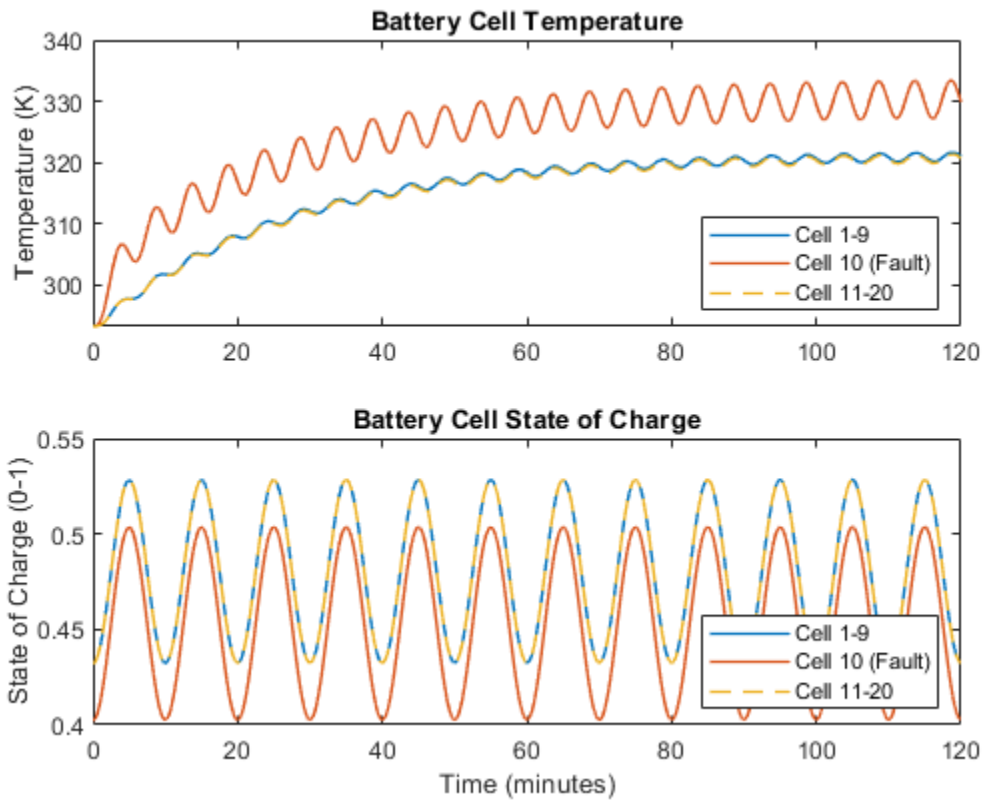
# Lithium-Ion Battery Pack with Fault Using Arrays

This example shows how to simulate a battery pack that consists of multiple series-connected cells. It also shows how you can introduce a fault into one of the cells to see the impact on battery performance and cell temperatures. The battery pack is modeled in Simscape™ language by connecting cell models in series using arrays. You can represent the fault by defining different parameters for the faulty cell.

**Array of Components**

Simscape language allows you to declare arrays of components using `for` loops. You can also use `for` loops to specify connections between the member components. For example, you can declare multiple battery cells and connect them together. For more information, see the "Case Study — Battery Pack with Fault Using Arrays".

**Model**
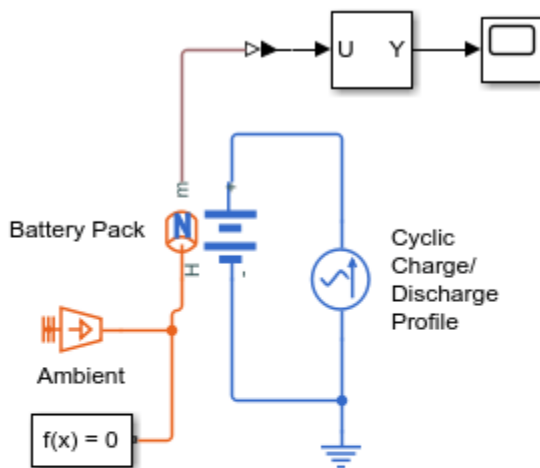


## Lithium-Ion Battery Pack with Fault Using Arrays

1. Plot temperature & SOC for different cells (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
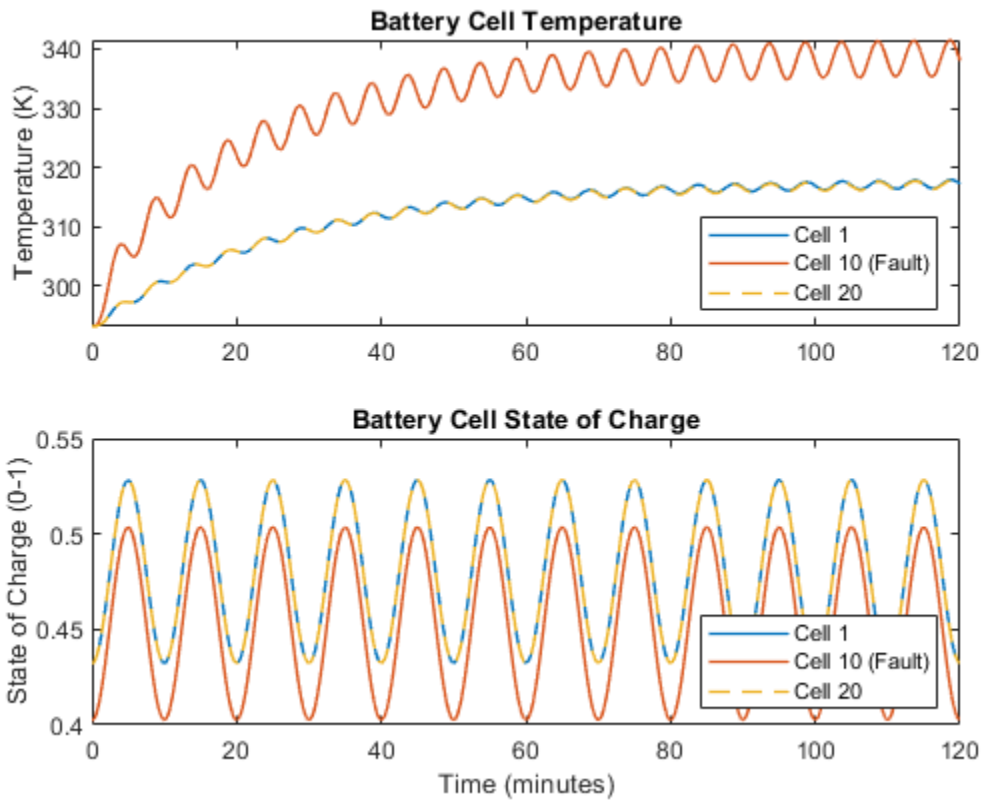
# Lithium Battery Cell - One RC-Branch Equivalent Circuit

This example shows how to model a lithium cell using the Simscape™ language to implement the elements of an equivalent circuit model with one RC branch. For the defining equations and their validation, see T. Huria, M. Ceraolo, J. Gazzarri, R. Jackey. "High Fidelity Electrical Model with Thermal Dependence for Characterization and Simulation of High Power Lithium Battery Cells," IEEE International Electric Vehicle Conference, March 2012.

A simple thermal model is used to model battery temperature. It is assumed that cooling is primarily via convection, and that heating is primarily from internal resistance. A battery pack can be modeled by connecting multiple copies of the battery cell block in series.

**Model**



### Lithium Battery Cell - One RC-Branch Equivalent Circuit

1. Plot current and SOC for battery (see code)
2. Explore simulation results using sscexplore
3. Open custom library used by Battery Cell block
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Lithium Cell 1RC Subsystem**
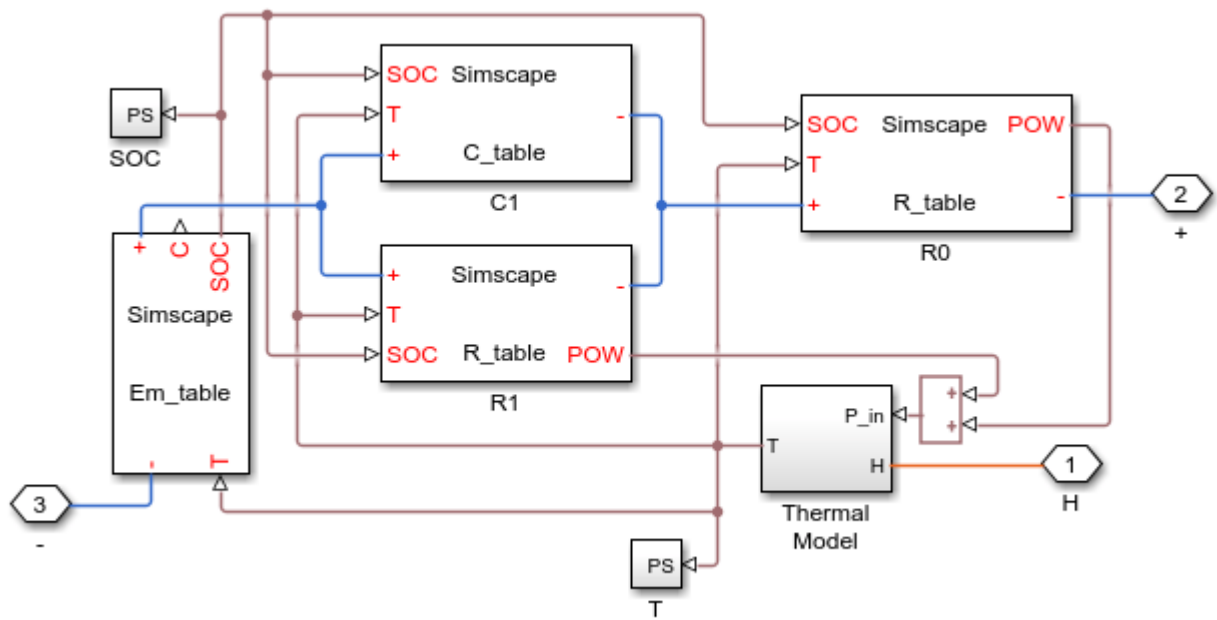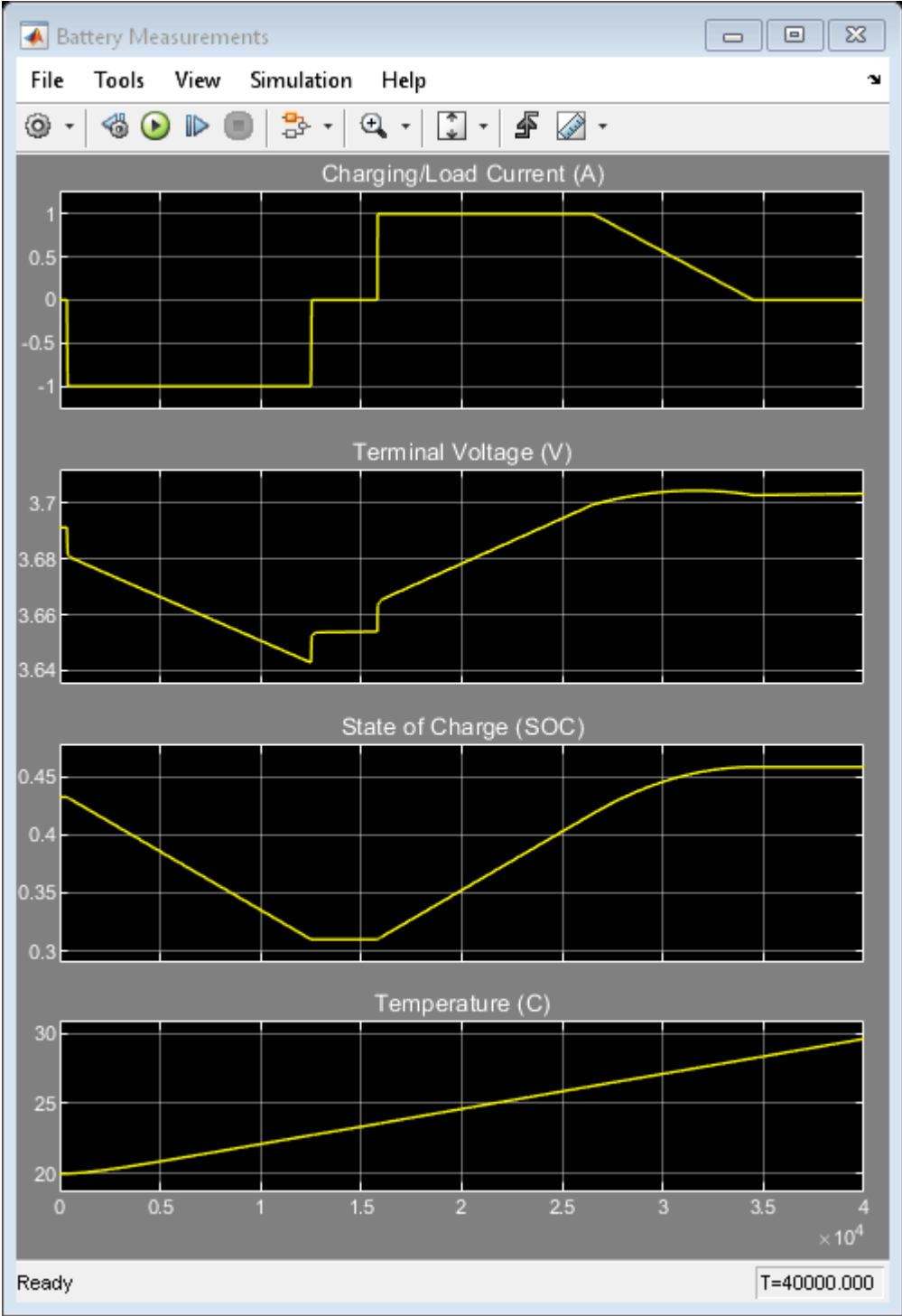
**Simulation Results from Scopes**

**Simulation Results from Simscape Logging**
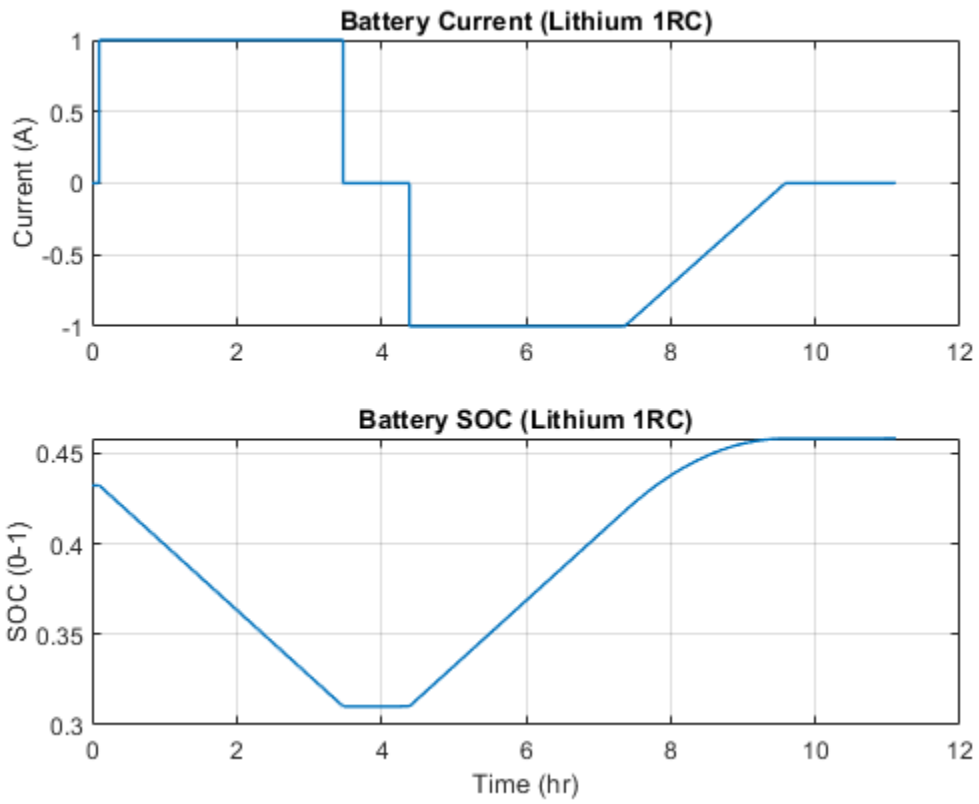
# Lithium Battery Cell - Two RC-Branch Equivalent Circuit

This example shows how to model a lithium cell using the Simscape™ language to implement the elements of an equivalent circuit model with two RC branches. For the defining equations and their validation, see T. Huria, M. Ceraolo, J. Gazzarri, R. Jackey. "High Fidelity Electrical Model with Thermal Dependence for Characterization and Simulation of High Power Lithium Battery Cells," IEEE International Electric Vehicle Conference, March 2012.

A simple thermal model is used to model battery temperature. It is assumed that cooling is primarily via convection, and that heating is primarily from internal resistance. A battery pack can be modeled by connecting multiple copies of the battery cell block in series.

**Model**



### Lithium Battery Cell - Two RC-Branch Equivalent Circuit

1. Plot current and SOC for battery (see code)
2. Explore simulation results using sscexplore
3. Open custom library used by Lithium Cell 2RC block
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Lithium Cell 2RC Subsystem**
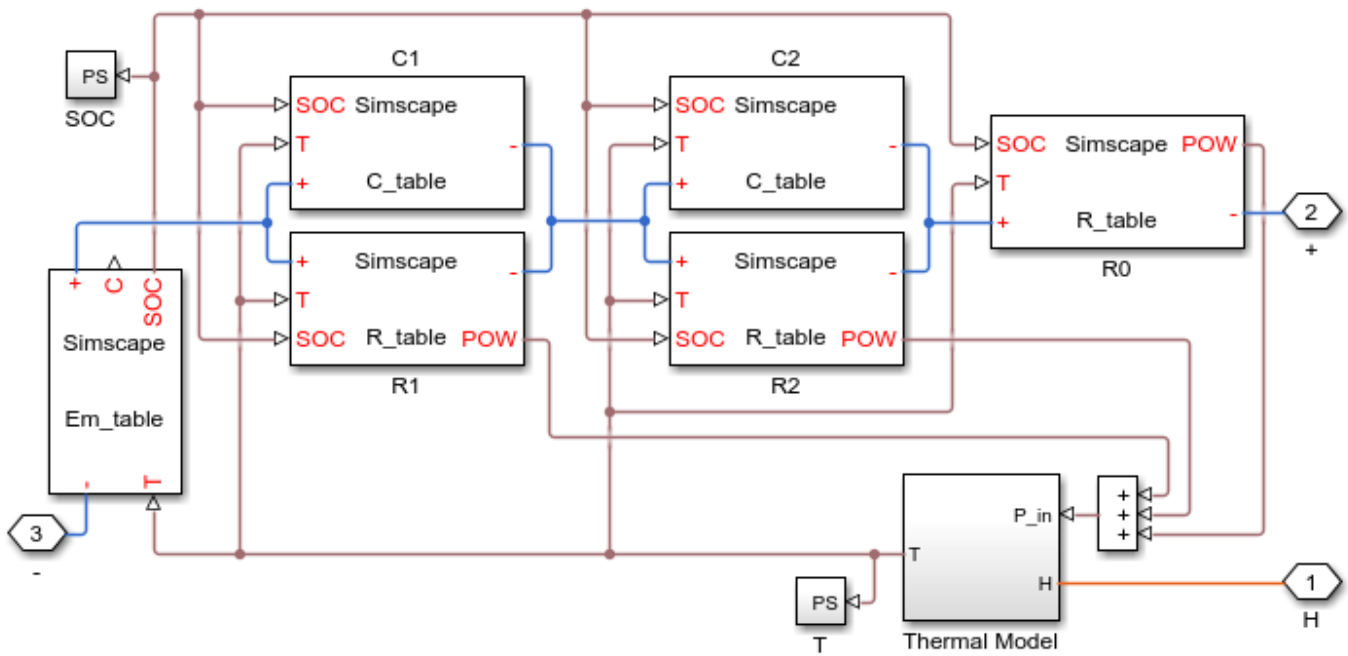
**Simulation Results from Scopes**
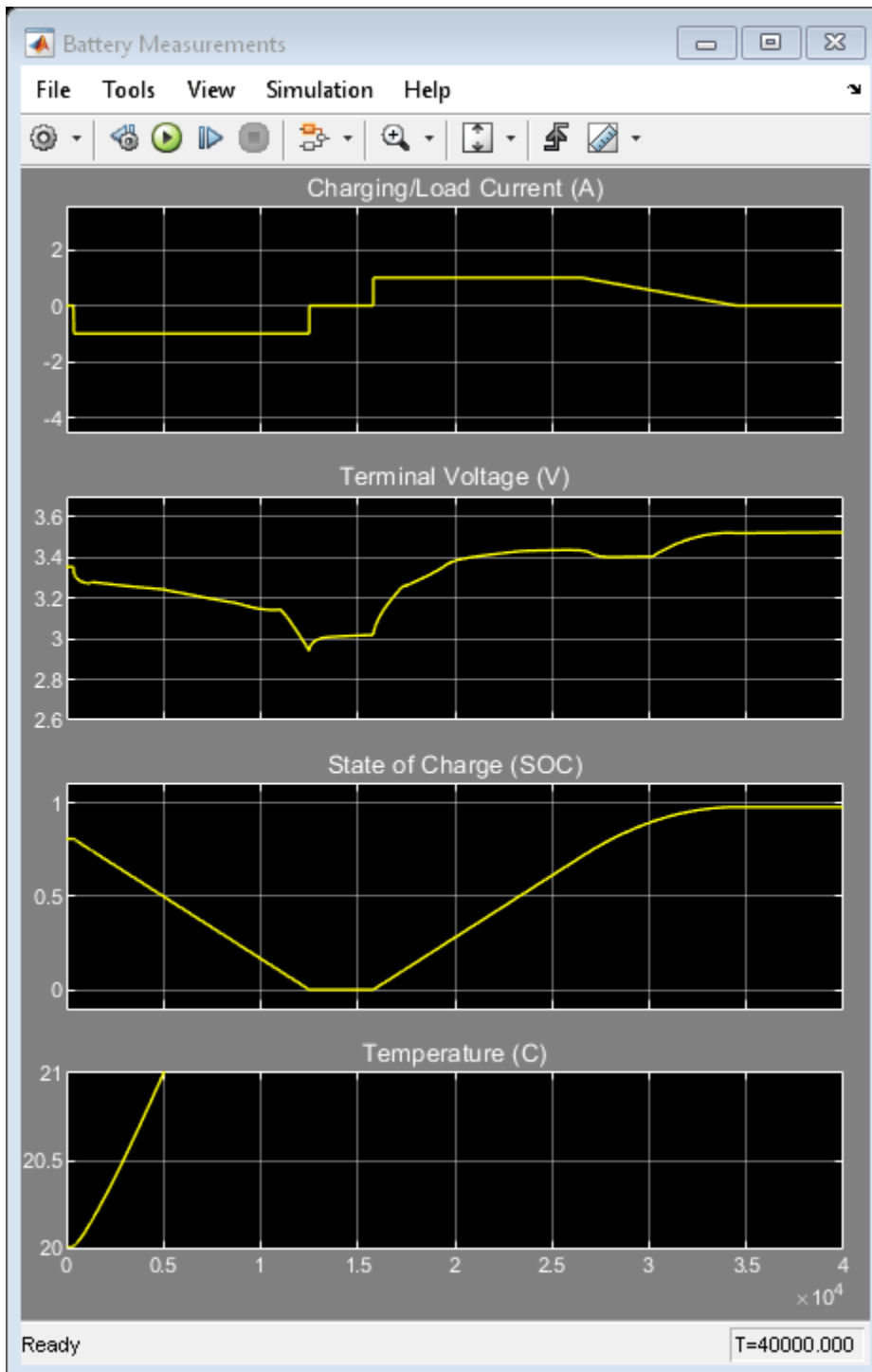
**Simulation Results from Simscape Logging**
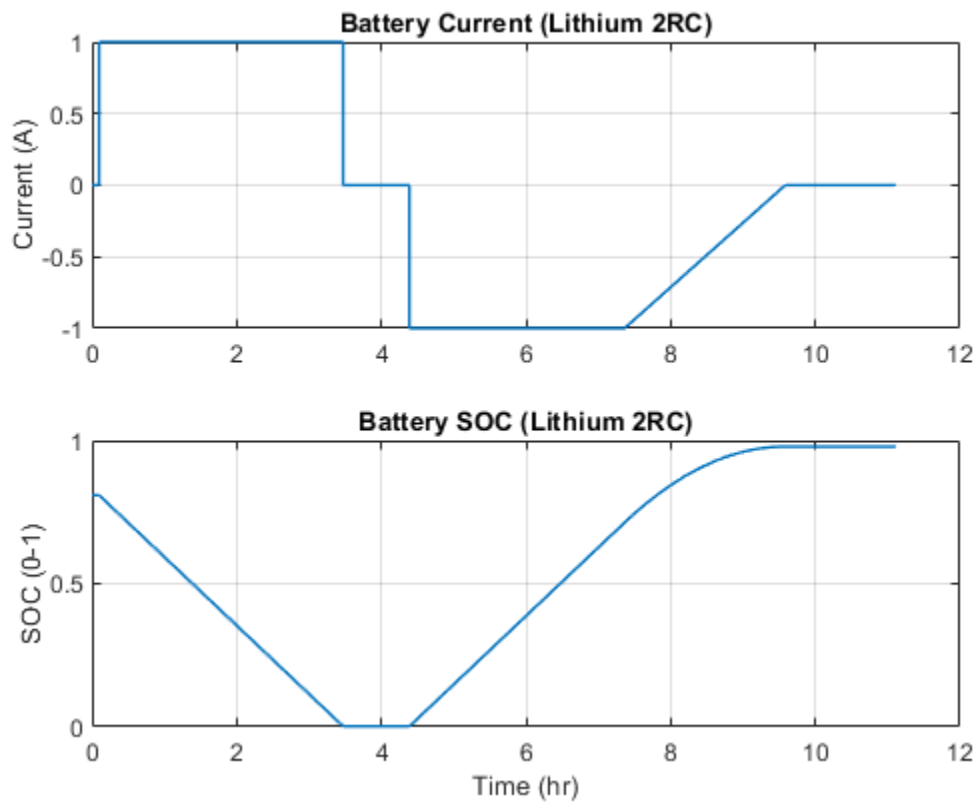
# Lithium Pack Thermal Runaway

This example shows how to model a thermal runaway in a lithium-ion battery pack. The model measures the cell heat generation, the cell-to- cell heat cascade, and the subsequent temperature rise in the cells, based on the design. The cell thermal runaway abuse heat is calculated using calorimeter data. Simulation is run to evaluate the number of cells that go into runaway mode, when just one cell is abused. To delay or cancel the cell-to-cell thermal cascading, this example models a thermal barrier between the cells.
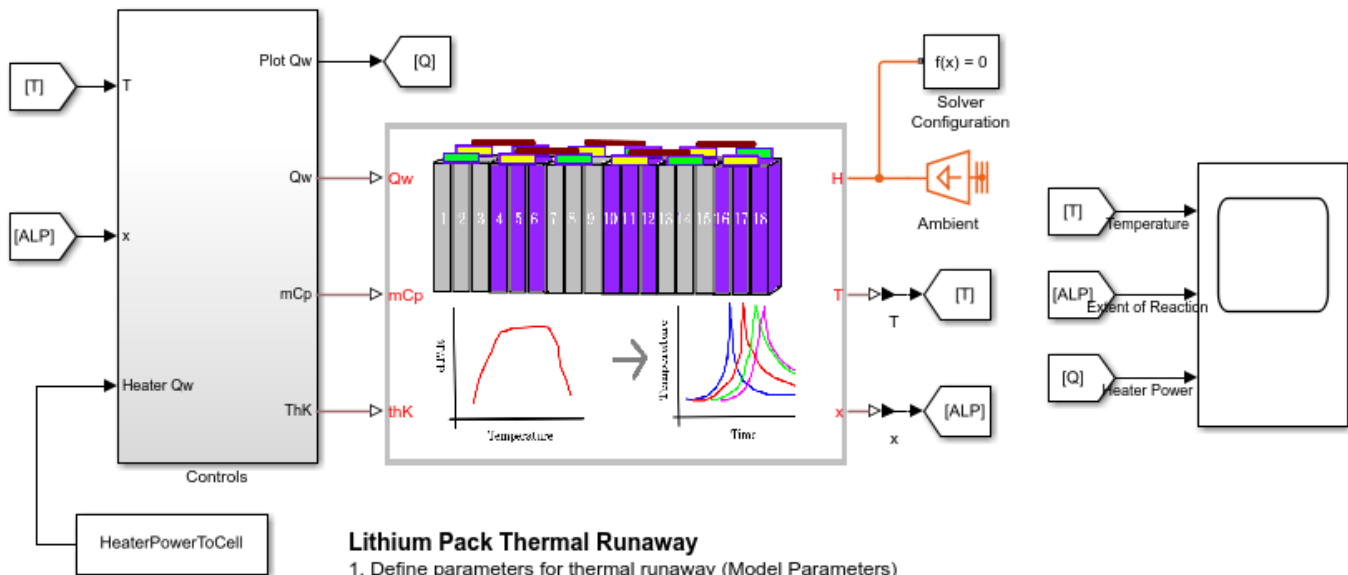


**Lithium Pack Thermal Runaway**
1. Define parameters for thermal runaway (Model Parameters)
2. Plot results for cell temperature and extent of reaction (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2020-2021 The MathWorks, Inc.

### Model Overview

The example models a battery pack that consists of eight pouch-shaped lithium-ion cells. The cells are in contact with each other. An external heater abuses the first cell. The heater heats the first cell enough to start the thermal runaway reaction. During the abuse period, the model uses the calorimeter data to estimate the cell self-heat generation and simulates the time needed by the other cells to go into their own respective runaway reactions. To stop the thermal cascading, this example models a thermal barrier between cell 4 and cell 5. Then, it calculates the thickness of this barrier to prevent the fifth cell to go into thermal runaway. These are the parameters in the battery pack:

- **Temperature (T) vector over which dT/dt is tabulated, T** — Temperature values at which the derivative of the temperature with time is defined, specified as an array of scalars. This data is typically obtained from a calorimeter test on a single lithium-ion cell.

- **Rate of temperature change (dT/dt) vector, dT/dt** — Derivative of the temperature with time, specified as an array of scalars of the same size of the **Temperature (T) vector over which dT/dt is tabulated, T** parameter. This data is typically obtained from a calorimeter test on a single lithium-ion cell.

- **Heat of abuse reaction** — Heat of the chemical reaction modeled using calorimetry data, specified as a scalar.

- **Active reactant mass as a fraction of cell mass** — Mass of the reactant or the active material used in the thermal abuse reaction, specified as a fraction greater than 0. The value of this fraction is equal to the mass of the reactant divided by the total mass of the cell.

- **Number of cells in stack** — Number of cells in the battery pack, specified as an integer greater than one.

- **Cell height** — Cell height, specified as a positive scalar.

- **Cell width** — Cell width, specified as a positive scalar.

- **Cell thickness** — Cell thickness, specified as a positive scalar.

- **Cell density** — Cell density, specified as a positive scalar.

- **Cell specific heat** — Cell specific heat, specified as a positive scalar.

- **Heat transfer coefficient to ambient** — Cell heat transfer coefficient, specified as a positive scalar.

- **Cell thermal conductivity** — Cell through-plane thermal conductivity value, specified as a positive scalar.

- **Cell initial temperature vector** — Cell initial temperature, specified as a vector. The number of elements in this vector must be equal to the value of the **Number of cells in stack** parameter.

- **Cell-to-cell gap length vector** — Distance between the individual cells, specified as a vector. The number of elements in this vector must be equal to the value of the **Number of cells in stack** parameter - 1.

- **Cell-to-cell gap thermal mass vector** — Thermal mass of the material in the gap between each cell, specified as a vector. The number of elements in this vector must be equal to the value of the **Number of cells in stack** parameter - 1.

- **Cell-to-cell gap thermal conductivity vector** — Thermal conductivity of the material in the gap between each cell, specified as a vector. The number of elements in this vector must be equal to the value of the **Number of cells in stack** parameter - 1.

To define the external heat input, the thermal mass change, and the thermal conductivity of each cell, specify these inputs:

- **Qw** — External heat input to each cell, specified as a vector of scalars.

- **mCp** — Thermal mass change of each cell, specified as a fraction greater than 0. To obtain the actual thermal mass of the cell, this value is multiplied to the cell thermal mass. The **mCp** port value models the changes in the cell thermal mass as the cell reacts. In this example, the **mCp** port value does not change with time or cell reaction.

- **thK** — Thermal conductivity change of each cell, specified as a fraction greater than 0. To obtain the actual thermal conductivity of the cell, this value is multiplied to the **Cell thermal conductivity** parameter. The **thK** port value models the changes in the cell thermal conductivity due to the gases being vented out and the cell becoming hollow. In this example, the **thK** port value does not change with time or cell reaction.

To access cell temperature output and extent of reaction, use these two outputs:

- **T** — Temperature of all the cells in the battery pack, specified as a vector of scalar.

- **x** — Extent of reaction for all the cells in the battery pack, specified as a vector of scalar.

**Controls Overview**

The Controls subsystem manages the heater operations. The heater provides a constant power to the first cell in the module, equal to the value of the `HeaterPowerToCell` workspace variable in the `ssc_lithium_pack_thermalRunaway_ini.m` file. The Heater Power = f(ExtentReaction) and Heater Power = f(T) blocks in the Controls subsystem check for the heater power input based on the measurements of the cell temperature. If the cell temperature is greater than the temperature cut-off limit, specified by the `stopHeaterWhenTempAbove` workspace variable, the heater switches off.
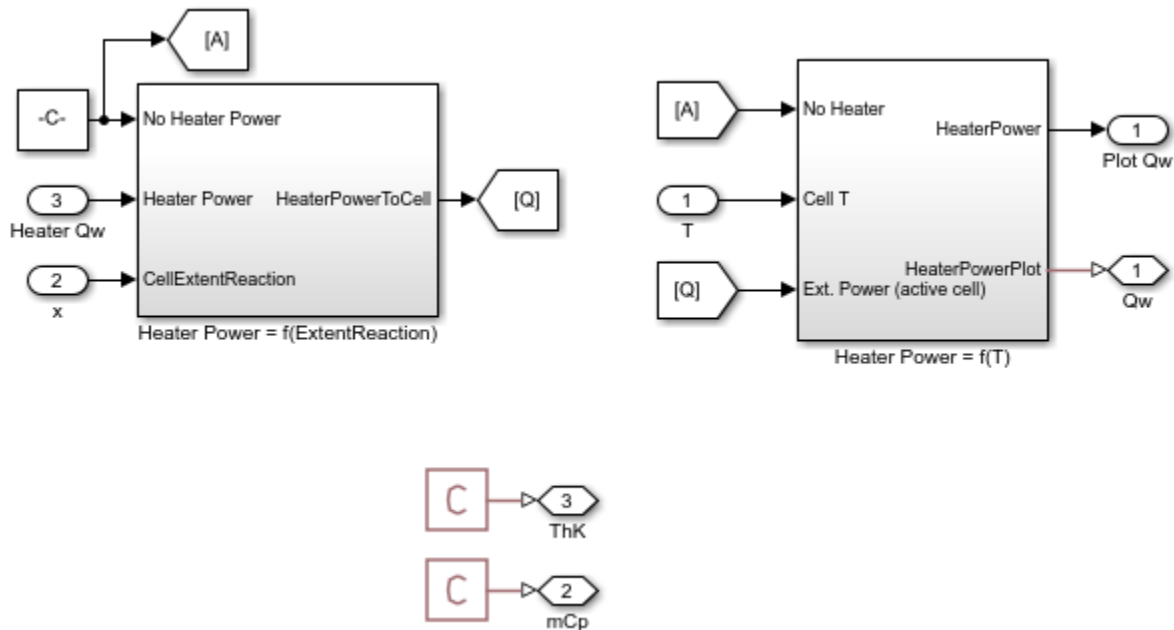


**Simulation Results**

The workspace variables in the `ssc_lithium_pack_thermalRunaway_ini.m` file set all the parameters and inputs. The initial temperature of all cells is 300 K and the `stopHeaterWhenTempAbove` workspace variable is equal to 443 K. The heater provides a constant power equal to 500 W to the first cell. When the cell temperature reaches the value specified in the `stopHeaterWhenTempAbove`, the heater switches off. Then, the cell- to-cell thermal cascading process begins. The first cell experiences a thermal runaway reaction, followed by all subsequent cells. The heat that you must remove from the battery pack is directly proportional to the number of cells that experience a runaway reaction. In this example, the battery pack can safely contain a total thermal energy equal to the thermal energies of four cells. To slow down or stop the cascading and prevent the damage of the cells, you must add thermal barriers between the cells. This example models a thermal barrier between the fourth and fifth cell. The **Cell-to-cell gap** parameter models the characteristics of the thermal barrier. You can edit this parameter by specifying these workspace variables in the `ssc_lithium_pack_thermalRunaway_ini.m` file:

- **cellToCellGapLen(1,4)** is equal to 0.005, or 5 millimeters.
- **cellToCellGapThermalMass(1,4)** is equal to 50 J/K.

**20-61**

- **cellToCellGapThermalK(1,4)** is equal to 0.05 W/m*K.

With these specifications, the thermal runaway stops at the fourth cell. The fifth, sixth, seventh, and eighth cells do not experience a thermal runaway. The results show how a 5 millimeters thermal barrier is enough to manage the spread of heat due to thermal cascading and runaway reactions.

**Simulation Results Without Thermal Barrier**

These plots show the cell temperature rise and thermal runaway in all the cells of the pack when there is no thermal barrier between the fourth and fifth cells.
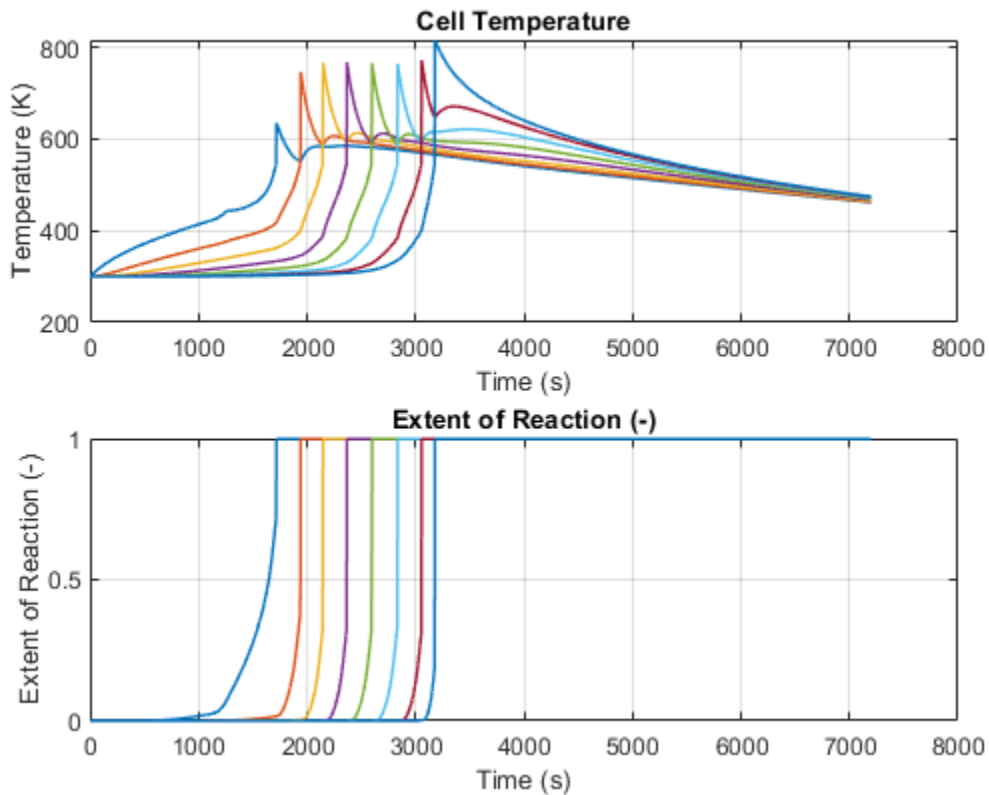
# Nonlinear Bipolar Transistor

This model shows an implementation of a nonlinear bipolar transistor based on the Ebers-Moll equivalent circuit. R1 and R2 set the nominal operating point, and the small signal gain is approximately set by the ratio R3/R4. The 1uF decoupling capacitors have been chosen to present negligible impedance at 1KHz. The model is configured for linearization so that a frequency response can be generated.

The model shows how more complex elements (in this case a transistor) can be built up from the fundamental electrical elements in the Foundation library. Note that the Start simulation from steady state option has been set in the Solver Configuration block.

For more background on the use of piecewise linear diodes in transistor modeling, see: Cel, J. "Ebers-Moll model of bipolar transistor with idealized diodes", International Journal of Electronics, Vol. 89, No. 1, January 2002, p.7-18.

**Model**
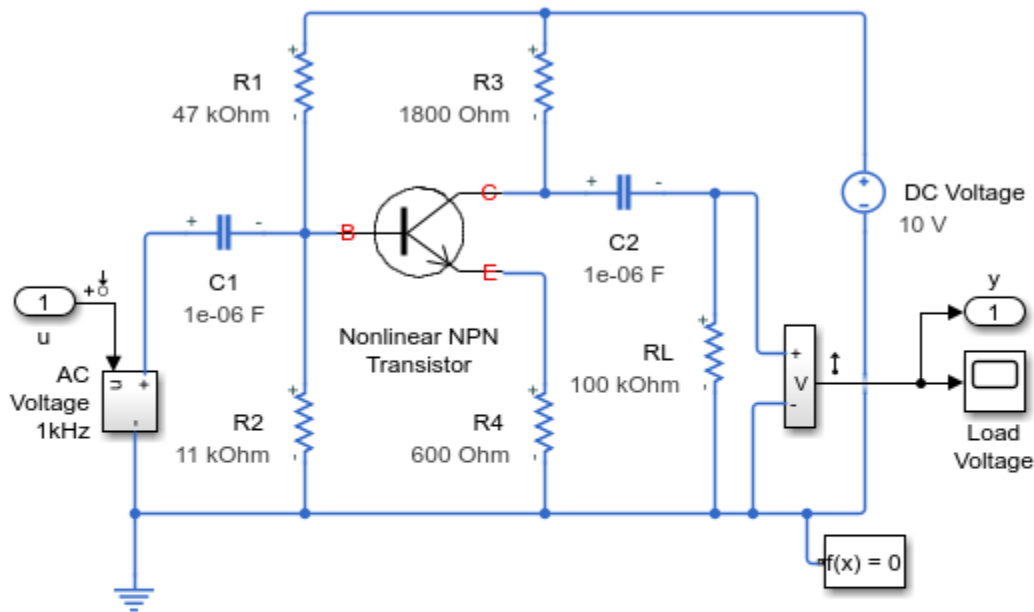


**Nonlinear Bipolar Transistor**

1. Plot voltages at transistor terminals (see code)
2. Linearize circuit to view frequency response (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Nonlinear NPN Transistor Subsystem**
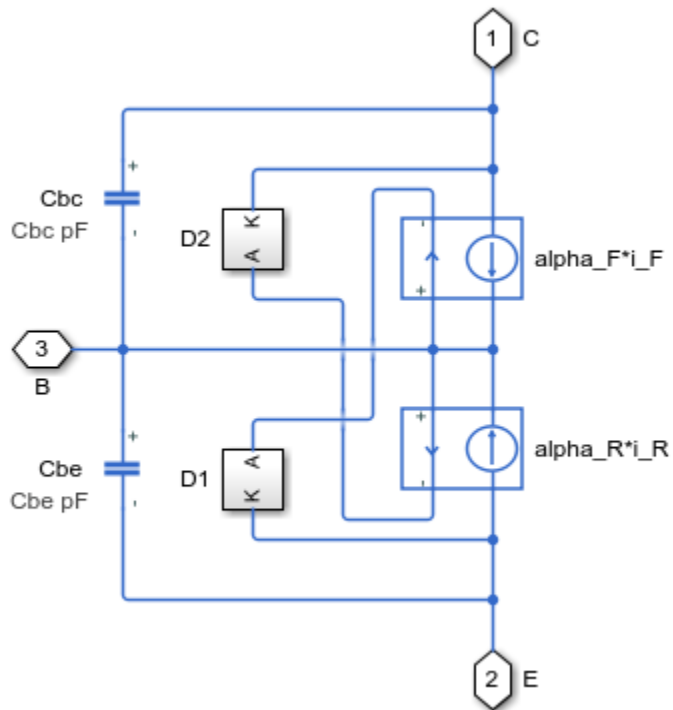
**Simulation Results from Simscape Logging**
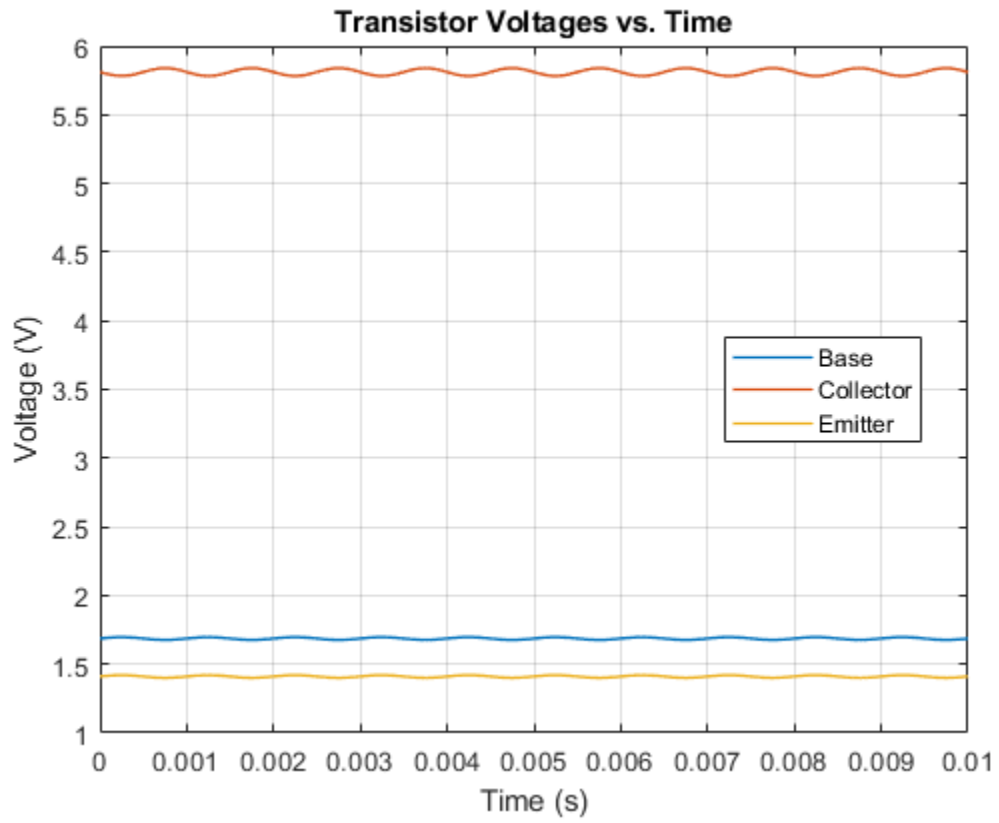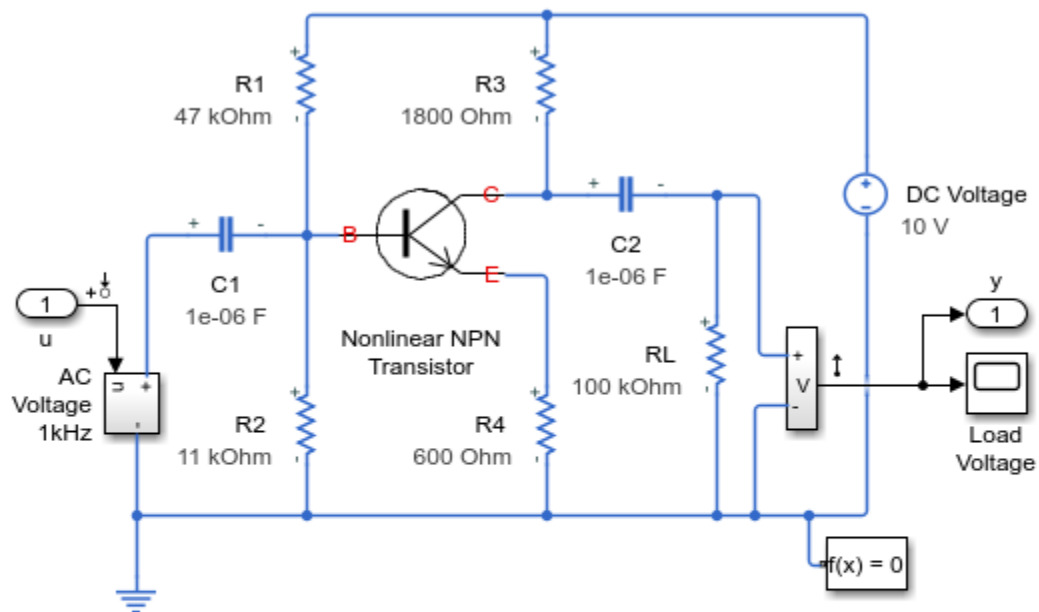
**Frequency Response**



## Nonlinear Bipolar Transistor

1. Plot voltages at transistor terminals (see code)
2. Linearize circuit to view frequency response (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.
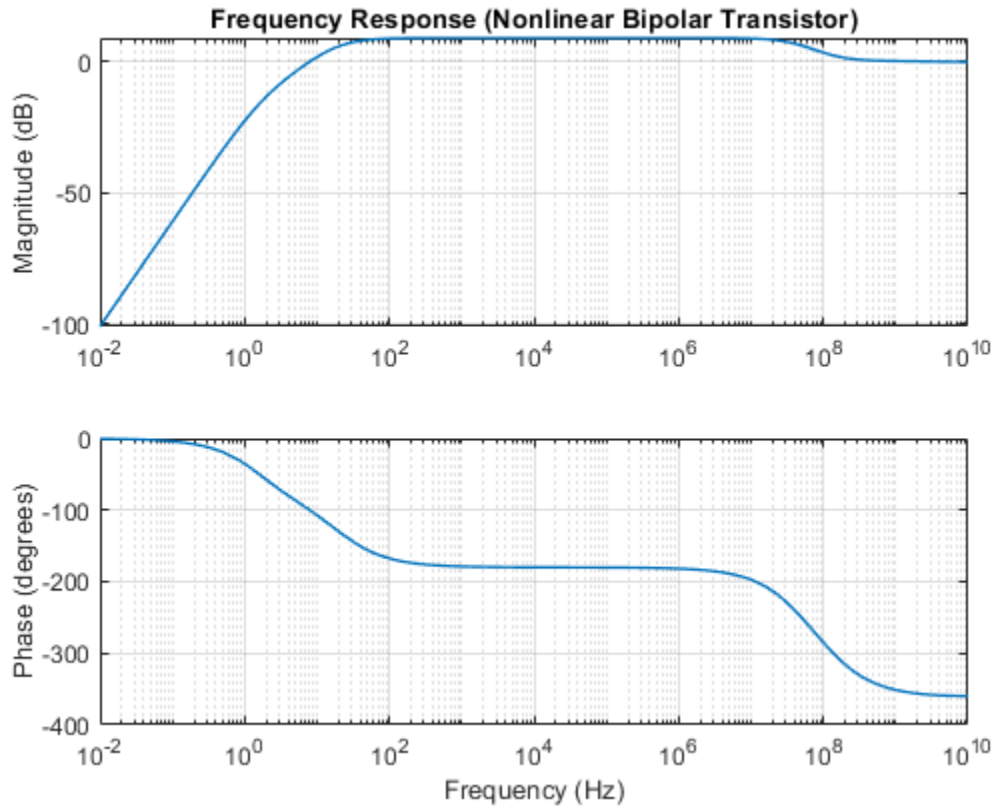
# Small-Signal Bipolar Transistor

This model shows the use of a small-signal equivalent transistor model to assess performance of a common-emitter amplifier. The 47K resistor is the bias resistor required to set nominal operating point, and the 470 Ohm resistor is the load resistor. The transistor is represented by a hybrid-parameter equivalent circuit with circuit parameters h_ie (base circuit resistance), h_oe (output admittance), h_fe (forward current gain), and h_re (reverse voltage transfer ratio). Parameters set are typical for a BC107 Group B transistor. The gain is approximately given by -h_fe*470/h_ie =-47. The 1uF decoupling capacitor has been chosen to present negligible impedance at 1KHz compared to the input resistance h_ie, so the output voltage should be 47*10mV = 0.47V peak.

The model also shows how more complex elements (in this case a transistor) can be built up from the fundamental electrical elements in the Foundation library.

**Model**
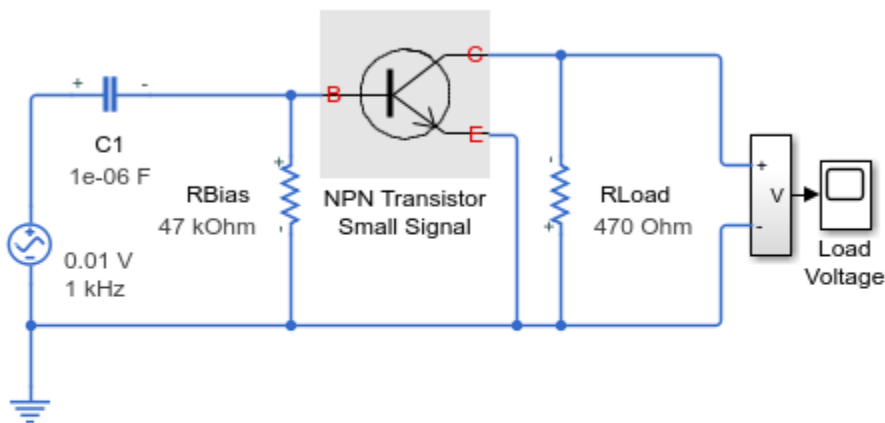


### Small-Signal Bipolar Transistor

1. Plot voltages in cylinder (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

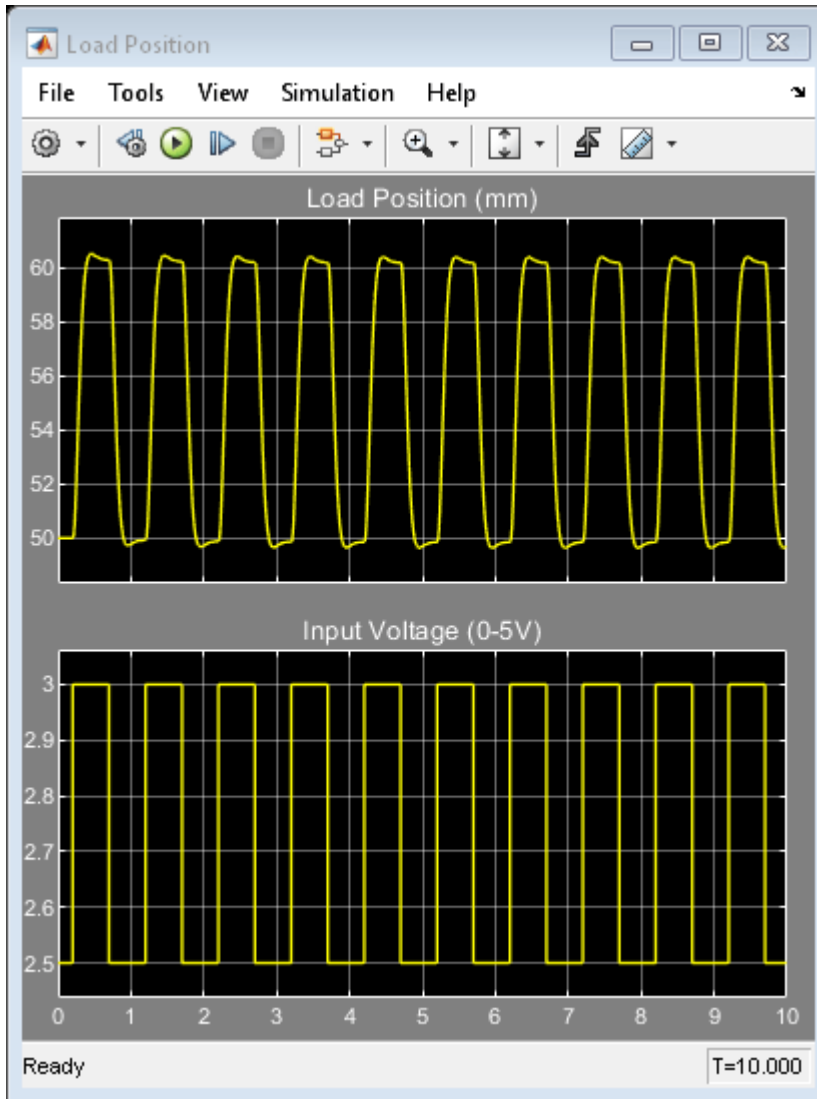Copyright 2015-2021 The MathWorks, Inc.

**NPN Transistor Small Signal Subsystem**
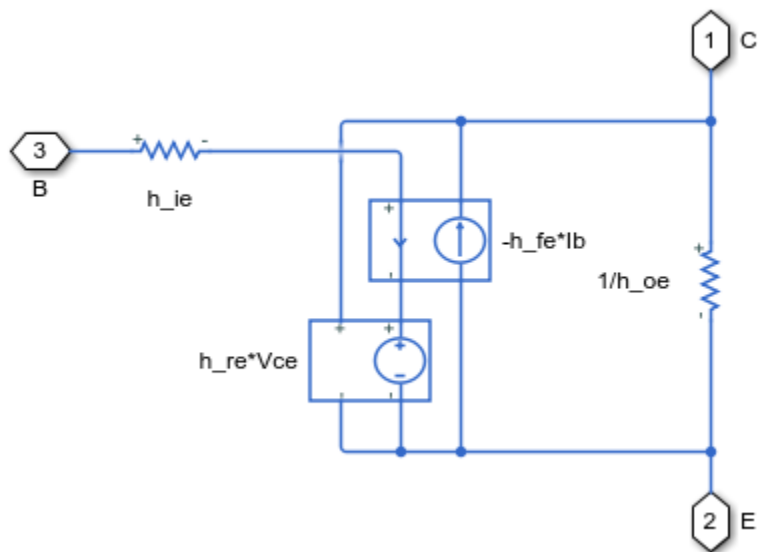


**Simulation Results from Simscape Logging**
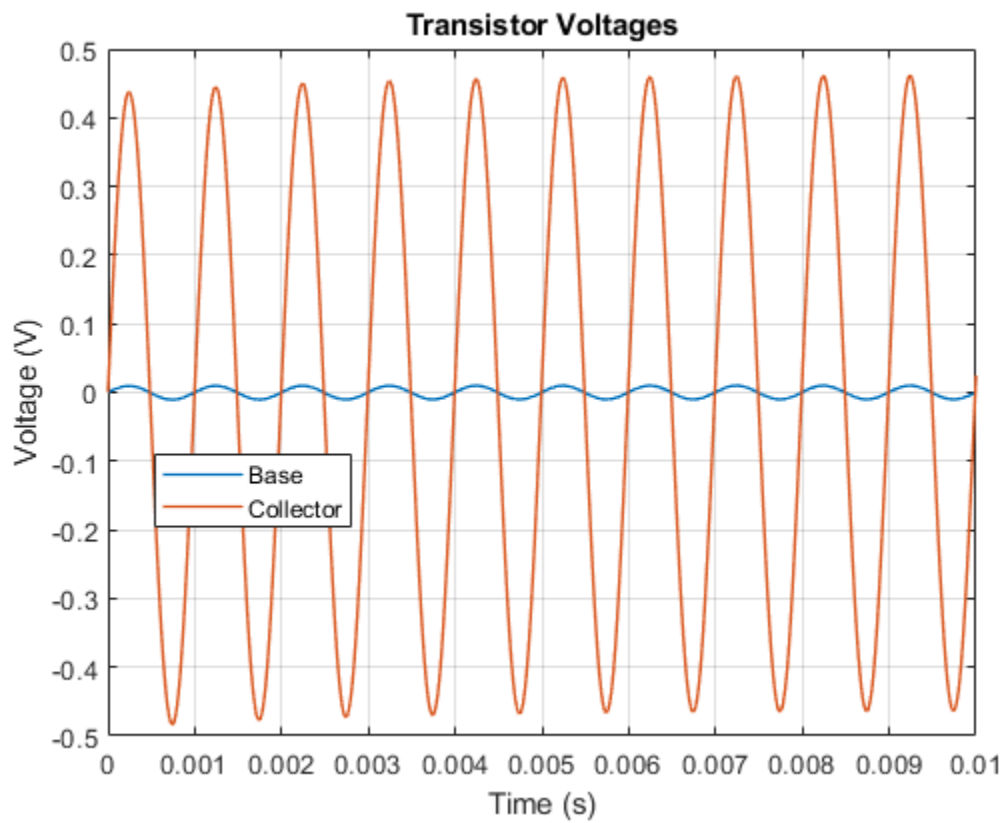
# Band-Limited Op-Amp

This example shows how higher fidelity or more detailed component models can be built from the Foundation library blocks. The model implements a band-limited op-amp. It includes a first-order dynamic from inputs to outputs, and gives much faster simulation than if using a device-level equivalent circuit, which would normally include multiple transistors. This model also includes the effects of input and output impedance (Rin and Rout in the circuit), but does not include nonlinear effects such as slew-rate limiting.

The op-amp is connected open-loop in this circuit, that is, in the form of a differential amplifier. This is to show the effect of the first-order dynamics. Including any feedback will reduce the impact of the dynamics on the circuit input-output response.

**Model**



## Band-Limited Op-Amp

1. Linearize circuit to view frequency response (see code)
2. Explore simulation results using sscexplore
3. Open op-amp examples:
   noninverting, inverting, differentiator, finite gain
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Band-Limited Op-Amp Subsystem**
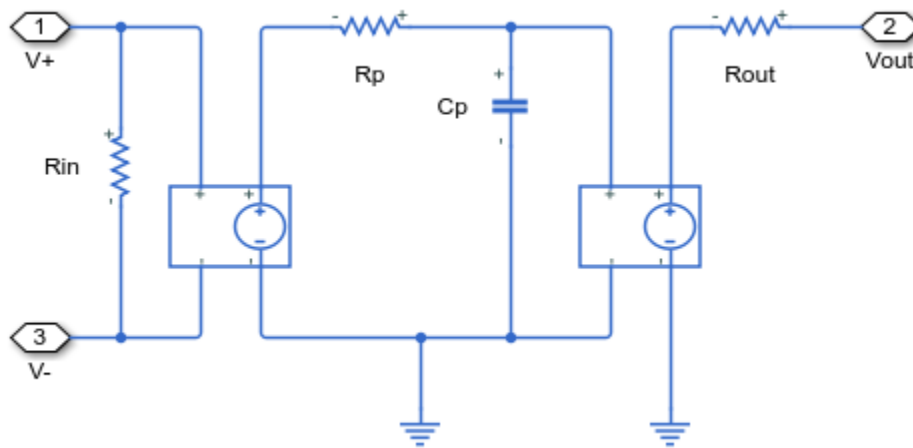


**Frequency Response**
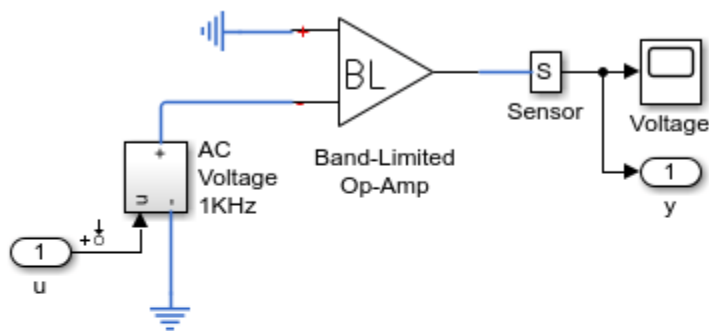


## Band-Limited Op-Amp

1. Linearize circuit to view frequency response (see code)
2. Explore simulation results using sscexplore
3. Open op-amp examples:
   noninverting, inverting, differentiator, finite gain
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.
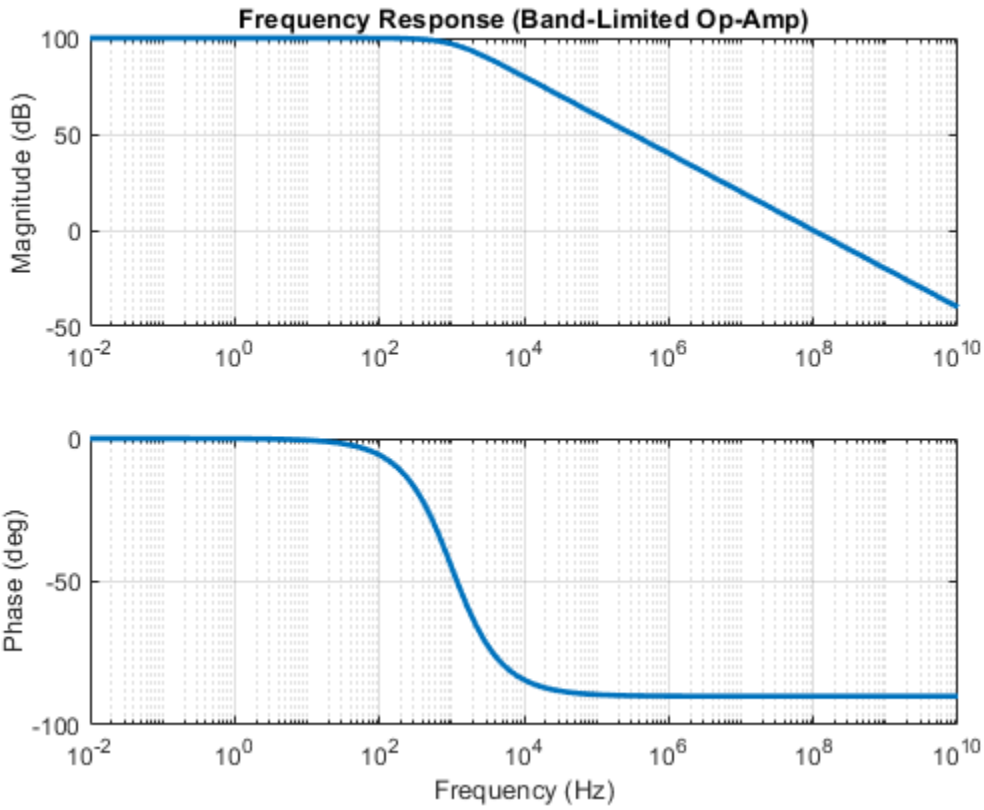
# Finite-Gain Op-Amp

This example shows how higher fidelity or more detailed component models can be built from the Foundation library blocks. The Op-Amp block in the Foundation library models the ideal case whereby the gain is infinite, input impedance infinite, and output impedance zero. The Finite Gain Op-Amp block in this example has an open-loop gain of 1e5, input resistance of 100K ohms and output resistance of 10 ohms. As a result, the gain for this amplifier circuit is slightly lower than the gain that can be analytically calculated if the op-amp gain is assumed to be infinite.

**Model**
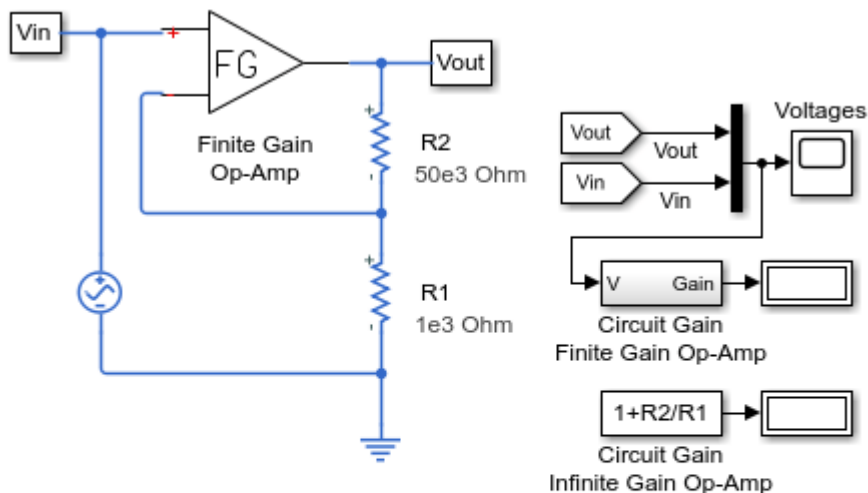


**Finite-Gain Op-Amp**

1. Plot voltages for op-amp circuit (see code)
2. Explore simulation results using sscexplore
3. Open op-amp examples:
   noninverting, inverting, differentiator, band-limited
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Finite Gain Op-Amp Subsystem**
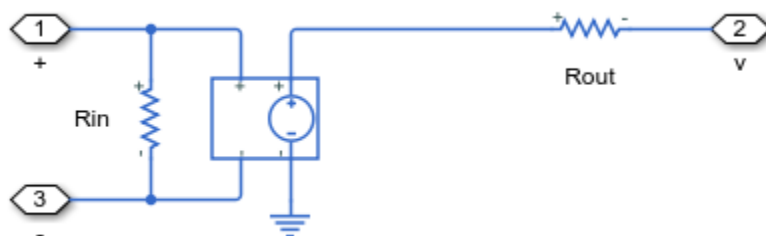


**Simulation Results from Simscape Logging**

Plot "Finite Gain Op-Amp Circuit Voltages" shows the input and output voltages for the circuit. If the circuit used an infinite gain op-amp with no input and output resistances defined, the gain would be

1+R2/R1 = 51. Since this model uses an op-amp with finite gain plus input and output resistances, the circuit gain is slightly less.

**Finite Gain Op-Amp Circuit Voltages**

Gain of Circuit with:
Finite Gain Op-Amp   = 50.97
Infinite Gain Op-Amp = 51
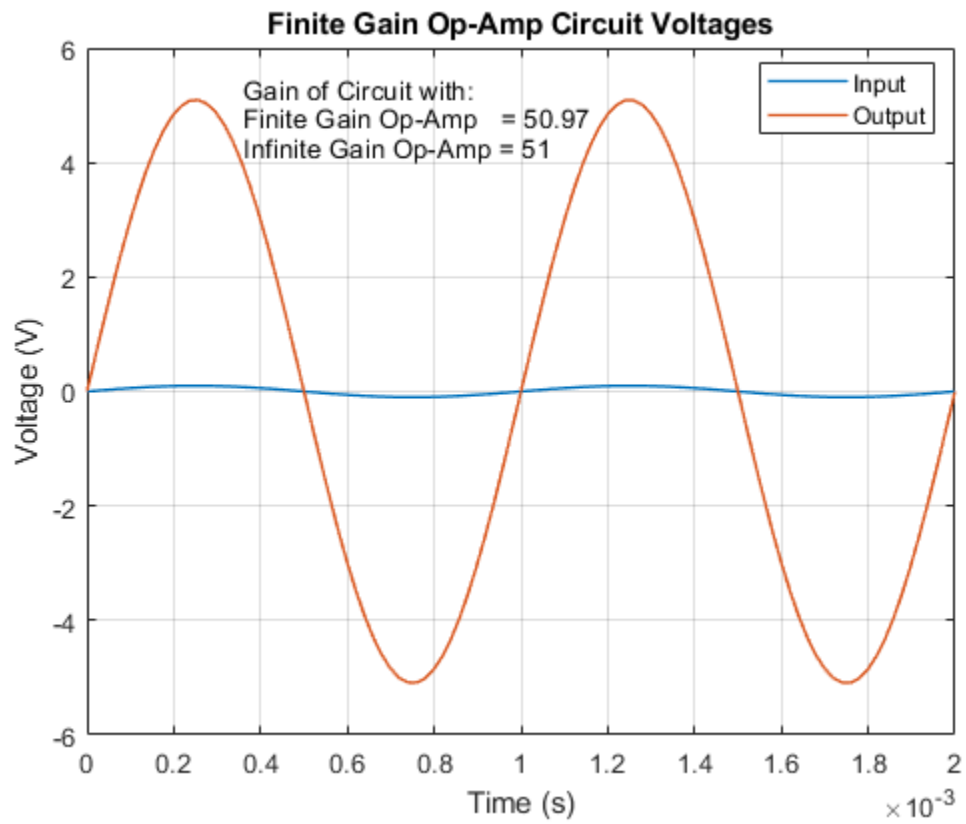
# Op-Amp Circuit - Differentiator

This model shows a differentiator, such as might be used as part of a PID controller. It also illustrates how numerical simulation issues can arise in some idealized circuits. The model runs with the capacitor series parasitic resistance set to its default value of 1e-6 Ohms. Setting it to zero results in a warning and a very slow simulation. See the User's Guide for further information.

**Model**
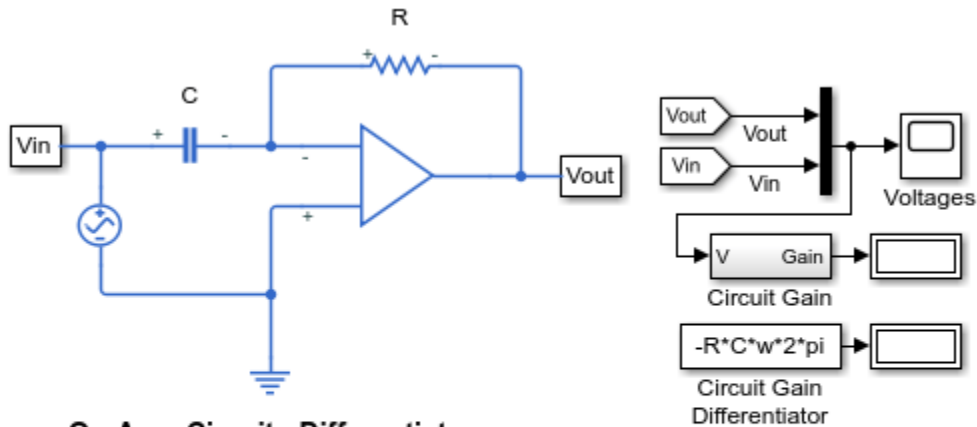


**Op-Amp Circuit - Differentiator**

1. Plot voltages for op-amp (see code)
2. Explore simulation results using sscexplore
3. Open op-amp examples:
   noninverting, inverting, finite gain, band-limited
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
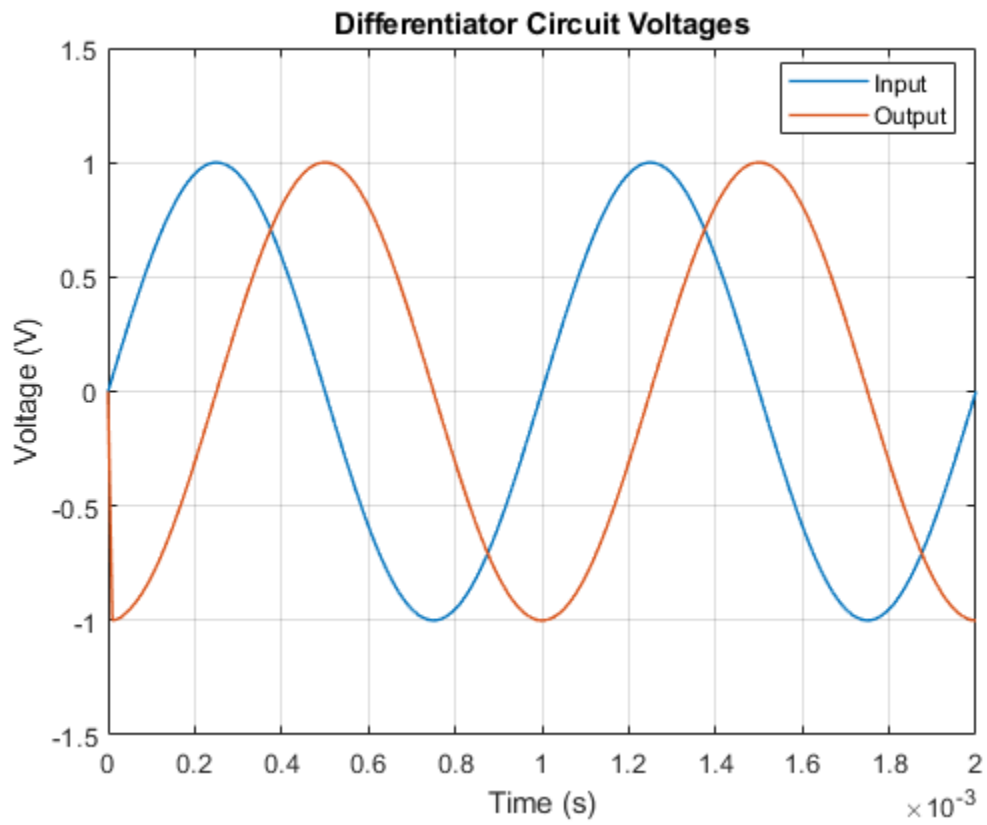
# Op-Amp Circuit - Inverting Amplifier

This model shows a standard inverting op-amp circuit. The gain is given by -R2/R1, and with the values set to R1=1K Ohm and R2=10K Ohm, the 0.1V peak-to-peak input voltage is amplified to 1V peak-to-peak. As the Op-Amp block implements an ideal (i.e. infinite gain) device, this gain is achieved regardless of output load.

**Model**
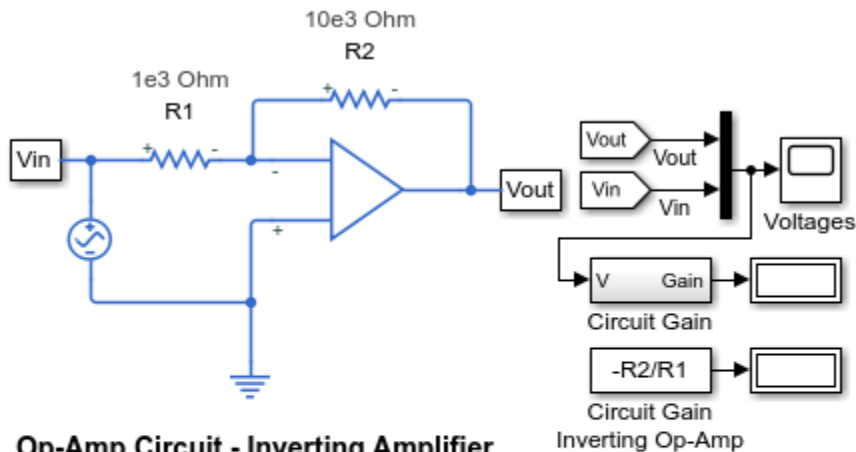


**Op-Amp Circuit - Inverting Amplifier**

1. Plot voltages for op-amp circuit (see code)
2. Explore simulation results using sscexplore
3. Open op-amp examples:
   noninverting, differentiator, finite gain, band-limited
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
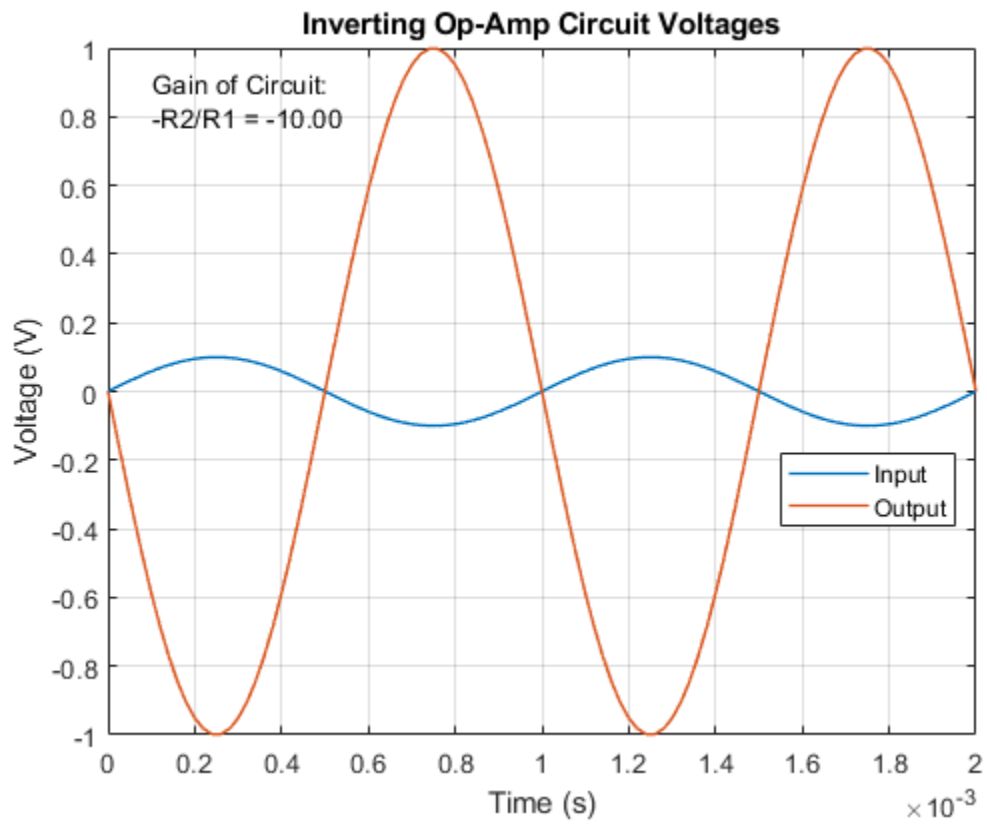
# Op-Amp Circuit - Noninverting Amplifier

This model shows a noninverting op-amp circuit. The gain is given by 1+R2/R1, and with the values set to R1=1K Ohm and R2=10K Ohm, the 0.1V peak-to-peak input voltage is amplified to 1.1V peak-to-peak. As the Op-Amp block implements an ideal (i.e. infinite gain) device, this gain is achieved regardless of output load.

**Model**
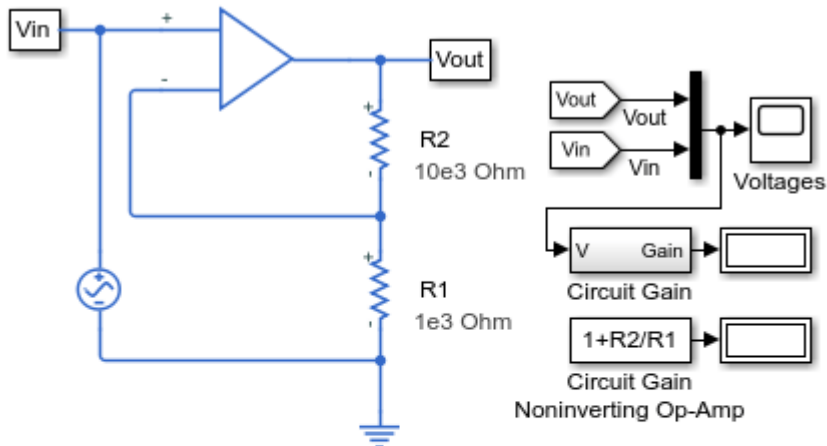


### Op-Amp Circuit - Noninverting Amplifier

1. Plot voltages for op-amp circuit (see code)
2. Explore simulation results using sscexplore
3. Open op-amp examples:
   inverting, differentiator, finite gain, band-limited
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
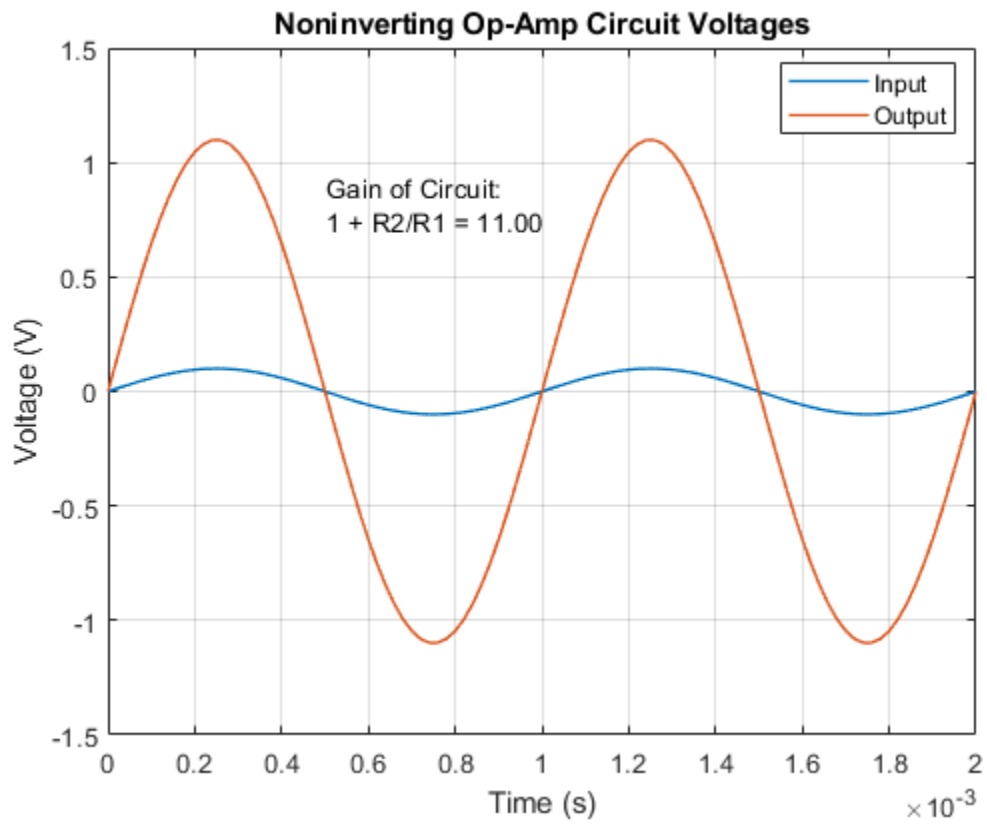
# Nonlinear Inductor

This example shows an implementation of a nonlinear inductor where the inductance depends on the current. For best numerical efficiency, the underlying behavior is defined in terms of a current-dependent flux. In order to differentiate the flux to get voltage, a magnetizing lag is included. Simulation results are relatively insensitive to this lag provided that it is at least an order of magnitude faster than the fastest frequency of interest. Parameters for the source and lag are defined in the MATLAB® workspace so that the expected voltage can be calculated analytically.

Here the nonlinear flux-current relationship is defined by a tanh curve. This exhibits the typical curve shape whereby the flux saturates for large currents due to saturation of (for example) an iron core. To model a specific flux-current relationship, replace the tanh function with a PS Lookup Table (1D) from the Simscape™ Foundation Library.

**Model**
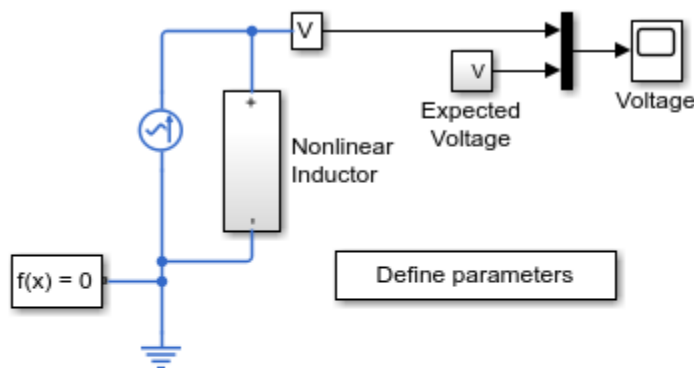


## Nonlinear Inductor

1. Plot current and flux for inductor (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Nonlinear Inductor Subsystem**
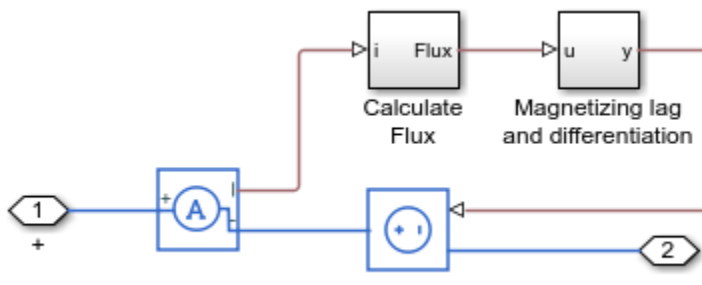
**Simulation Results from Simscape Logging**
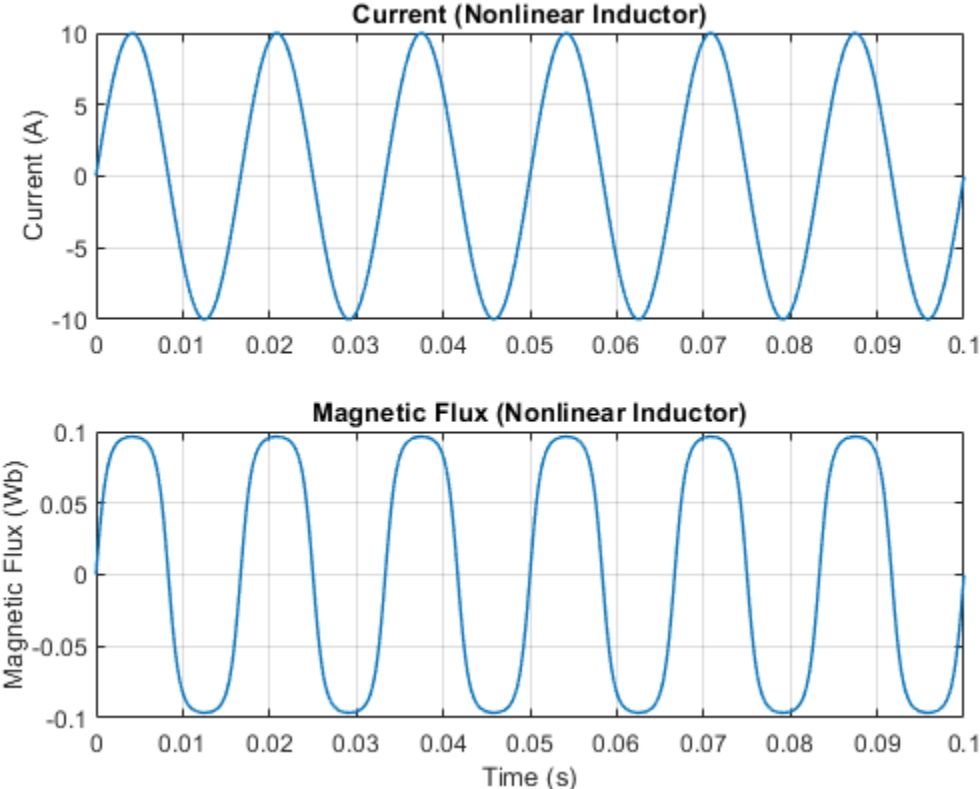
# Full-Wave Bridge Rectifier

This example shows an ideal AC transformer plus full-wave bridge rectifier. It converts 120 volts AC to 12 volts DC. The transformer has a turns ratio of 14, stepping the supply down to 8.6 volts rms, i.e. 8.6*sqrt(2) = 12 volts pk-pk. The full-wave bridge rectifier plus capacitor combination then converts this to DC. The resistor represents a typical load.

The model can be used to size the capacitor required for a specified load. For a given size of capacitor, as the load resistance is increased, the ripple in the DC voltage increases. The model can also be used to drive an application circuit in order to assess the effect of the ripple.

**Model**
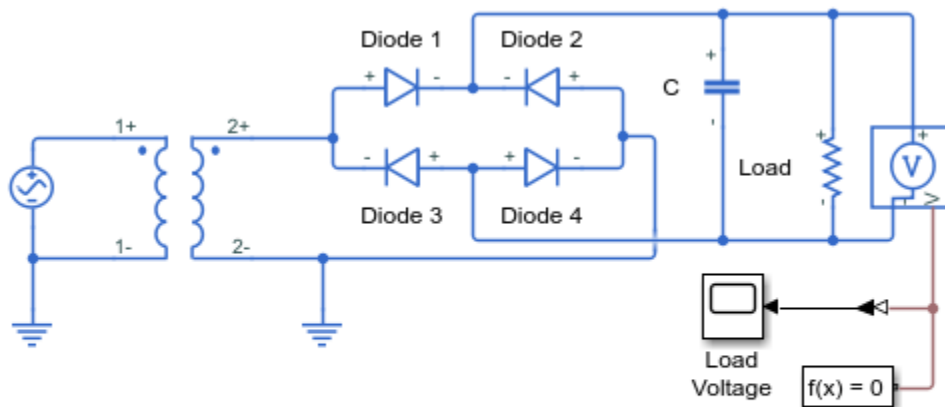


**Full-Wave Bridge Rectifier**

1. Plot voltages and currents of rectifier (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

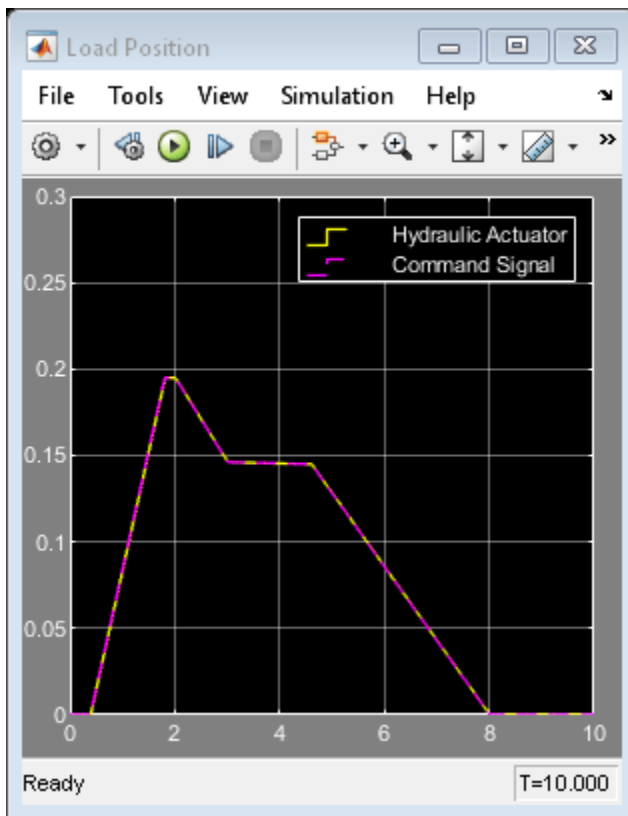**Simulation Results from Scopes**
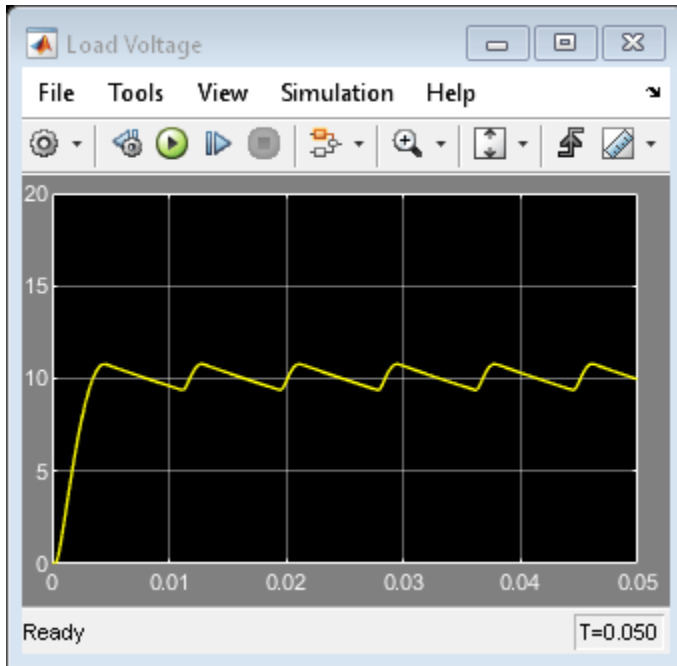


**Simulation Results from Simscape Logging**

Plot "Bridge Rectifier Voltages and Currents" shows how AC voltage is converted to DC Voltage. The dark blue line is the AC voltage on the source side of the bridge. There are two paths for current flow through the diode bridge. The alternating peaks through diodes 1&4 and diodes 2&3 show that current flow reaching the capacitor is flowing in the same direction even though the polarity of the voltage is changing. The ripple in the load voltage corresponds to the charging and discharging of the capacitor.
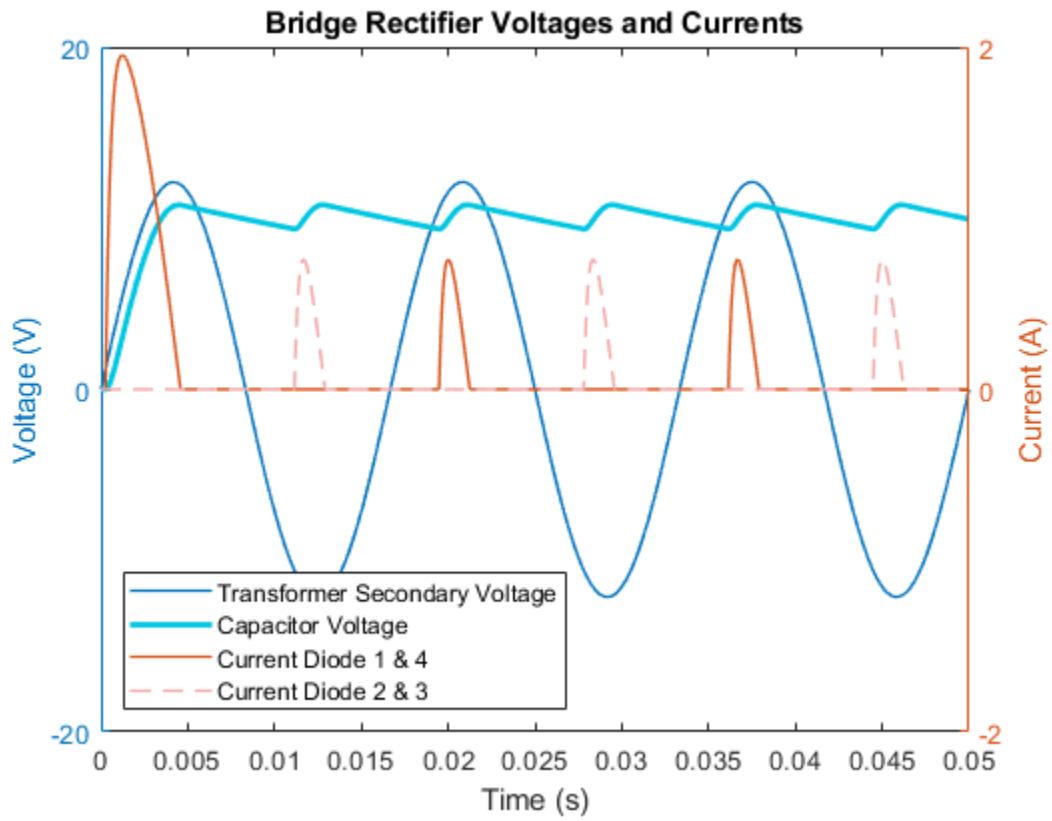
# Circuit Breaker

This example shows how to model a circuit breaker. The electromechanical breaker mechanism is approximated with a first-order time constant, and it is assumed that the mechanical force is proportional to load current. This simple representation is suitable for use in a larger model of a complete system. When the 20V supply is applied at one second, it results in a current that exceeds the circuit breaker current rating, and hence the breaker trips. The reset is then pressed at three seconds, and the voltage is ramped up. The breaker then trips just beyond the circuit breaker current rating.

**Model**
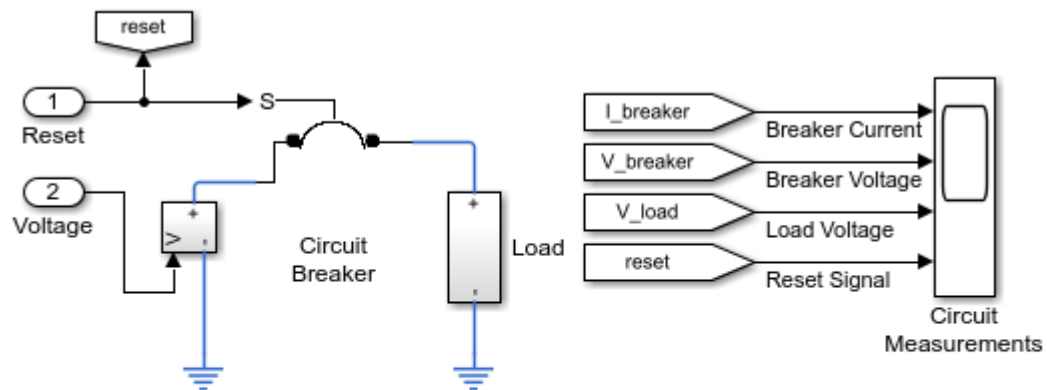


## Circuit Breaker

1. Plot current and voltage in circuit breaker (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
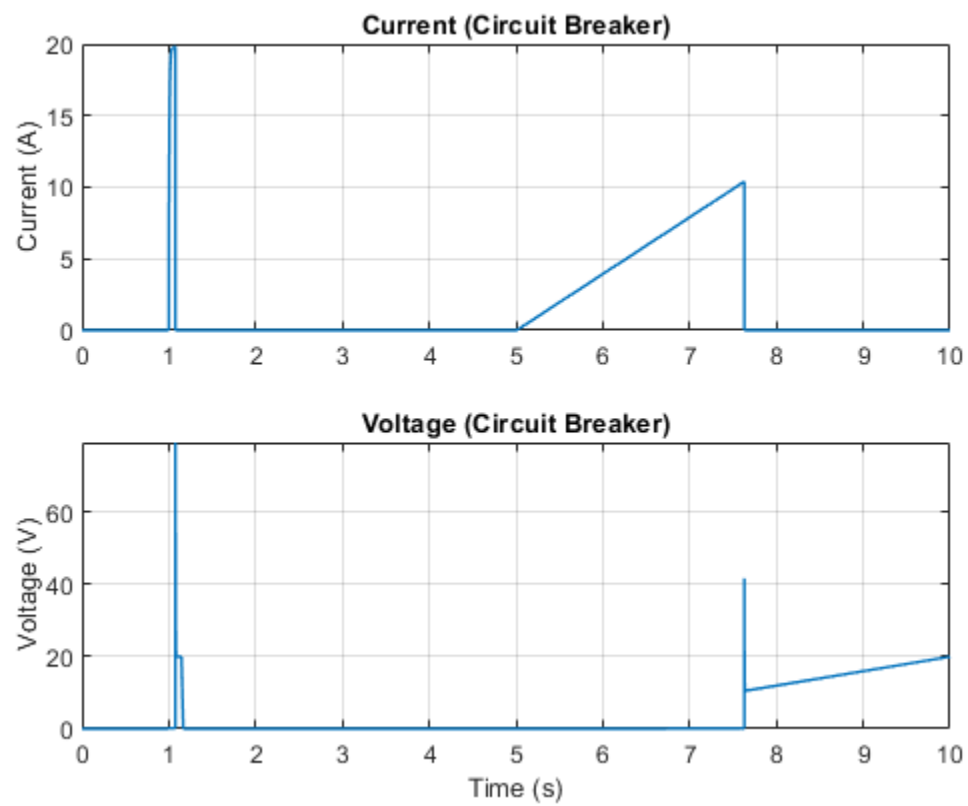
# Circuit Breaker with Probe Block

This example models a circuit breaker using a Simscape Probe block to access the current and voltage within the electrical switch block and another Simscape Probe block to access the voltage across a load.

**Model**
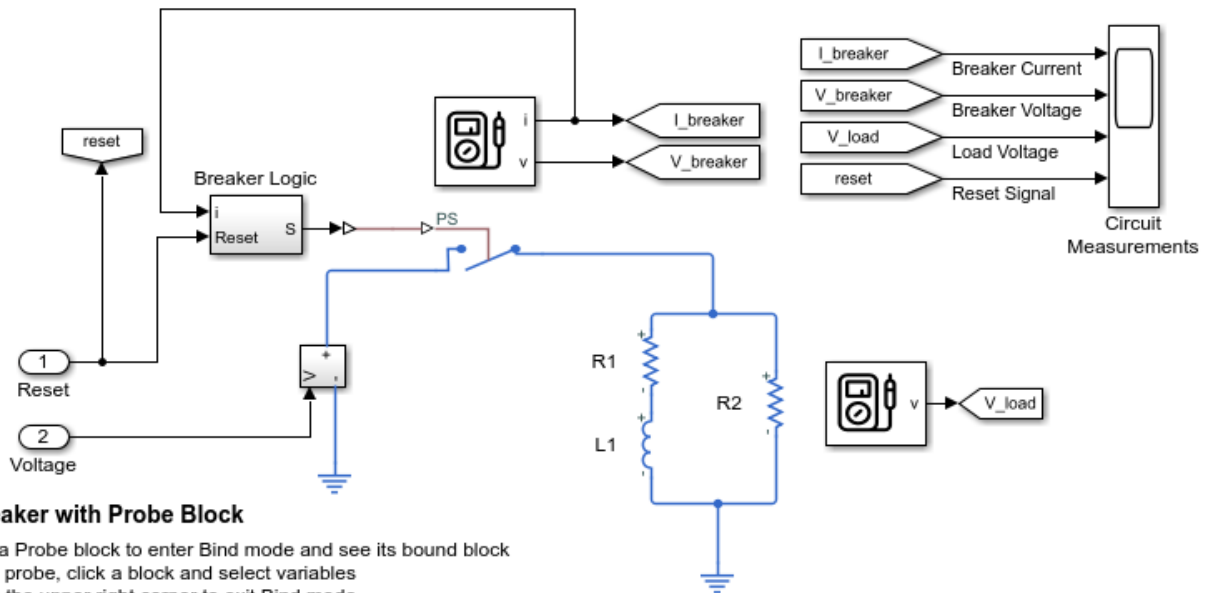


### Circuit Breaker with Probe Block

1. Double-click a Probe block to enter Bind mode and see its bound block
2. To rebind the probe, click a block and select variables
3. Click the X in the upper right corner to exit Bind mode
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

# Solenoid

This example shows a solenoid with a spring return. The solenoid is modeled as an inductance whose value L depends on the plunger position x. The back emf for a time-varying inductance is given by:

$$v = L\frac{di}{dt} + i\frac{dL}{dt} \qquad (1)$$

As L depends on x, this becomes:

$$v = L\frac{di}{dt} + i\frac{dL}{dx}\frac{dx}{dt} \qquad (2)$$

$\frac{dL}{dx}$ can be derived from manufacturer force-stroke data using the relationship:

$$force = 0.5 * i^2\frac{dL}{dx} \qquad (3)$$

$\frac{dL}{dx}$ can then be integrated to get L as a function of x.

In the model, equation 2 is rearranged to solve for i, and then implemented using Physical Signal blocks. A controlled current source then constrains the current to equate to i.

**Model**
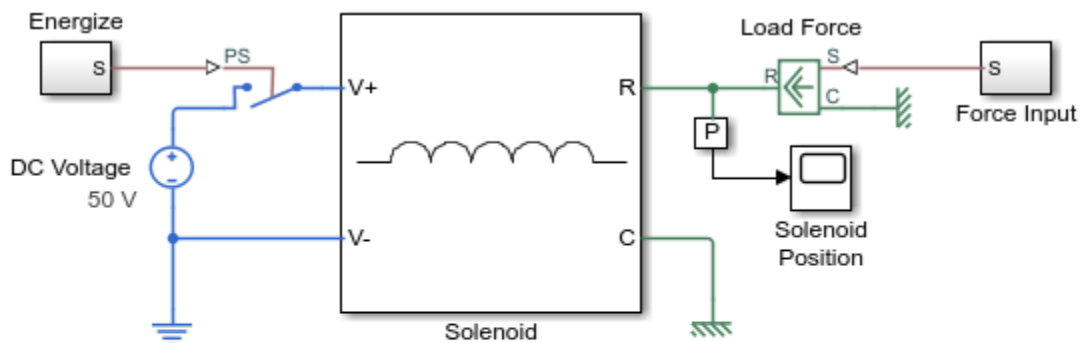


## Solenoid

1. Plot current and force of solenoid (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Solenoid Subsystem**
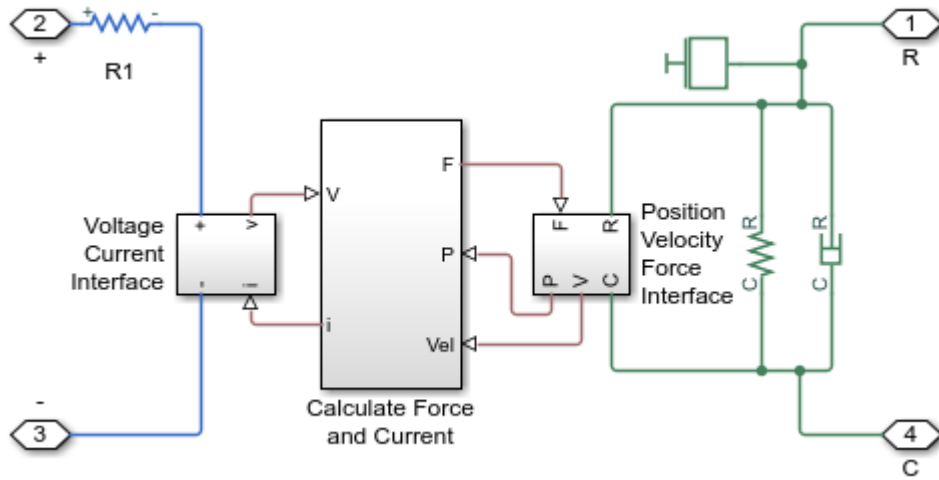


**Simulation Results from Simscape Logging**
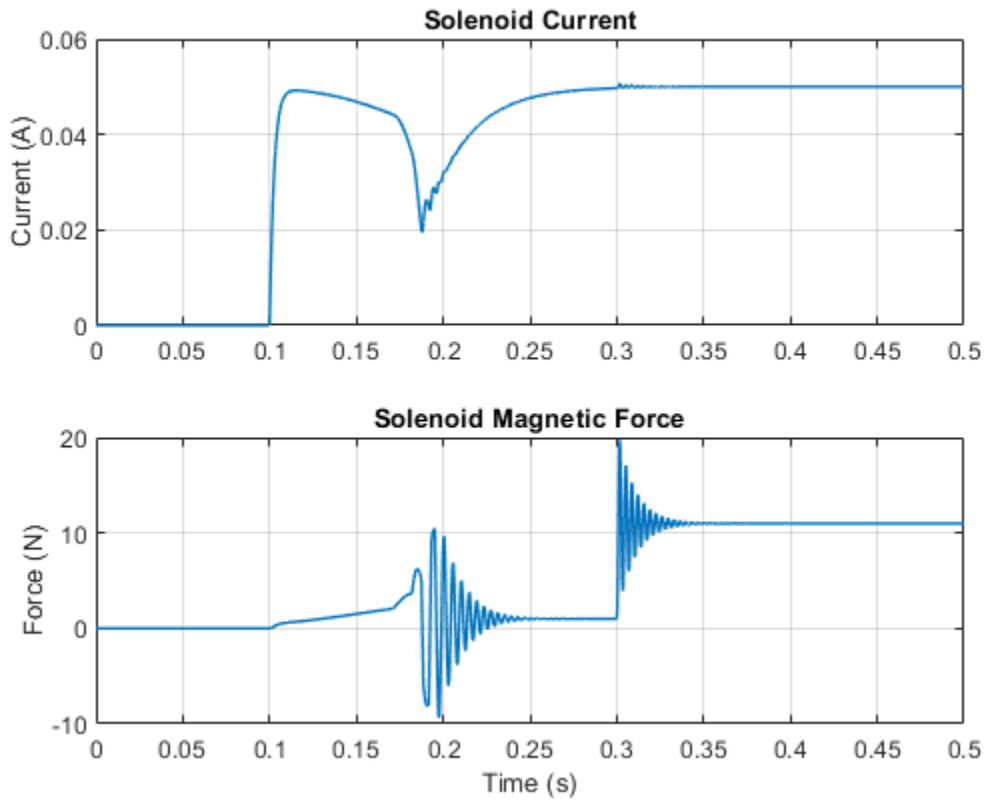
# Operating Point RLC Transient Response

This example shows the response of a DC power supply connected to a series RLC load. The goal is to plot the output voltage response when a load is suddenly attached to the fully powered-up supply. This is done using a Simscape operating point.

First, the power supply is connected to an open circuit and simulated until it reaches steady state. An operating point object is extracted from the resultant Simscape log. This operating point is used to initialize the model and verify that it is in steady state. Next, the load is changed to the series RLC circuit and the responses are compared with and without the operating point. Finally, a parameter sweep is done to compare the results with different values of the load inductance.

**Model**

The power supply consists of a DC voltage connected to an inductor, resistor, and capacitor. The values are chosen to demonstrate an underdamped open circuit response when powering up the supply. The load is a variant subsystem with an open circuit and a series RLC circuit.

```
model = 'ssc_op_rlc_transient_response';
open_system(model);
```



**Operating Point RLC Transient Response**

1. Plot open circuit response (see code)
2. Plot Series RLC circuit response (see code)
3. Perform parameter sweep on range of load inductance values (see code)
4. Explore simulation results using sscexplore
5. Learn more about this example

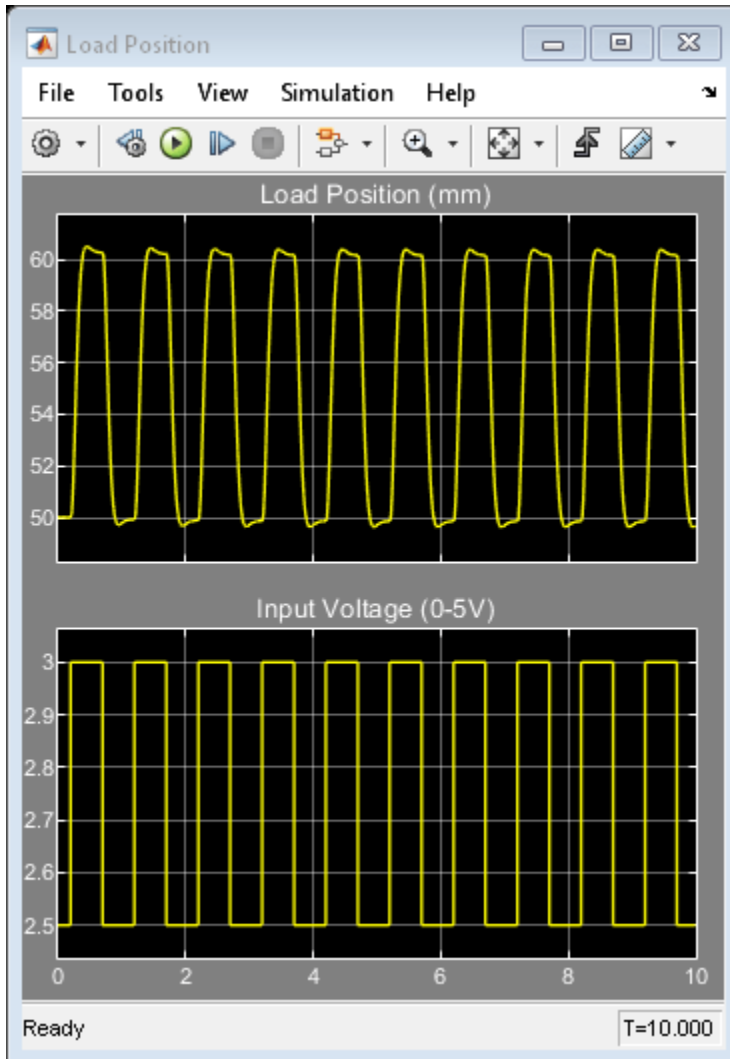Copyright 2017-2021 The MathWorks
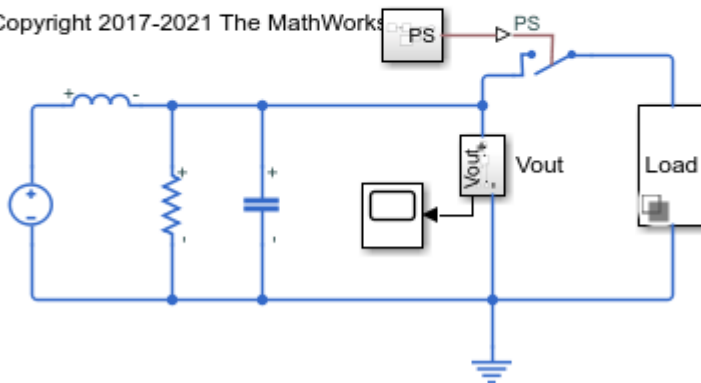
**Transient Open Circuit Response**

First, simulate to obtain the open circuit response of the power supply. The simulation is run long enough for the power supply to reach steady-state. This is the state that we want to start in when we experiment with different loads.

```
set_param('ssc_op_rlc_transient_response/Load', 'LabelModeActiveChoice', 'OpenCircuit');
sim(model);
```

**20-91**
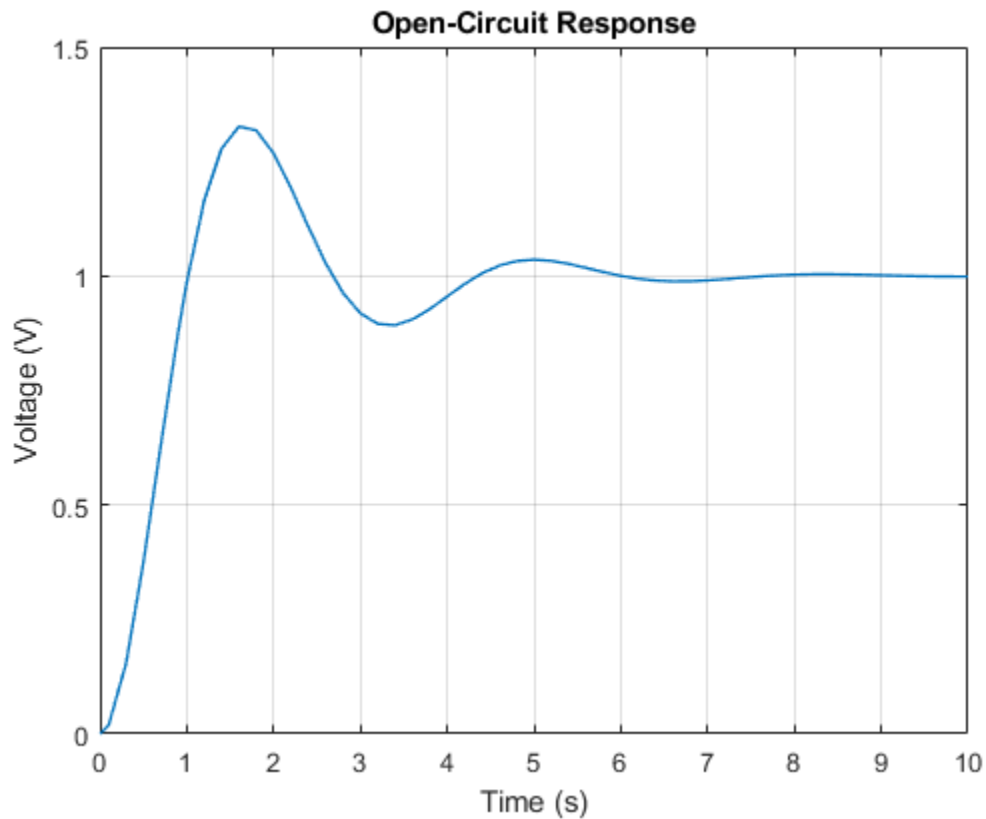
**Create Operating Point from Simscape Log**

Extract a steady-state Simscape operating point for the model by using the simscape.op.create function and the Simscape log that resulted from the previous simulation. Use '10' as the time because the simulation had reached an approximate steady state by that time.

```
op_steadystate = simscape.op.create(simlog_ssc_op_rlc_transient_response, 10);
```

Remove the operating point for the Load block since it will be irrelevant in subsequent experiments.

```
op_steadystate = remove(op_steadystate, 'Load')
```

```
op_steadystate =

  OperatingPoint with children:

  OperatingPoints:

   ChildId                  Size
   _____   ____

   'Capacitor'              1x1
   'DC Voltage'             1x1
   'Electrical Reference'   1x1
   'Inductor'               1x1
   'Series Resistance'      1x1
   'Step Input'             1x1
```

```
'Switch'                    1x1
'Vout'                      1x1
```

**Open Circuit Response with Operating Point**

Validate the operating point by initializing the open circuit model with the operating point. The result is a flat line representing the fully energized power supply.

```
set_param(model, 'SimscapeUseOperatingPoints', 'on', 'SimscapeOperatingPoint', 'op_steadystate')
sim(model);
```
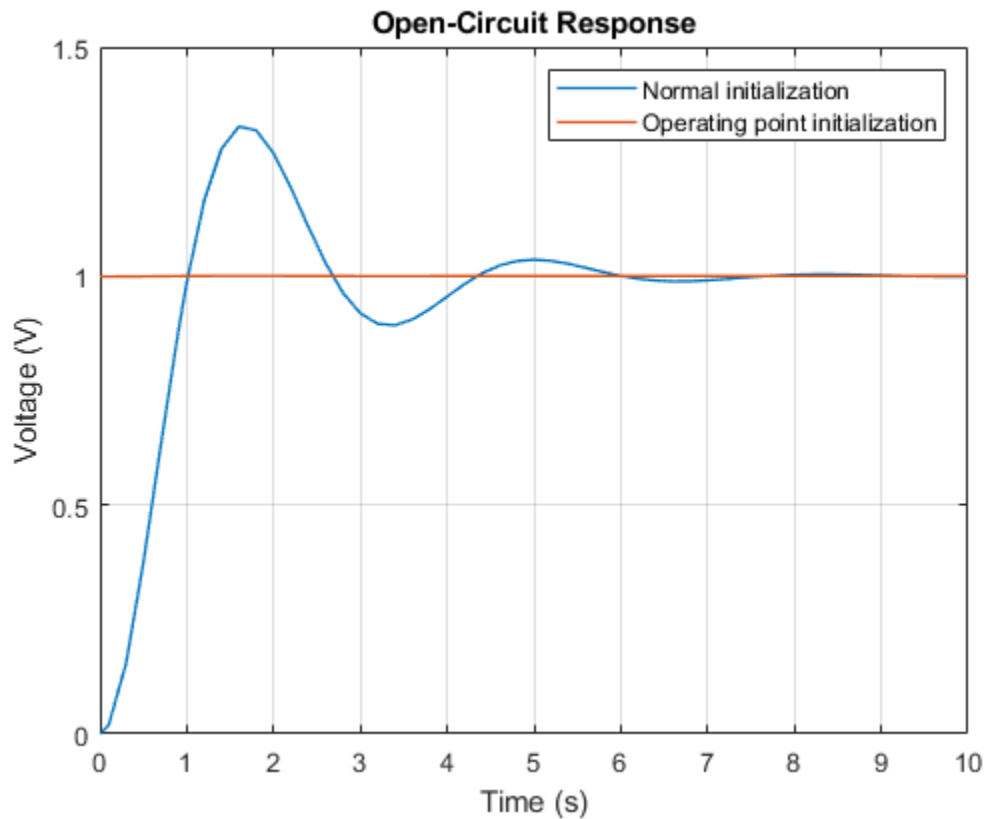


**Transient RLC Response Without Operating Point**

Change the load to the RLC series circuit and analyze the results. First, simulate without the operating point to show the combined response of the supply powering up and the load attached after 1 second. The results show what would happen if the load was attached while the supply was still powering up.

```
L_load = 1e-1;
set_param('ssc_op_rlc_transient_response/Load', 'LabelModeActiveChoice', 'RLC');
set_param(model, 'SimscapeUseOperatingPoints', 'off');
sim(model);
```

**RLC Load Response**
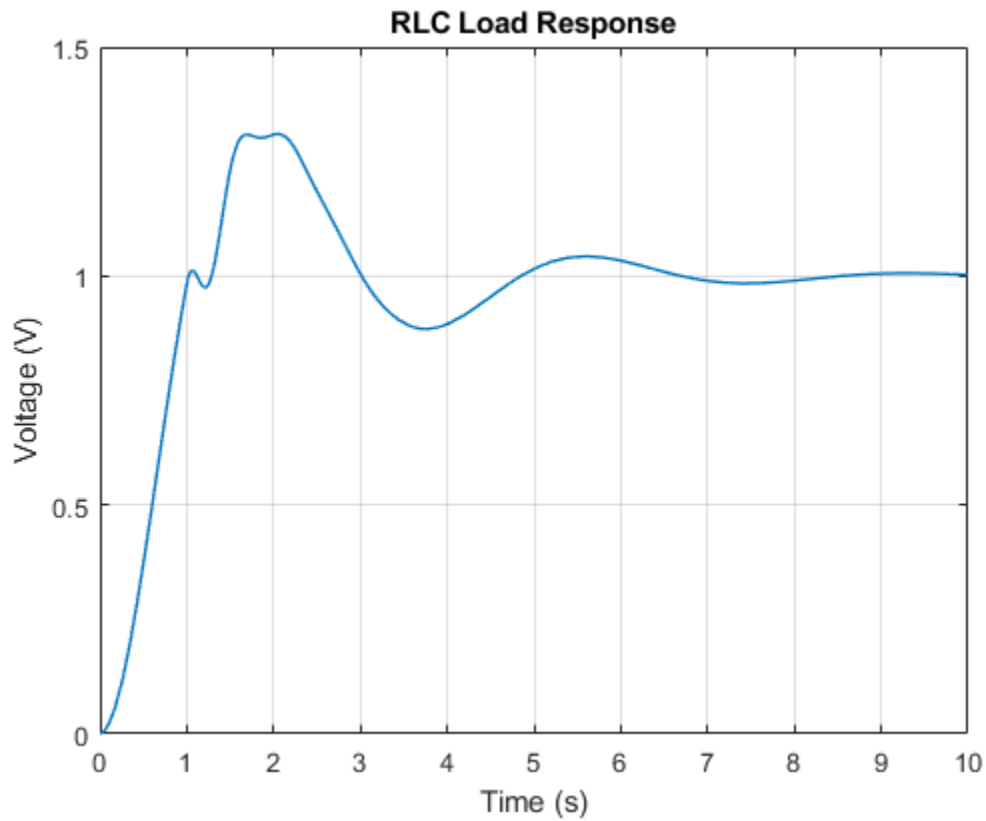
**Transient RLC Response with Operating Point**

Next, enable operating point initialization to see the desired response. The circuit is in steady state until we attach the load at 1 second.

```
set_param(model, 'SimscapeUseOperatingPoints','on');
sim(model);
```

**RLC Load Response**
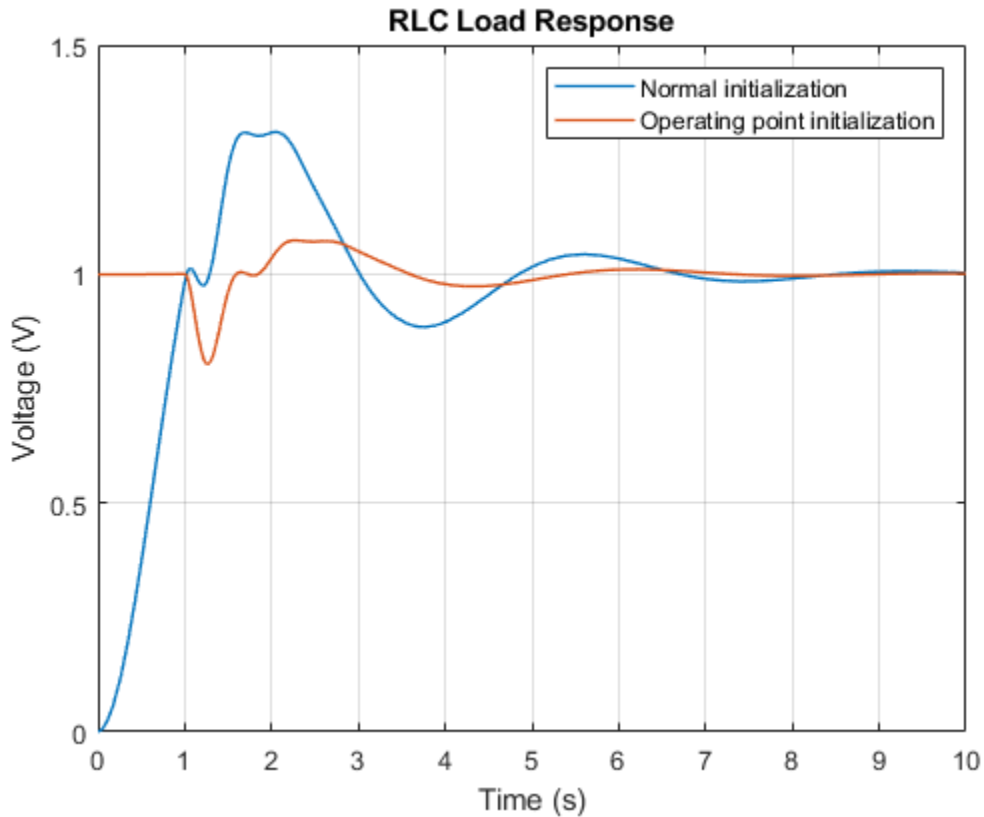


**Perform Parameter Sweep over Range of Load-Inductance Values**

Reuse the operating point in a series of simulations to compare the results across a range of load inductance values. Since the inductance load is run-to-run tunable, simulate the model in fast restart mode to avoid recompilation.

```
set_param(model,'FastRestart', 'on');
lValues = linspace(1e-2, 2e-1, 5);
hold on;
for idx = 1:numel(lValues)
    L_load = lValues(idx);
    out = sim(model);
    t = out.simlog_ssc_op_rlc_transient_response.Vout.Vs.V.series.time;
    Vout = out.simlog_ssc_op_rlc_transient_response.Vout.Vs.V.series.values('V');
    plot(t, Vout, 'LineWidth', 1);
end
hold off;
```
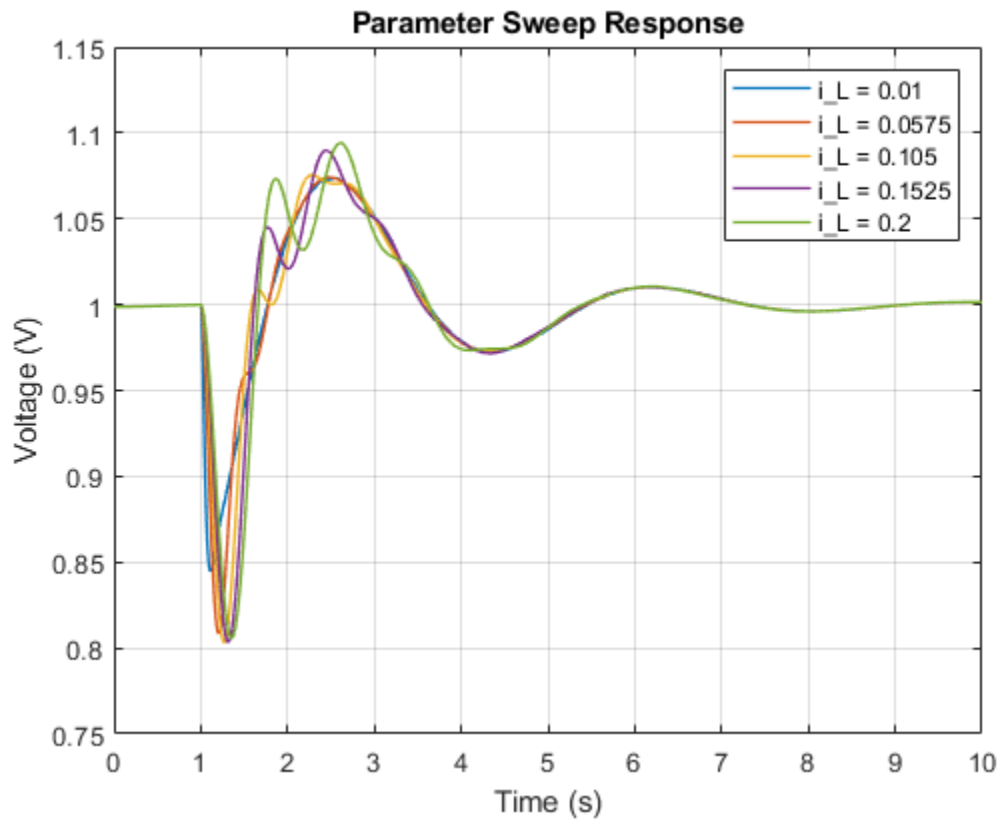
# Solenoid with Magnetic Blocks

This example shows how to model a solenoid with spring return. When unpowered, the spring pulls the plunger +5mm away from the center of the coil. Turning on the power supply at t=0.1 seconds pulls the plunger into the center of the coil. At t=0.3s, a 10N external load is added to the plunger.

This model is similar to the example model ssc_solenoid, except the solenoid here is modeled using magnetic blocks rather than physical signal blocks. The current through the solenoid produces a magnetomotive force (mmf) which drives a flux through the magnetic core of the solenoid. A reluctance force is produced which drives the plunger to close the air gap, initially 5mm in length. The flux in the magnetic core increases as the air gap length decreases.

**Model**
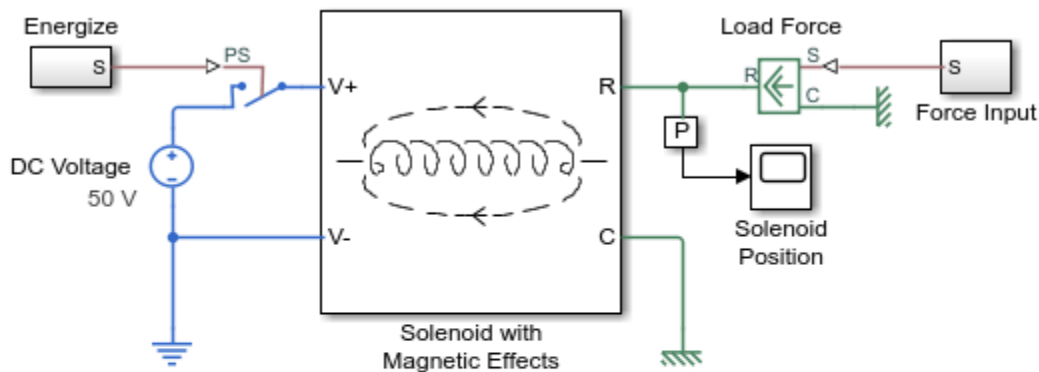


### Solenoid with Magnetic Blocks

1. Plot position and flux of solenoid (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Solenoid with Magnetic Effects Subsystem**
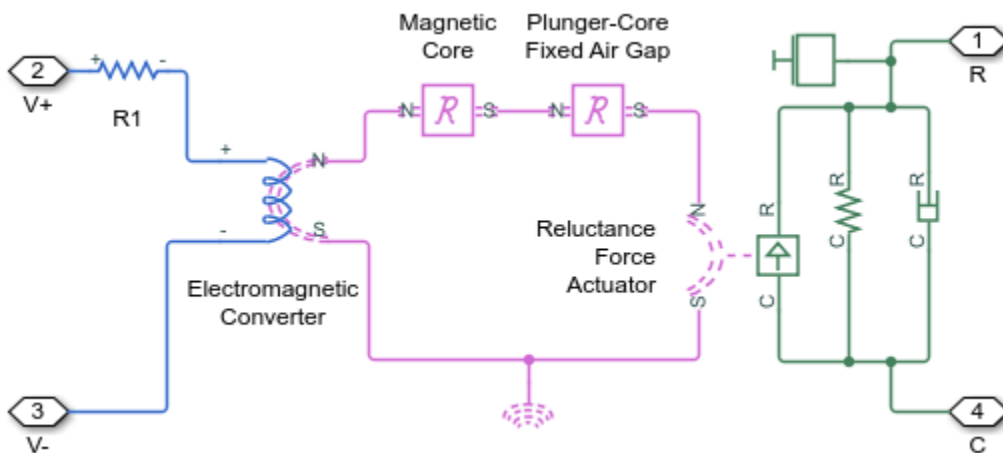
**Simulation Results from Scopes**
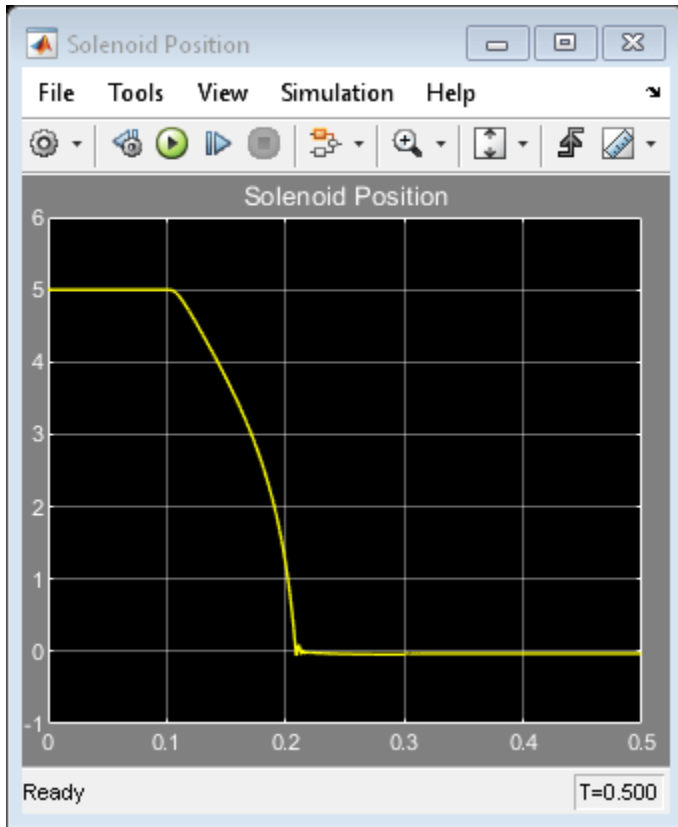
**Simulation Results from Simscape Logging**
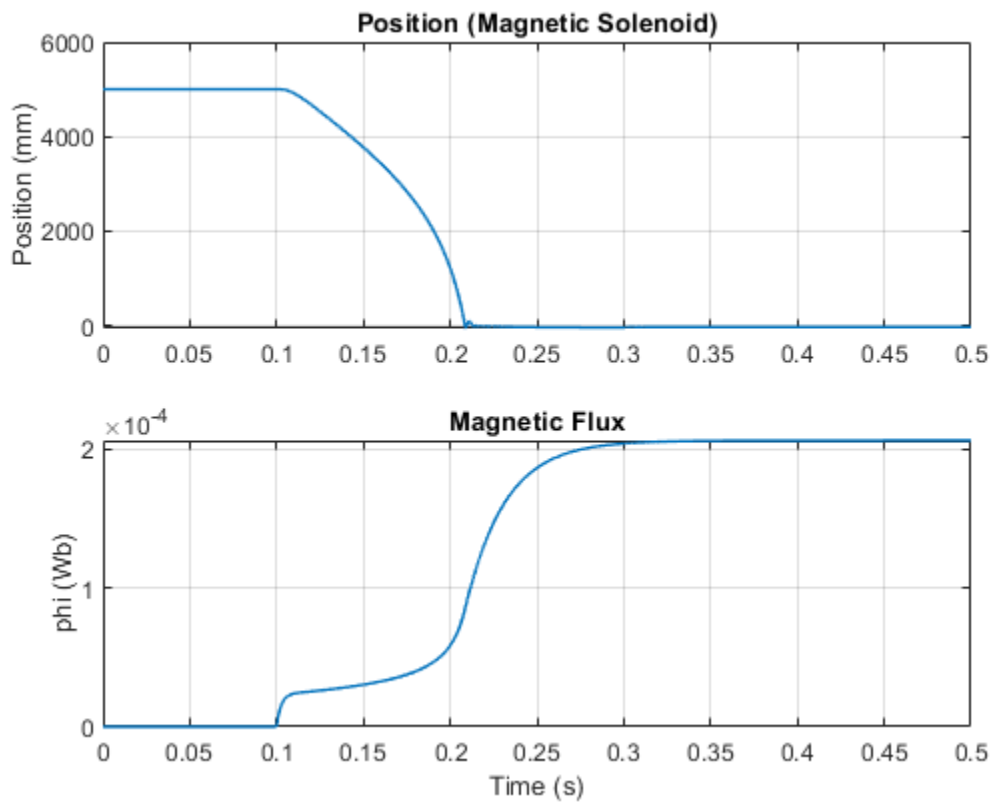
# Electrical Transformer
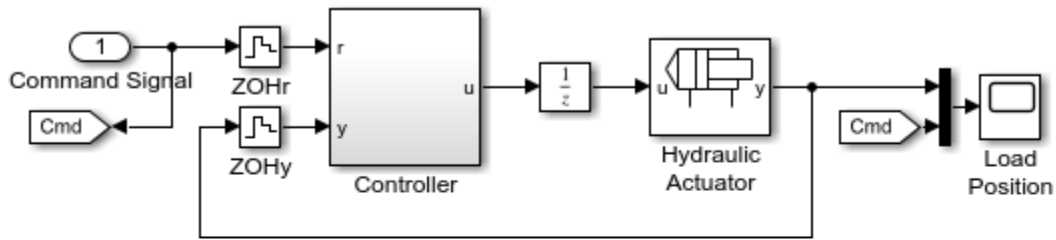
This example shows how to model a transformer using fundamental magnetic library blocks. The transformer is rated 50W, 60 Hz, 120V/12V and assumed to have an efficiency of 94%, no-load magnetizing current of 1% and a leakage reactance of 2.3%. Core losses are not modeled and the core material B-H characteristic is assumed to be linear.

The transformer initially operates under no-load conditions. At time t=0.5s, the rated load is turned on. Because of the winding resistances and leakage inductances, the secondary voltage drops from 12 Vrms to 11.3 Vrms and the transformer supplies a full load current of 3.9 A rms on its secondary. The flux and mmf scopes show the operating conditions during no-load and full-load. Compare the leakage flux with the core flux, and observe the primary and secondary-side mmfs.

**Model**
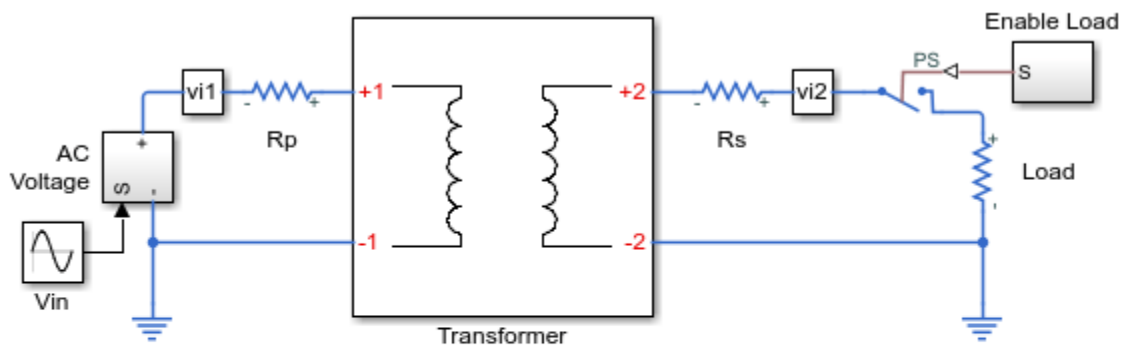


### Electrical Transformer

1. Plot fluxes in transformer (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example
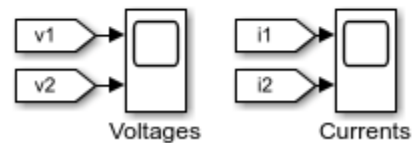
Copyright 2015-2021 The MathWorks, Inc.

**Transformer Subsystem**
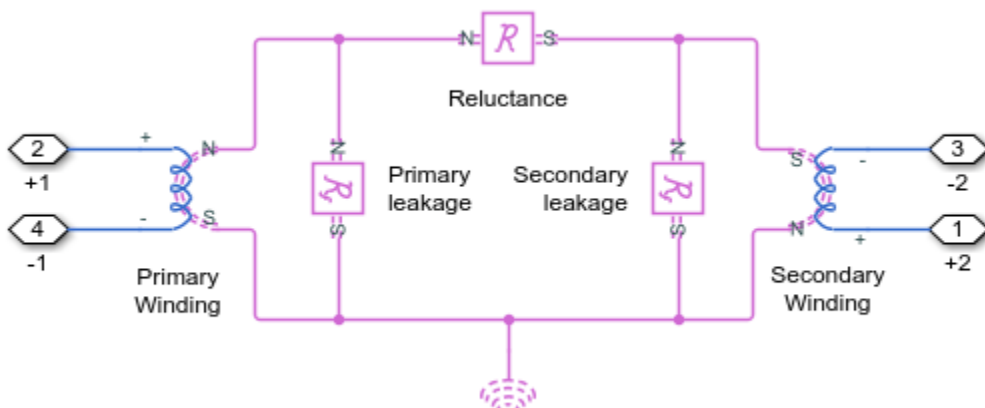
**Simulation Results from Simscape Logging**

# Hydraulic Actuator with Analog Position Controller

This example shows how the Foundation library can be used to model systems that span electrical, mechanical and hydraulic domains. In the model, a hydraulic system controls mechanical load position in response to a voltage reference demand. If the reference demand is zero, then the hydraulic actuator (and load) displacement is zero, and if the reference is +5 volts, then the displacement is 100mm.

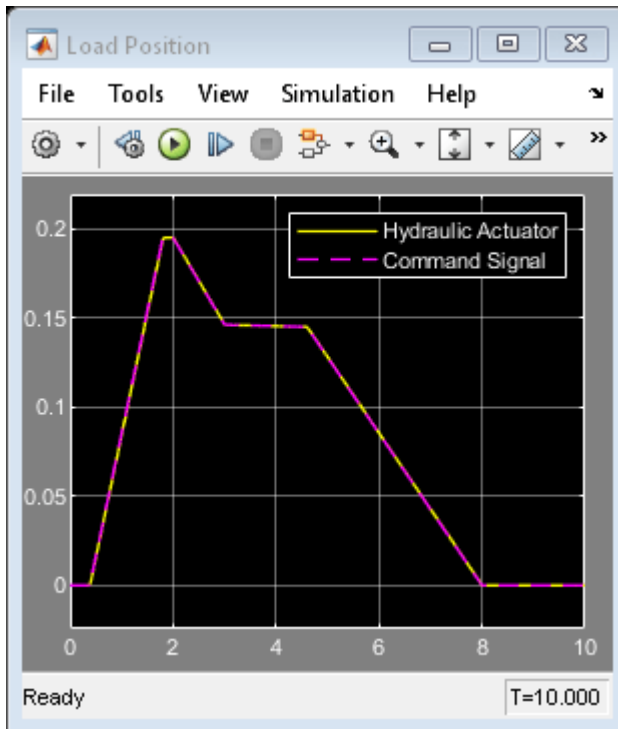The proportional plus integral controller is implemented using op-amps, the final stage of which is set up as a current source. This then drives a torque motor with resistance and inductance terms modeled. The torque motor directly actuates the spool valve, which in turn controls the main hydraulic circuit supplying the hydraulic ram. Finally, the ram drives a generic mechanical load.

A model with this level of fidelity is ideal to support the servo-valve controller design and testing. It includes the electromechanical high frequency modes that affect stability margins, as well as nonlinear flow rate effects when large demands are made from the hydraulics.

**Model**



## Hydraulic Actuator with Analog Position Controller

1. Plot cylinder pressures in a figure (see code)
2. View cylinder pressures in Simulation Data Inspector
3. Browse simulation results using Simulation Data Inspector
4. Explore simulation results using sscexplore
3. Learn more about this example

**Actuator Control Circuit Subsystem**



**Spool Valve Subsystem**

**Motor Control Circuit Subsystem**

**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

The figure below shows the cylinder pressures and load position plotted in a MATLAB figure. You can also view the data in the Simscape Results Explorer and the Simulation Data Inspector.

# Hydraulic Actuator with Analog Position Controller and Dashboard Blocks

This example shows how to use the Foundation library to model systems that span electrical, mechanical and hydraulic domains.

In the model, a hydraulic system controls mechanical load position in response to a voltage reference demand. If the reference demand is zero, then the hydraulic actuator (and load) displacement is zero, and if the reference is +5 volts, then the displacement is 100 mm.

The proportional plus integral controller is implemented using op-amps, the final stage of which is set up as a current source. This then drives a torque motor with resistance and inductance terms modeled. The torque motor directly actuates the spool valve, which in turn controls the main hydraulic circuit supplying the hydraulic ram. Finally, the ram drives a generic mechanical load.

A model with this level of fidelity is ideal to support the servo-valve controller design and testing. It includes the electromechanical high frequency modes that affect stability margins, as well as nonlinear flow rate effects when large demands are made from the hydraulics.

This example includes animations of the actuator and pressure gauges created using a Vertical Gauge block and two Circular Gauge blocks from the Simulink / Dashboard / Customizable Blocks library.

**Model**



**Hydraulic Actuator with Analog Position Controller and Dashboard Blocks**

1. Plot cylinder pressures in a figure (see code)
2. View cylinder pressures in Simulation Data Inspector
3. Browse simulation results using Simulation Data Inspector
4. Explore simulation results using sscexplore
3. Learn more about this example

### Actuator Control Circuit Subsystem



### Spool Valve Subsystem

**Motor Control Circuit Subsystem**

**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

The figure below shows the cylinder pressures and load position plotted in a MATLAB figure. You can also view this data in the Simulation Data Inspector by clicking the **View cylinder pressures** hyperlink in the model canvas or by clicking the **Data Inspector** button on the model **Simulation** tab.

# Hydraulic Actuator with Digital Position Controller

This example shows a two-way valve acting in a closed-loop circuit together with a double-acting cylinder. The controller is represented as a continuous-time transfer function plus a transport delay. The delay allows for the computational time (one discrete time period) plus the zero-order hold (half a discrete time period) when implemented in discrete time. The model is configured for linearization so that a frequency response can be generated. To configure for faster desktop simulation, comment through the transport delay and increase the solver maximum step size.

**Model**



## Hydraulic Actuator with Digital Position Controller

1. Plot pressures in hydraulic cylinder (see code)
2. Linearize the hydraulic plant (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Hydraulic Actuator Subsystem**

**Hydraulic Cylinder Subsystem**



**Simulation Results from Scopes**

**Simulation Results from Simscape Logging**



**Frequency Response**



## Hydraulic Actuator with Digital Position Controller

1. Plot pressures in hydraulic cylinder (see code)
2. Linearize the hydraulic plant (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Frequency Response (Hydraulic Actuator)

# Hydraulic Actuator Configured for HIL Testing

This example shows a physical system model and controller configured for HIL testing. It is derived from example Hydraulic Actuator with Digital Position Controller, ssc_hydraulic_actuator_digital_control. The model has been configured for HIL testing by performing the following steps:

1. Make the controller discrete time: The Transport Delay block is replaced by a Unit Delay block. This represents the worst case delay whereby it takes a full computational frame time for the controller output u to be updated based on current input values for r and y. Zero-Order Hold blocks are added to all analog measurements (ZOHr and ZOHy) to represent digital sampling of continuous time measurements. The controller itself must be made discrete time, so the continuous time first order filter converted to a discrete time filter using a Tustin transformation. In this example the discrete sample time is made a parameter, the advantage of this being that it can be easily adjusted if necessary to ensure the controller calculations do not cause a frame time overrun.

2. Partition each HIL component into its own subsystem. The hydraulic plant is already in its own subsystem, so we just need to group the controller into its own subsystem. This partitioning helps if just part of the model is to be run in HIL, or controller and plant are to be run on separate HIL systems.

3. Set fixed step local solver option, setting fixed step to the controller sample time ts. Make ts as large as possible whilst retaining simulation fidelity required. Sometimes the plant may require a different sample time, typically smaller, than the controller. In this case ts=0.001 is small enough for the plant and controller. Determine how many Nonlinear iterations are required for convergence; some models may need more than the default 3.

4. Run Performance Advisor checks relating to Simscape™.

5. Review the model for any simplifications that the Performance Advisor did not find. One method is to linearize the model to look for fast eigenvalues in the resulting A-matrix, and then to relate these back to model components. Applying to ssc_hydraulic_actuator_digital_control reveals that dynamic compressibility in the Hydraulic Cylinder can be removed to avoid an oscillatory dynamic pole pair. This change has been made to the model.

6. Configure code generation target: see Simulink® Coder Deployment documentation

**Model**



**Hydraulic Actuator Configured for HIL Testing**
1. Plot pressures in hydraulic cylinder (see code)
2. Compare results of variable and fixed-step simulation (see code)
3. Open Hydraulic Actuator with Digital Position Controller
4. Explore simulation results using sscexplore
5. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Hydraulic Actuator Subsystem**

**Hydraulic Cylinder Subsystem**



**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

Plot "Cylinder Pressures" shows how the cylinder pressure varies during simulation. It corresponds with the opening of the valve. The opening of the valve is set by the controller so that the actuator tracks a reference signal.

Plot "Cylinder Pressure, Solver Types" shows the effect of solver type on simulation results. The variable-step simulation uses smaller simulation steps to accurately capture the system dynamics. The fixed-step simulation results are close but do not exactly match the variable-step simulation results, for the solver is not permitted to adjust its step size. The fixed-step solver settings should be adjusted until the fixed-step simulation results are within an acceptable range of the variable-step simulation results.

# Cavitation Cycle

This model shows how the fluid in a custom double acting cylinder can cavitate and recover. During the repeating cycle of oscillating pressure, the absolute pressure does not get below absolute zero as the bulk modulus of the homogeneous liquid-gas mixture decreases accordingly at low pressures.

**Model**
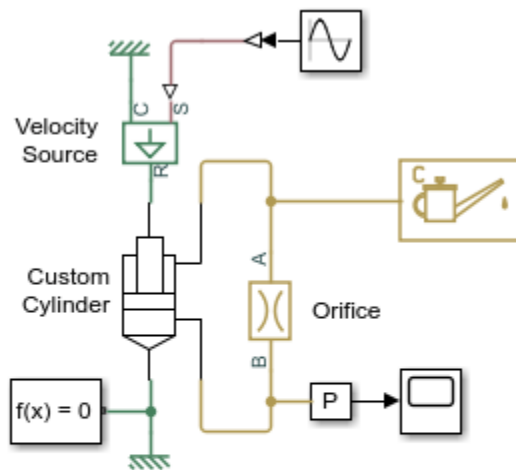


**Cavitation Cycle**
1. Plot pressures in cylinder (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2004-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**

The plots below show the pressures in the cylinder chambers and the piston position.

# Hydraulic Actuator with Analog Position Controller

This example shows how the Foundation library can be used to model systems that span electrical, mechanical and isothermal liquid domains. In the model, a hydraulic system implemented in the isothermal liquid domain controls the mechanical load position in response to a voltage reference demand. If the reference demand is zero, then the hydraulic actuator (and load) displacement is zero, and if the reference is +5 volts, then the displacement is 100 mm.

The proportional plus integral controller is implemented using op-amps, the final stage of which is set up as a current source. This then drives a torque motor with resistance and inductance terms modeled. The torque motor directly actuates the spool valve, which in turn controls the main hydraulic circuit supplying the hydraulic ram. Finally, the ram drives a generic mechanical load.

A model with this level of fidelity is ideal to support the servo-valve controller design and testing. It includes the electromechanical high frequency modes that affect stability margins, as well as nonlinear flow rate effects when large demands are made from the hydraulics.

**Model**



## Hydraulic Actuator with Analog Position Controller

1. Plot cylinder pressures in a figure (see code)
2. View cylinder pressures in Simulation Data Inspector
3. Browse simulation results using Simulation Data Inspector
4. Explore simulation results using sscexplore
5. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Actuator Control Circuit Subsystem**



**Spool Valve Subsystem**

**Motor Control Circuit Subsystem**

**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

The figure below shows the cylinder pressures and load position plotted in a MATLAB figure. You can also view the data in the Simscape Results Explorer and the Simulation Data Inspector.

# Hydraulic Actuator with Analog Position Controller and Dashboard Blocks

This example shows how to use the Foundation library to model systems that span electrical, mechanical and isothermal liquid domains.

In the model, a hydraulic system implemented in the isothermal liquid domain controls the mechanical load position in response to a voltage reference demand. If the reference demand is zero, then the hydraulic actuator (and load) displacement is zero, and if the reference is +5 volts, then the displacement is 100 mm.

The proportional plus integral controller is implemented using op-amps, the final stage of which is set up as a current source. This then drives a torque motor with resistance and inductance terms modeled. The torque motor directly actuates the spool valve, which in turn controls the main hydraulic circuit supplying the hydraulic ram. Finally, the ram drives a generic mechanical load.

A model with this level of fidelity is ideal to support the servo-valve controller design and testing. It includes the electromechanical high frequency modes that affect stability margins, as well as nonlinear flow rate effects when large demands are made from the hydraulics.

This example includes animations of the actuator and pressure gauges created using a Vertical Gauge block and two Circular Gauge blocks from the Simulink / Dashboard / Customizable Blocks library.

**Model**



**Hydraulic Actuator with Analog Position Controller and Dashboard Blocks**

1. Plot cylinder pressures in a figure (see code)
2. View cylinder pressures in Simulation Data Inspector
3. Browse simulation results using Simulation Data Inspector
4. Explore simulation results using sscexplore
5. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

## Actuator Control Circuit Subsystem

Op-Amp 2

Integrator
Vo=-V1/(R8*C2*s)

R8

C2

R5    R6

Add integrator
term (R6/R5)Vo

Op-Amp 1

1 P+

2 P-

4 D+

5 D-

Summing amplifier
with 1st order filter

R3

Proportional
term -R7/R3

Op-Amp 3

R7

3 Vi

## Spool Valve Subsystem

1 Vi

Vi  A

Torque Motor, Controller
and Spool Dynamics

Reverse
Direction

Area TA    Area PA    Area PB    Area TB

4 A    5 B

Orifice TA    Orifice PA    Orifice PB    Orifice TB

2 P    3 T

**Motor Control Circuit Subsystem**

**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

The figure below shows the cylinder pressures and load position plotted in a MATLAB figure. You can also view this data in the Simulation Data Inspector by clicking the **View cylinder pressures** hyperlink in the model canvas or by clicking the **Data Inspector** button on the model **Simulation** tab.

# Hydraulic Actuator with Digital Position Controller

This example shows a two-way valve acting in a closed-loop circuit together with a double-acting cylinder implemented in the isothermal liquid domain. The controller is represented as a continuous-time transfer function plus a transport delay. The delay allows for the computational time (one discrete time period) plus the zero-order hold (half a discrete time period) when implemented in discrete time. The model is configured for linearization so that a frequency response can be generated. To configure for faster desktop simulation, comment through the transport delay and increase the solver maximum step size.

**Model**



## Hydraulic Actuator with Digital Position Controller

1. Plot pressures in hydraulic cylinder (see code)
2. Linearize the hydraulic plant (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Hydraulic Actuator Subsystem**

**Hydraulic Cylinder Subsystem**



**Simulation Results from Scopes**

**Simulation Results from Simscape Logging**



**Frequency Response**



## Hydraulic Actuator with Digital Position Controller

1. Plot pressures in hydraulic cylinder (see code)
2. Linearize the hydraulic plant (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

Frequency Response (Hydraulic Actuator)

# Hydraulic Actuator Configured for HIL Testing

This example shows a physical system model and controller configured for HIL testing. It is derived from example Hydraulic Actuator with Digital Position Controller, ssc_isothermal_liquid_actuator_digital_control. The model has been configured for HIL testing by performing the following steps:

1. Make the controller discrete time: The Transport Delay block is replaced by a Unit Delay block. This represents the worst case delay whereby it takes a full computational frame time for the controller output u to be updated based on current input values for r and y. Zero-Order Hold blocks are added to all analog measurements (ZOHr and ZOHy) to represent digital sampling of continuous time measurements. The controller itself must be made discrete time, so the continuous time first order filter converted to a discrete time filter using a Tustin transformation. In this example the discrete sample time is made a parameter, the advantage of this being that it can be easily adjusted if necessary to ensure the controller calculations do not cause a frame time overrun.

2. Partition each HIL component into its own subsystem. The hydraulic plant is already in its own subsystem, so we just need to group the controller into its own subsystem. This partitioning helps if just part of the model is to be run in HIL, or controller and plant are to be run on separate HIL systems.

3. Set fixed step local solver option, setting fixed step to the controller sample time ts. Make ts as large as possible while retaining simulation fidelity required. Sometimes the plant may require a different sample time, typically smaller, than the controller. In this case ts=0.001 is small enough for the plant and controller. Determine how many Nonlinear iterations are required for convergence; some models may need more than the default 3.

4. Run Performance Advisor checks relating to Simscape™.

5. Review the model for any simplifications that the Performance Advisor did not find. One method is to linearize the model to look for fast eigenvalues in the resulting A-matrix, and then to relate these back to model components. Applying to ssc_il_actuator_digital_control reveals that dynamic compressibility in the Hydraulic Cylinder can be removed to avoid an oscillatory dynamic pole pair. This change has been made to the model.

6. Configure code generation target: see Simulink® Coder Deployment documentation

**Model**



**Hydraulic Actuator Configured for HIL Testing**
1. Plot pressures in hydraulic cylinder (see code)
2. Compare results of variable and fixed-step simulation (see code)
3. Open Hydraulic Actuator with Digital Position Controller
4. Explore simulation results using sscexplore
5. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Hydraulic Actuator Subsystem**

**Hydraulic Cylinder Subsystem**



**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

Plot "Cylinder Pressures" shows how the cylinder pressure varies during simulation. It corresponds with the opening of the valve. The opening of the valve is set by the controller so that the actuator tracks a reference signal.

Plot "Cylinder Pressure, Solver Types" shows the effect of solver type on simulation results. The variable-step simulation uses smaller simulation steps to accurately capture the system dynamics. The fixed-step simulation results are close but do not exactly match the variable-step simulation results, for the solver is not permitted to adjust its step size. The fixed-step solver settings should be adjusted until the fixed-step simulation results are within an acceptable range of the variable-step simulation results.

# Entrained Air Effects

This model shows the effects of entrained air in a custom double acting cylinder. During the repeating cycle of oscillating pressure, the absolute pressure does not get below absolute zero as the bulk modulus of the homogeneous liquid-gas mixture decreases accordingly at low pressures.

**Model**
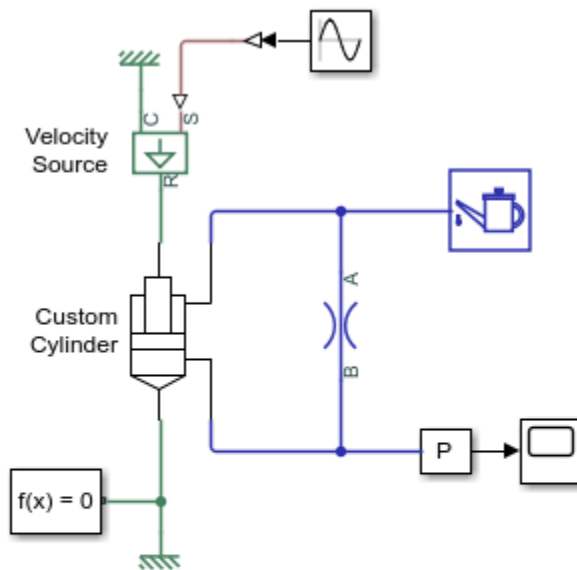


## Entrained Air Effects

1. Plot pressures in cylinder (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2018-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**

The plots below show the pressures in the cylinder chambers and the piston position.
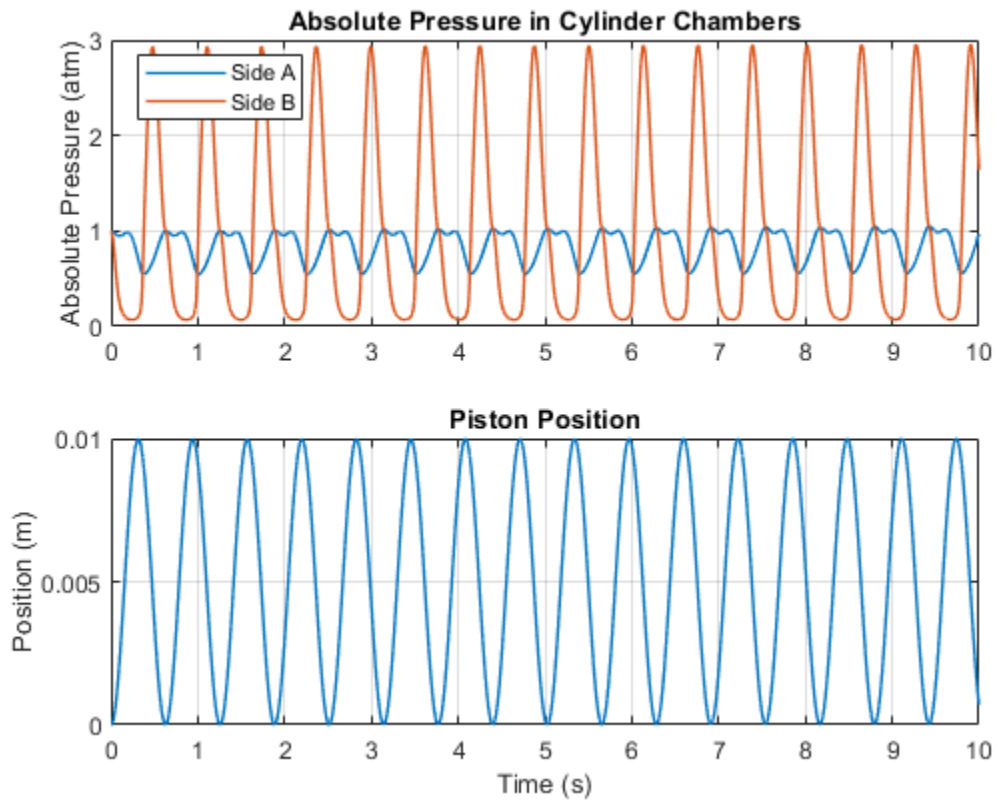
# Hydraulic Fluid Warming Due to Losses

This example model shows how Foundation Library thermal liquid components can be used to estimate the impact of viscous losses on an actuating system's temperature over a long time scale. A pump supplies energy to the system to actuate a double-acting cylinder periodically. The pressure losses within directional valve converts the energy into heat. Heat transfer through the cylinder casing to the environment provides a heat sink.

**Model**
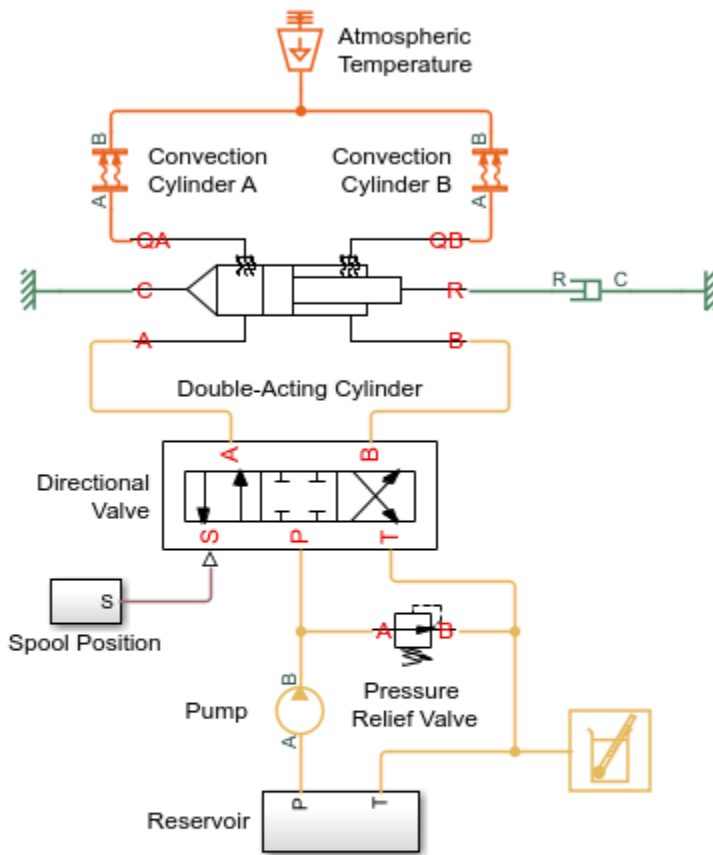


### Hydraulic Fluid Warming Due to Losses

1. Plot fluid properties (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.
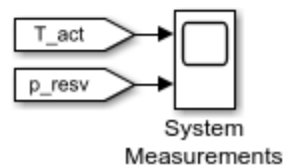
**Directional Valve Subsystem**
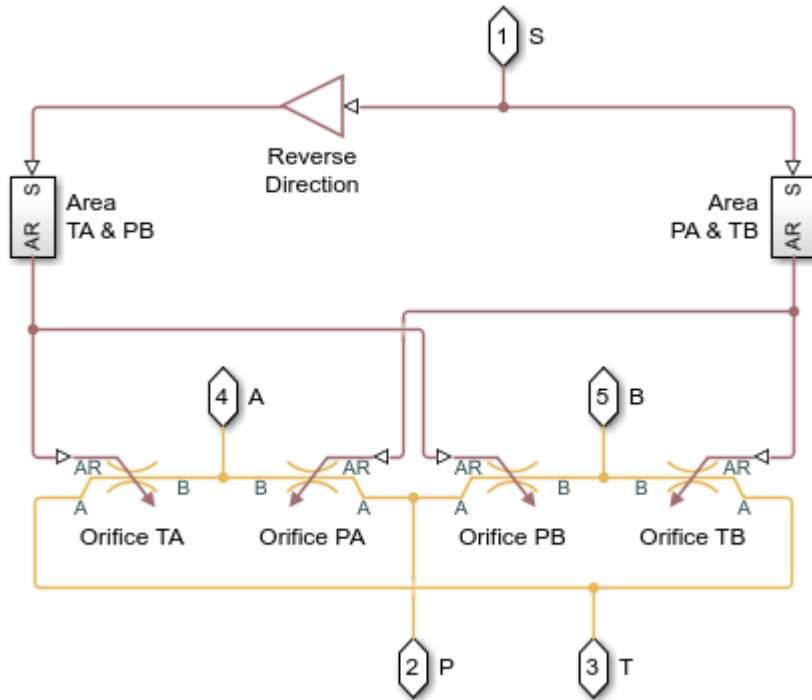


**Double-Acting Cylinder Subsystem**
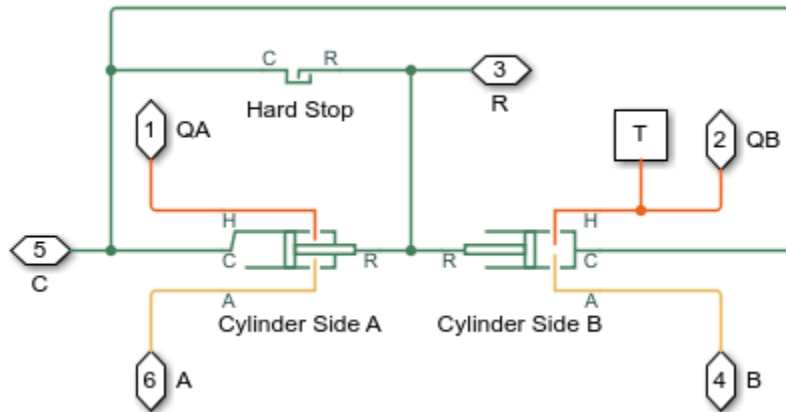
**Simulation Results from Scopes**
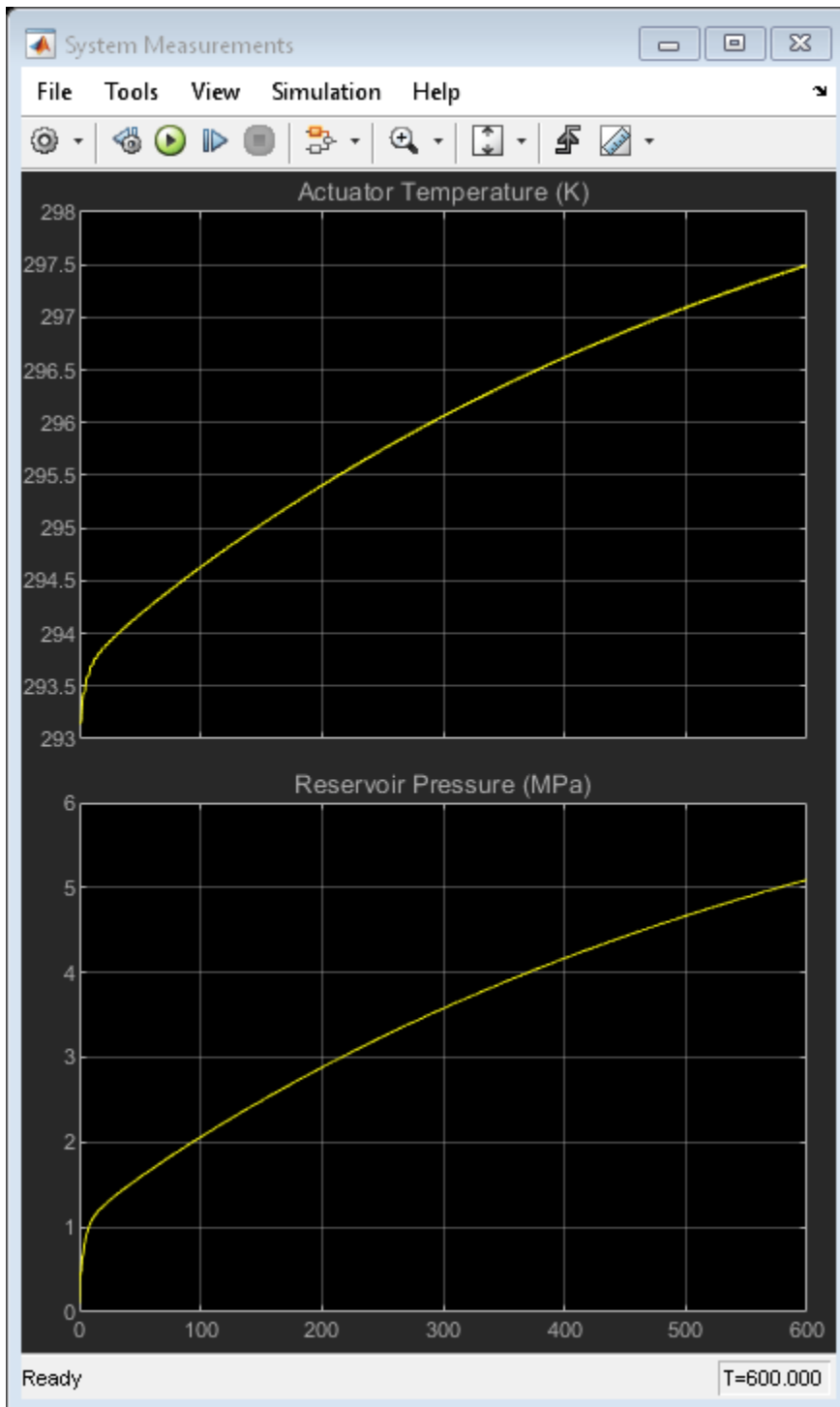
**Fluid Properties**
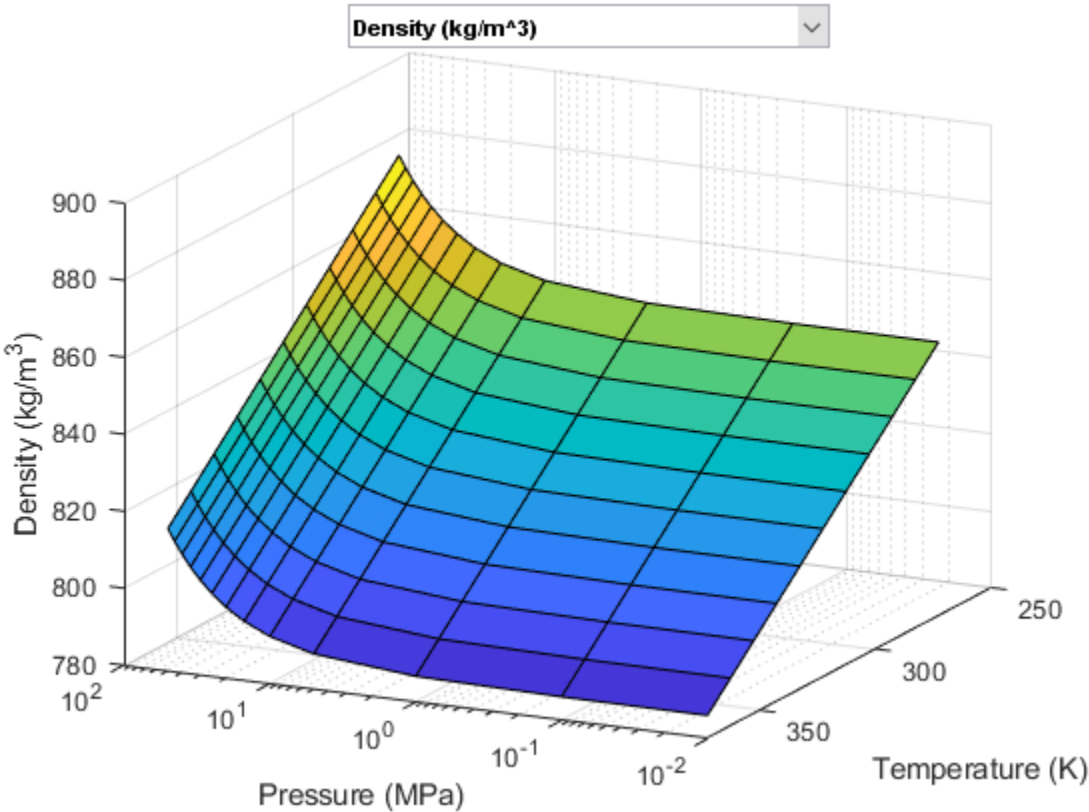
# Optimal Pipeline Geometry for Heated Oil Transportation

This example model shows the opposing effects of viscous warming and conductive cooling on the temperature of a buried pipeline segment for heated oil transportation. A requirement for these systems is for the crude oil to stay well above the pour point to provide a continuous steady flow. The system is modeled using the Thermal Liquid Foundation Library. Crude oil thermodynamic properties are stored in workspace matrices and are used to populate the Thermal Liquid Settings block mask parameters.

The model can be used to optimize the pipe diameter. In this example it is assumed that the outer diameter of the insulator is fixed, and it is the inner diameter that is to be optimized. An optimal design is where the heating and thermal loss terms balance so that the temperature stays constant through the segment. This model reproduces the study detailed in 'Optimal Pipeline Geometries and Oil Temperatures for Least Rates of Energy Expenditure During Crude-oil Transmission', S.D. Probert and C. Y. Chu, Applied Energy 14(1983) 1-31.

**Model**
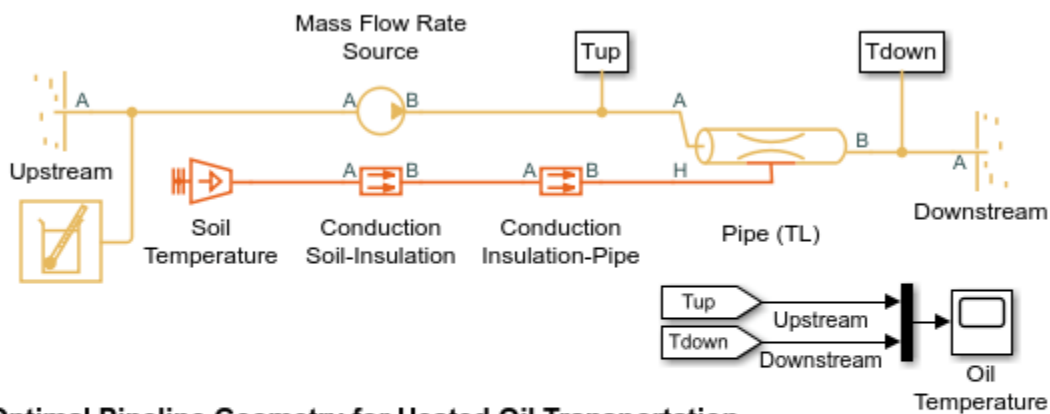


### Optimal Pipeline Geometry for Heated Oil Transportation

1. Optimize pipe insulation inner diameter (see code)
2. Plot fluid properties (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Scopes**

**Optimization**



Energy Transfer to/from Oil
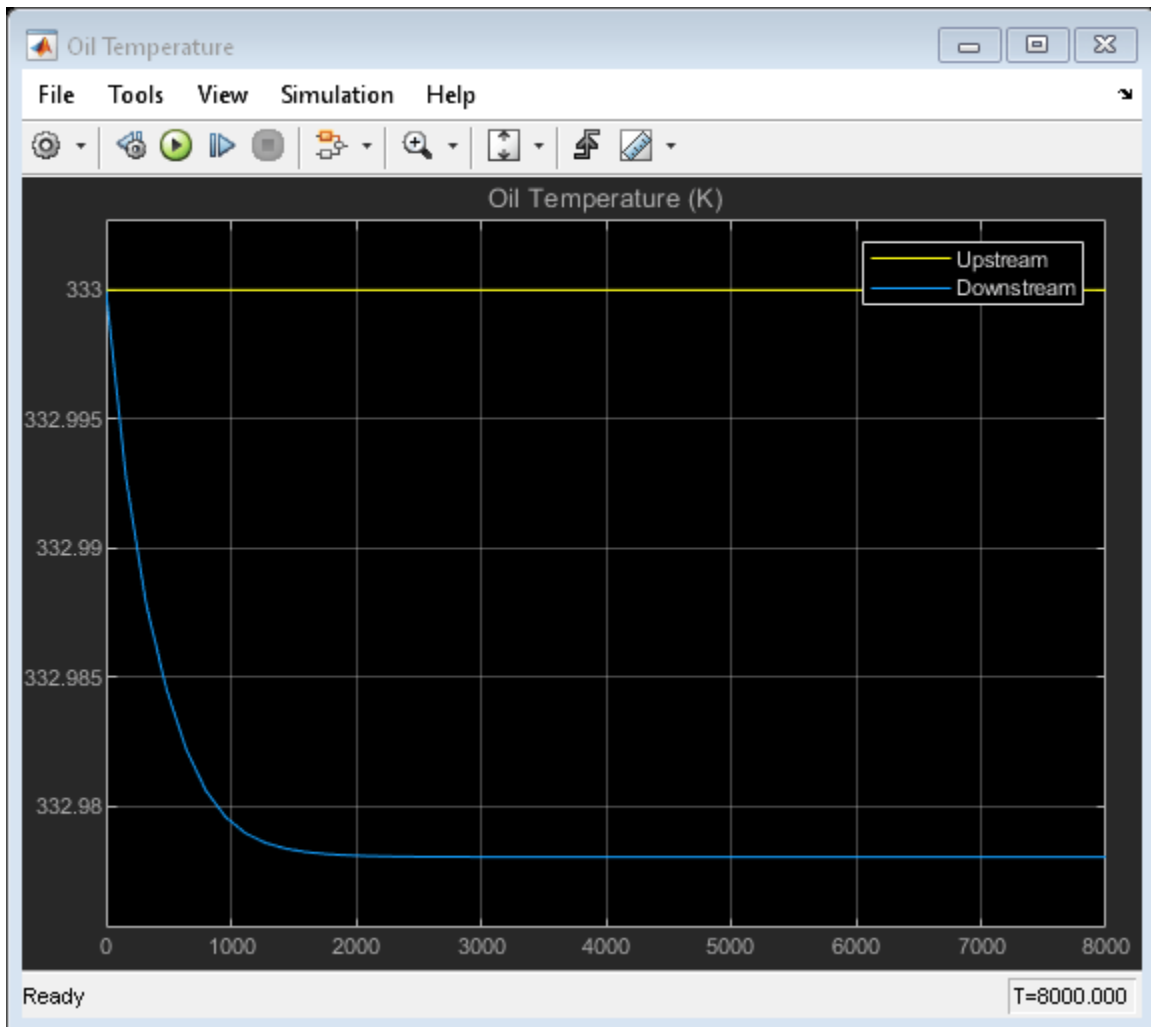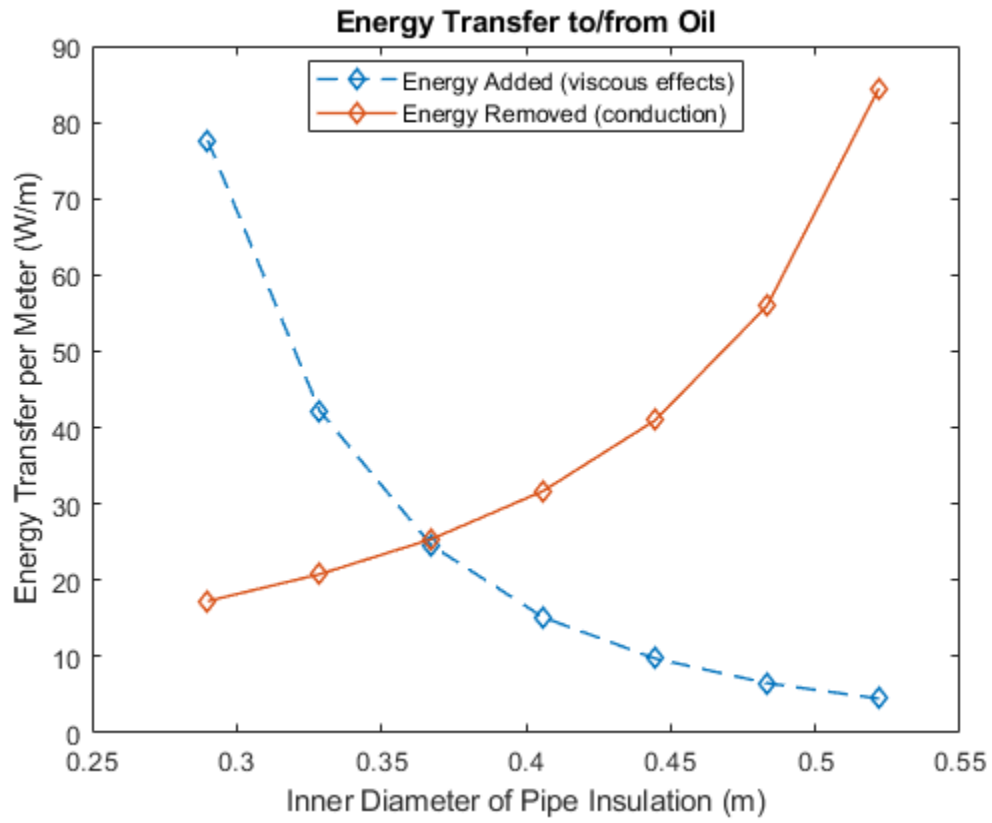
**Fluid Properties**
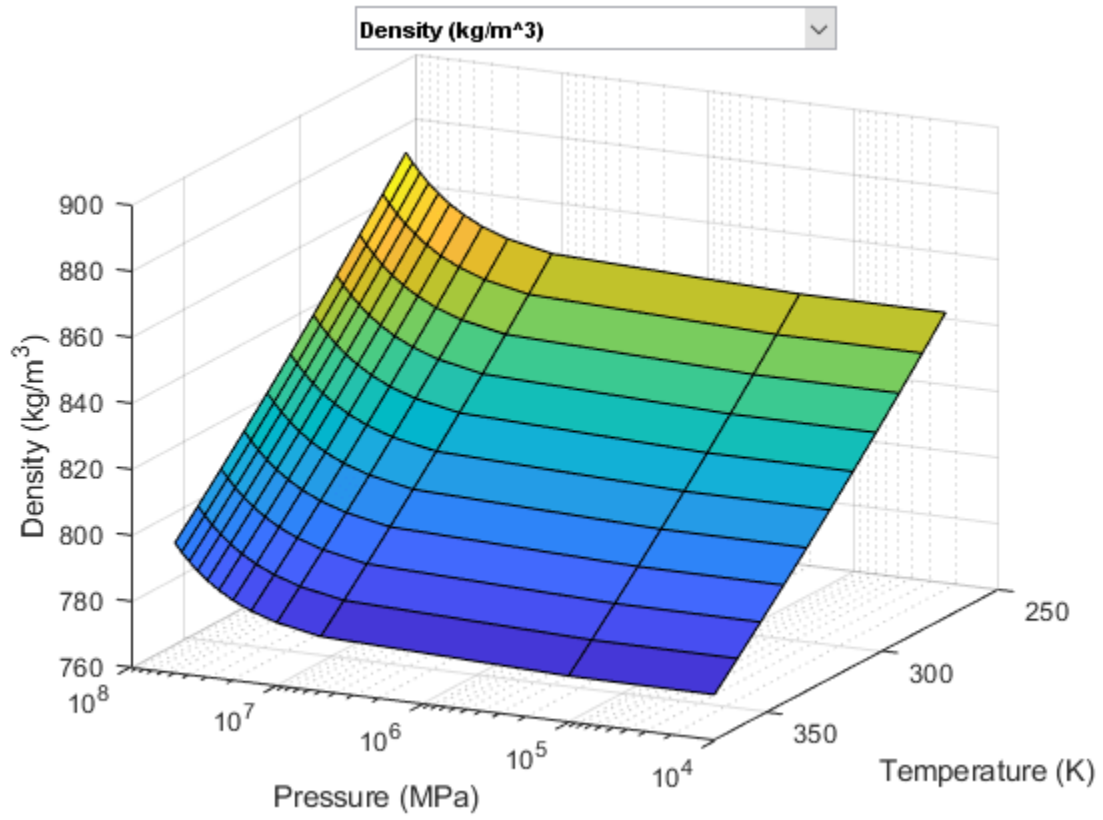
# Water Hammer Effect

This example model shows how the Thermal Liquid foundation library can be used to model water hammer in a long pipe. After slowly establishing a steady flow within the pipe by opening a valve accordingly, the same valve is shut within a few milliseconds. This triggers a water hammer effect. The valve is modeled using a Variable Local Restriction (TL) block and the pipe is divided into four segments using the Pipe (TL) block. Breaking the pipe into more segments increases fidelity at the expense of simulation performance.

The Pipe (TL) block can be configured to include or neglect dynamic compressibility and fluid inertia. Water hammer effect is reproduced in this model if the Valve Signal is set to Fast and both dynamic compressibility and inertia are enabled. See the documentation for the Pipe (TL) block for more details.

**Model**
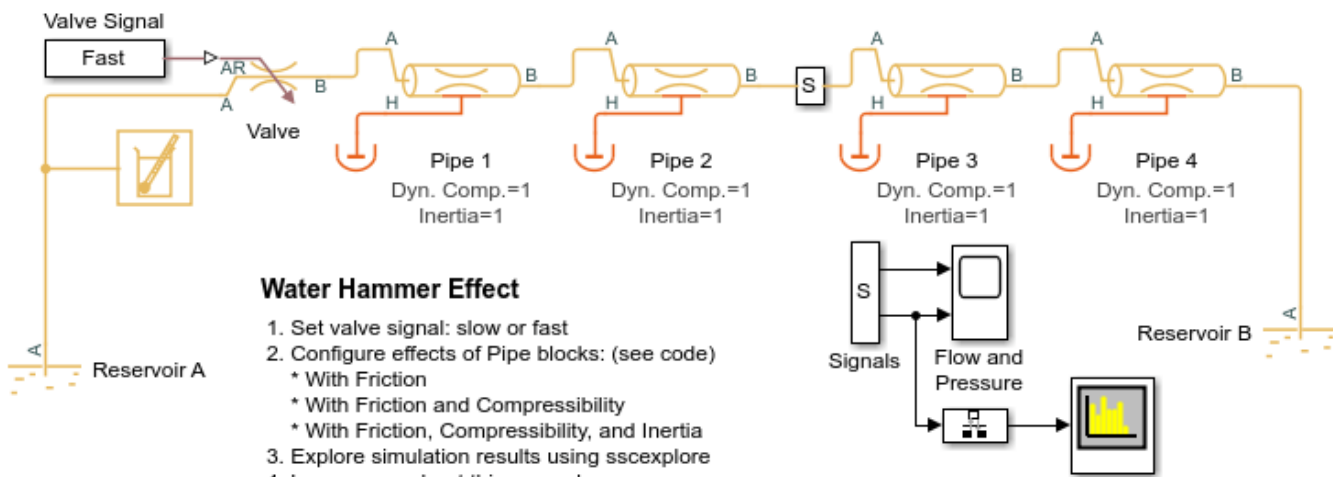
**Simulation Results from Scopes**
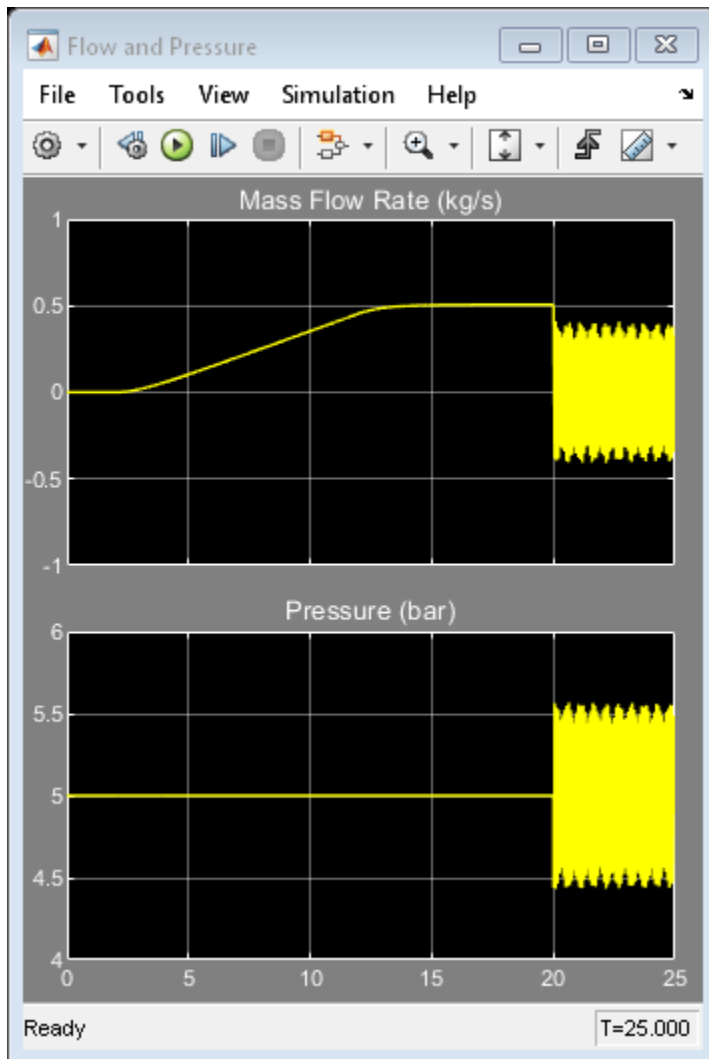
**Power Spectral Density**
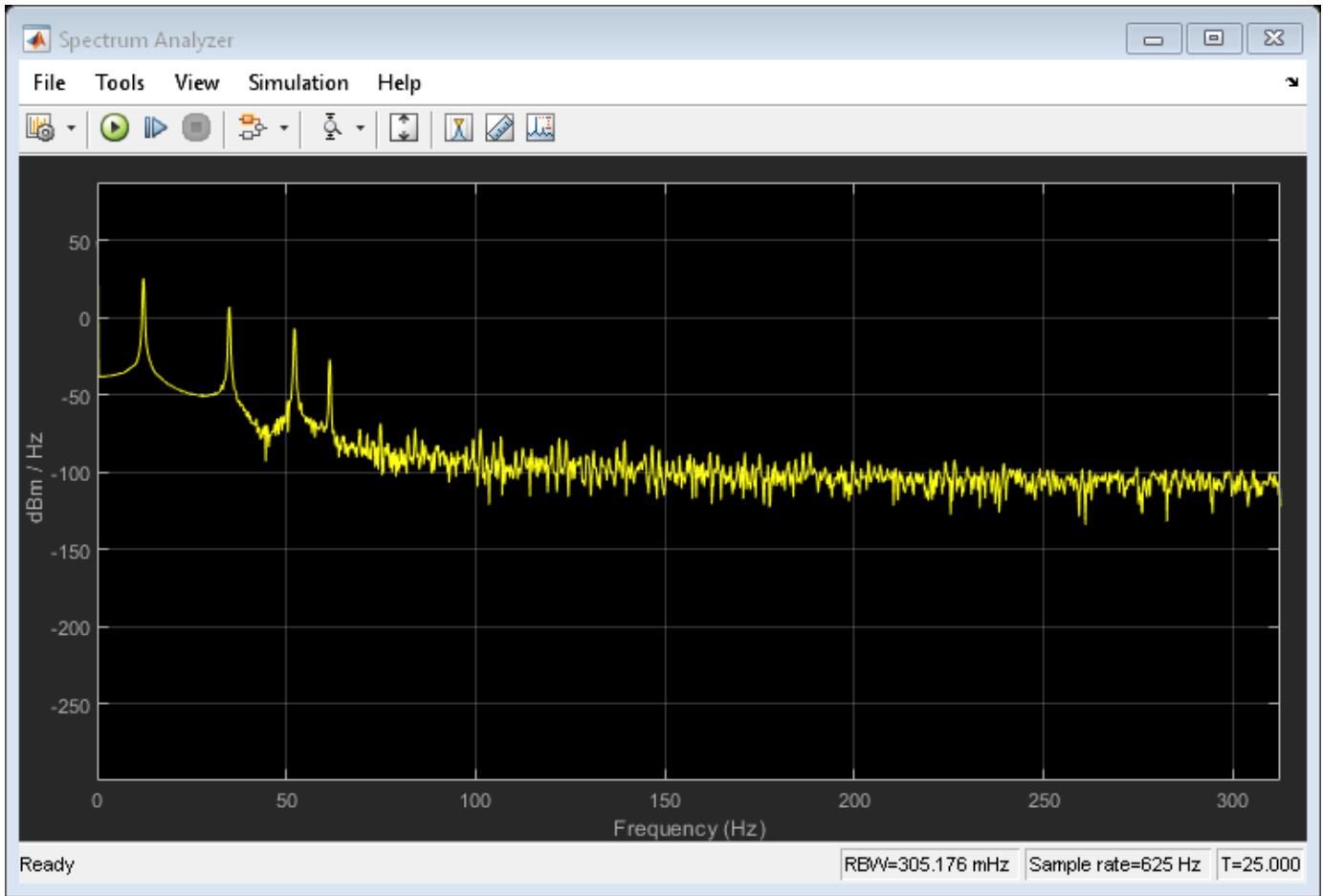
# Engine Cooling System

This example shows how to model a basic engine cooling system using custom thermal liquid blocks. A fixed-displacement pump drives water through the cooling circuit. Heat from the engine is absorbed by the water coolant and dissipated through the radiator. The system temperature is regulated by the thermostat, which diverts flow to the radiator only when the temperature is above a threshold.

The custom thermal liquid blocks include the Fixed-Displacement Pump, the Fluid Jacket, the Radiator, and the Thermostat. Click on the source code link on the block dialog to inspect the code and see how existing Thermal Liquid Library blocks can be modified to suit a specific application.

The Fluid Jacket and the Radiator are modifications of the Pipe (TL) block. These components represent an internal volume of liquid to model the effects of dynamic compressibility and thermal capacity using the mass and energy conservation equations. Default priorities for the pressure and temperature are set to high to provide initial conditions for the liquid state.

The Fixed-Displacement Pump is a modification of the Mass Flow Rate Source (TL) block. The Thermostat is a modification of the Local Restriction (TL) block. Both components are assumed to contain negligible volume of liquid. Therefore, they are assumed quasi-steady.

All four components inherit from foundation.thermal_liquid.two_port_dynamic or foundation.thermal_liquid.two_port_steady base classes, which implements common equations to calculate the energy flow rate based on a smoothed upwind method. This method allows energy to be convected downstream, enabling the proper propagation of information throughout the thermal liquid network.

**Model**
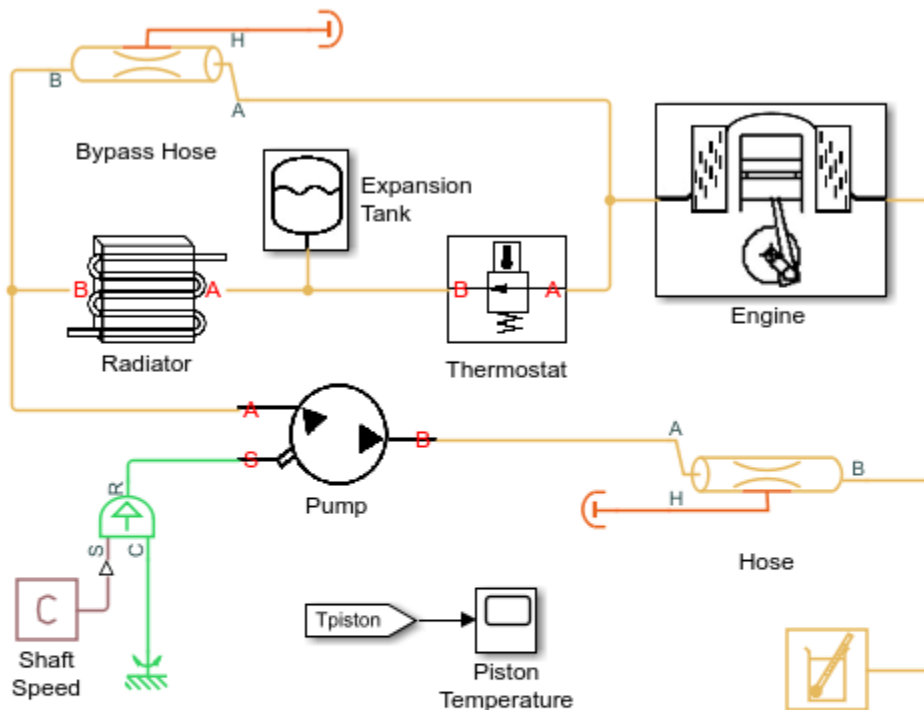


## Engine Cooling System

1. Plot temperature and mass flow in system (see code)
2. Plot density of coolant in system (see code)
3. Plot fluid properties (see code)
4. Explore simulation results using sscexplore
5. Open the cooling components library
6. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Engine Subsystem**
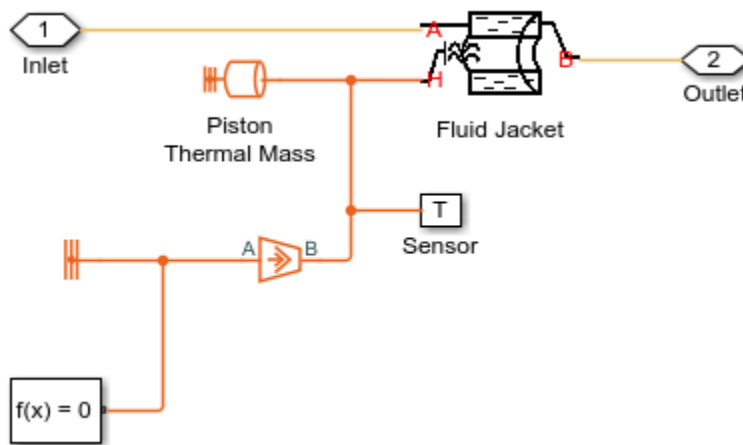


**Simulation Results from Simscape Logging**

These plots show the effect of opening the thermostat in the engine cooling system. The temperature of the piston climbs steadily until the thermostat opens. At that point, the flow of coolant through the radiator climbs sharply and the flow of coolant through the bypass hose decreases. Because coolant passing through the radiator releases heat to the atmosphere, the piston temperature rises more slowly.
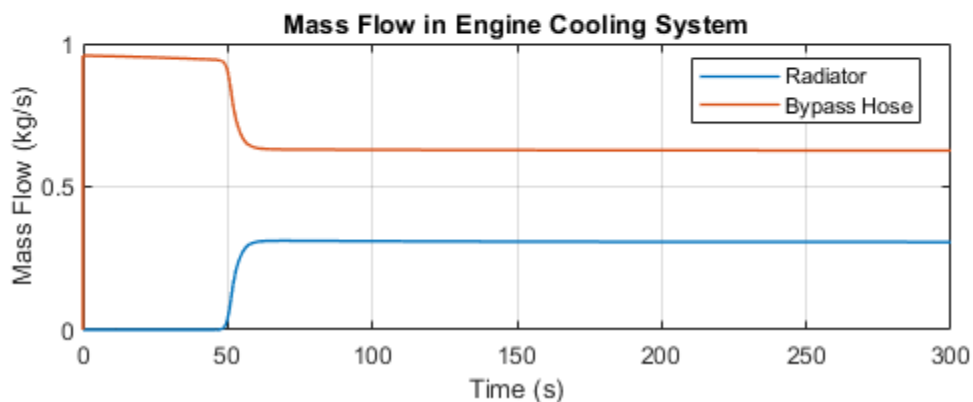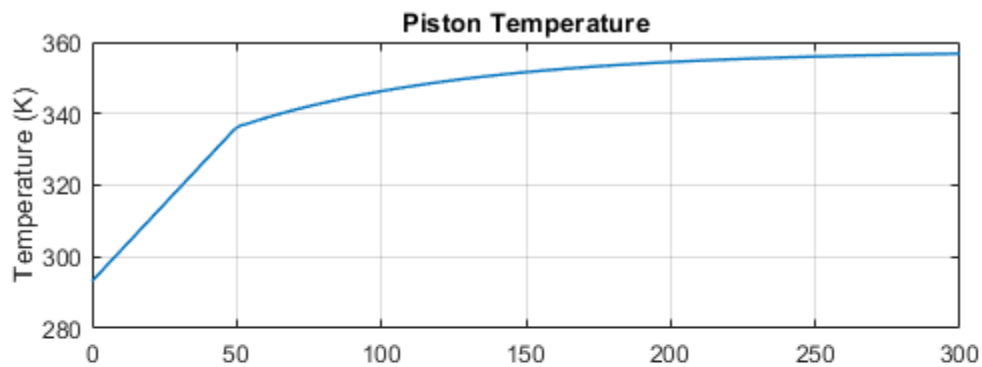
This plot shows the density of the coolant at different locations in the cooling system over time. The density of the coolant varies throughout the network based on the local temperature and pressure.
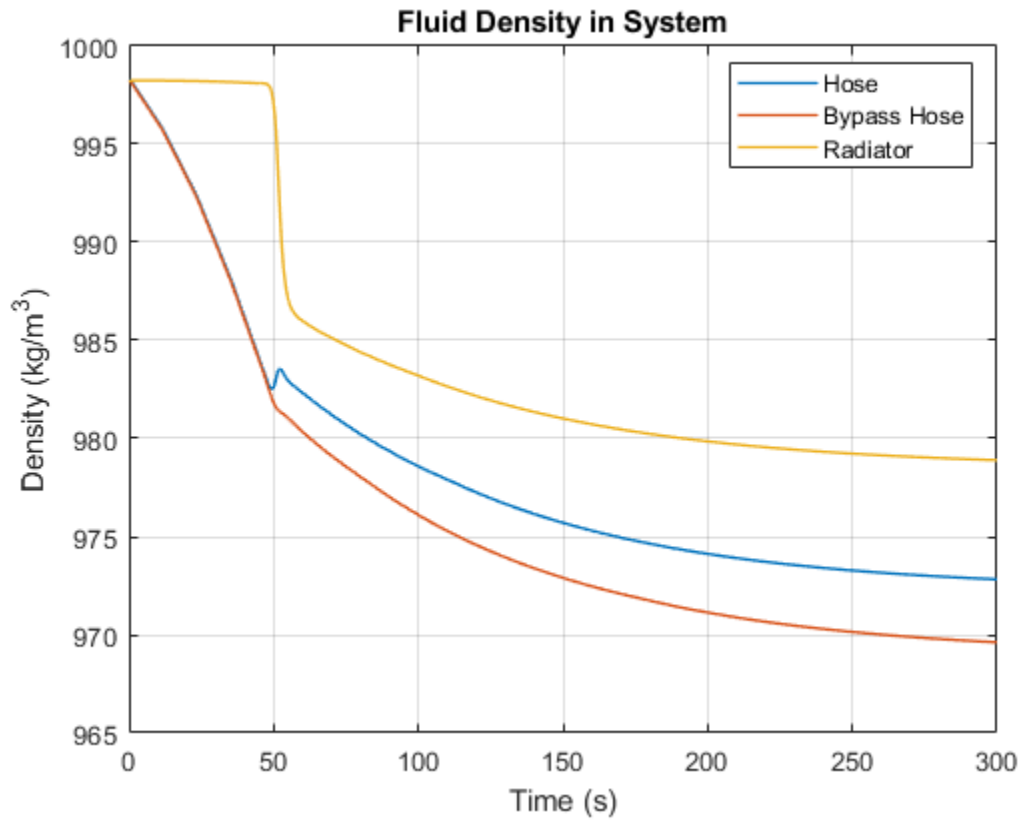
**Fluid Properties**
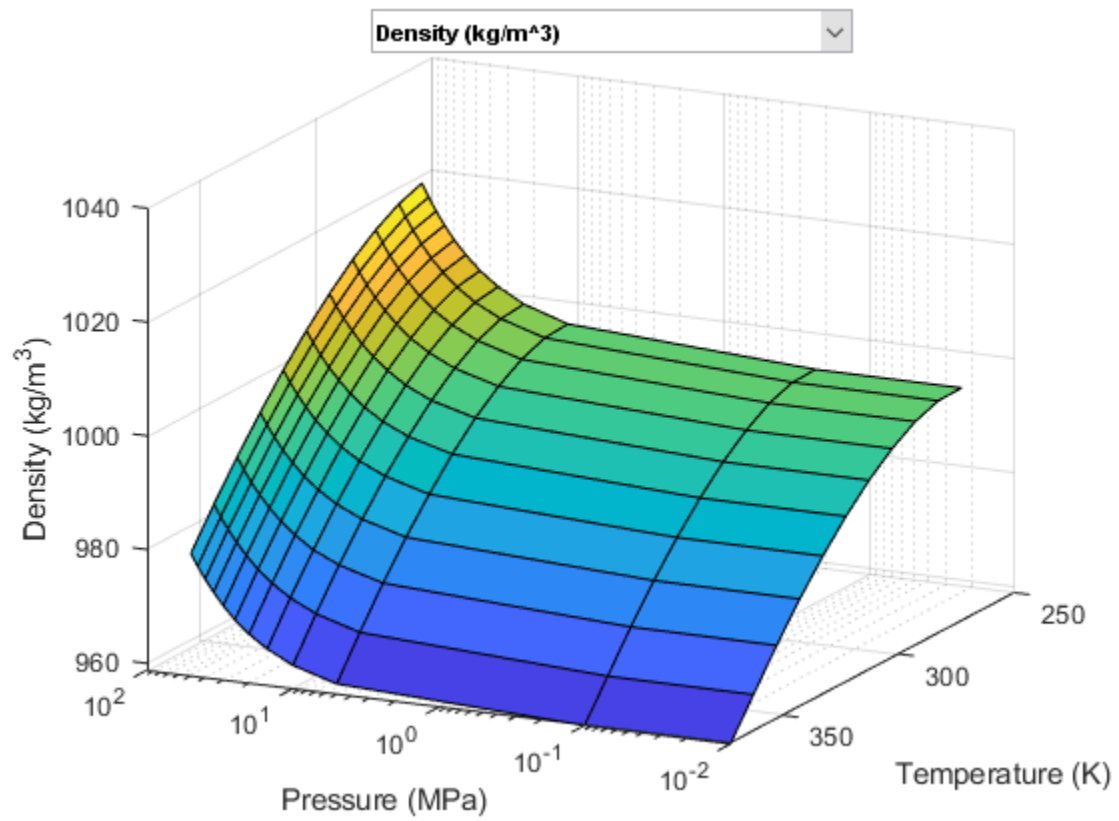
# Pneumatic Actuation Circuit

This example shows how the Foundation Library gas components can be used to model a controlled pneumatic actuator. The Directional Valve is a masked subsystem created from Variable Local Restriction (G) blocks to model the opening and closing of the flow paths. The Double-Acting Actuator is a masked subsystem created from Translational Mechanical Converter (G) blocks to model the interface between the gas network and the mechanical translational network.

During the simulation, the valve spool moves to its maximum positive displacement, causing the actuator to extend to its maximum stroke. The valve spool is then centered and the load is held. Next, the valve spool moves to its maximum negative displacement, causing the actuator to retract to its minimum stroke. The valve spool is then centered and the load is held. Thermal convection between the pipes and the environment allows the circuit to dissipate heat from the work done by the pressure source.

**Model**
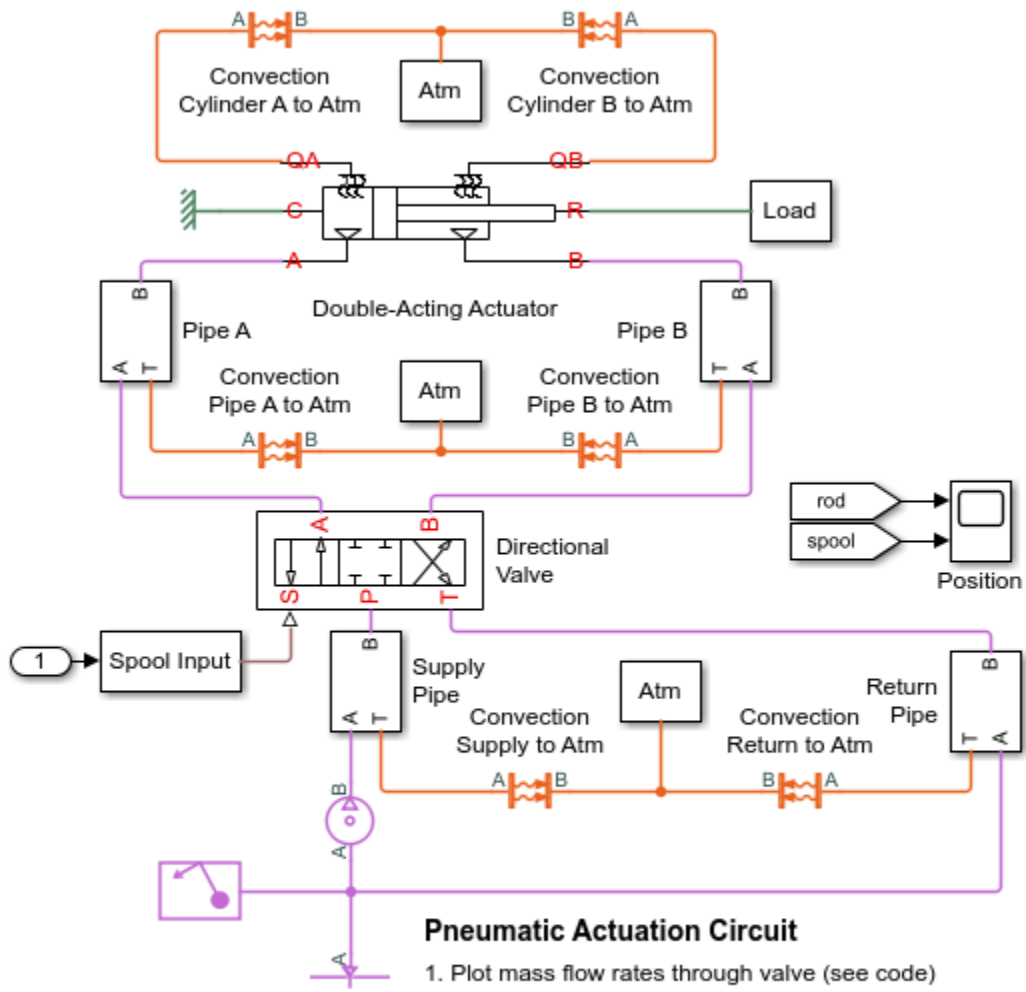


**Pneumatic Actuation Circuit**

1. Plot mass flow rates through valve (see code)
2. Plot pressures and temperatures of actuator (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2016-2021 The MathWorks, Inc.

**Directional Valve Subsystem**
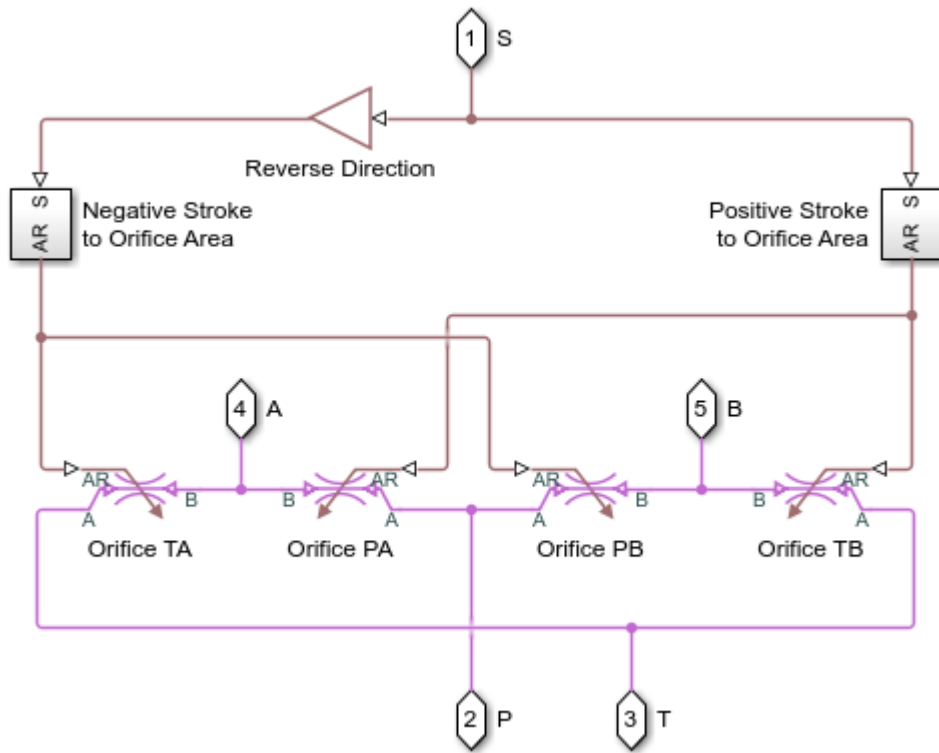
## Double-Acting Actuator Subsystem
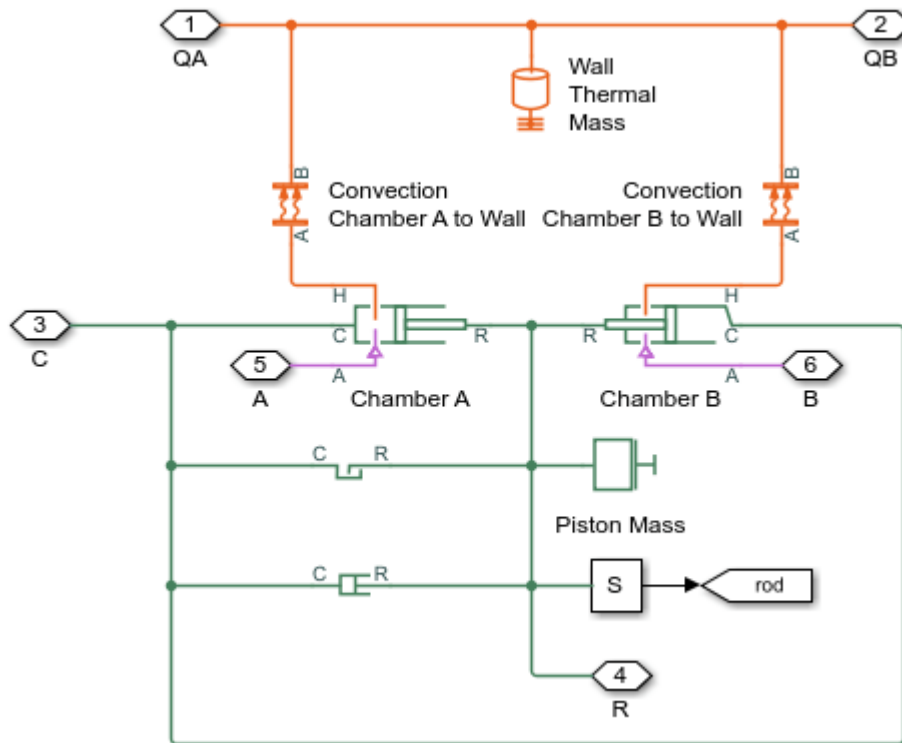


## Supply Pipe Subsystem
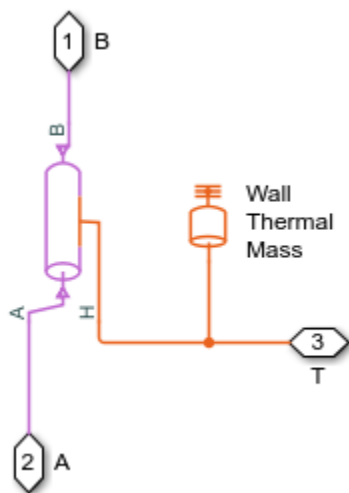
**Simulation Results from Scopes**
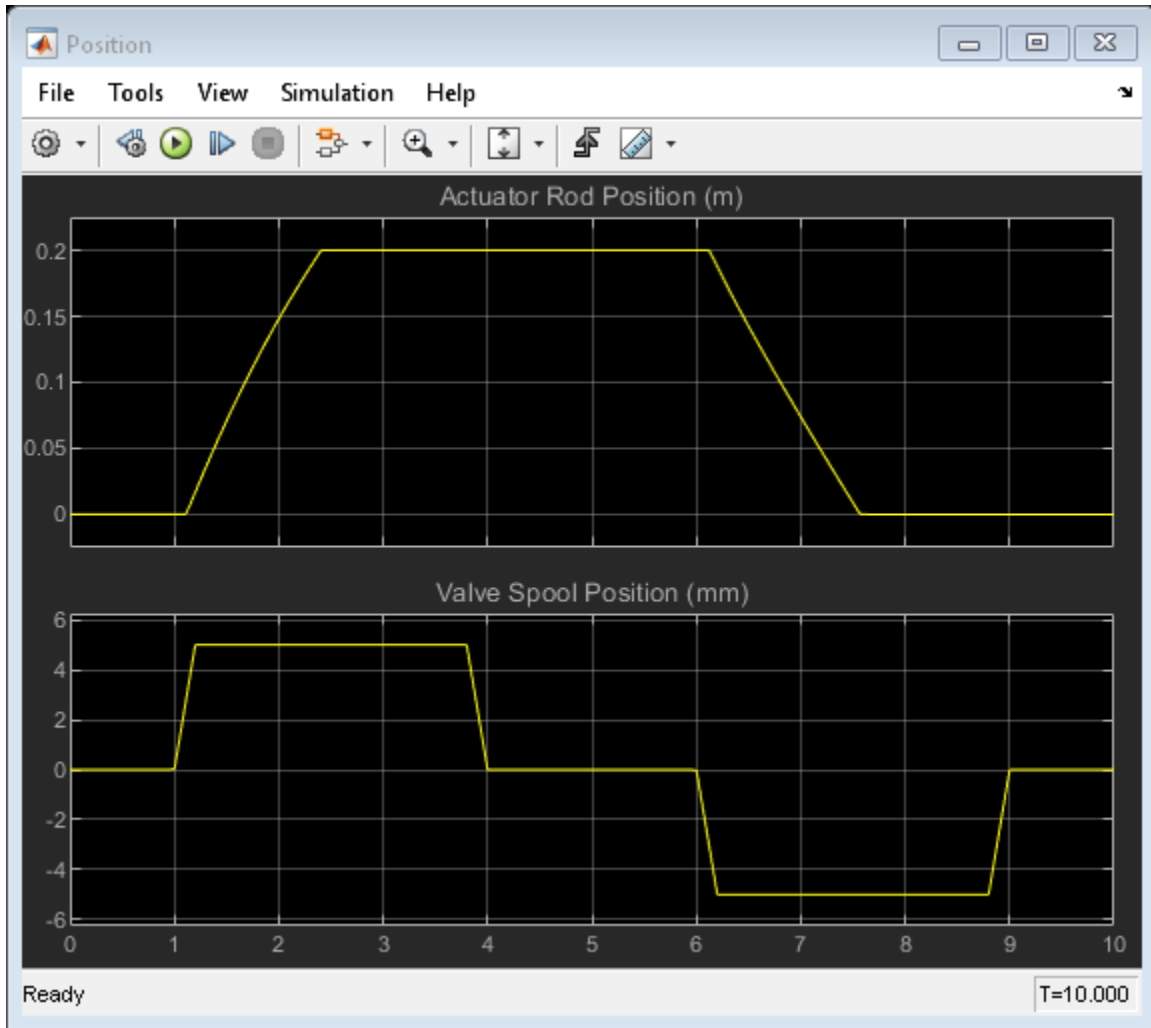


**Simulation Results from Simscape Logging**

When the valve spool position is positive, Orifice PA and Orifice TB open, allowing gas to flow from port P to port A and from port B to port T. When the valve spool position is negative, Orifice PB and Orifice TA open, allowing as to flow from port P to port B and from port A to port T.
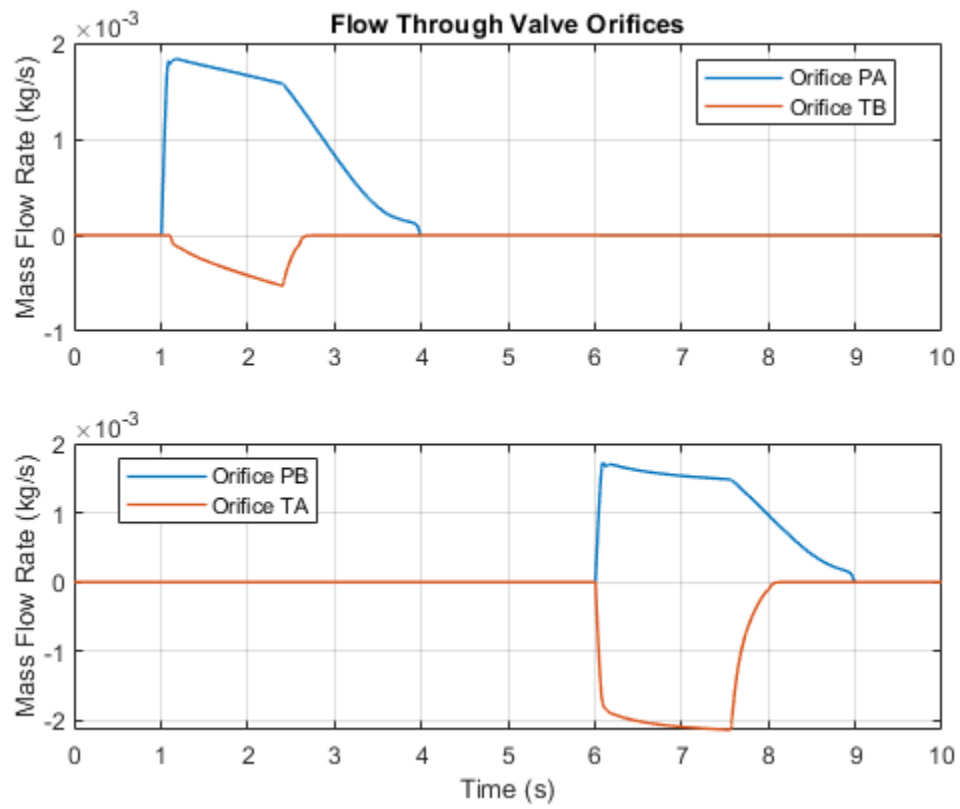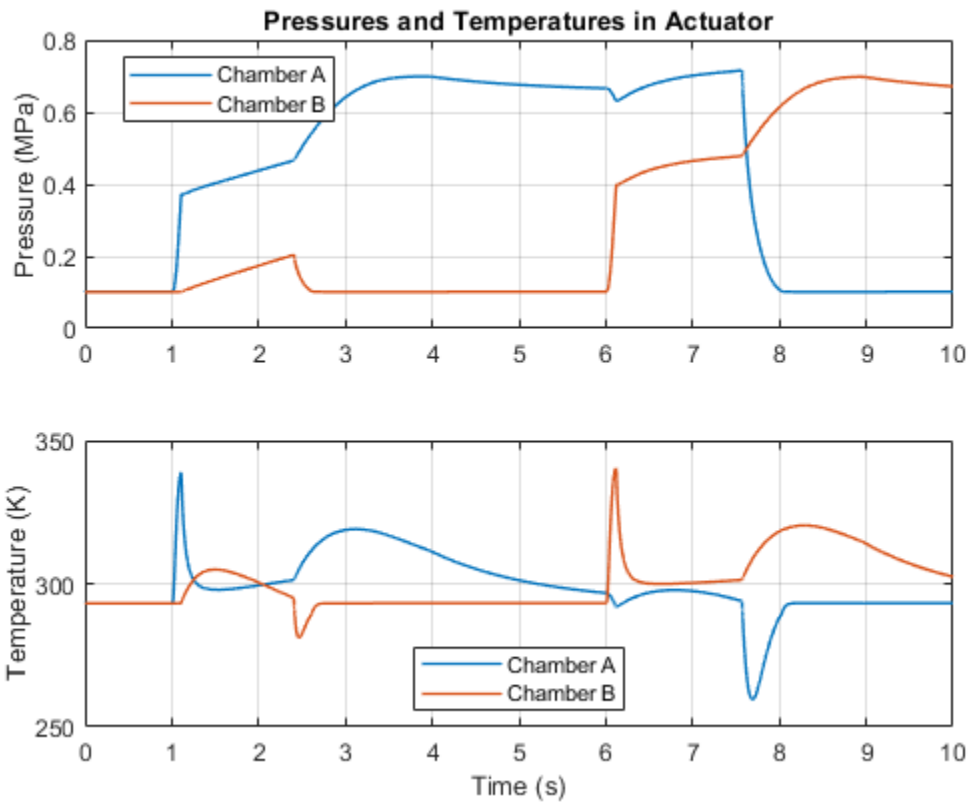
# Pneumatic Motor Circuit

This example shows how a pneumatic vane motor can be modeled using the Simscape™ language. The Pneumatic Motor component is built using the Simscape Foundation gas domain. It inherits from the `foundation.gas.two_port_steady` base class, which contains common equations that implement the upwind energy flow rate and the gas properties at the ports. The Pneumatic Motor subclass implements equations that describe behaviors specific to the component, such as the motor torque and flow rate characteristics and the mass and energy balance. The Pneumatic Motor block is inserted into the model using the Simscape Component block without the need to generate a separate library.

The motor is controlled by the Directional Valve, which is a masked subsystem created from Variable Local Restriction (G) blocks to model the opening and closing of the flow paths. During the simulation, the valve spool moves to its maximum positive displacement, causing the motor to spin up in the positive direction. The valve spool is then centered and the motor brakes back to zero speed. Next, the valve spool moves to its maximum negative displacement, causing the motor to spin up in the negative direction. The valve spool is then centered and the motor again brakes back to zero speed.

**Model**



**Pneumatic Motor Circuit**

1. Plot motor characteristics (see code)
2. Plot power and torque delivered to load (see code)
3. Plot pressures and flow rate in motor (see code)
4. See motor code referenced by Simscape Component
5. Explore simulation results using sscexplore
6. Learn more about this example

Copyright 2016-2021 The MathWorks, Inc.

**Directional Valve Subsystem**
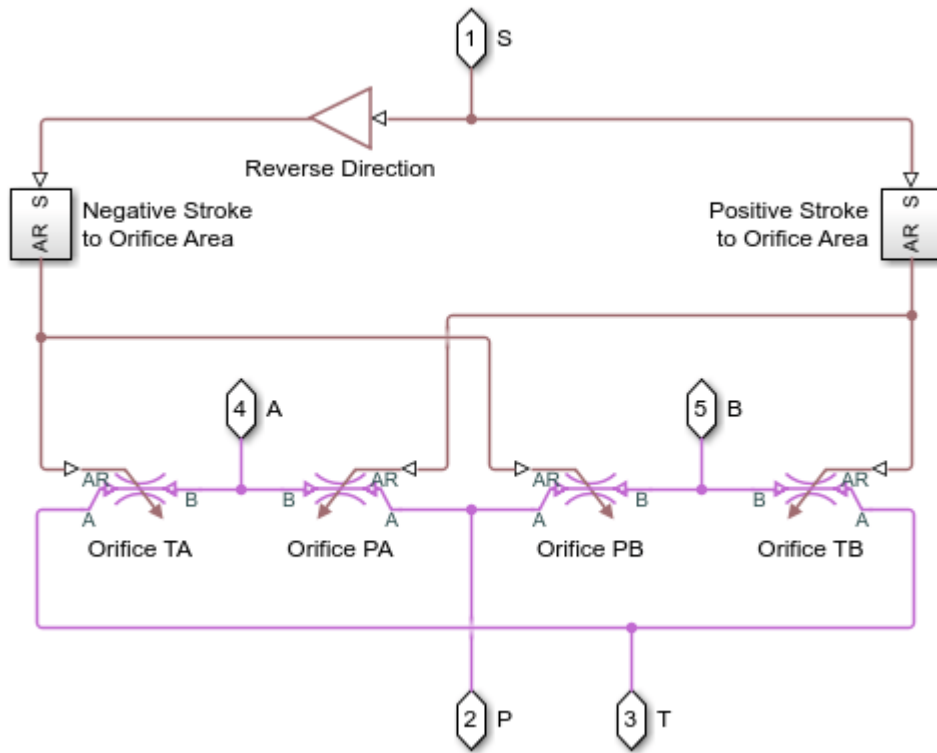


**Supply Pipe Subsystem**
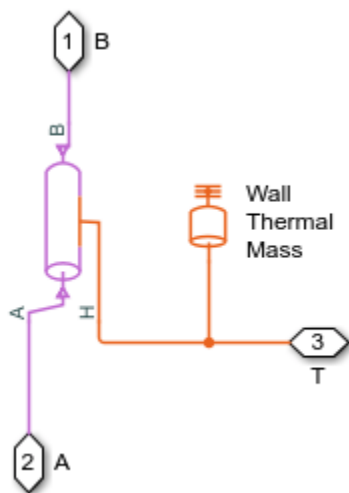
**Simulation Results from Scopes**
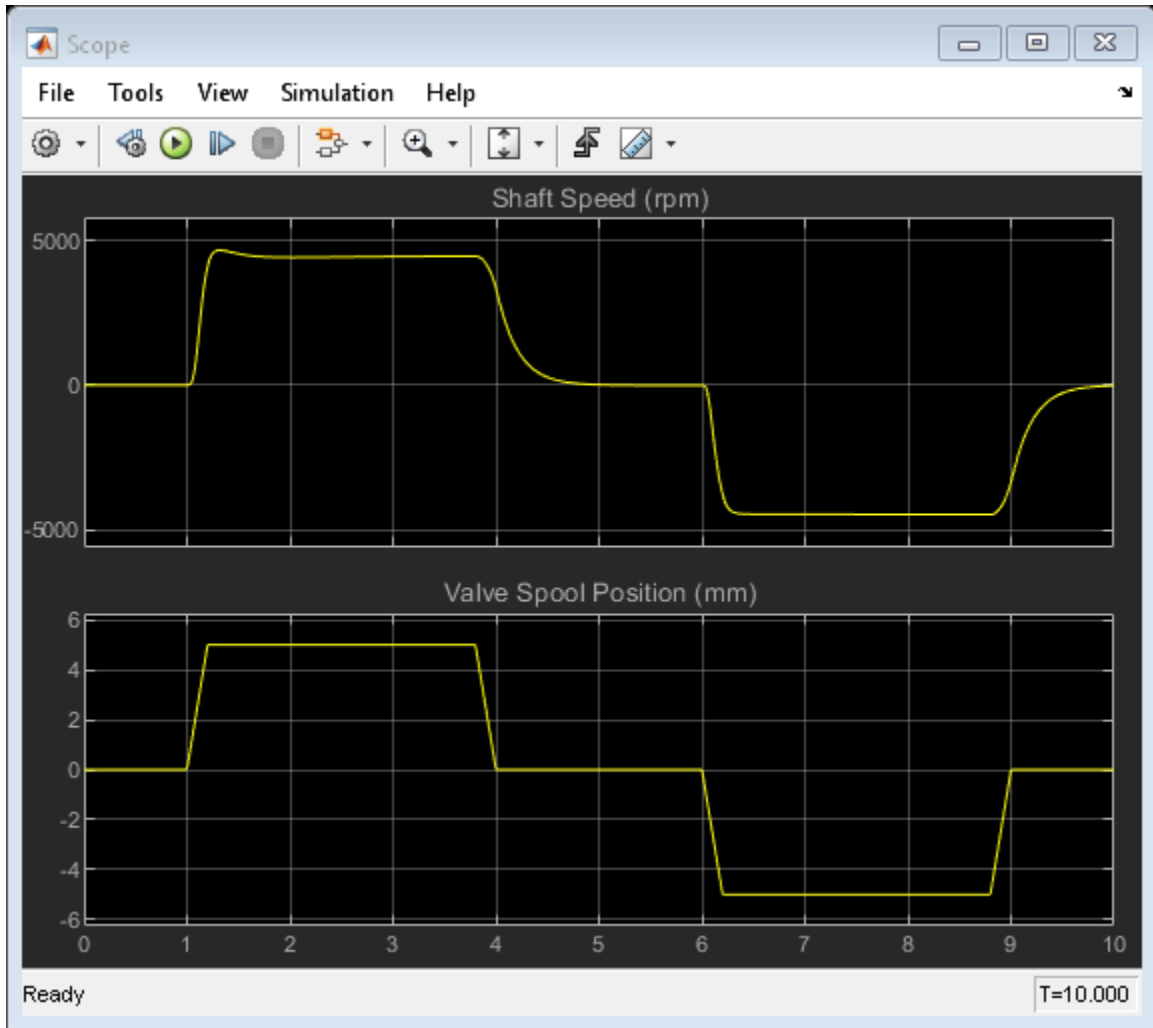
**Plot Motor Characteristics**
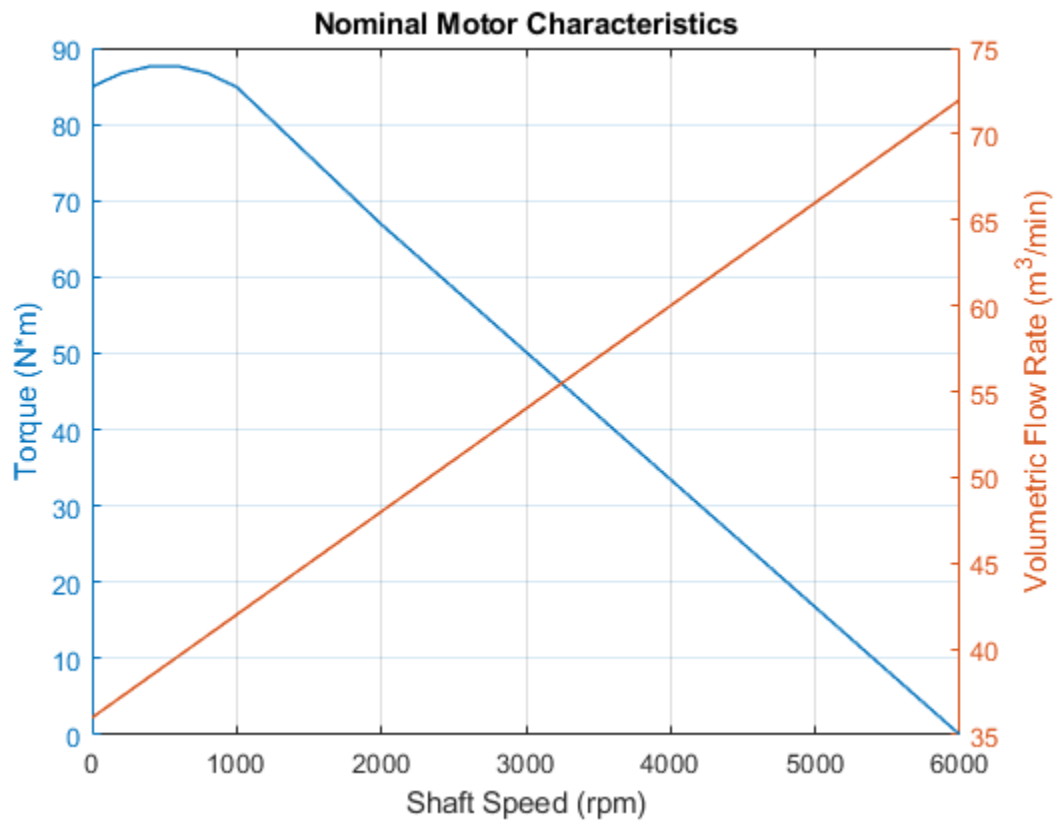
**Simulation Results from Simscape Logging**
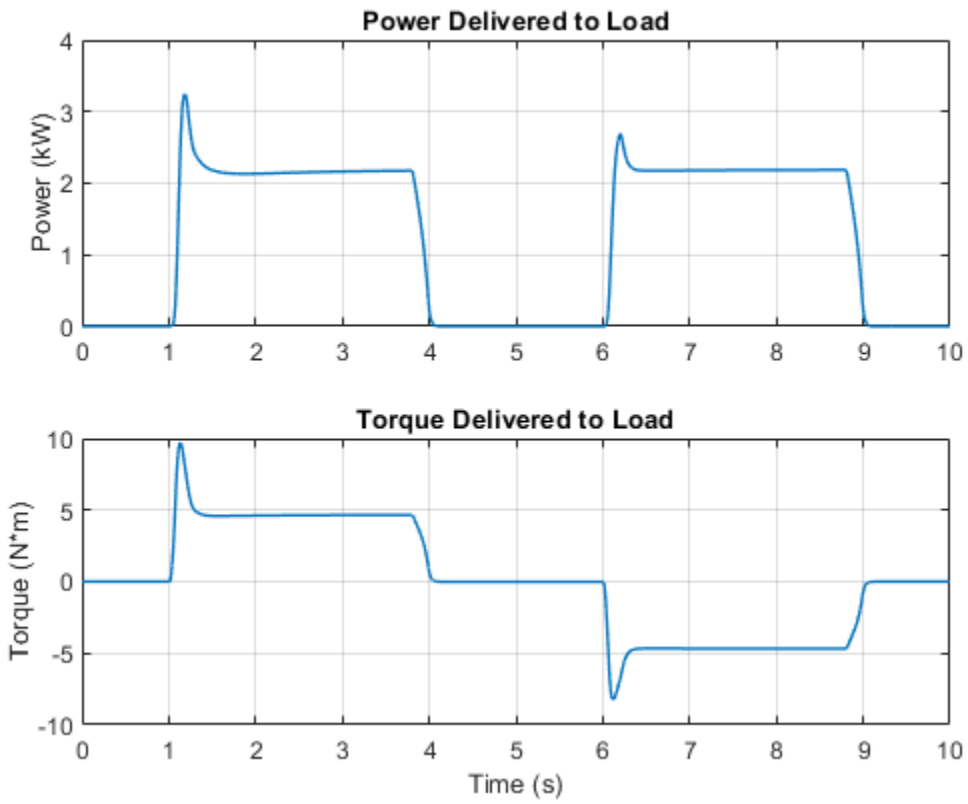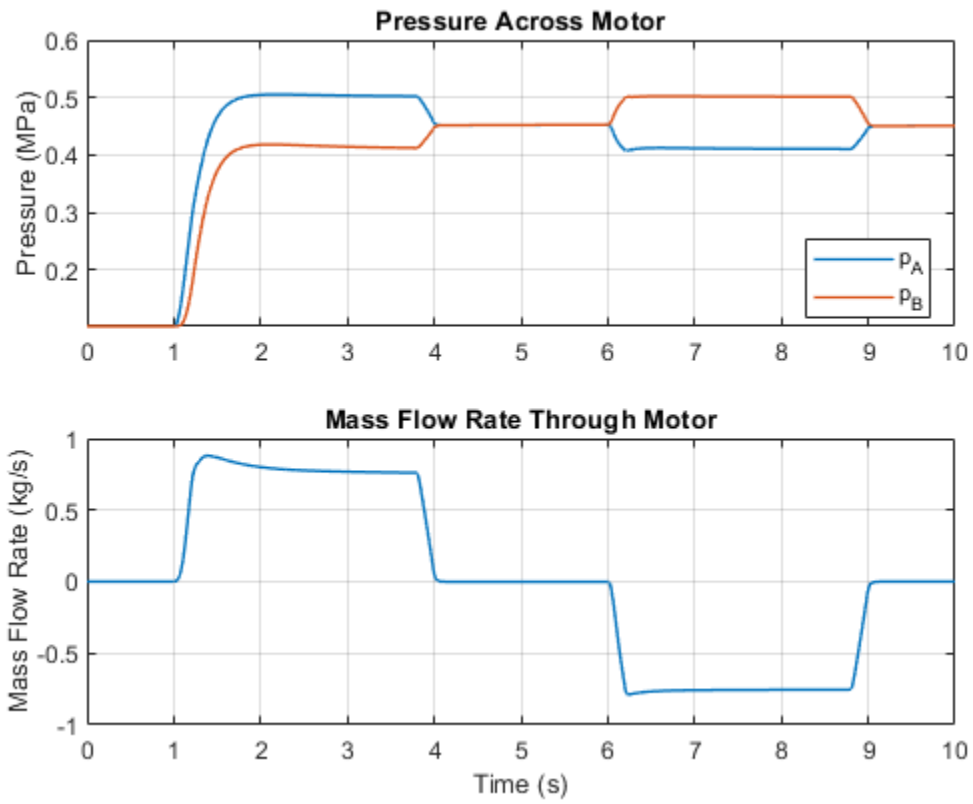
# Choked Flow in Gas Orifice

This example shows the choking behavior of a gas orifice modeled by the Local Restriction (G) block. The Controlled Reservoir (G) blocks are used to set up controlled pressure and temperature boundary conditions from the Signal Builder to test the gas orifice.

At the start, the flow is unchoked. Mass flow rate increases as the downstream pressure decreases. At 3.3 s, the flow becomes choked. The choked mass flow rate depends only on the upstream conditions, so further decrease in the downstream pressure does not cause a further increase in the choked mass flow rate. However, at 6 s, the upstream pressure increases, so the choked mass flow rate increases again.

**Model**
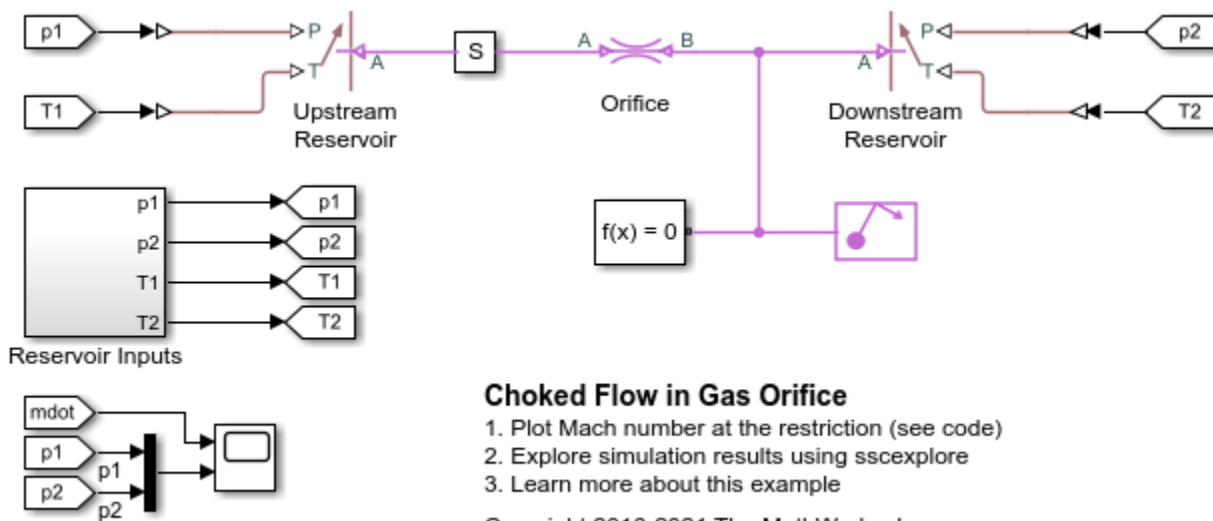


**Choked Flow in Gas Orifice**
1. Plot Mach number at the restriction (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2016-2021 The MathWorks, Inc.

**Simulation Results from Scopes**
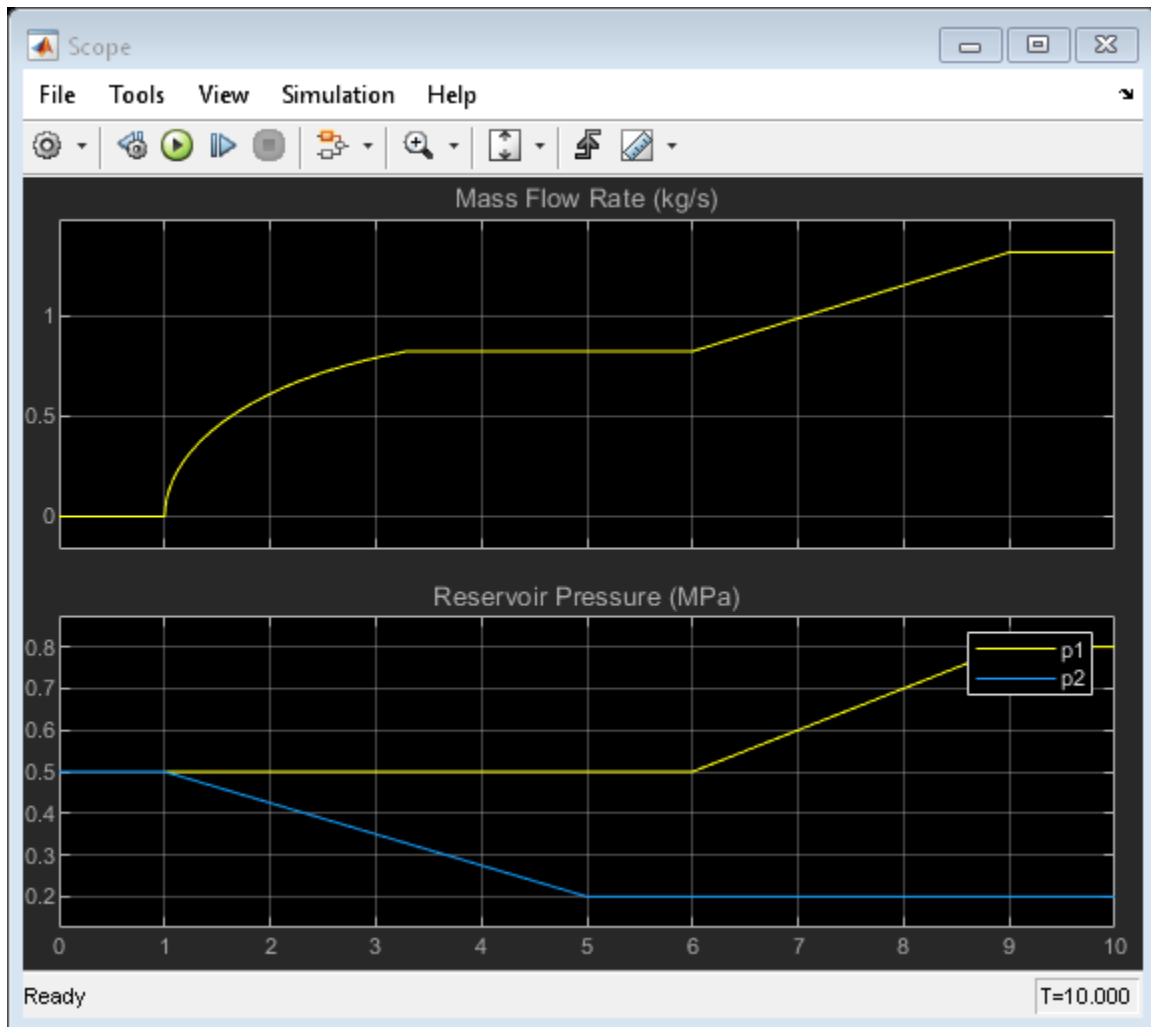
**Simulation Results from Simscape Logging**
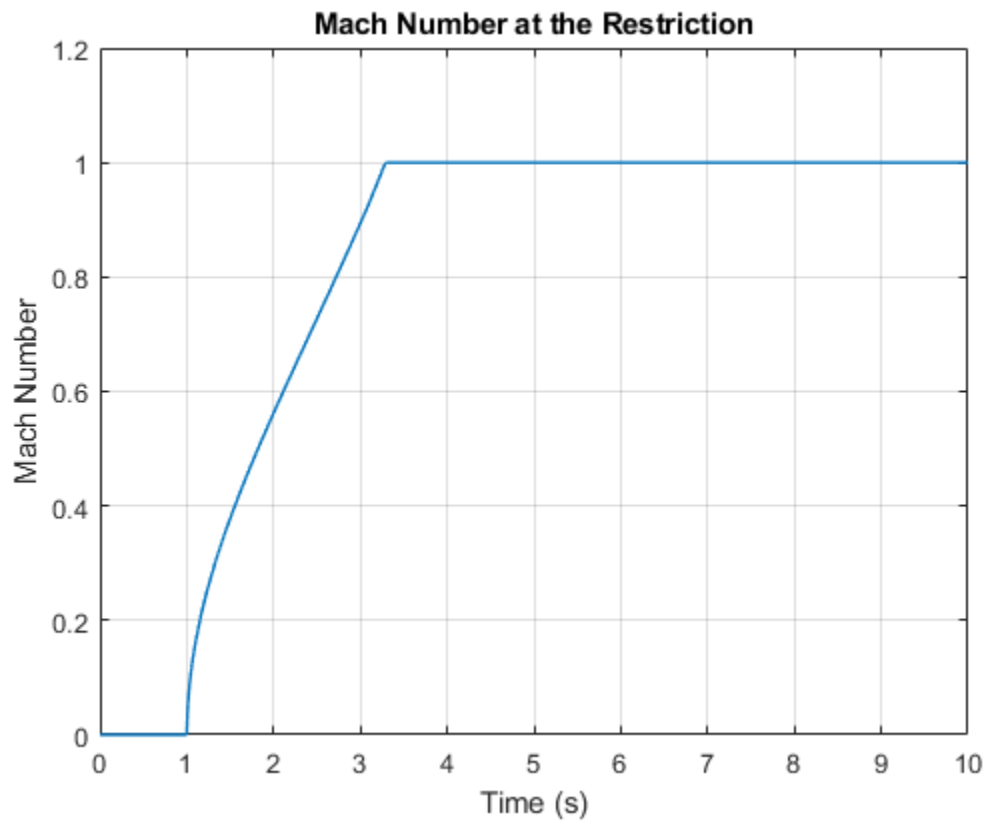
# Brayton Cycle (Gas Turbine) with Custom Components

This example models a gas turbine auxiliary power unit (APU) based on the Brayton Cycle. The Compressor and Turbine blocks are custom components based on the Simscape™ Foundation Gas Library. The power input to the system is represented by heat injection into the combustor; actual combustion chemistry is not modeled. A single shaft connects the compressor and the turbine so that the power from the turbine drives the compressor. The APU is a free turbine that further expands the exhaust stream to produce output power.

Three PID controllers regulate the shaft speed, the turbine inlet temperature, and the compressor surge margin. System inputs are defined for three scenarios: varying shaft speed, varying surge margin, and varying APU vane opening. Running the first scenario produces the typical operating line on the compressor map. Running the second and third scenarios show where the maximum power output and maximum global efficiency occurs.

### Model

**Controller Subsystem**

**Simulation Results from Scopes**

**Animation of Simscape Logging Results**

This figure shows an animation of the Brayton Cycle on a temperature-entropy diagram over time. The five cycle points on the figure correspond to the sensor measurements labeled "T,s 1" to "T,s 5", respectively.

**Simulation Results from Simscape Logging**



Compressor Map
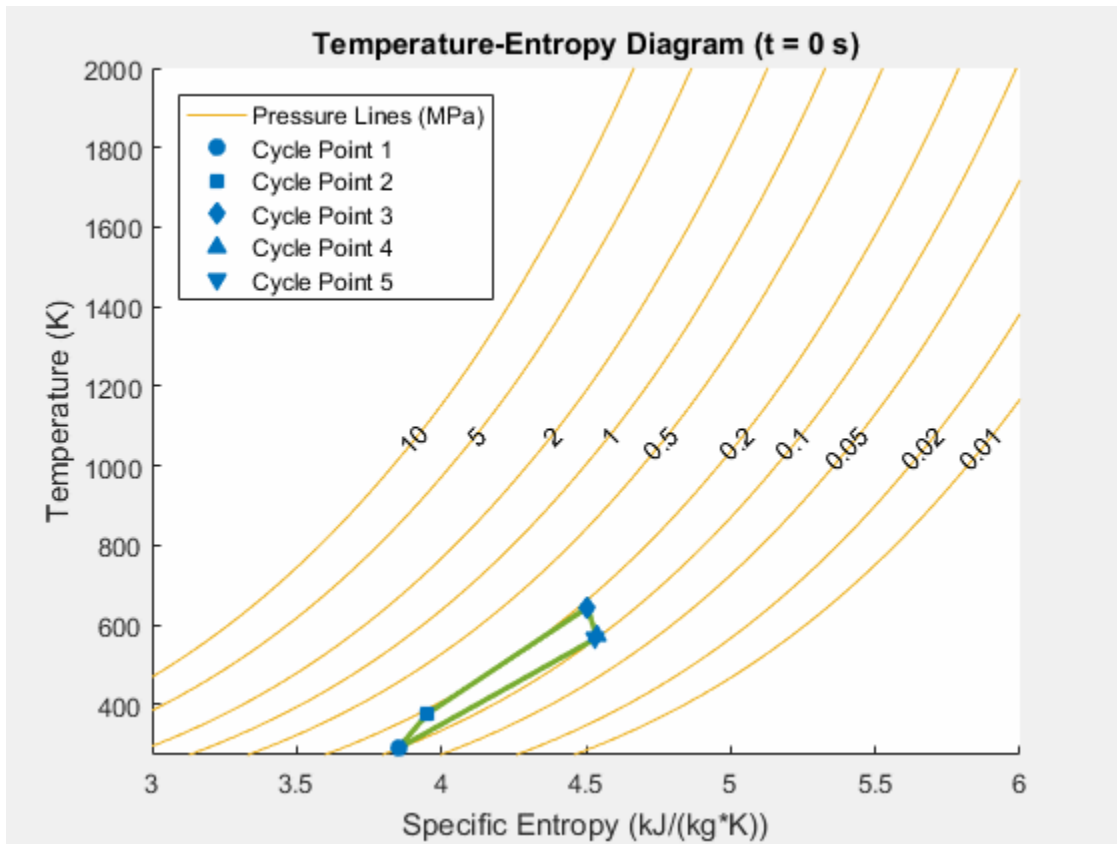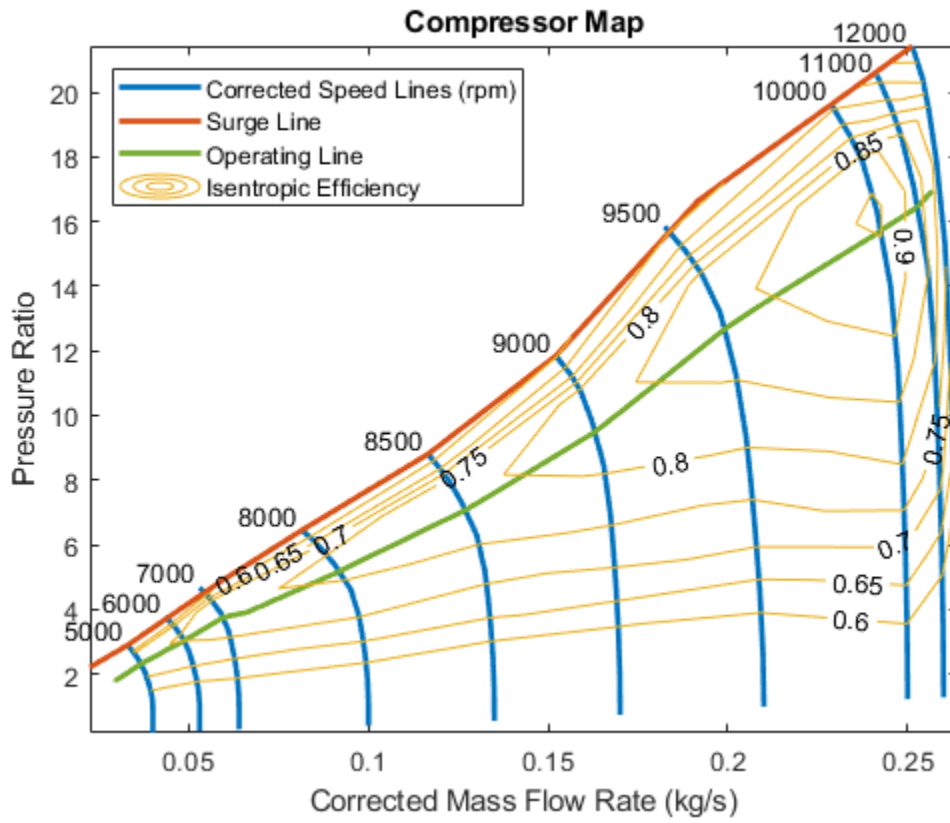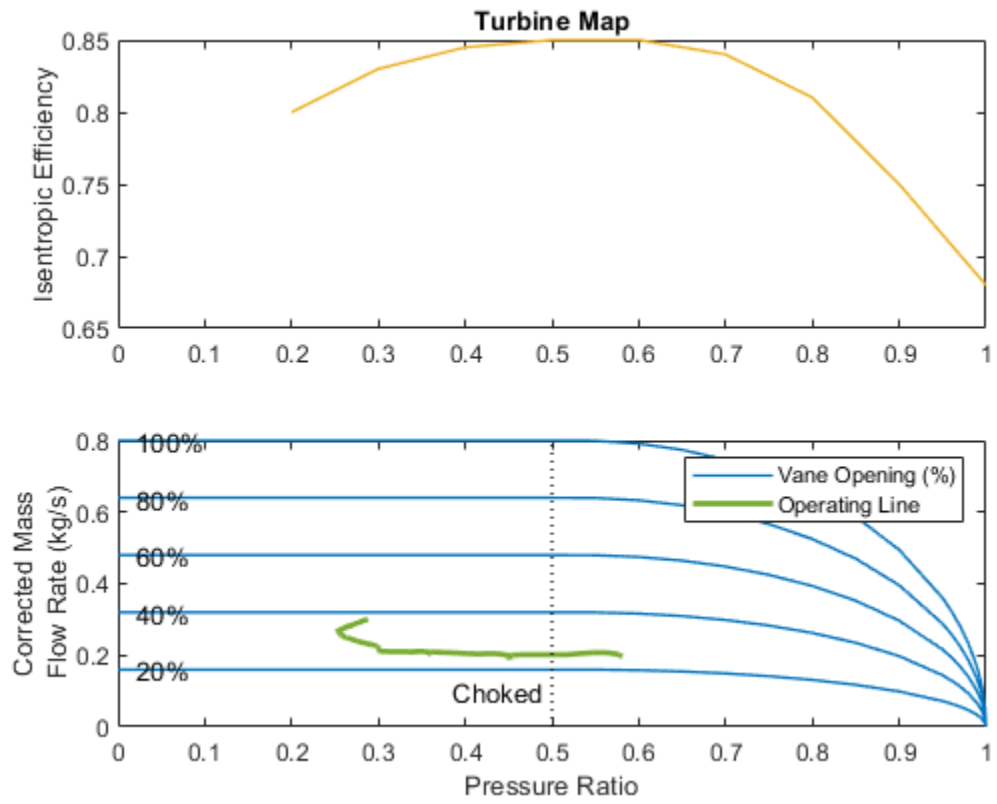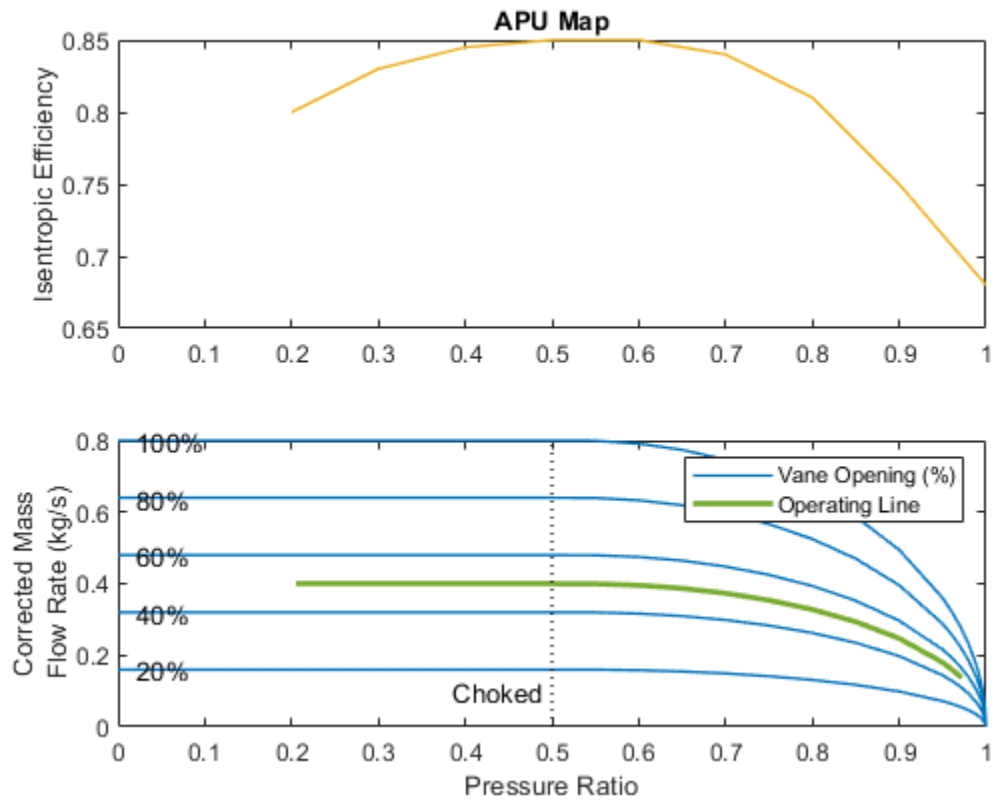
# Building Ventilation

This example models a ventilation circuit in a building. The air volume inside the building is divided into four zones. The ventilation unit blows cool air into Zone 1 and extracts air from Zone 3. The extracted air can be optionally recycled back into Zone 1. A door in Zone 4 can be opened to vent air out to the atmosphere.

### Model



**Building Ventilation**

1. Set System Inputs to vary Fan Control, Fan Temperature, or Disturbances
2. Plot mass flow rate between zones (see code)
3. Plot pressure in zones (see code)
4. Explore simulation results using sscexplore
5. Learn more about this example

Copyright 2016-2021 The MathWorks, Inc.
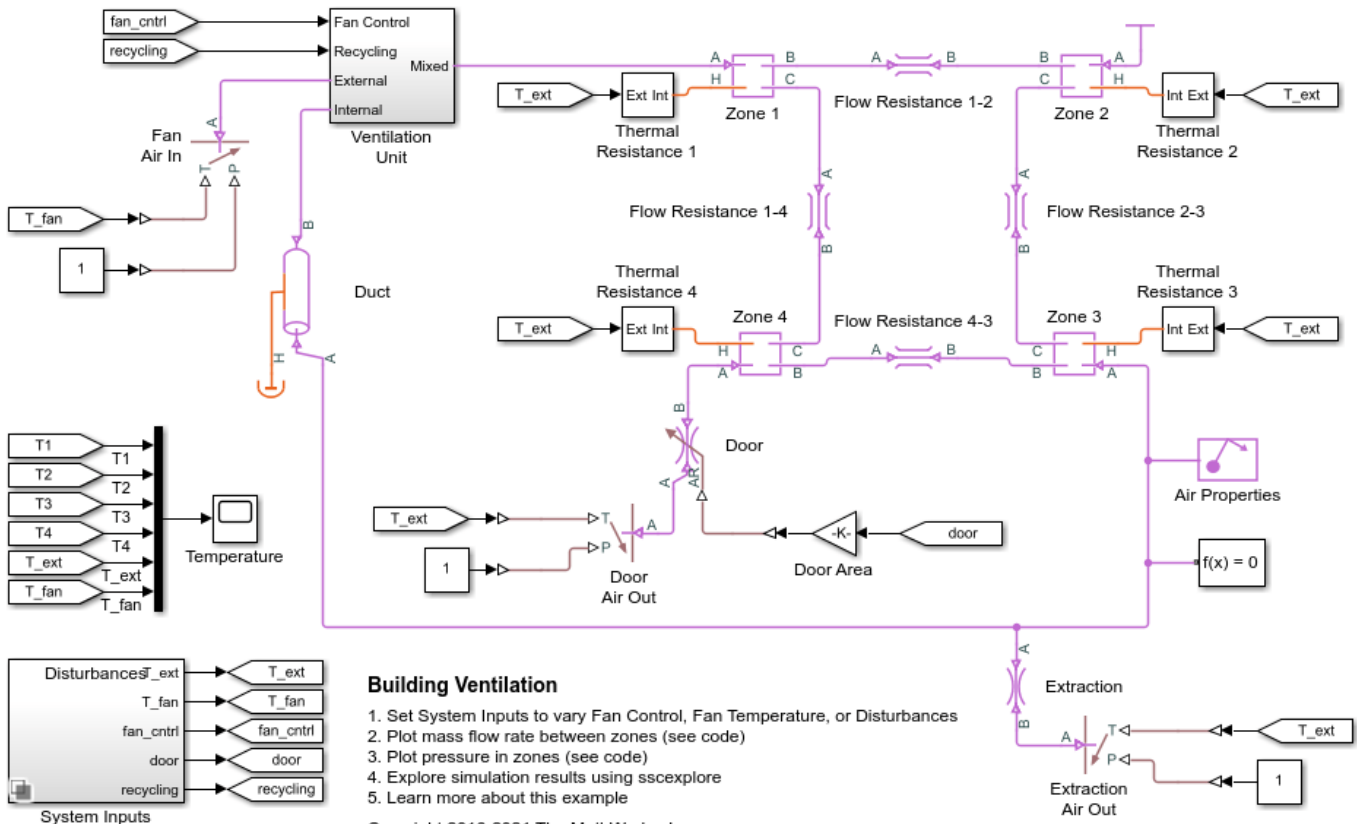
**Thermal Resistance 1 Subsystem**
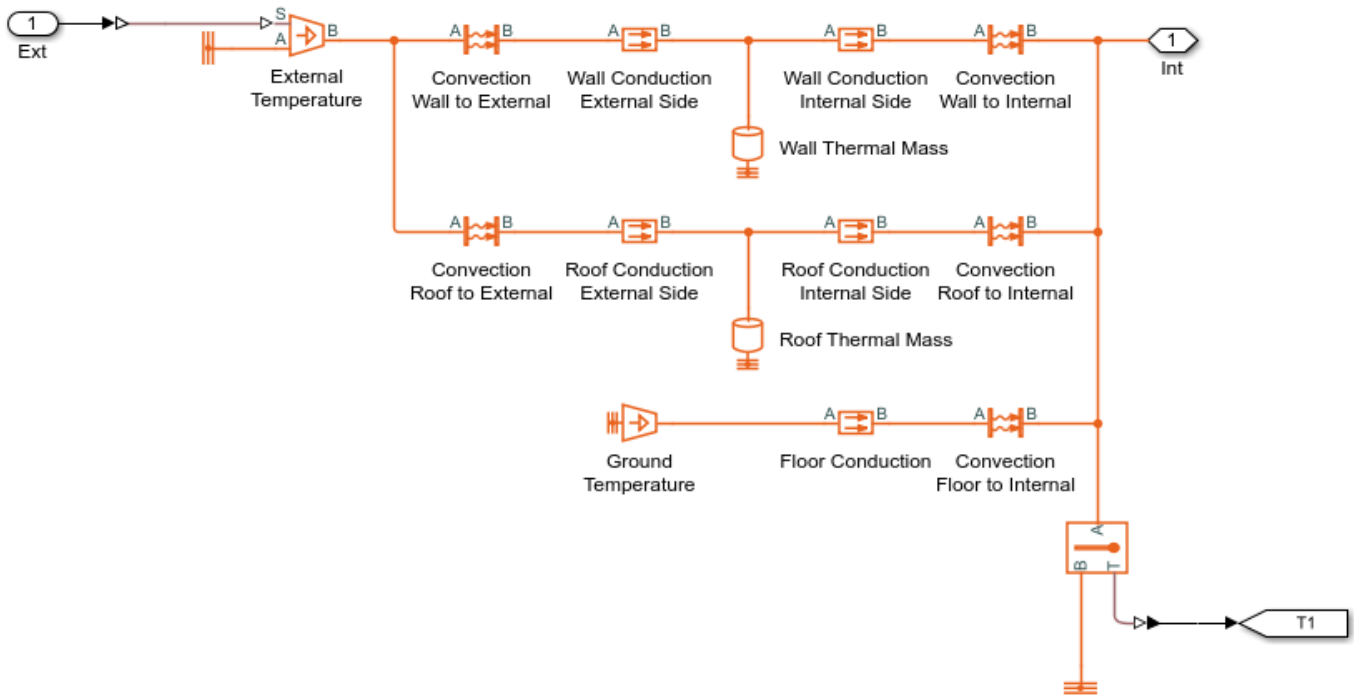


**Ventilation Unit Subsystem**
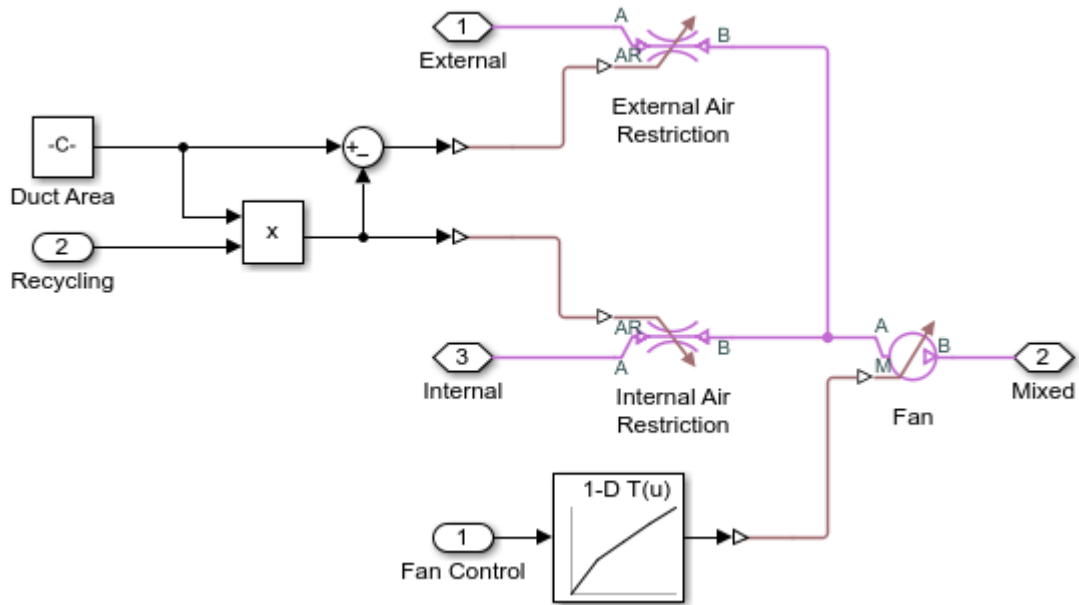
**Simulation Results from Scopes**
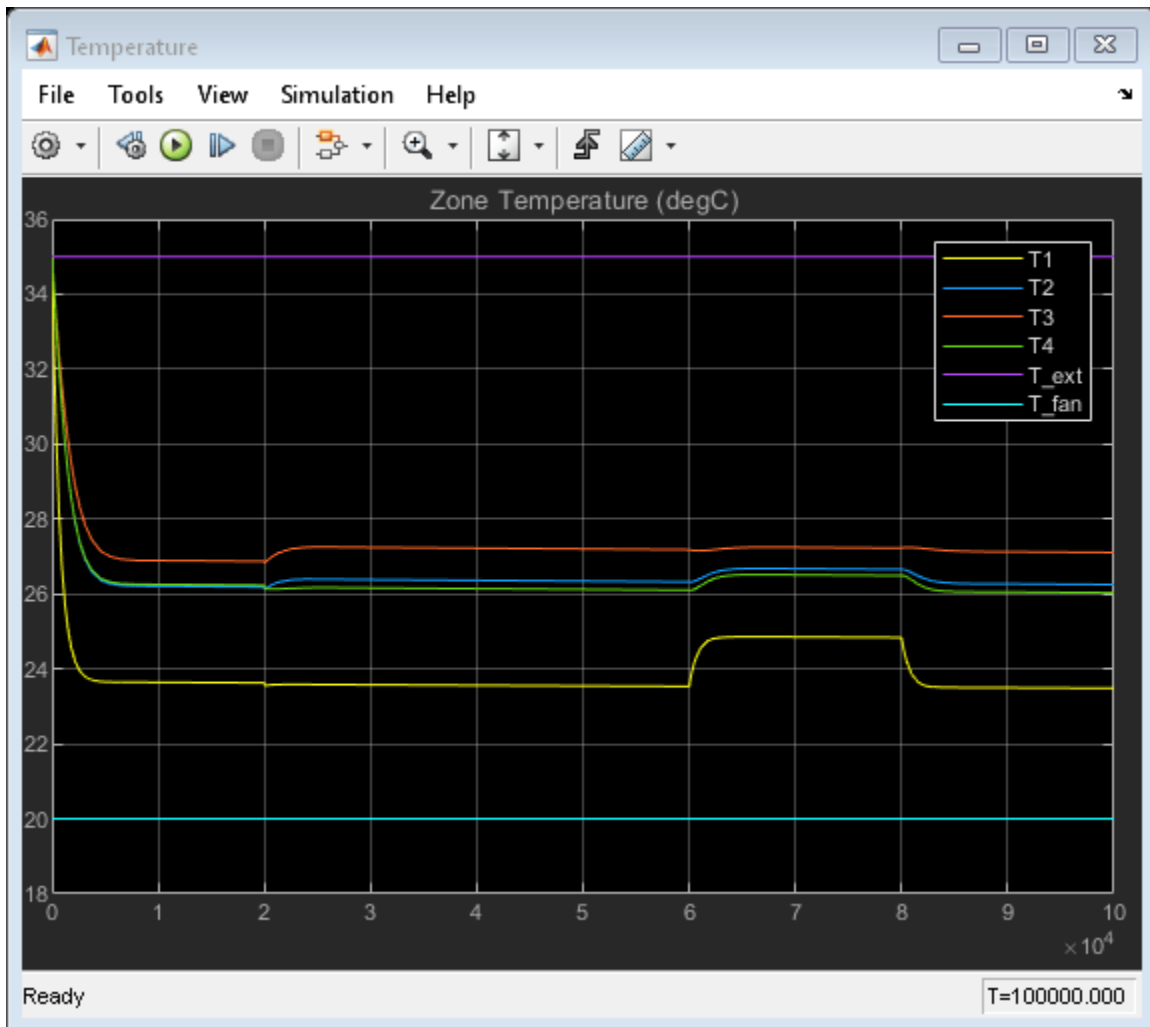
**Simulation Results from Simscape Logging**
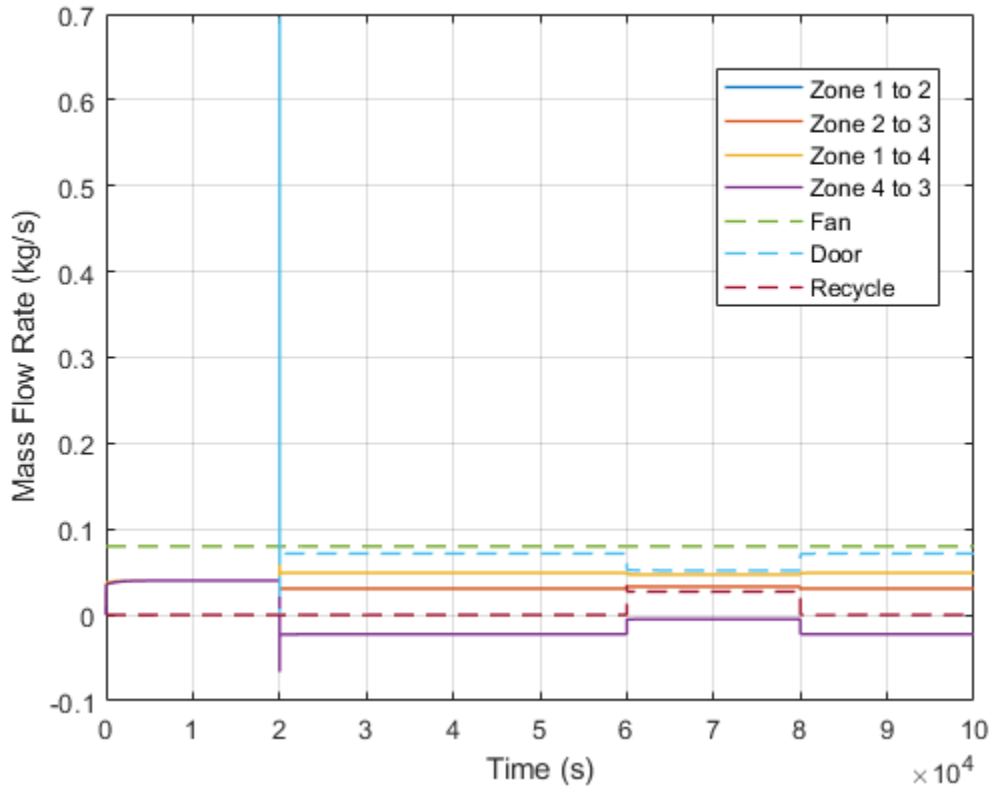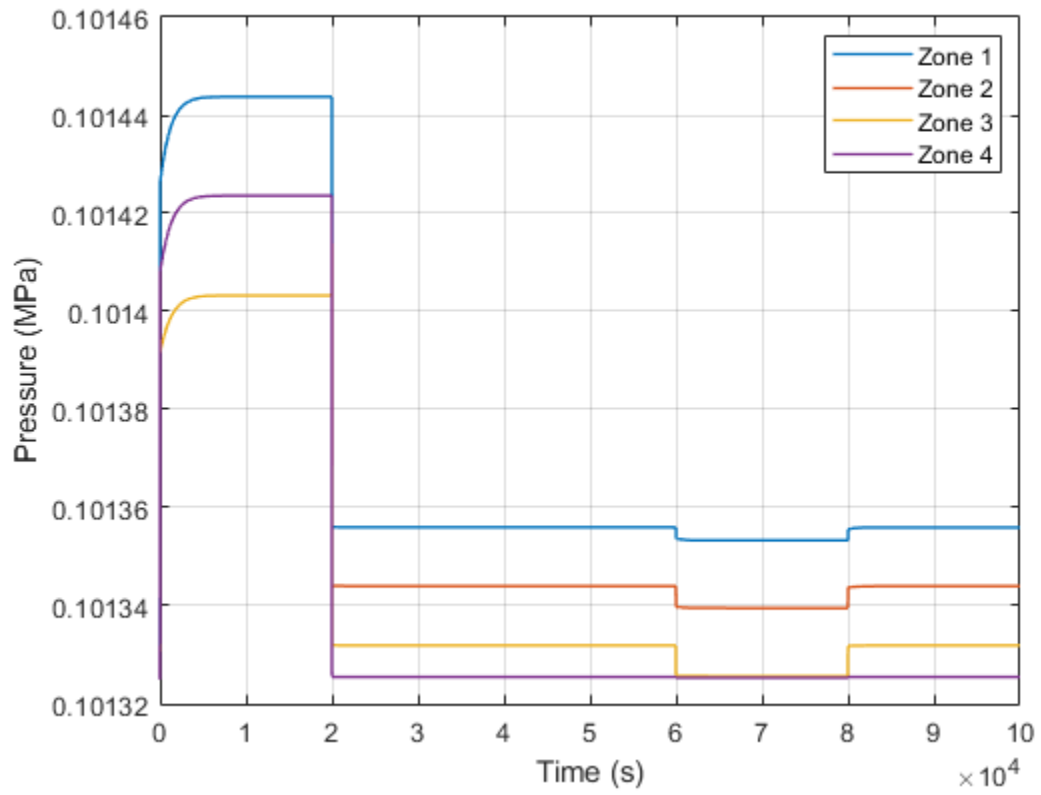
# Gamma Stirling Engine

This example shows how to model a Gamma Stirling engine using gas, thermal, and mechanical Simscape™ components and domains.

Stirling engines absorb heat from an external source to partially transform it into mechanical power, and dissipate the rest in a cold thermal sink. The external heat source is the key difference with internal combustion engines, which produce heat from combustion reactions in the gas inside the system. In Stirling engines the gas is inert (for example, air, in this case).

**Model overview**

The most typical designs of Stirling engines are alpha, beta and gamma configurations. In this example we only model the gamma configuration, which consists of two pistons connected with a passage pipe.

The first piston is called Displacer, which is a double-acting cylinder with two chambers, one is the heater, absorbing heat from a flame, and the other is the cooler, dissipating heat to the ambient. The overall volume of the displacer piston is constant, although gas flows from the cooler to the heater and vice-versa as the piston head moves. The flow between them is through the so-called Regenerator. The Regenerator is a pipe that allows flow between cooler and heater in the displacement piston. It is normally implemented as a piston head with smaller radius than the cylinder, allowing leakage.

The second piston is called Power piston, and is a single-acting cylinder with variable volume connected to the displacer through a passage pipe. This piston produces the torque and power.

Both displacer and power pistons are connected through two slider-crank mechanisms to a flywheel. The displacer crank has a 90 degree delay from the power piston.



The displacer piston:

The regenerator:
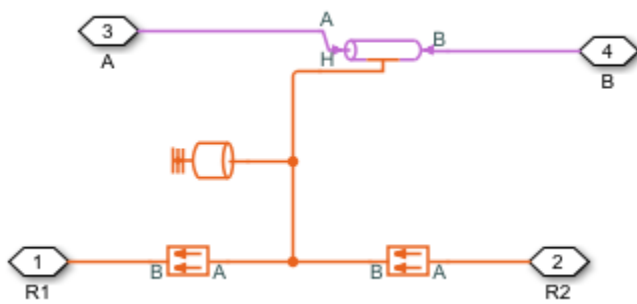


The regenerator also conducts heat from the heater to the cooler.

The power piston:
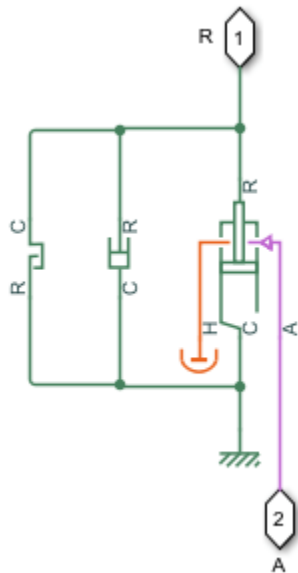
The slider-cranks and flywheel:
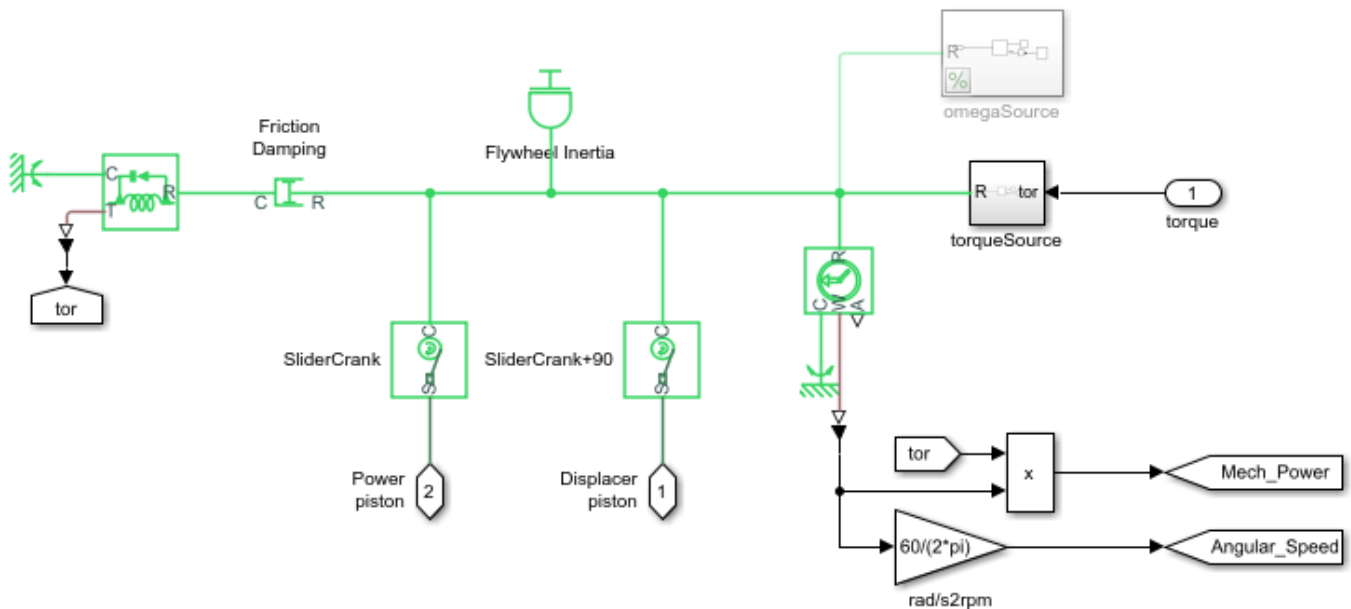


The user can choose to start the engine with a torque impulse and let it accelerate until steady-state, or force an angular speed, by commenting and uncommenting the torque source and the angular speed source.

The flame and ambient subsystems contain temperature sources and heat convection.

**Parameterization**

Most parameters in the Simscape™ blocks of this example have been stored as variables in the script ssc_stirling_engine_params for easy modification. Edit the script to change parameter values.

## Simulation results

The model simulates 15s of Stirling engine start-up, by applying an impulse at t = 5s to set the flywheel in initial motion.
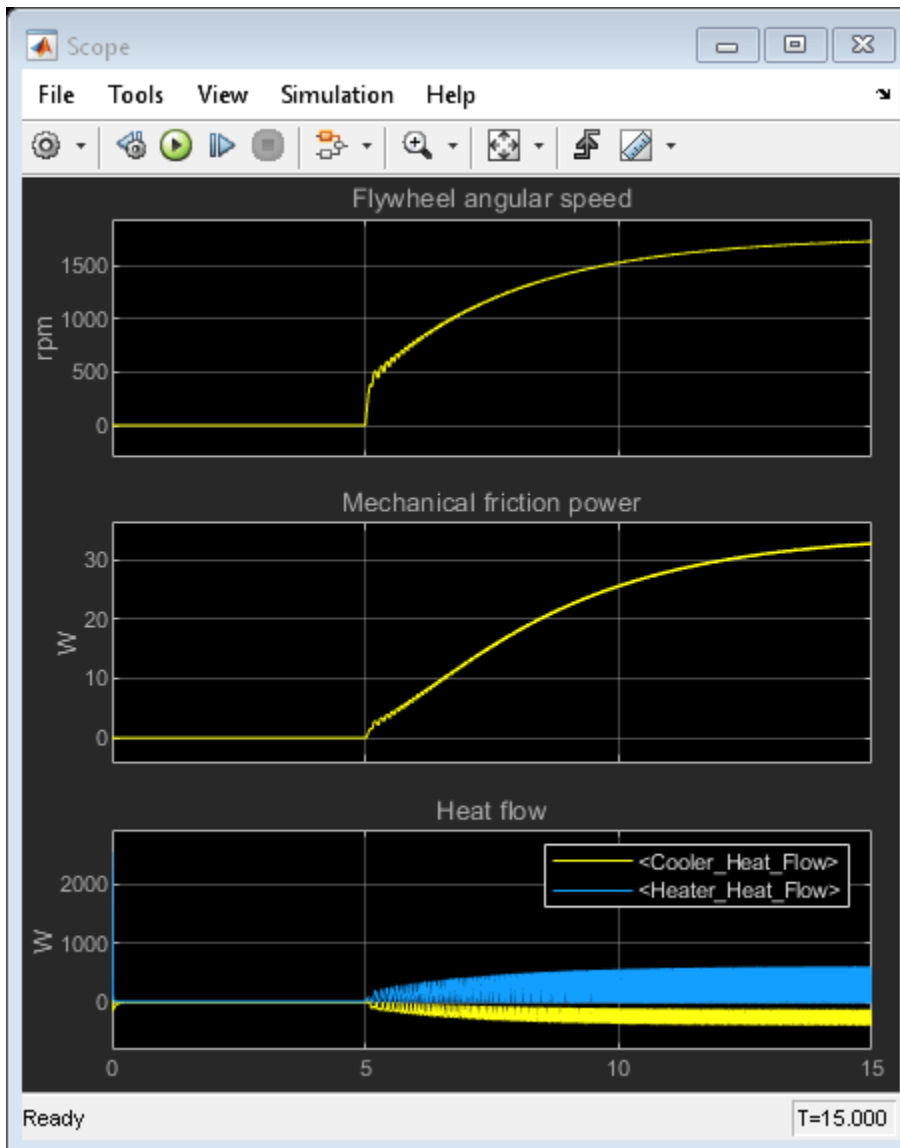


## P-V diagram of thermodynamic cycle

A key graph to consider in engine design is the P-V diagram of the thermodynamic cycle. It plots gas pressure and volume in the power piston during a revolution of the flywheel. In steady-state, this curve is closed and cyclical. The area enclosed by the curve is the mechanical work provided during one cycle. The total area under the curve is the heat absorbed during one cycle. The ratio between the two is the thermodynamic efficiency of the cycle. If we multiply work per cycle (or heat per cycle) with the number of cycles per second, we obtain the mechanical power (or the heat power absorbed)

```
Work per cycle: 1.3457J
Heat absorption per cycle: 11.4415J
```

```
Thermodynamic efficiency: 11.76%
Mechanical power: 38.7378W
Thermal power absorbed: 329.3652W
----------------------
```



P-V diagram

Work per cycle: 1.3457J

Heat absorbed per cycle: 11.4415J

Thermodynamic efficiency: 11.76%
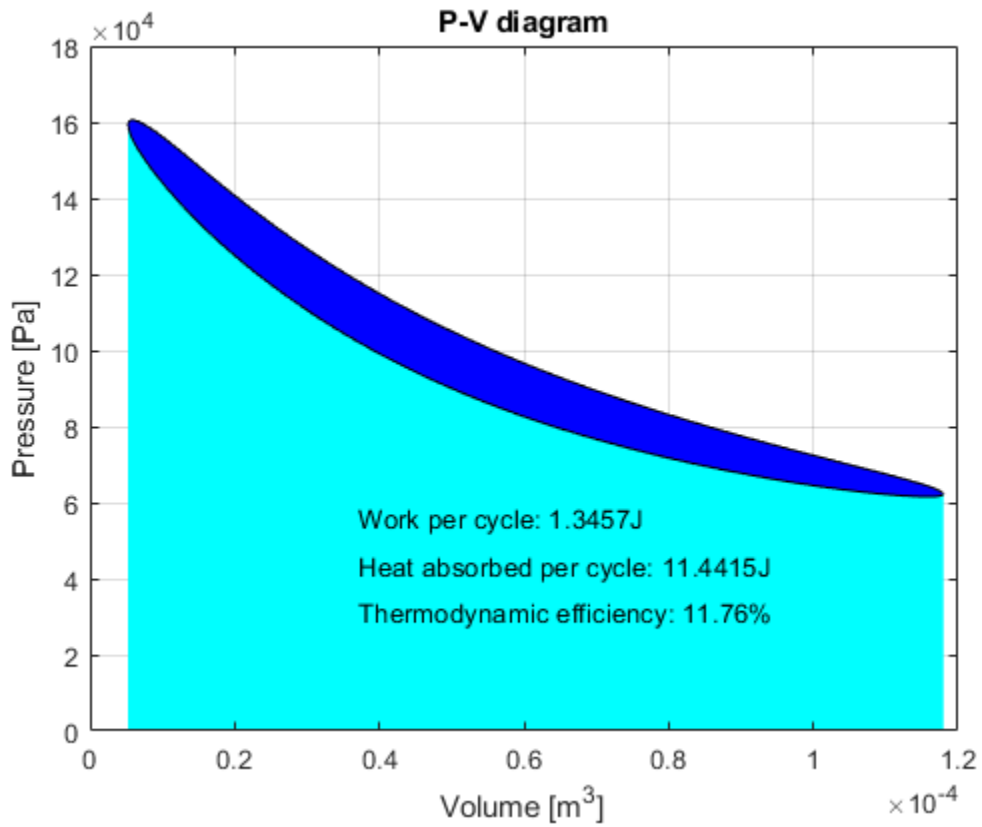
**Power and torque curves**

Another key performance indicator is the power-rpm curve and torque-rpm curve.

Power and Torque curves
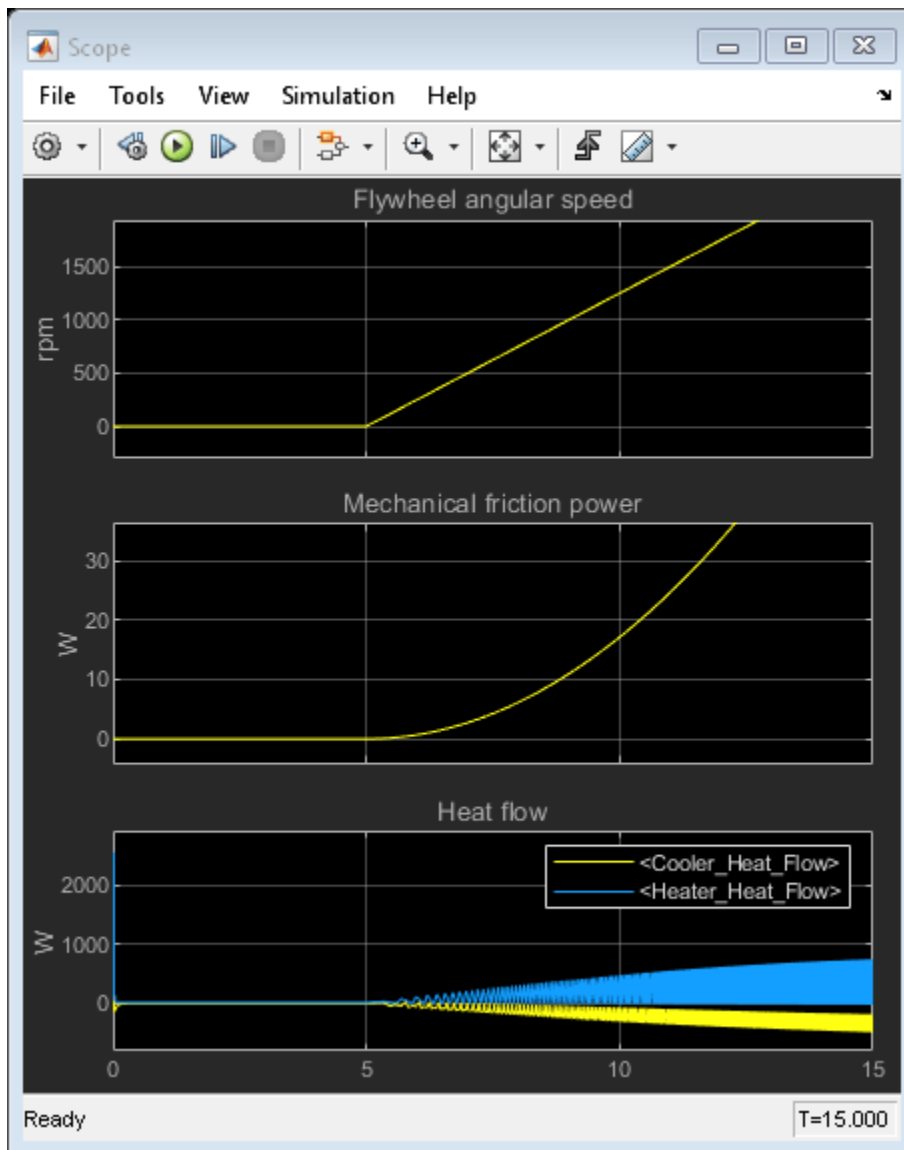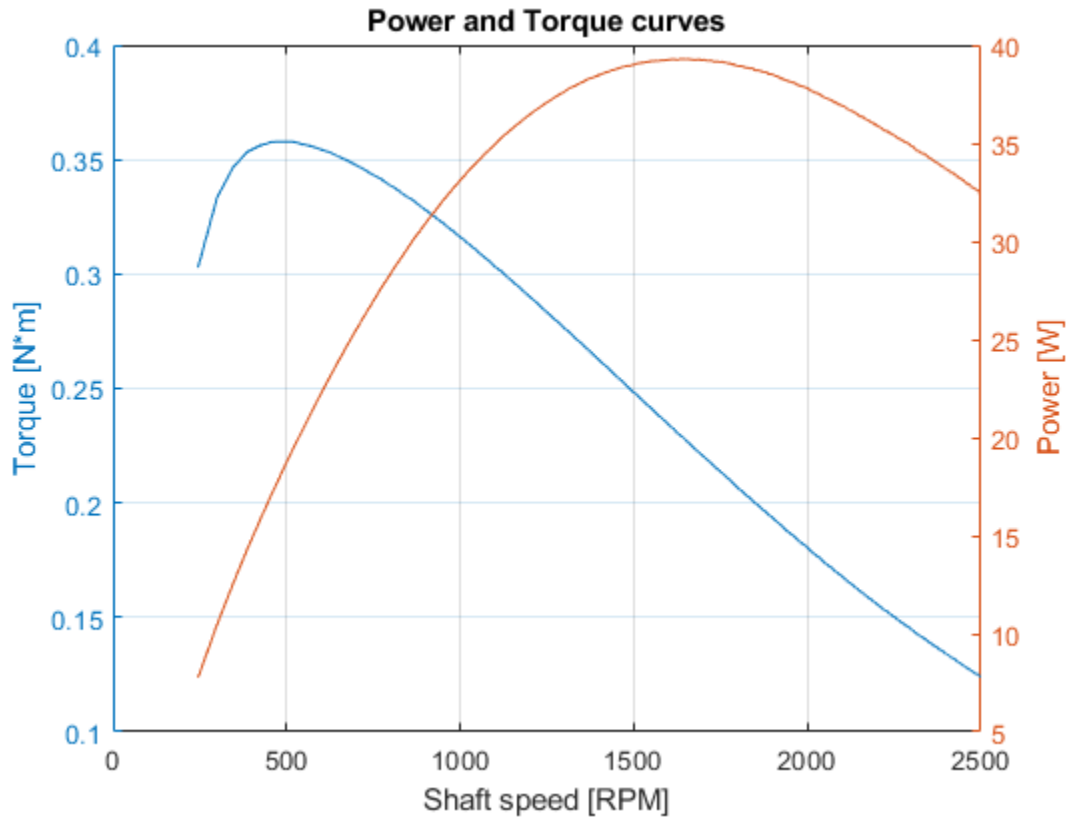
**Design optimization**

A great advantage of having a parameterized physical model is that optimization algorithms can be used to find optimal design parameters (for maximum efficiency or power). One of the possible design variables to optimize is power piston crank radius. In this section two values of power piston crank radius will be compared.

```
1st crank radius
Work per cycle: 1.3457J
Heat absorption per cycle: 11.4415J
Thermodynamic efficiency: 11.76%
Shaft speed: 1727.2132rpm
Mechanical power: 38.7378W
Thermal power absorbed: 329.3652W
----------------------
2nd crank radius
Work per cycle: 1.2816J
Heat absorption per cycle: 8.0983J
Thermodynamic efficiency: 15.83%
Shaft speed: 1613.5798rpm
Mechanical power: 34.4664W
Thermal power absorbed: 217.7864W
----------------------
```
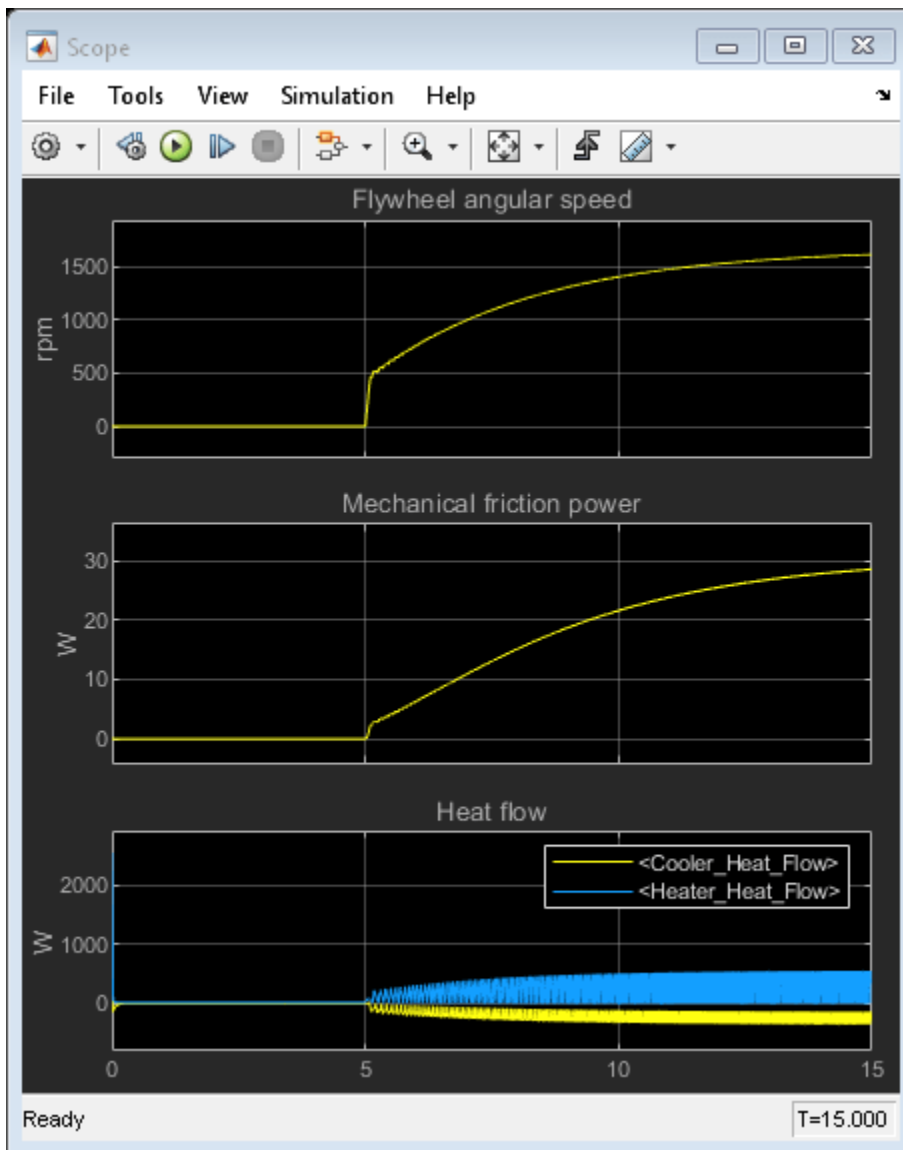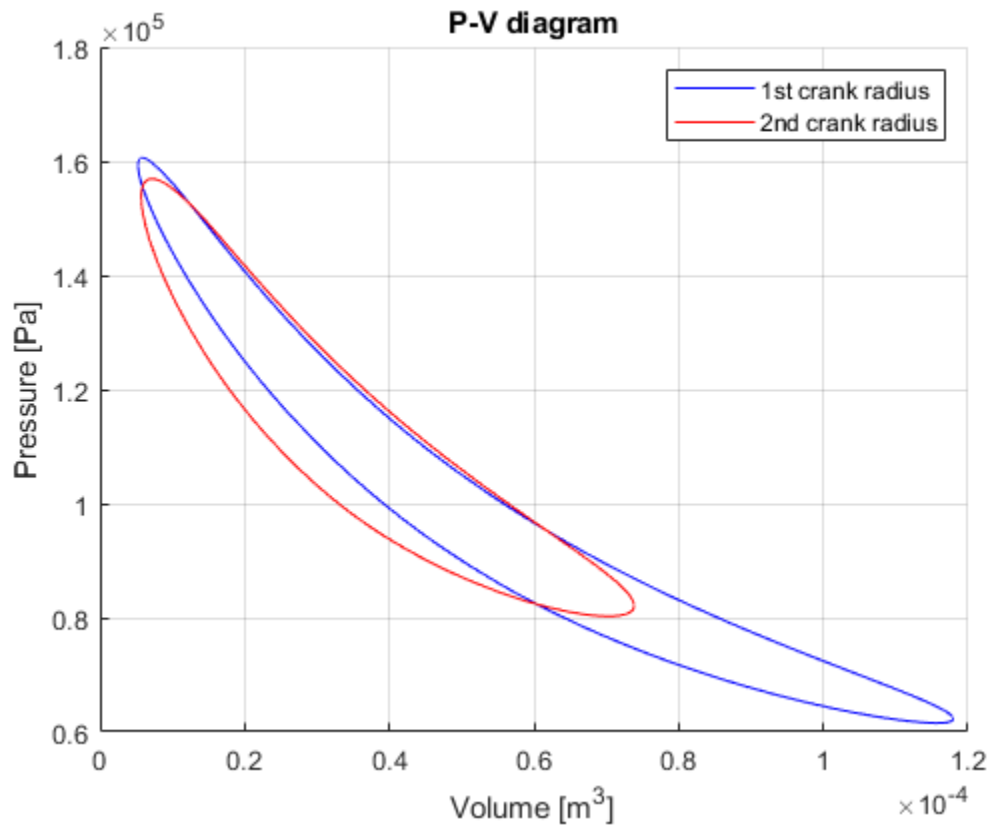
With the second value of crank radius we obtain lower shaft speed and lower power, but higher thermodynamic efficiency. This approach could be used in a multi-variable optimization process to find a global optimal design with genetic algorithms, for example.

# Vehicle HVAC System

This example models moist air flow in a vehicle heating, ventilation, and air conditioning (HVAC) system. The vehicle cabin is represented as a volume of moist air exchanging heat with the external environment. The moist air flows through a recirculation flap, a blower, an evaporator, a blend door, and a heater before returning to the cabin. The recirculation flap selects flow intake from the cabin or from the external environment. The blender door diverts flow around the heater to control the temperature.

The model can be simulated in two modes: predefined system inputs or manual system inputs. For predefined system inputs, the control settings for the HVAC system is specified in a signal builder block. For manual system inputs, the control settings can be adjusted at run time using the dashboard controls.

The heater core and the evaporator are basic implementations of heat exchangers using the e-NTU method. They are custom components based on the Simscape™ Foundation Moist Air library.

**Model**



### Vehicle HVAC System

1. Configure system inputs: Predefined or Manual
2. Plot psychrometric chart for cabin air (see code)
3. Plot condensation in evaporator (see code)
4. Explore simulation results using sscexplore
5. Open the HVAC example library
6. Learn more about this example

Copyright 2017-2021 The MathWorks, Inc.
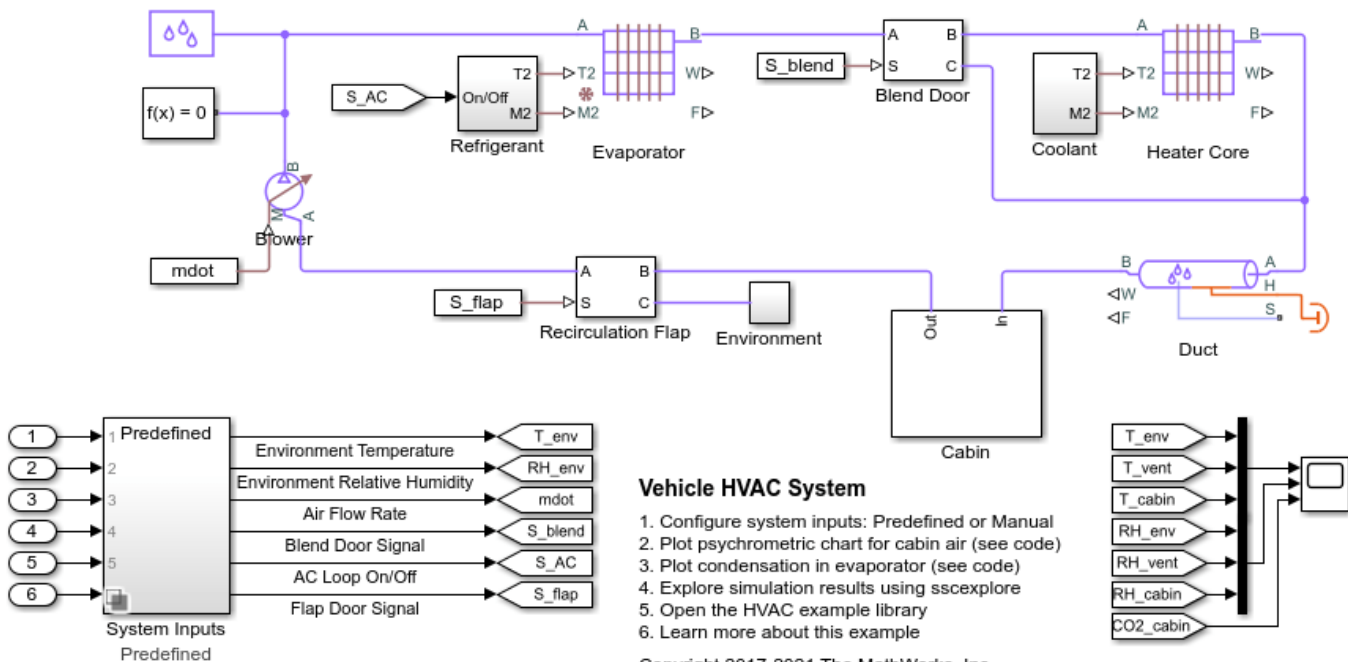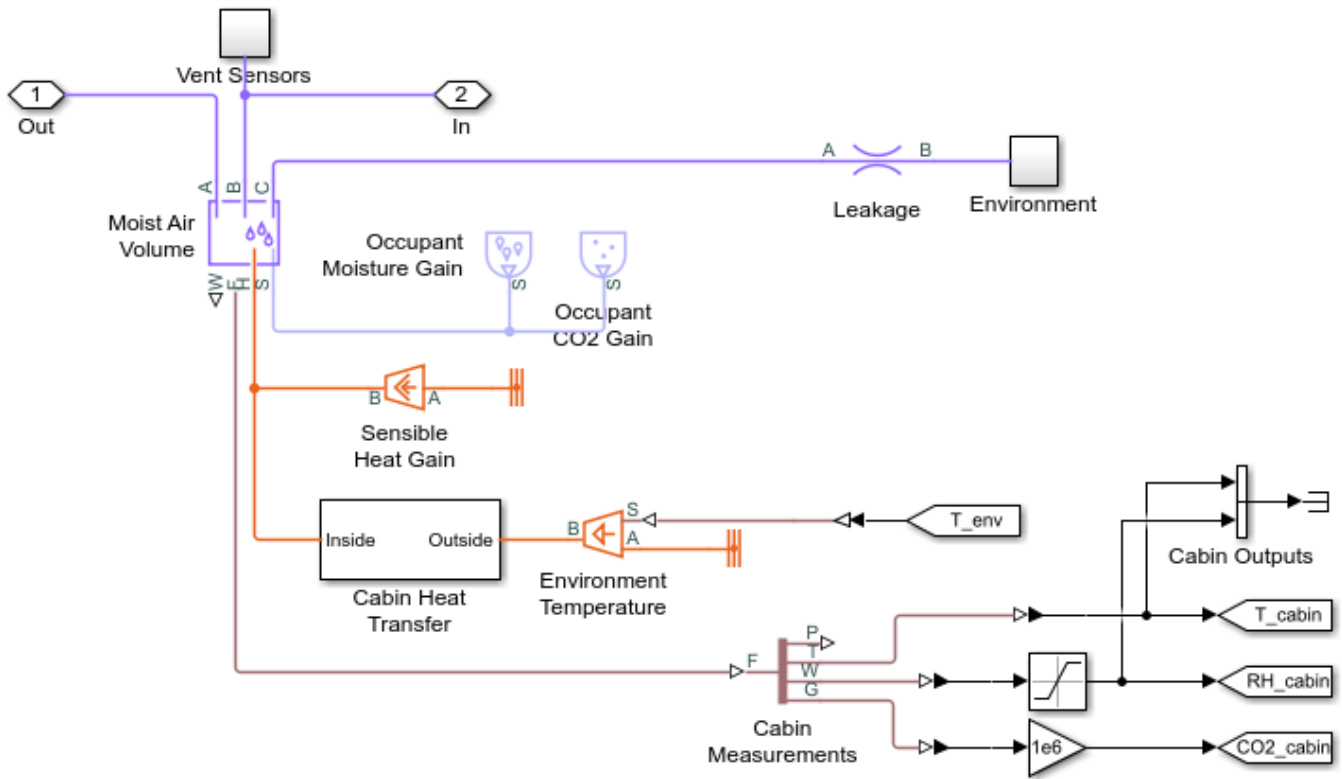
**Cabin Subsystem**

**Cabin Heat Transfer Subsystem**
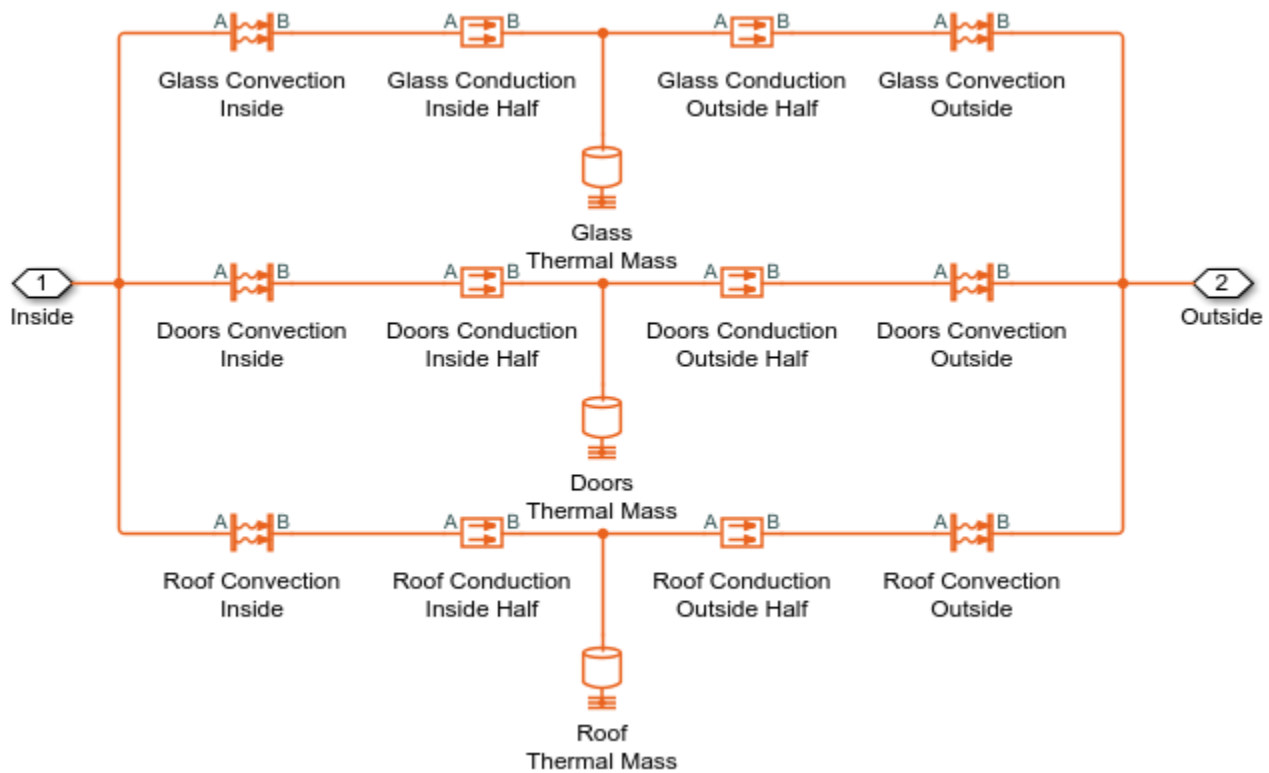
**Simulation Results from Scopes**
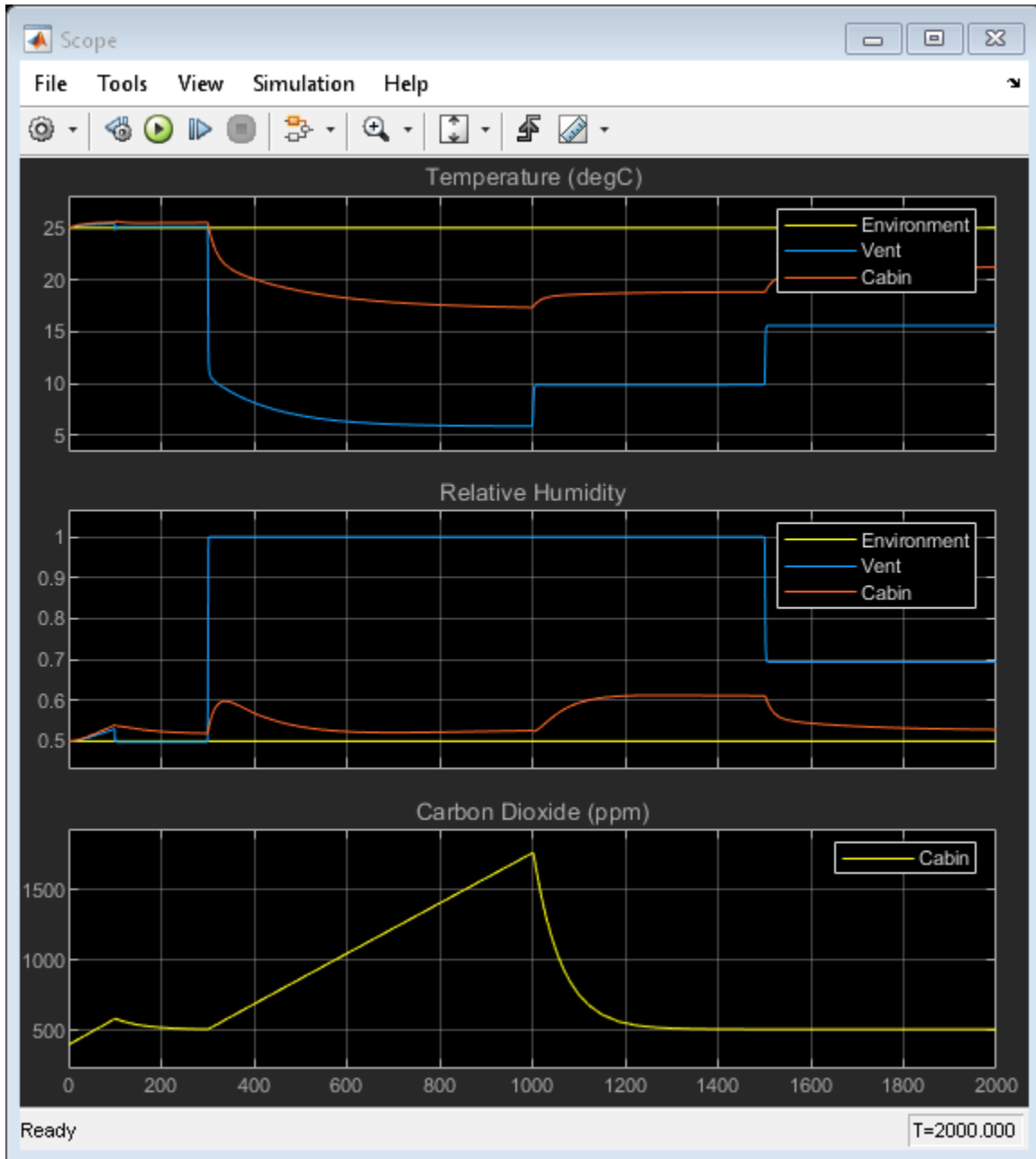


**Simulation Results from Simscape Logging**

This figure shows a psychrometric chart for the moist air inside the cabin. It is based on the ASHRAE Psychrometric Chart No. 1 and is used to convey thermodynamic properties of moist air, such as dry-bulb temperature, web-bulb temperature, relative humidity, humidity ratio, enthalpy, and volume. The trajectory of the state of the cabin moist air during simulation is plotted on the chart.

**Psychrometric Chart (Predefined Inputs)**
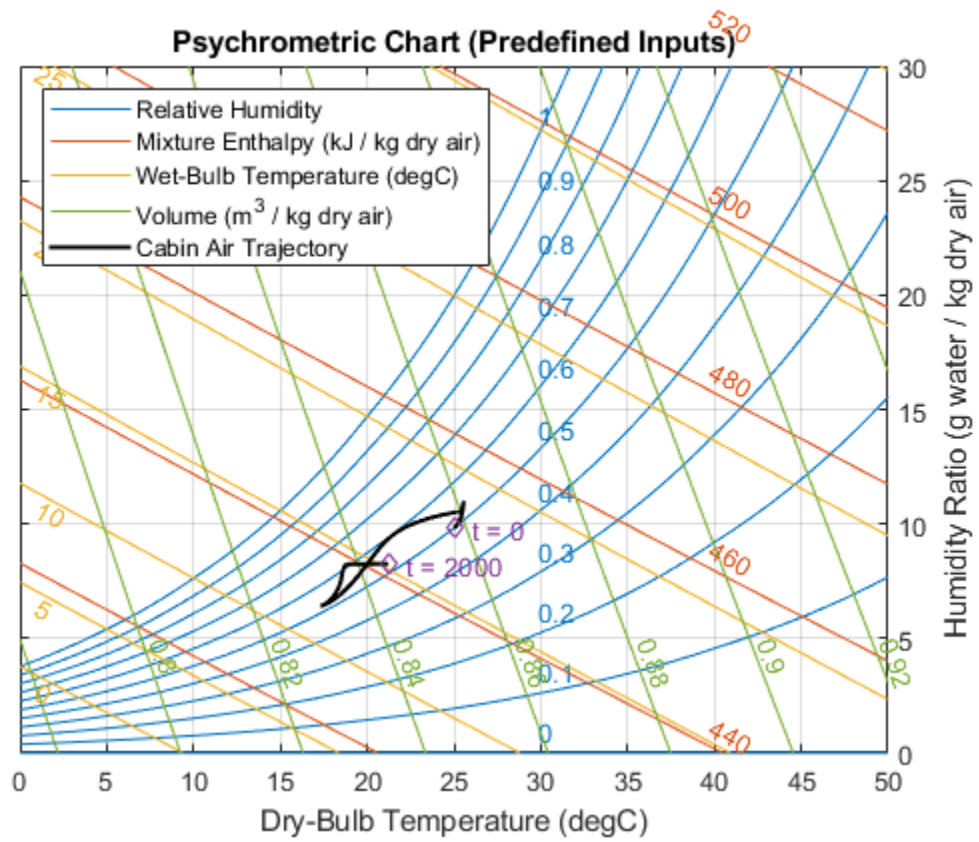
This plot shows rate of condensation in the evaporator when the air conditioning is turned on.
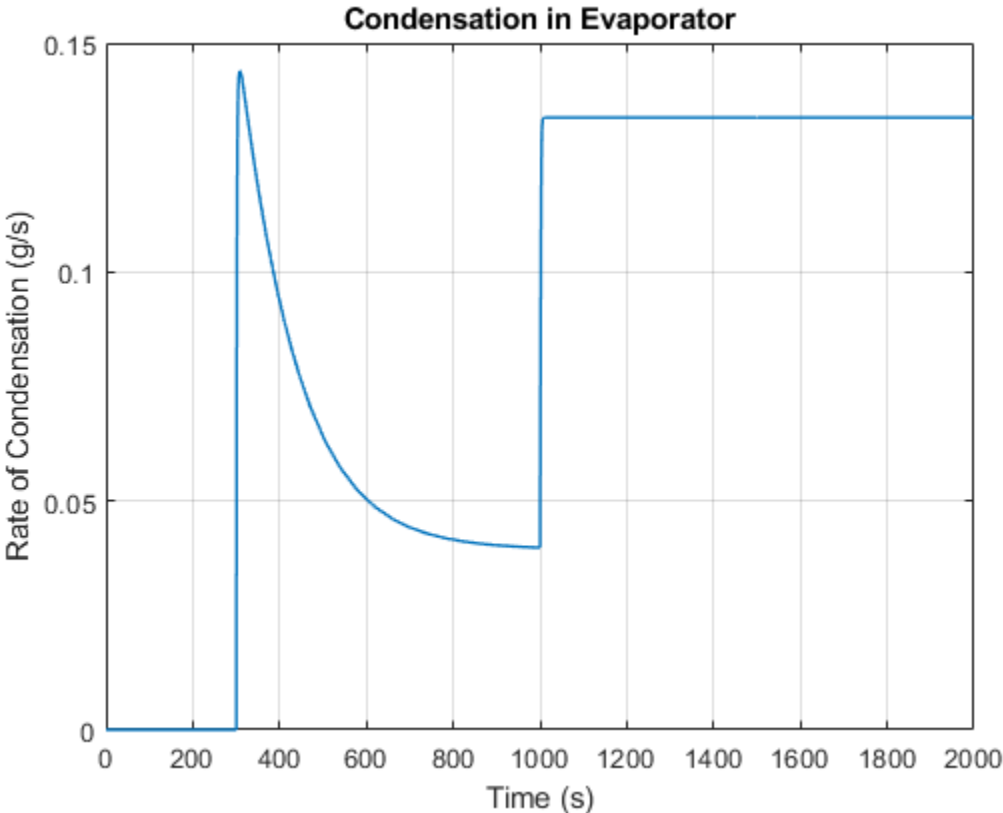
# Aircraft Environmental Control System

This example models an aircraft environmental control system (ECS) that regulates pressure, temperature, humidity, and ozone (O3) to maintain a comfortable and safe cabin environment. Cooling and dehumidification are provided by the air cycle machine (ACM), which operates as an inverse Brayton cycle to remove heat from pressurized hot engine bleed air. Some hot bleed air is mixed directly with the output of the ACM to adjust the temperature. Pressurization is maintained by the outflow valve in the cabin. This model simulates the ECS operating from a hot ground condition to a cold cruise condition and back to a cold ground condition.

Simple models of the compressor, turbine, and heat exchangers in the ACM are implemented as custom components based on the Simscape™ Foundation Moist Air library.

**Model**



**Aircraft Environmental Control System**

1. Plot flow rates through ACM (see code)
2. Plot condensation in turbine (see code)
3. Plot temperature-entropy diagram of ACM (see code)
4. Plot compressor map (see code)
5. Explore simulation results using sscexplore
6. Open the HVAC example library
7. Learn more about this example

Copyright 2017-2021 The MathWorks, Inc.
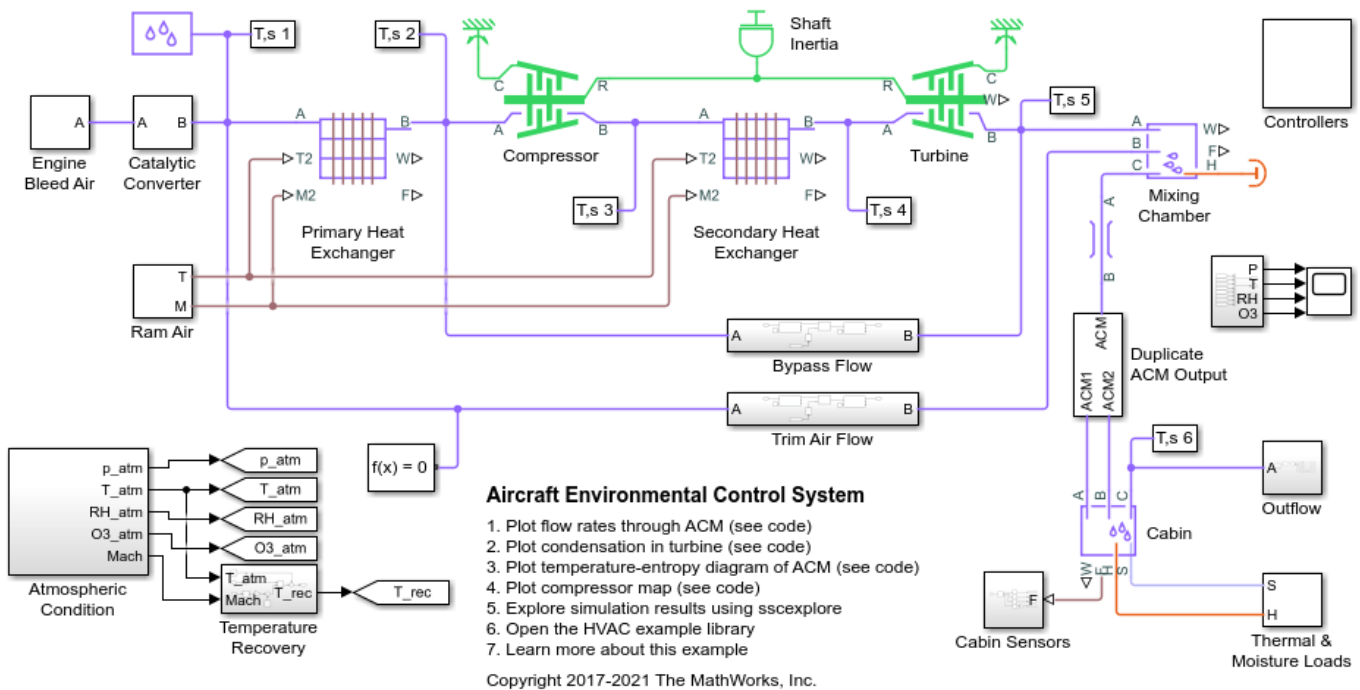
**Bypass Flow Subsystem**
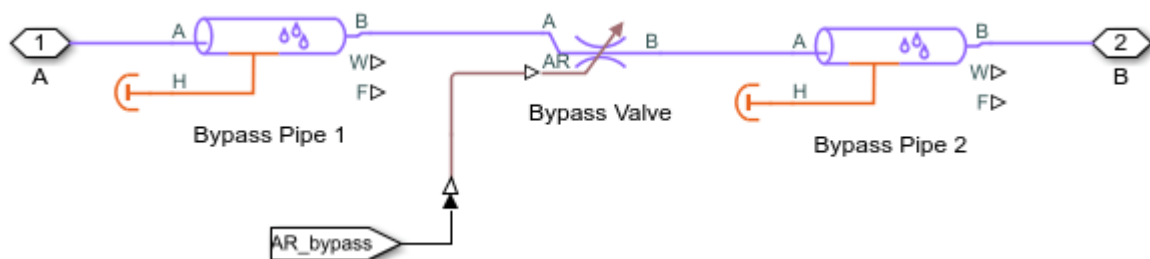
### Catalytic Converter Subsystem
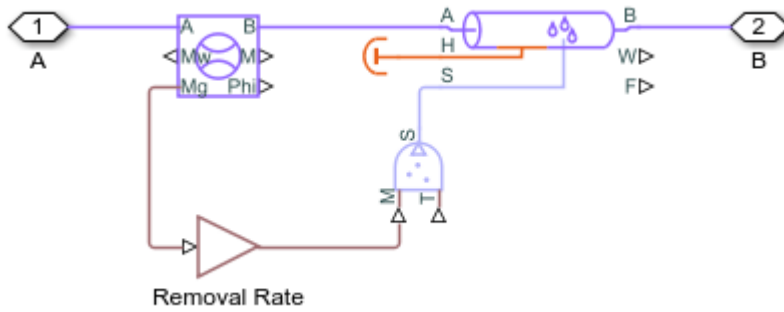


### Controllers Subsystem
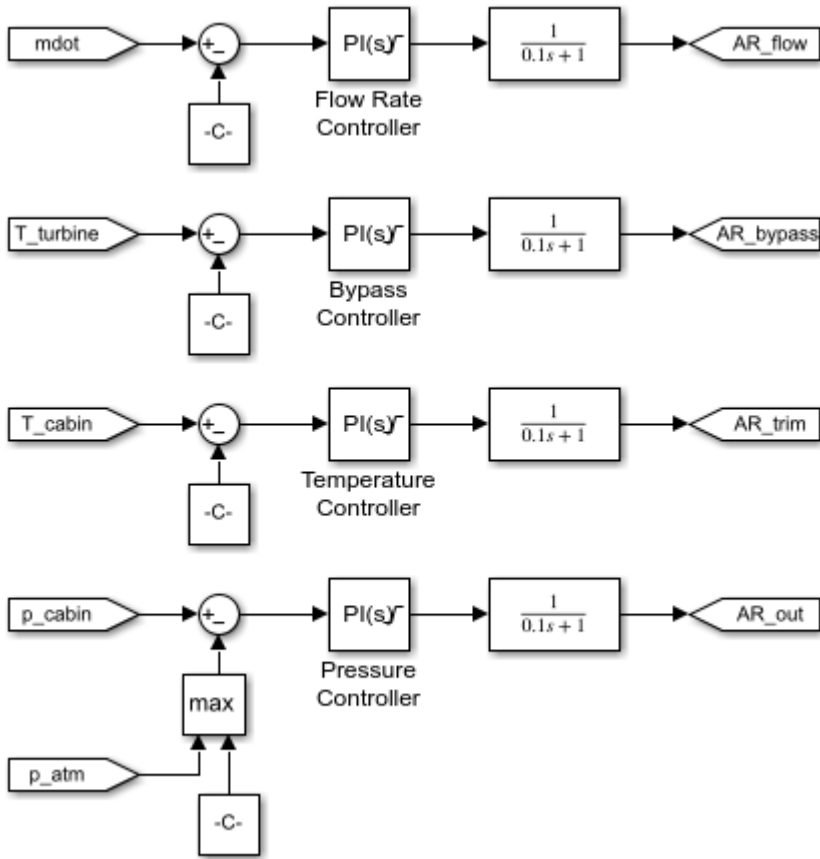
## Engine Bleed Air Subsystem
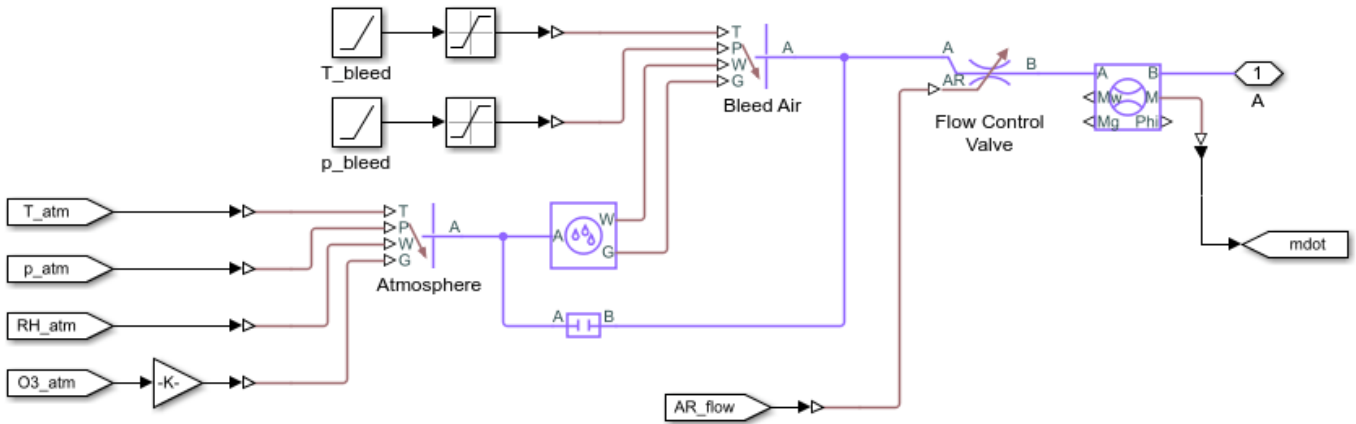


## Outflow Subsystem
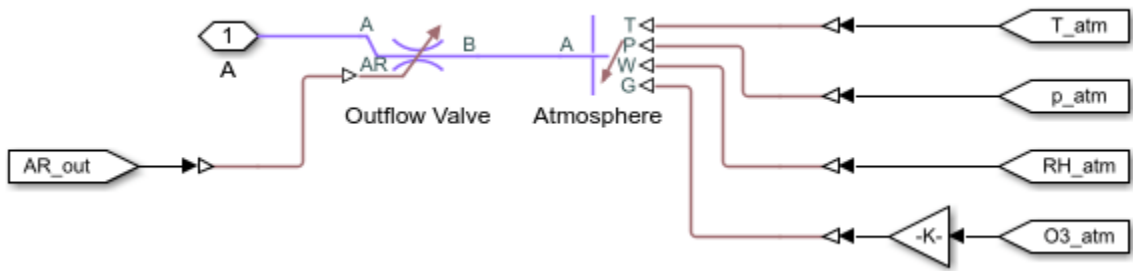


## Thermal & Moisture Loads Subsystem
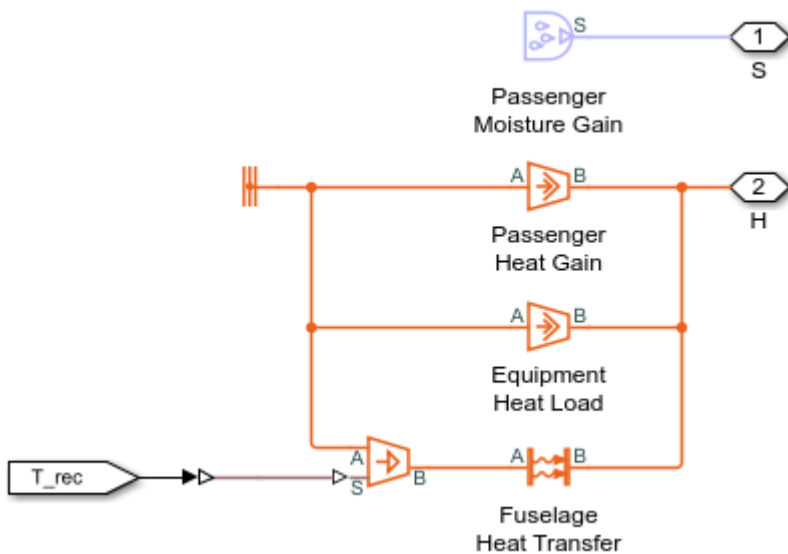
**Trim Air Flow Subsystem**
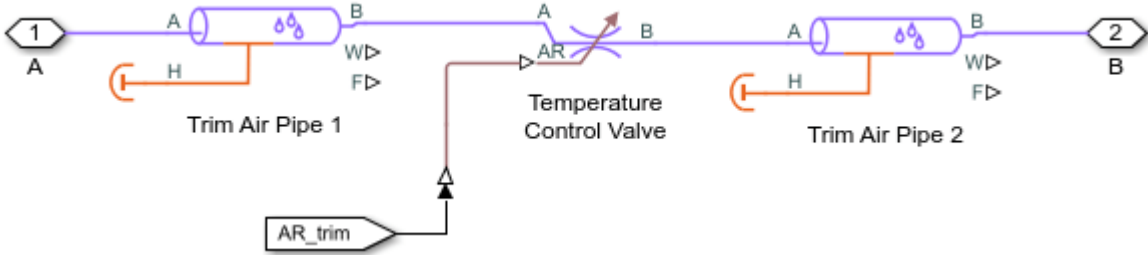
**Simulation Results from Scopes**
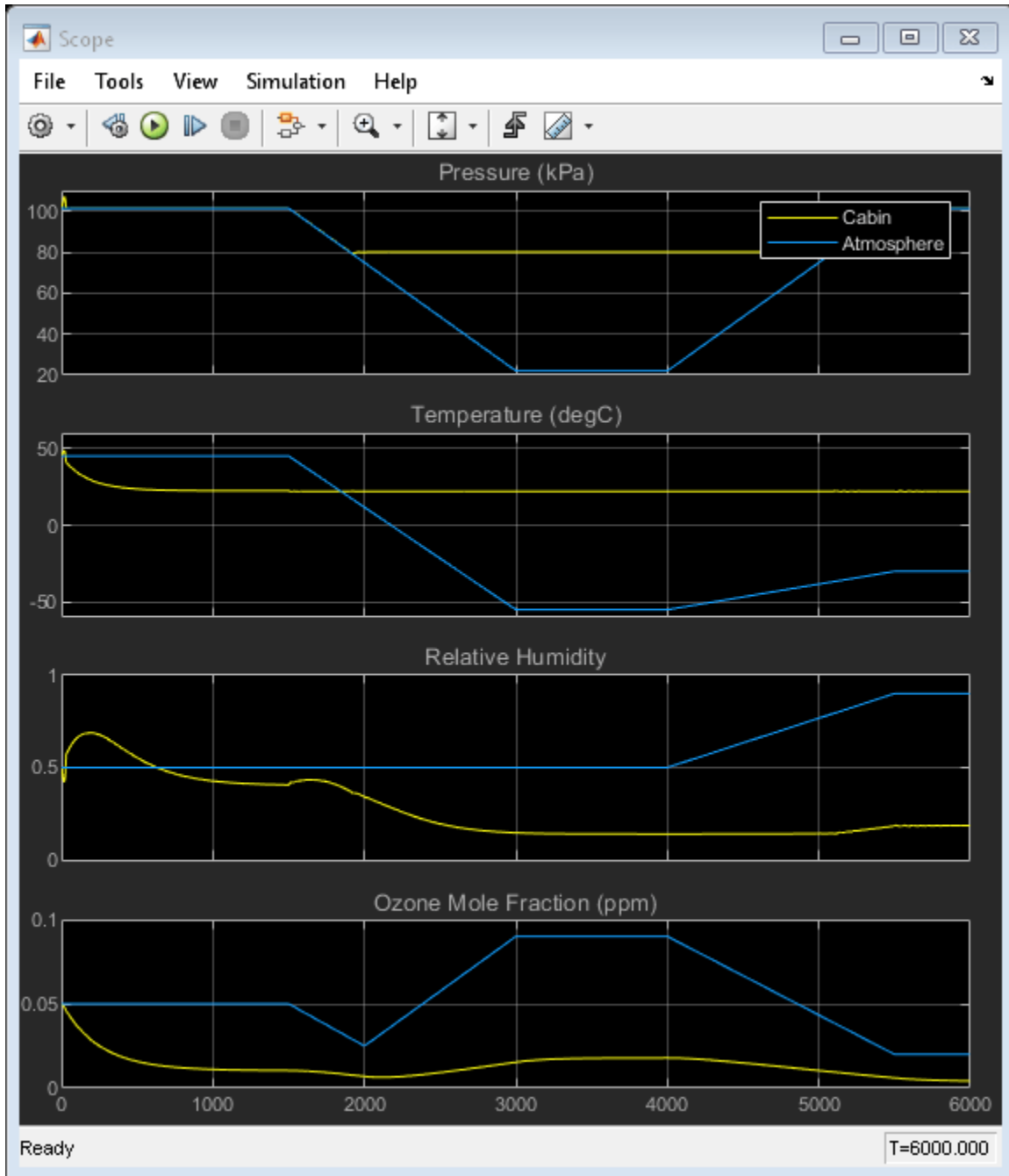


**Simulation Results from Simscape Logging**

This plot shows the effect of the valves that regulate the air flow through the system. As the external environment becomes cooler, some of the air flow bypasses the compressor and turbine to prevent the turbine outflow from becoming too cold. Furthermore, some hot engine bleed air is used as trim air to adjust the temperature of the air flow supplied to the cabin.
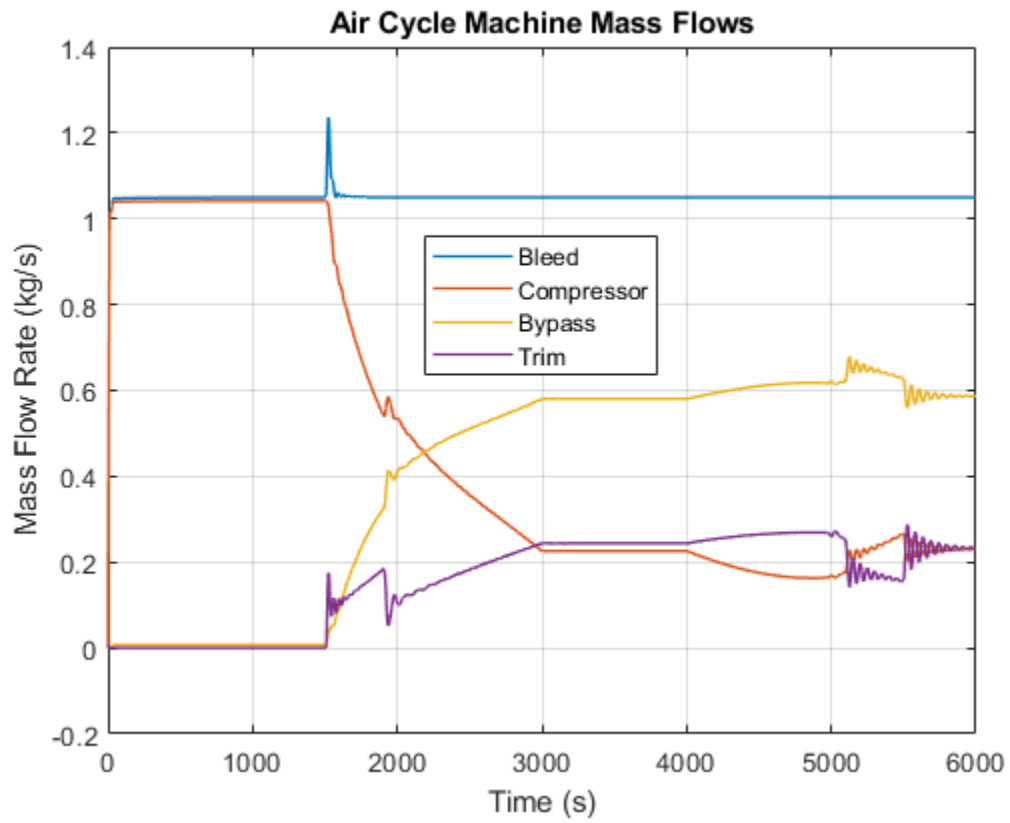
This plot shows the moisture removed from the hot humid air by the air cycle machine (ACM).
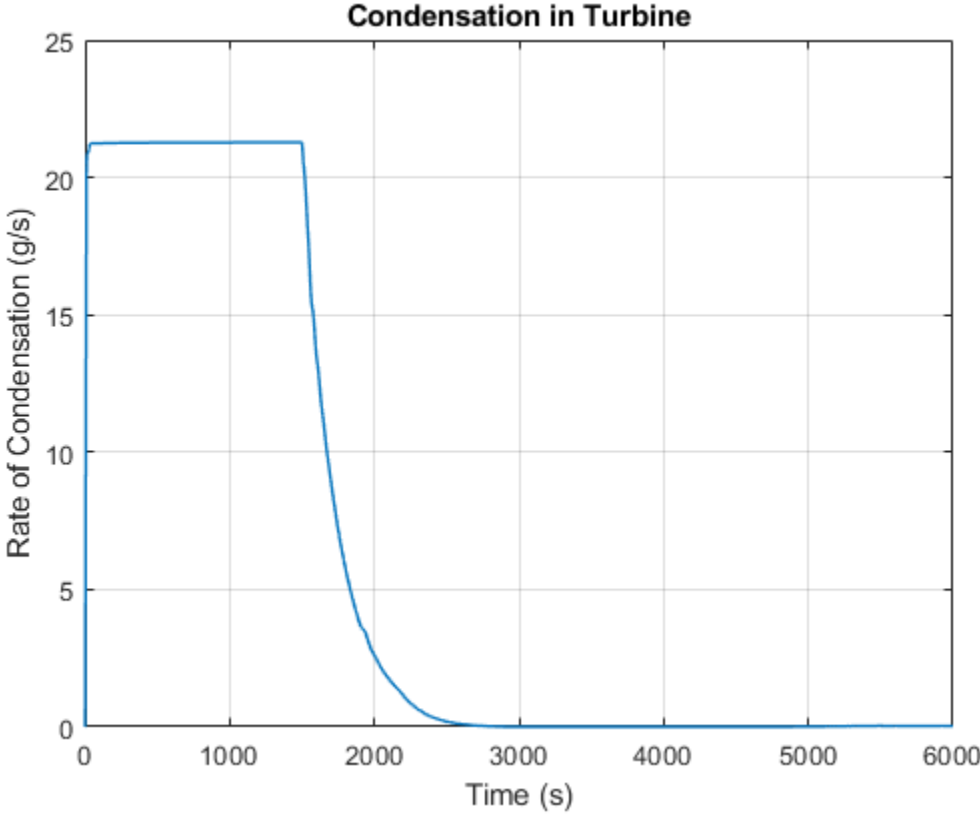
This plot shows thermodynamic states of the air flow in a temperature-entropy diagram when the air cycle machine (ACM) is cooling and dehumidifying the cabin. It is an inverse Brayton cycle that extracts work from the pressurized bleed air to cool the air flow below the ambient temperature.

**Temperature-Entropy Diagram (t = 1000 s)**
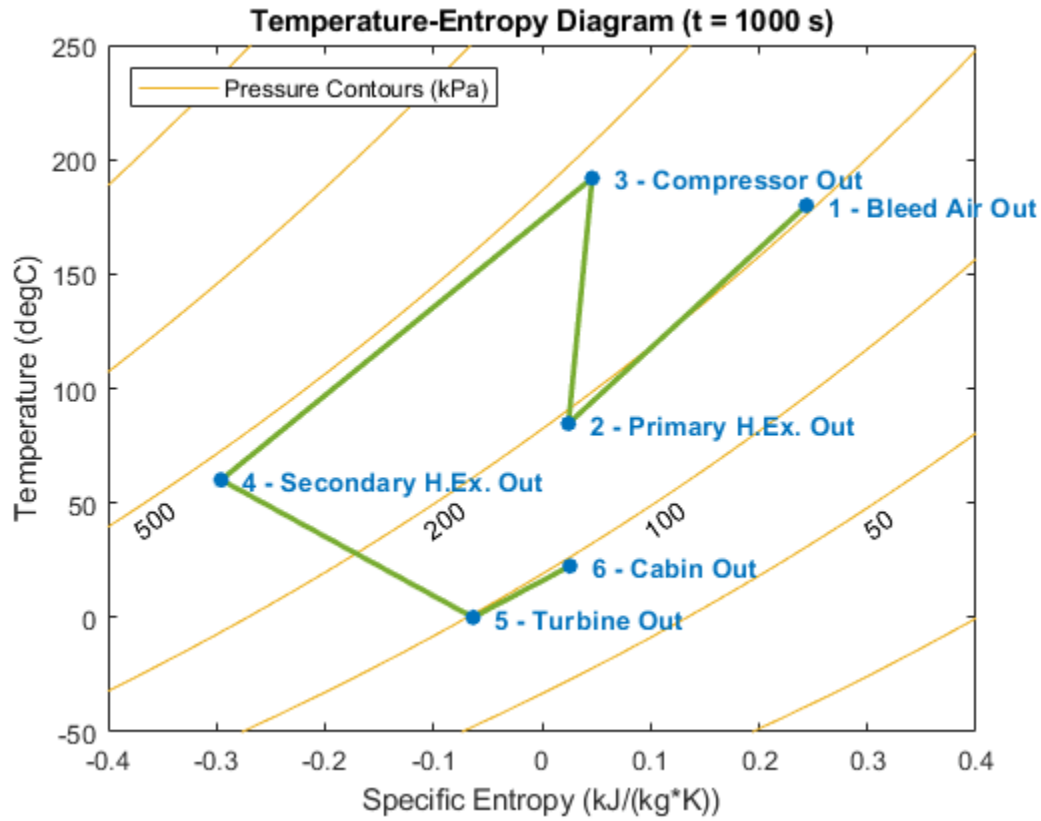
This plot shows a basic compressor map used by the air cycle machine (ACM) parameterized by 3 coefficients that control the shape and spacing of the corrected speed lines.
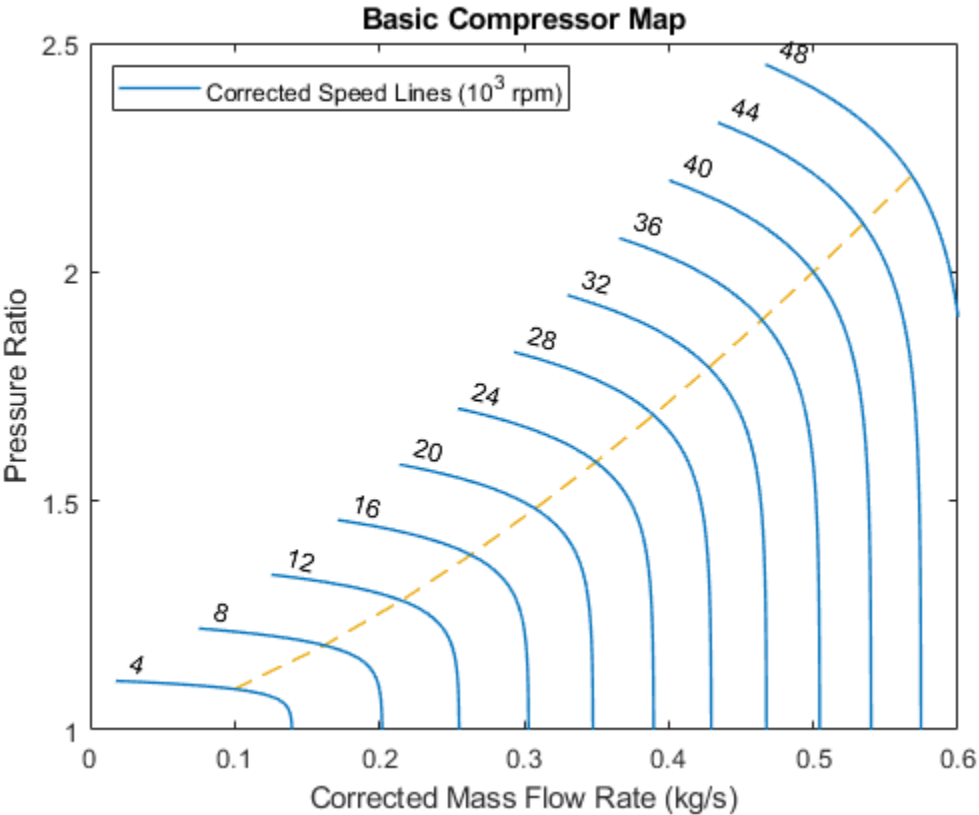
# PEM Fuel Cell System

This example shows how to model a proton exchange membrane (PEM) fuel cell stack with a custom Simscape™ block. The PEM fuel cell generates electrical power by consuming hydrogen and oxygen and producing water vapor. The custom block represents the membrane electrode assembly (MEA) and is connected two separate moist air networks: one for the anode gas flow and one for the cathode gas flow.

The two moist air networks represents different gas mixtures. The anode network consists of nitrogen (N2), water vapor (H2O), and hydrogen (H2), representing the fuel. The hydrogen is stored in the fuel tank at 70 MPa. A pressure-reducing valve releases hydrogen to the fuel cell stack at around 0.16 MPa. Unconsumed hydrogen is recirculated back to the stack. The cathode network consists of nitrogen (N2), water vapor (H2O), and oxygen (O2), representing air from the environment. A compressor brings air to the fuel cell stack at a controlled rate to ensure that the fuel cell is not starved of oxygen. A back pressure relief valve maintains a pressure of around 0.16 MPa in the stack and vents the exhaust to the environment.

The temperature and relative humidity in the fuel cell stack must be maintained at an optimal level to ensure efficient operation under various loading conditions. Higher temperatures increase thermal efficiency but reduce relative humidity, which causes higher membrane resistance. Therefore, in this model, the fuel cell stack temperature is kept at 80 degC. The cooling system circulates coolant between the cells to absorb heat and rejects it to the environment via the radiator. The humidifiers saturate the gas with water vapor to keep the membrane hydrated and minimize electrical resistance.

The custom MEA block is implemented in the Simscape code `FuelCell.ssc`. The output port F of the anode and cathode gas channel pipe blocks provide the gas mole fractions needed to model the fuel cell reaction. The removal of H2 and O2 from the anode and cathode gas flows are implemented by Controlled Trace Gas Source (MA) blocks. The production of H2O and the transport of water vapor across the MEA are implemented by Controlled Moisture Source (MA) blocks. The heat generated by the reaction is sent through the thermal port H to the connected Thermal Mass block. Refer to the comments in the code for additional details on the implementation.

See also the PEM Electrolysis System example.

References:

Dutta, Sandip, Sirivatch Shimpalee, and J. W. Van Zee. "Numerical prediction of mass-exchange between cathode and anode channels in a PEM fuel cell." *International Journal of Heat and Mass Transfer* 44.11 (2001): 2029-2042.

EG&G Technical Services, Inc. *Fuel Cell Handbook (Seventh Edition).* US Department of Energy, Office of Fossil Energy, National Energy Technology Laboratory, 2004.

Pukrushpan, Jay T., Anna G. Stefanopoulou, and Huei Peng. *Control of fuel cell power systems: principles, modeling, analysis and feedback design.* Springer-Verlag London, 2004.

Spiegel, Colleen. *PEM fuel cell modeling and simulation using MATLAB.* Elsevier, 2008.

## Model
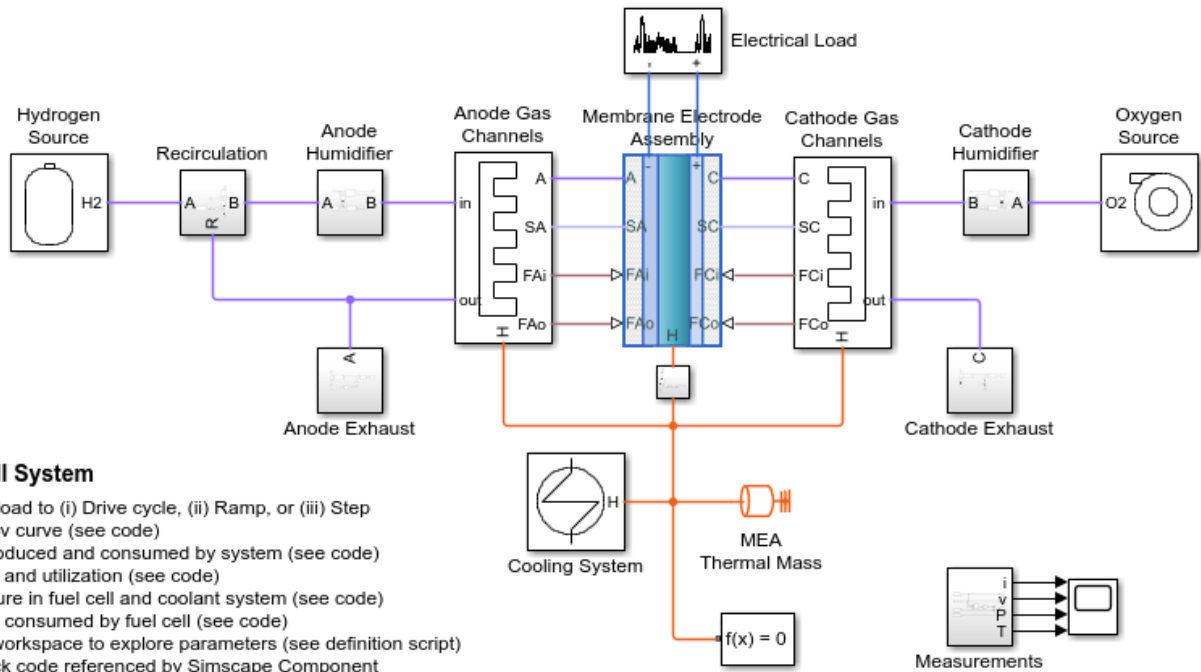


### PEM Fuel Cell System

1. Set electrical load to (i) Drive cycle, (ii) Ramp, or (iii) Step
2. Plot fuel cell i-v curve (see code)
3. Plot power produced and consumed by system (see code)
4. Plot efficiency and utilization (see code)
5. Plot temperature in fuel cell and coolant system (see code)
6. Plot hydrogen consumed by fuel cell (see code)
7. Open model workspace to explore parameters (see definition script)
8. See MEA block code referenced by Simscape Component
9. Explore simulation results using sscexplore
10. Learn more about this example

Copyright 2020-2021 The MathWorks, Inc.

## Anode Humidifier Subsystem
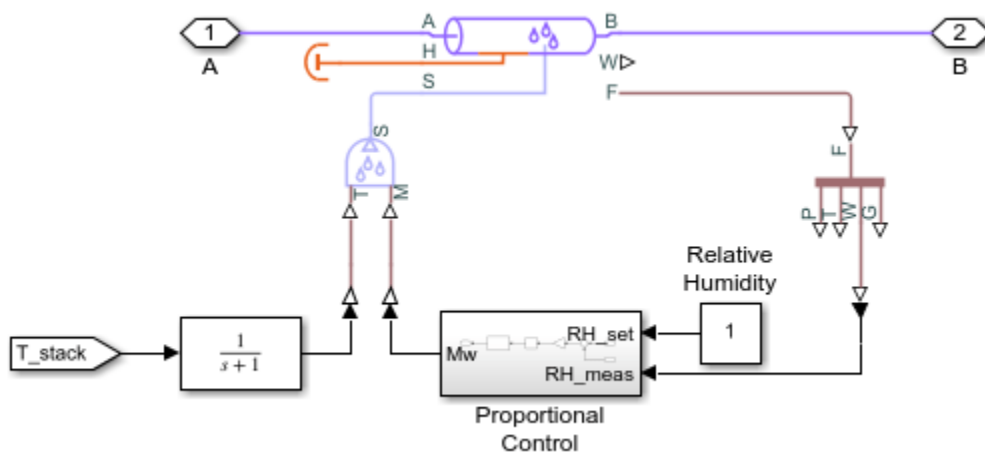
**Anode Exhaust Subsystem**
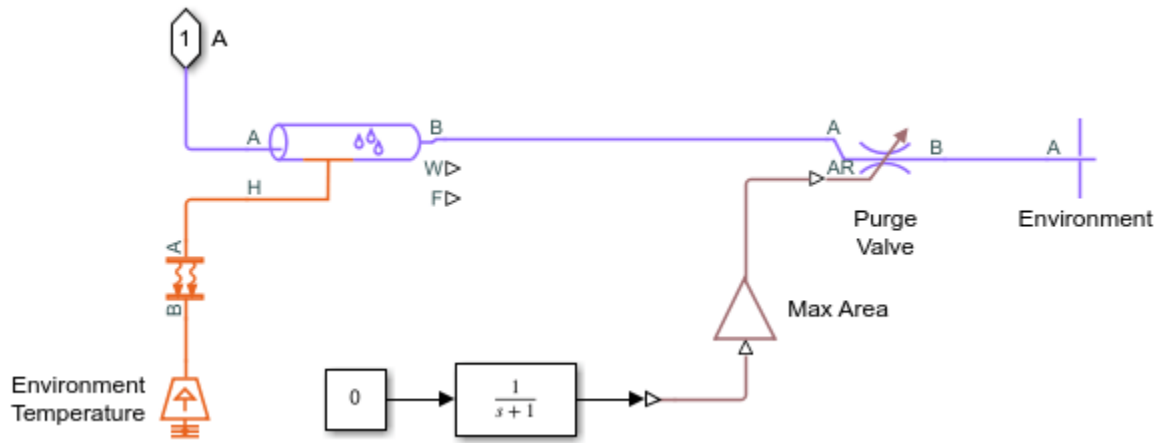


**Anode Gas Channels Subsystem**
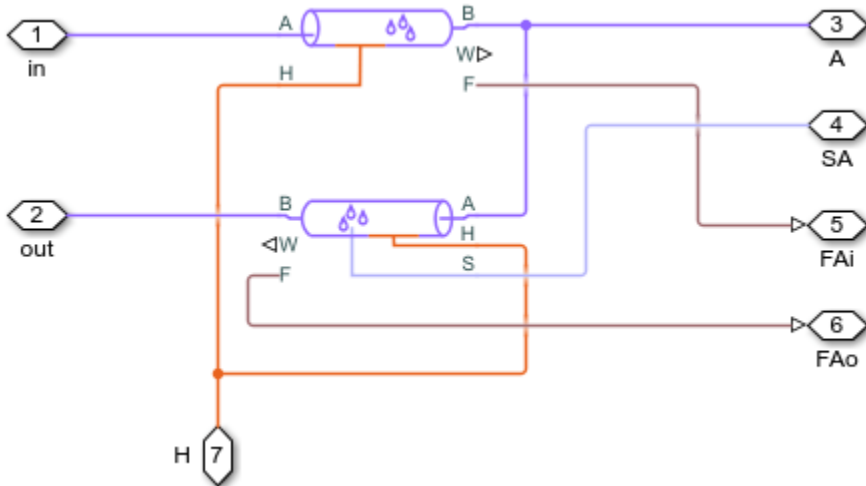
## Cathode Humidifier Subsystem
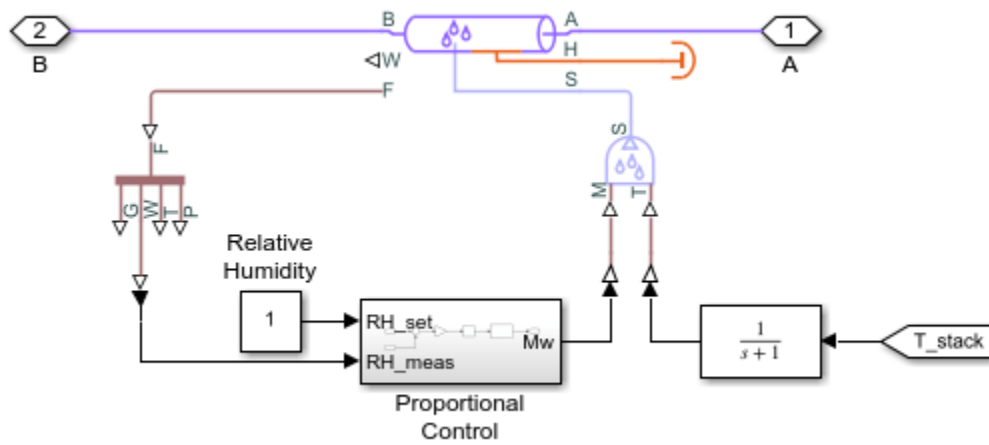


## Cathode Exhaust Subsystem
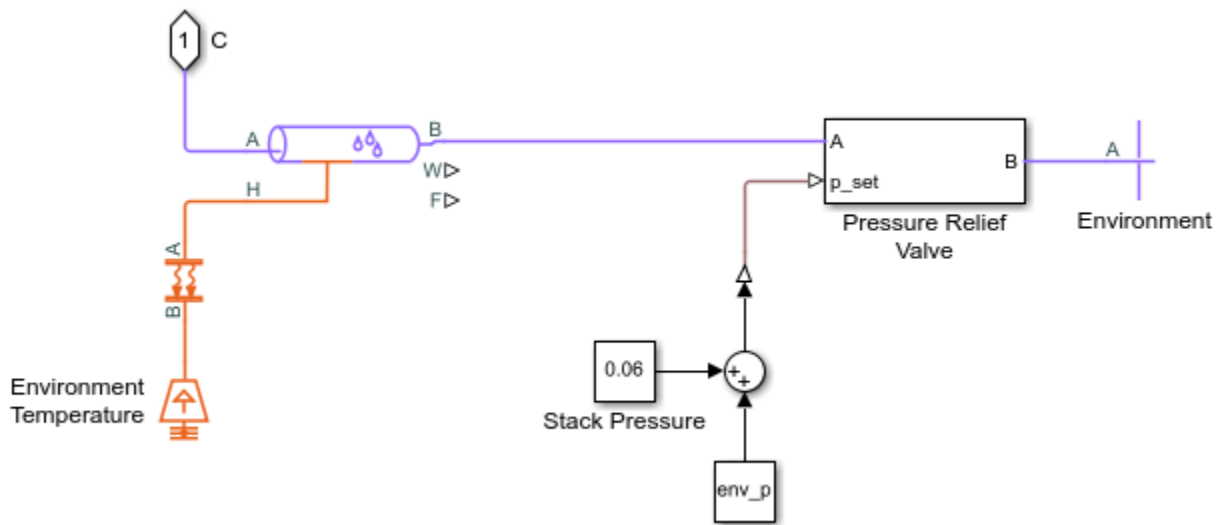
**Pressure Relief Valve Subsystem**



**Cathode Gas Channels Subsystem**

## Cooling System Subsystem
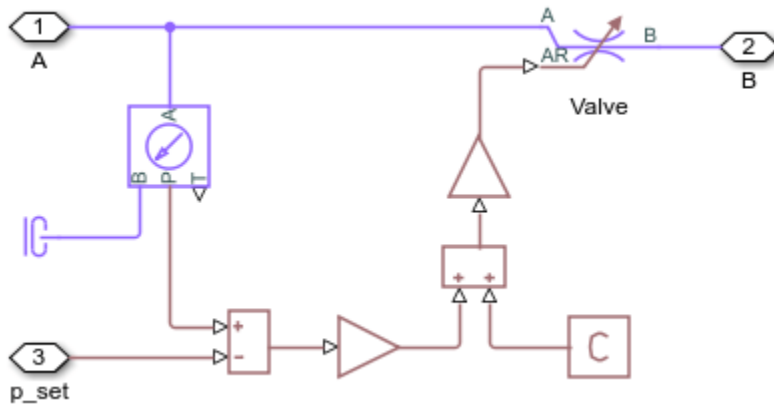


## Coolant Tank Subsystem
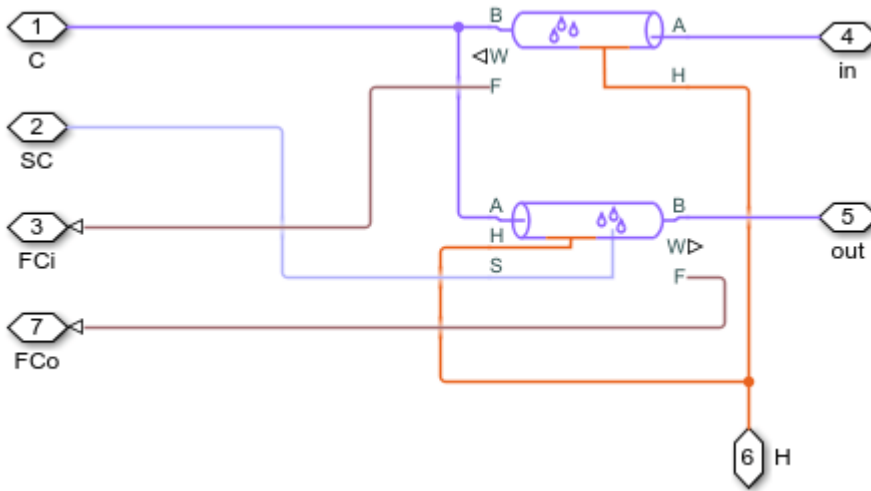
**Electrical Load Subsystem**
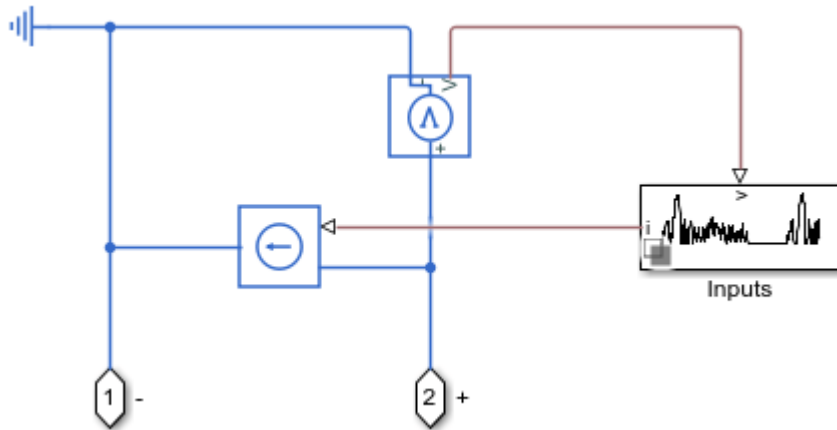


**Hydrogen Source Subsystem**
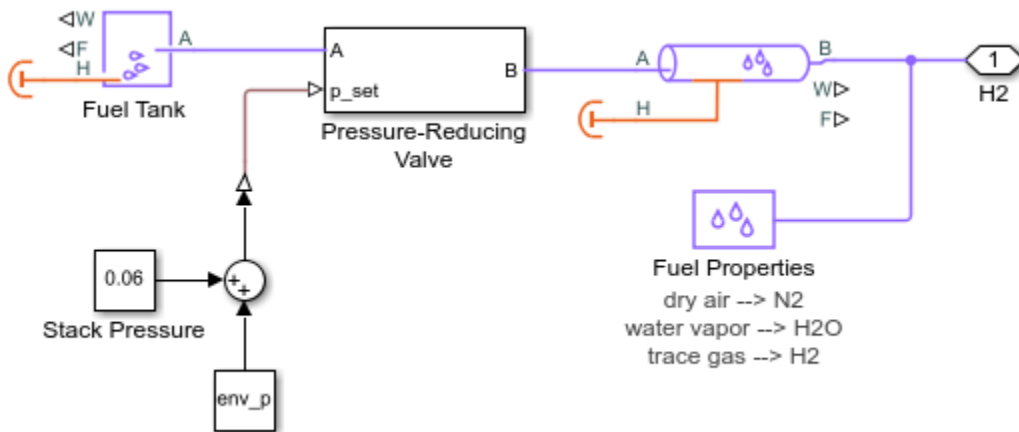
**Pressure-Reducing Valve Subsystem**
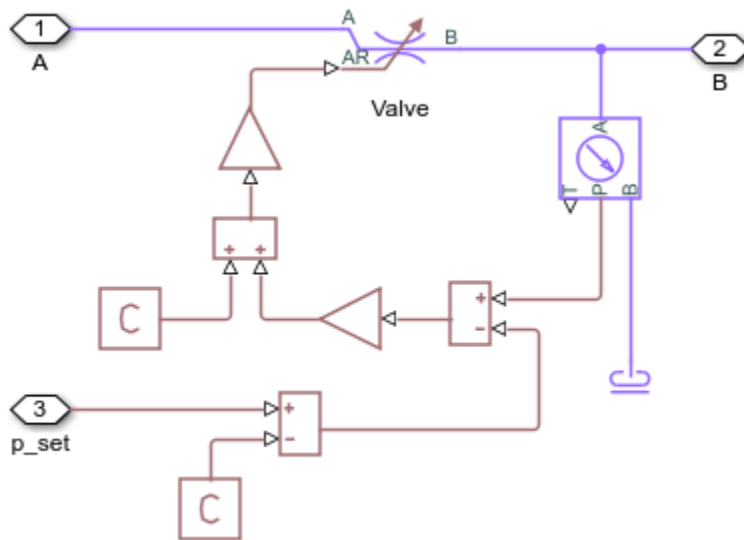


**Oxygen Source Subsystem**
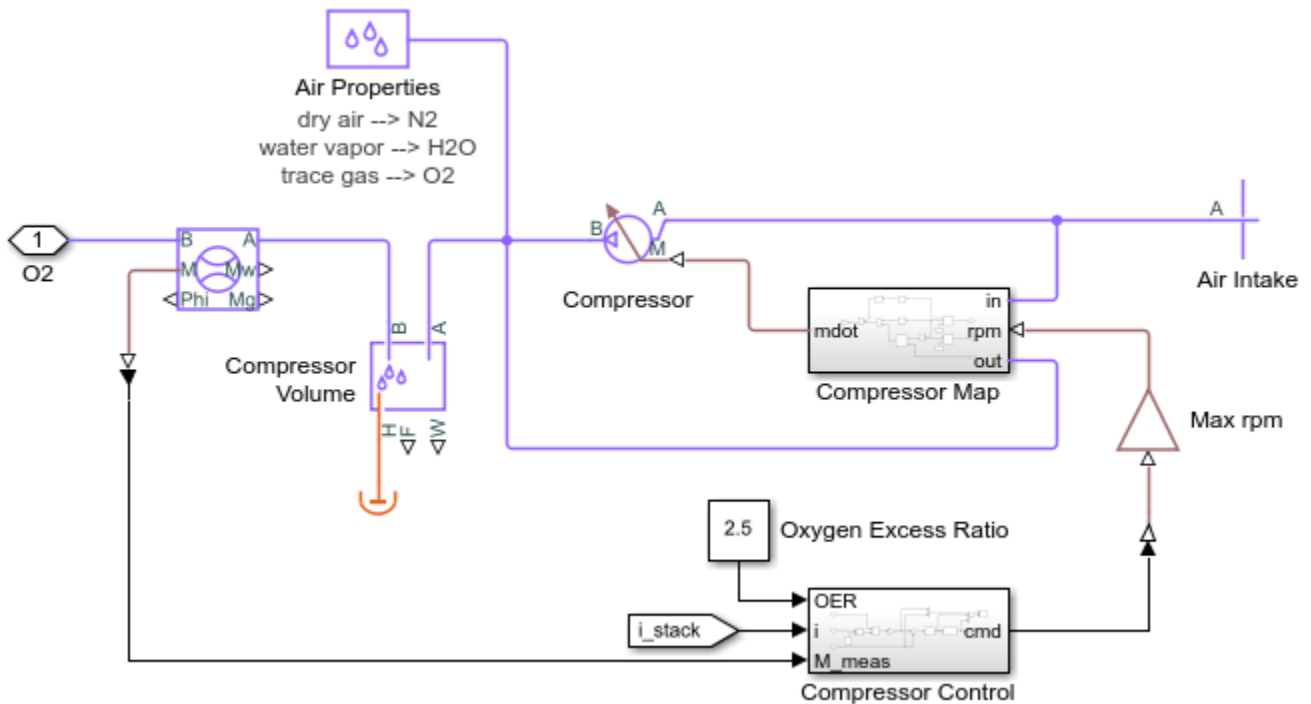
**Recirculation Subsystem**
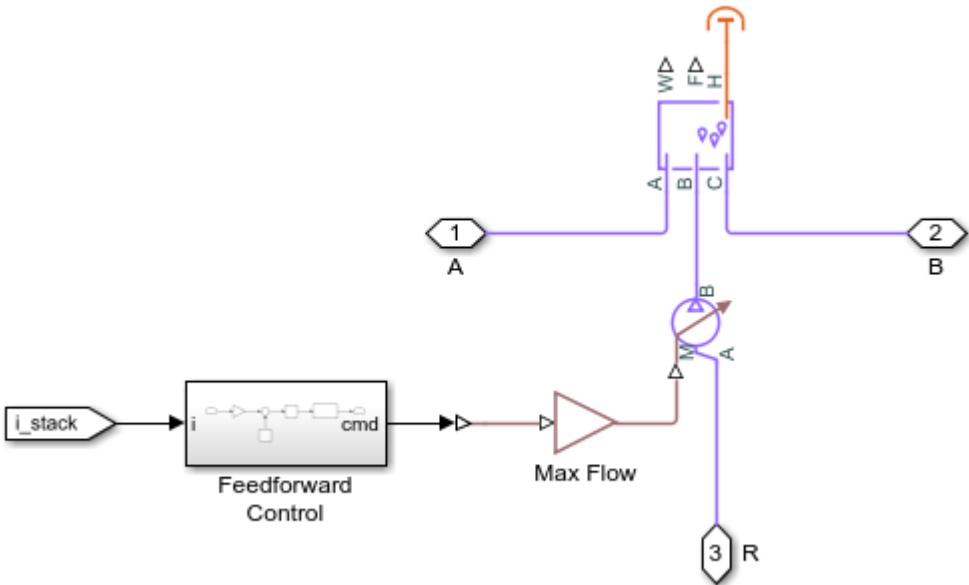
**Simulation Results from Scopes**
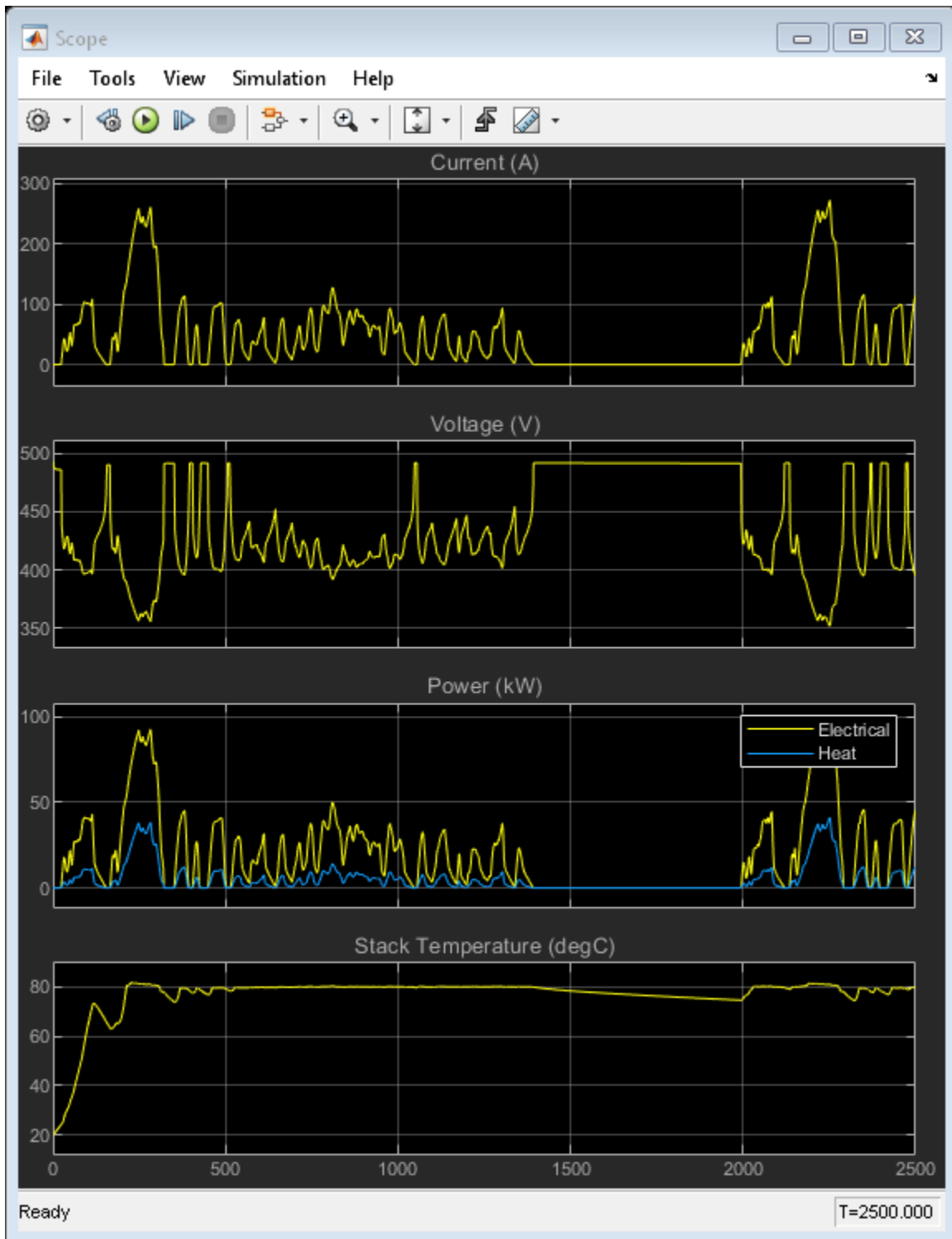
**Simulation Results from Simscape Logging**

This plot shows the current-voltage (i-v) curve of a fuel cell in the stack. As the current ramps up, an initial drop in voltage occurs due to electrode activation losses, followed by a gradual decrease in voltage due to Ohmic resistances. Near maximum current, a sharp drop in voltage occurs due to gas-transport-related losses.

This plot also shows the power produced by the cell. When the ramp scenario is selected, the power increases until a maximum power output, then decreases due to the high losses near maximum current.
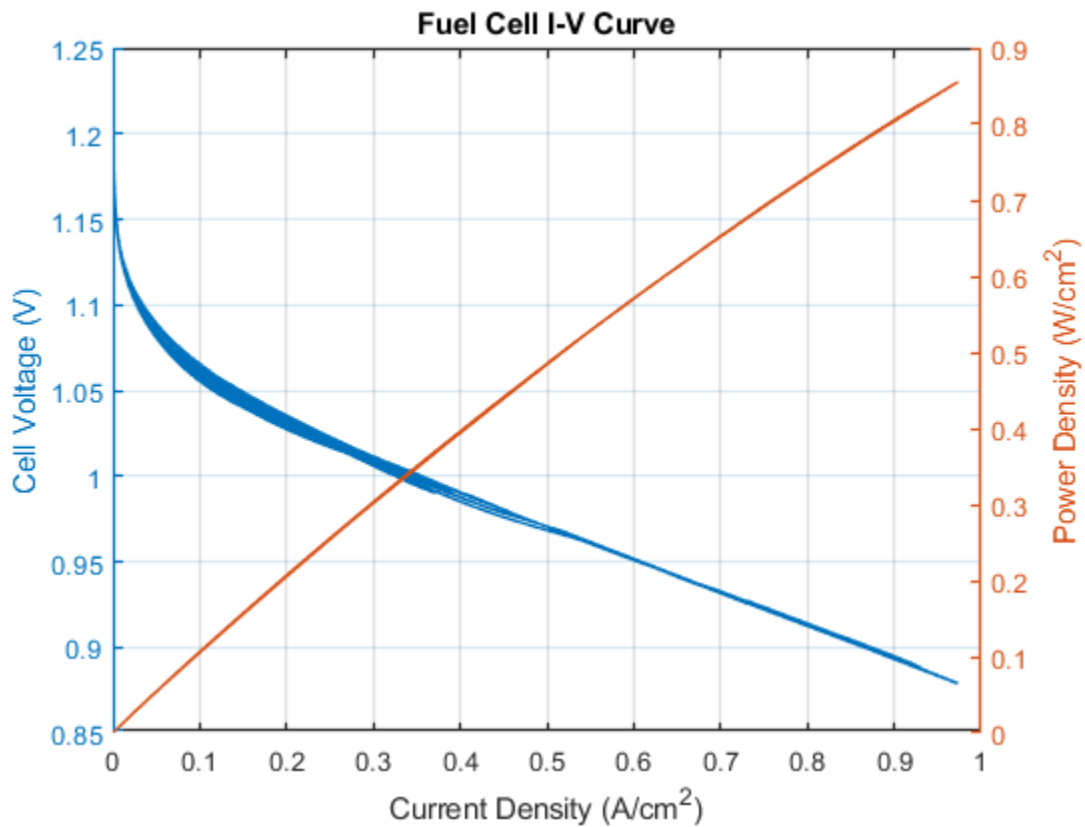


This plot shows the electrical power produced by the fuel cell stack as well as the power consumed by the cathode air compressor and the coolant pump to maintain stable and efficient system operation. As a result, the net power produced by the system is a few percent less than the power produced by the stack. Note that this model assumes an isentropic compressor. Accounting for compressor efficiency would decrease net power gain by another couple of percent.

This plot also shows the excess heat generated by the fuel cell stack, which must be removed by the cooling system. The maximum power produced by the fuel cell stack is 110 kW.
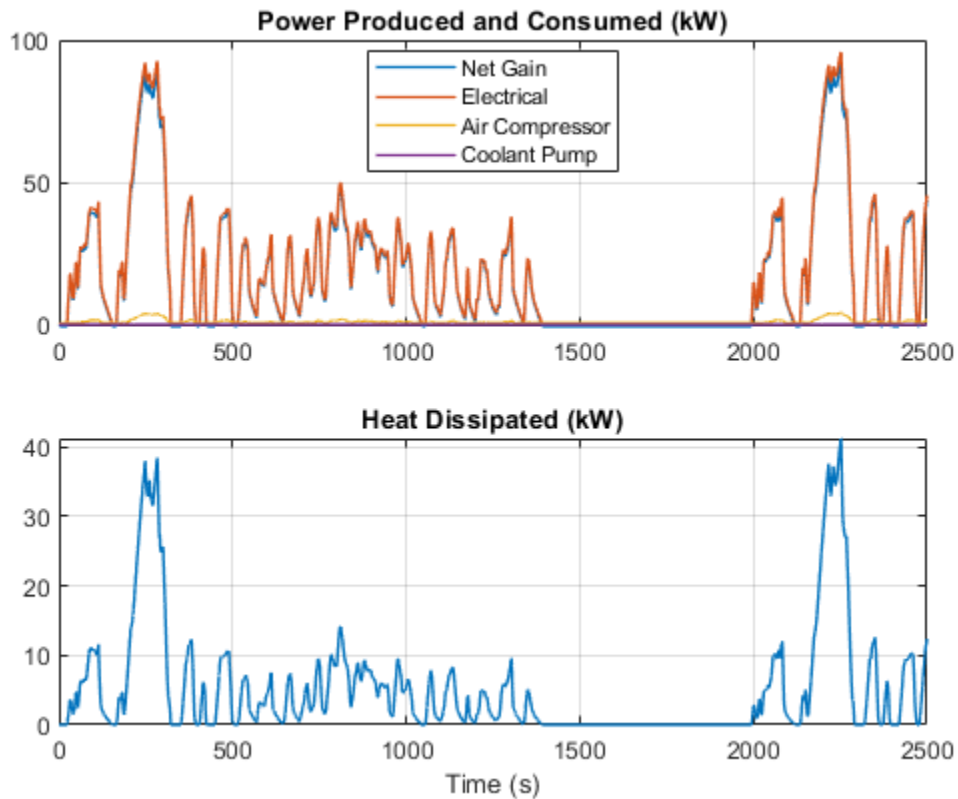
This plot show the thermal efficiency of the fuel cell and its reactant utilization fraction. The thermal efficiency indicates the fraction of the hydrogen fuel's energy that the fuel cell has converted to useful electrical work. The theoretical maximum efficiency for a PEM fuel cell is 83%. However, actual efficiency is around 60% due to internal losses. Near maximum current, the efficiency drops to around 45%.

The reactant utilization is the fraction of the reactants, H2 and O2, flowing into the fuel cell stack that has been consumed by the fuel cell. While higher utilization makes better use of the gases flowing through the fuel cell, it decreases the concentration of the reactants and thus reduces the voltage produced. Unconsumed O2 is vented to the environment, but unconsumed H2 is recirculated back to the anode to avoid waste. However, in practice, the H2 is periodically purged to remove contaminants.

**Thermal Efficiency (%)**



**Reactant Utilization (%)**
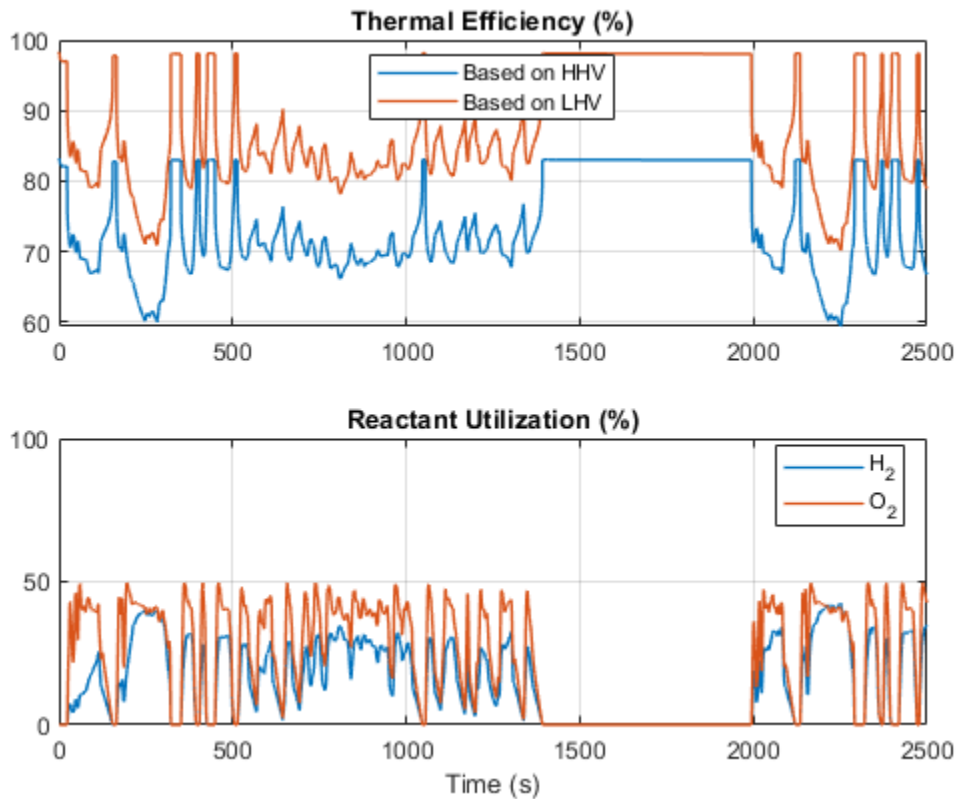


This plot shows the temperatures at various locations in the system. The fuel cell stack temperature is maintained at a maximum of 80 degC by the cooling system. Fuel flowing to the anode is warmed by the recirculated flow. Air flowing to the cathode is warmed by the compressor.

Maintaining an optimal temperature is critical to the operation of the fuel cell because higher temperatures lower the relative humidity which increases the membrane resistance. In this model, the cooling system is operated by a simple control of the coolant pump flow rate. The plot shows the temperature of the coolant after it has absorbed heat from the fuel cell stack and after it has rejected heat in the radiator.

**Temperature (degC)**

**Coolant Pump Mass Flow Rate (kg/s)**
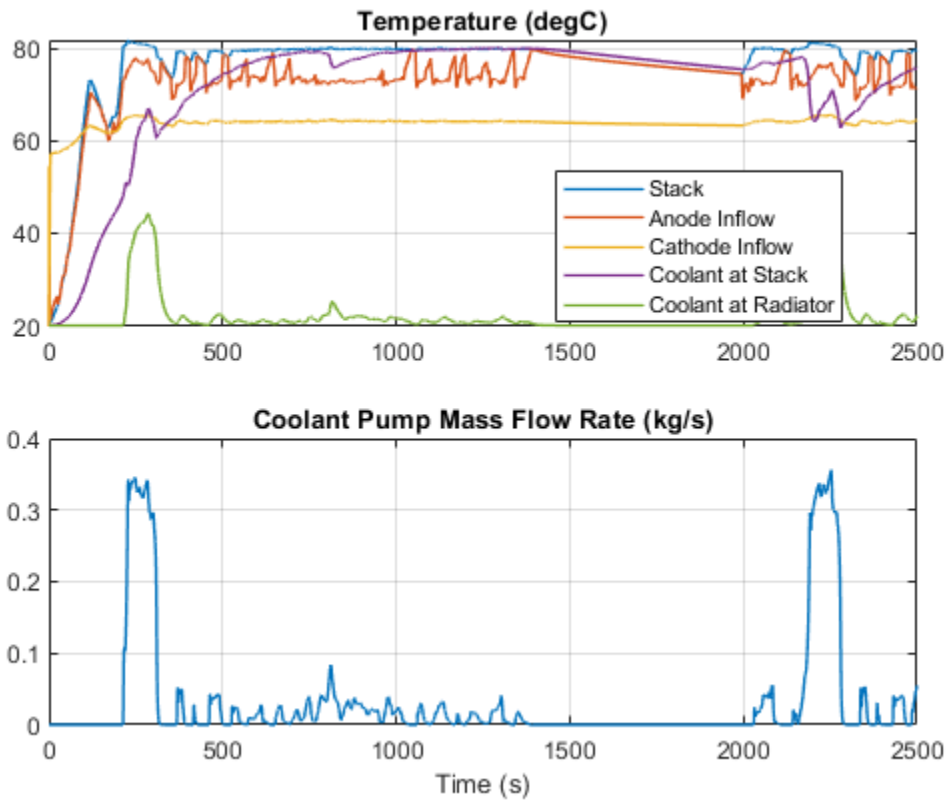
This plot shows the mass of hydrogen used during operation and the corresponding decrease in the hydrogen tank pressure. The energy of the consumed hydrogen fuel is converted to electrical energy.
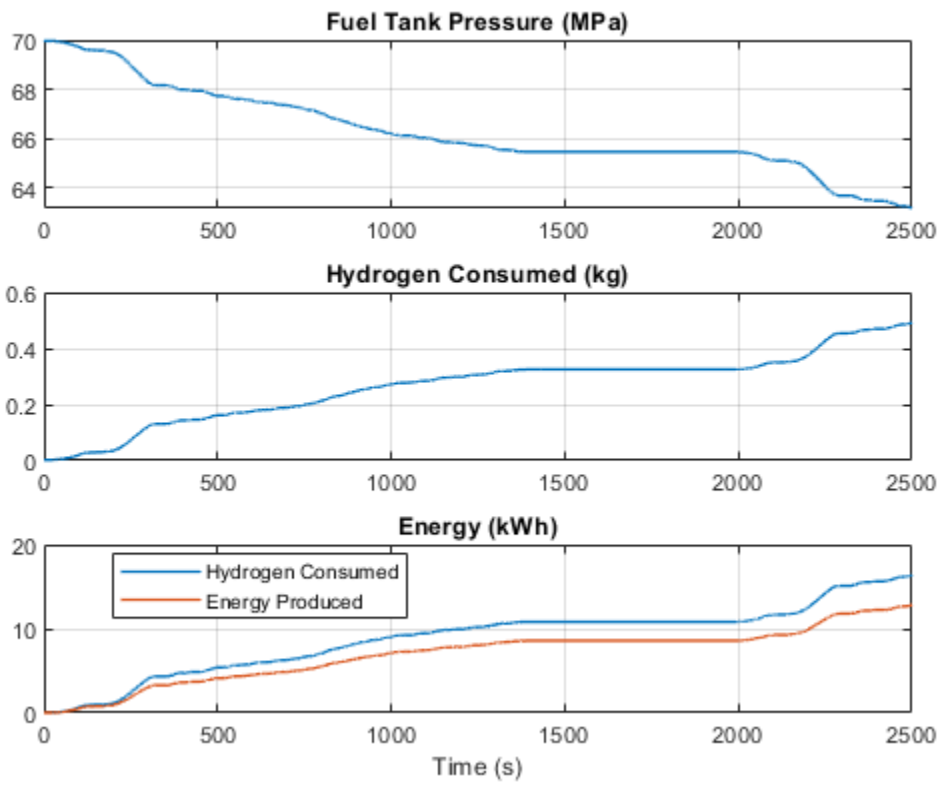
# PEM Electrolysis System

This example shows how to model a proton exchange membrane (PEM) water electrolyzer with a custom Simscape block. The PEM electrolyzer consumes electrical power to split water into hydrogen and oxygen. The custom block represents the membrane electrode assembly (MEA) and is connected to a thermal liquid network and two separate moist air networks: the thermal liquid network models the water supply, the anode moist air network models the oxygen flow, and the cathode moist air network models the hydrogen flow.

The circulation pump provides a continuous flow of water to the anode side of the electrolyzer. Consumed water is removed from the thermal liquid network and excess water is recirculated. Oxygen produced at the anode is carried away by the excess water flow; it is modeled separately by the anode moist air network. The separator tank models the balance of water and oxygen in the return flow before the oxygen is vented. The supply pump replenishes the system with fresh water.

Hydrogen produced at the cathode side along with any water that has been transported across the MEA is modeled by the cathode moist air network. The dehumidifier removes the unwanted water vapor from the hydrogen. A pressure regulator valve maintains a pressure of 3 MPa at the cathode in comparison to atmospheric pressure at the anode. The differential pressure across the MEA results in water transport due to hydraulic pressure that helps counteract the electro-osmosis drag and reduces the amount of water at the cathode side.

Unlike a fuel cell stack, a separate cooling network is not needed. Heat dissipated by the electrolyzer is carried away by the excess water and then rejected to the environment via the heat exchanger. The recirculating water is controlled to maintain a temperature of 80 degC in the electrolylzer.

The custom MEA block is implemented in the Simscape code `Electrolyzer.ssc`. The thermal liquid port H2O is used to remove water from the thermal liquid network. The produced H2 and O2 and the transported H2O are added to the two moist air networks using Controlled Trace Gas Source (MA) and Controlled Moisture Source (MA) blocks. The excess heat is sent through the thermal port H to the connected Thermal Mass block. Refer to the comments in the code for additional details on the implementation.

See also the PEM Fuel Cell System example.

References:

Liso, Vincenzo, et al. "Modelling and experimental analysis of a polymer electrolyte membrane water electrolysis cell at different operating temperatures." *Energies* 11.12 (2018): 3273.

Mo, Jingke, et al. "Thin liquid/gas diffusion layers for high-efficiency hydrogen production from water splitting." *Applied Energy* 177 (2016): 817-822.

## Model
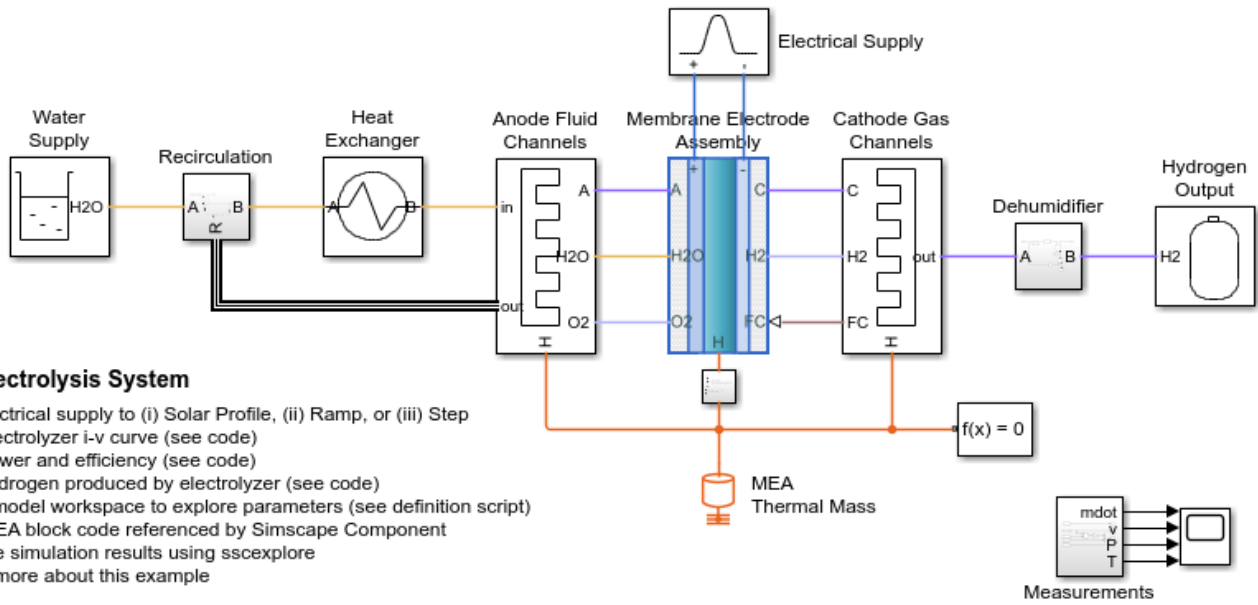


### PEM Electrolysis System

1. Set electrical supply to (i) Solar Profile, (ii) Ramp, or (iii) Step
2. Plot electrolyzer i-v curve (see code)
3. Plot power and efficiency (see code)
4. Plot hydrogen produced by electrolyzer (see code)
5. Open model workspace to explore parameters (see definition script)
6. See MEA block code referenced by Simscape Component
7. Explore simulation results using sscexplore
8. Learn more about this example

Copyright 2021 The MathWorks, Inc.

## Anode Fluid Channels Subsystem
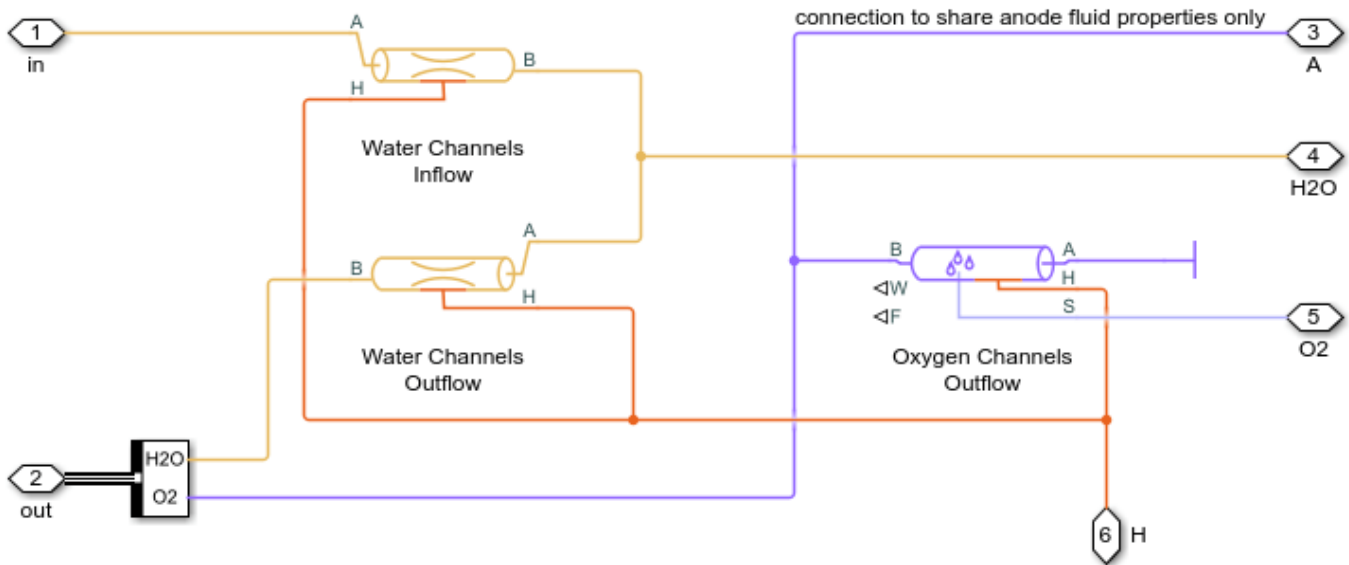
## Cathode Gas Channels Subsystem
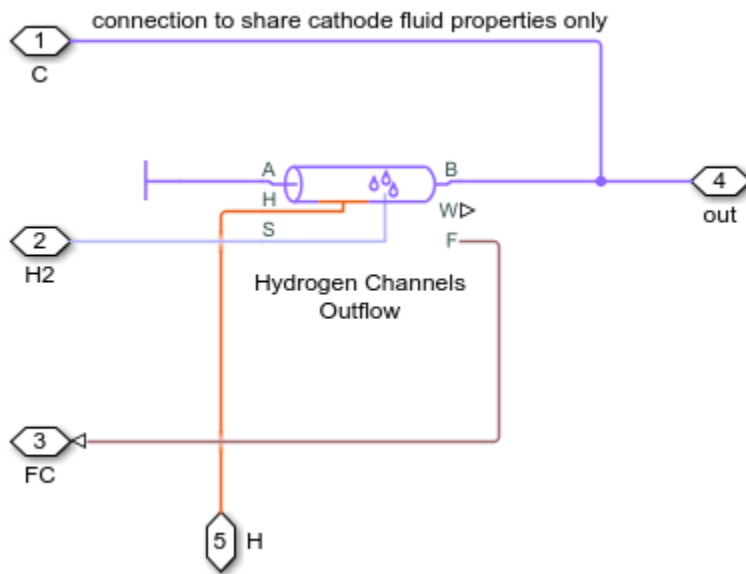


## Dehumidifier Subsystem
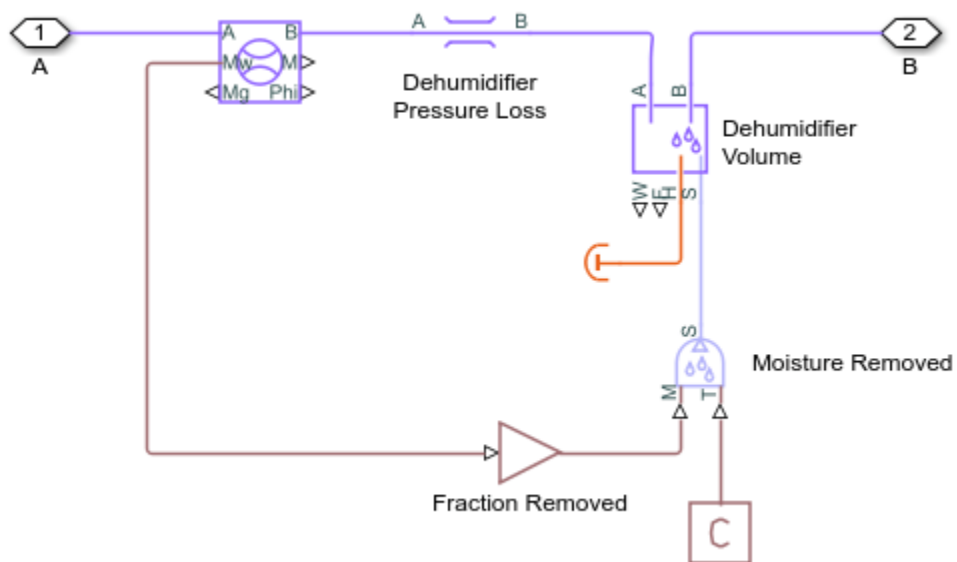
**Electrical Supply Subsystem**

Inputs

**Heat Exchanger Subsystem**

Heat Exchanger

Environment
Temperature
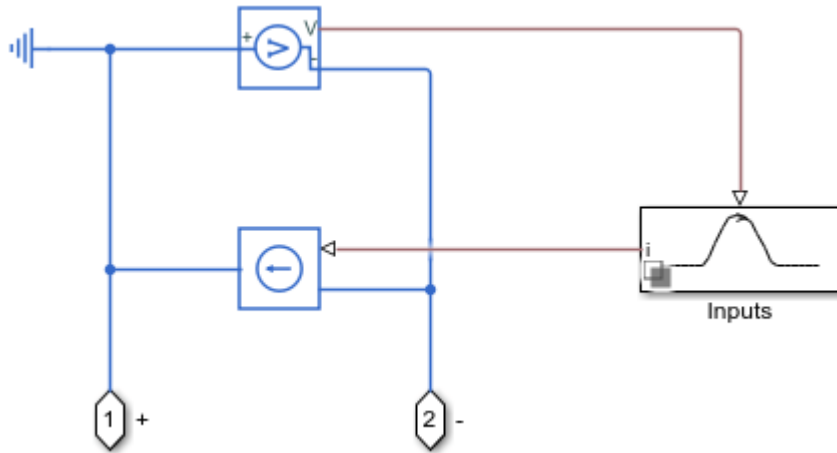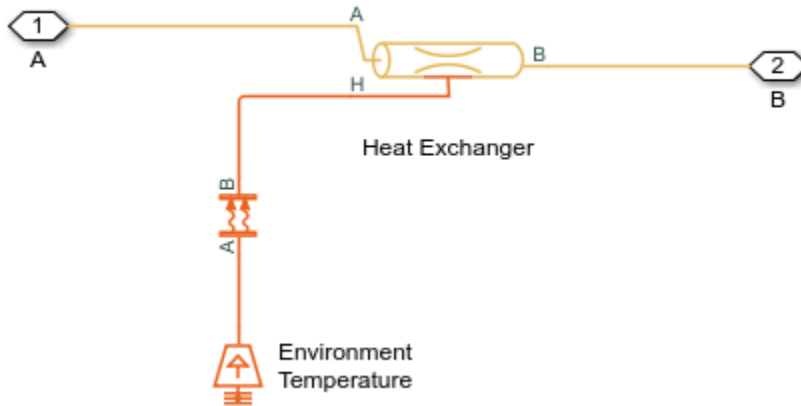
**Hydrogen Output Subsystem**

Pressure
Regulator

Valve
Area

Hydrogen Properties
dry air --> N2
water vapor --> H2O
trace gas --> H2

mdot_H2

Hydrogen
Reservoir

Setpoint [MPa]
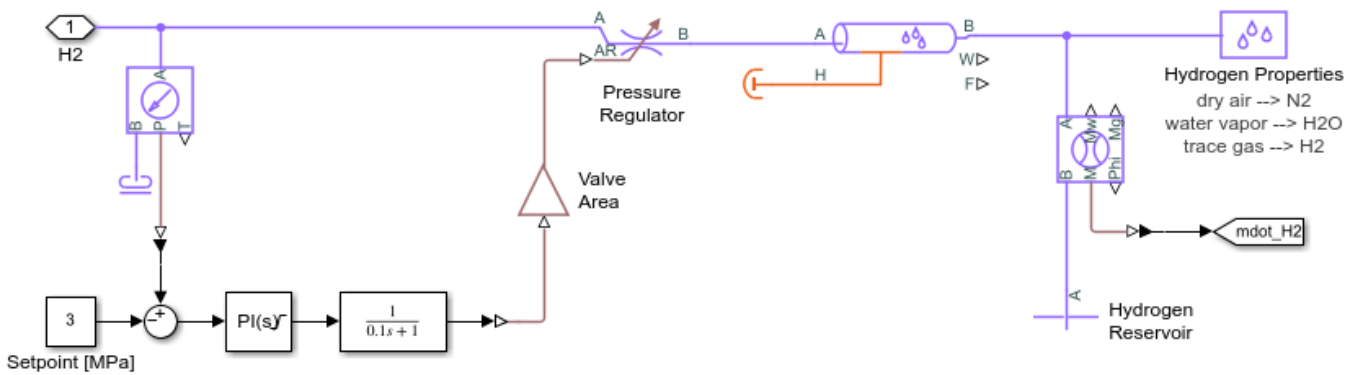
**Recirculation Subsystem**
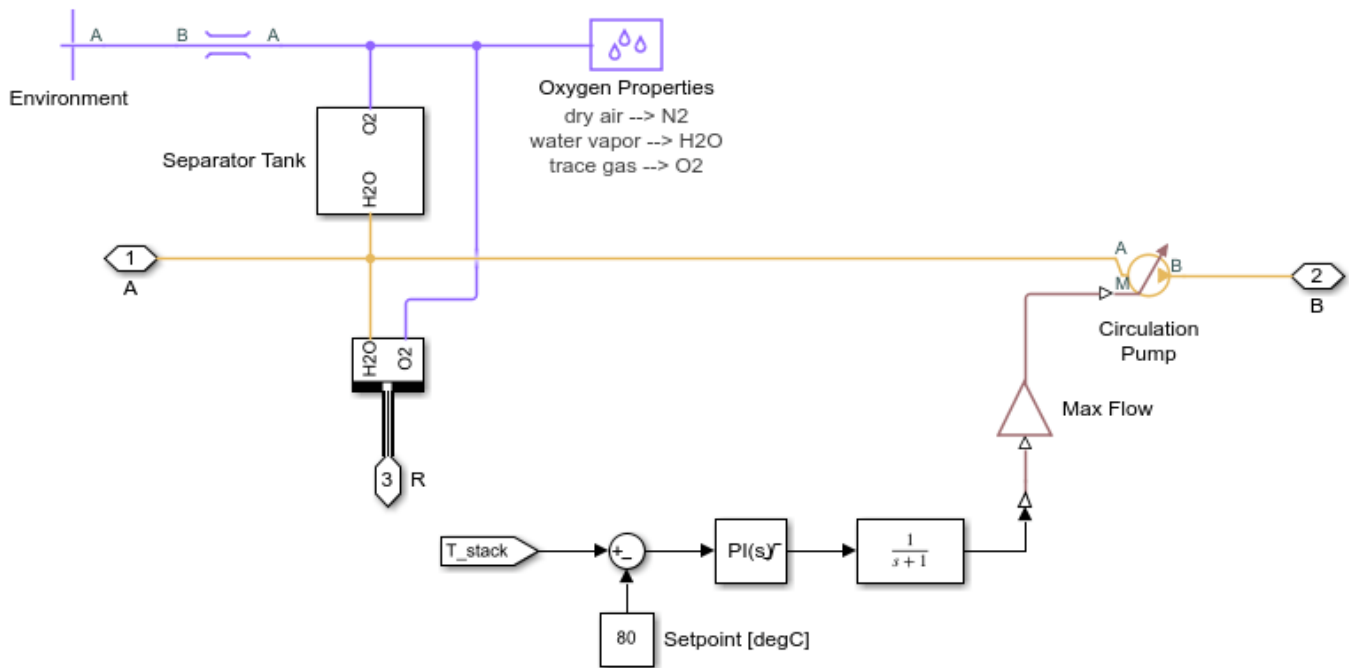


**Separator Tank Subsystem**
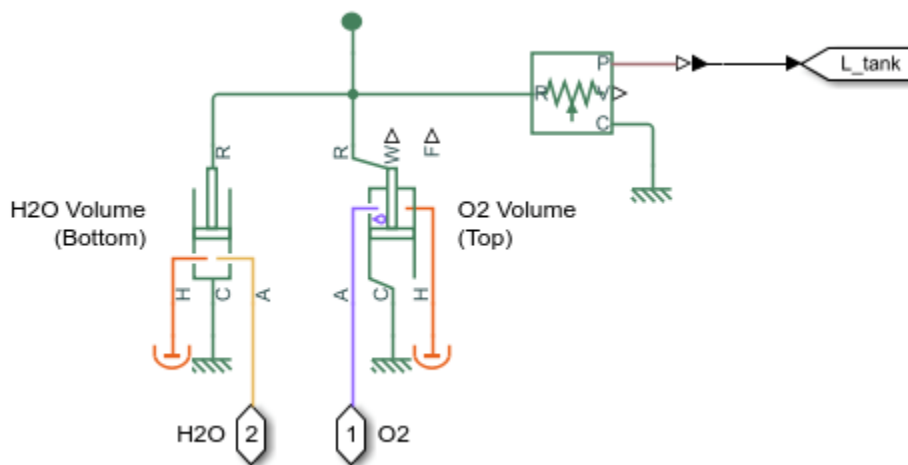
**Water Supply Subsystem**
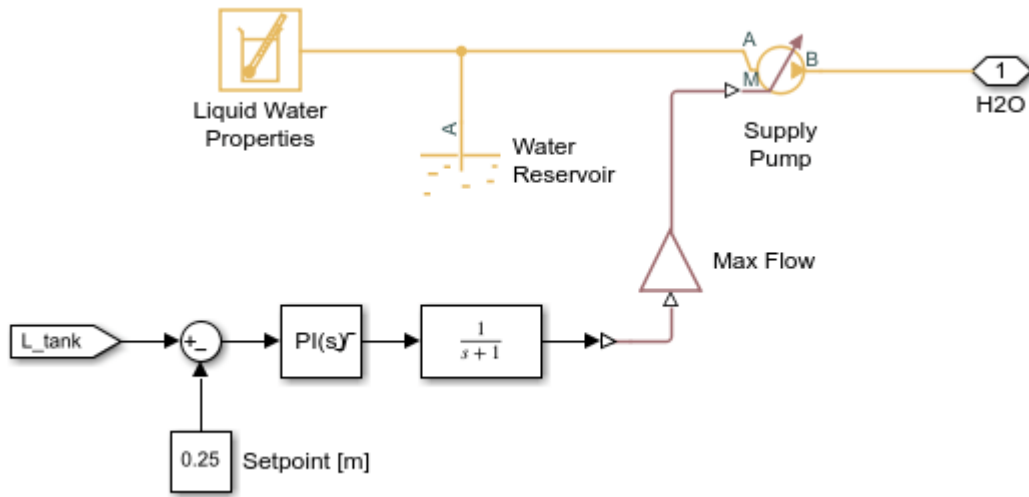
**Simulation Results from Scopes**
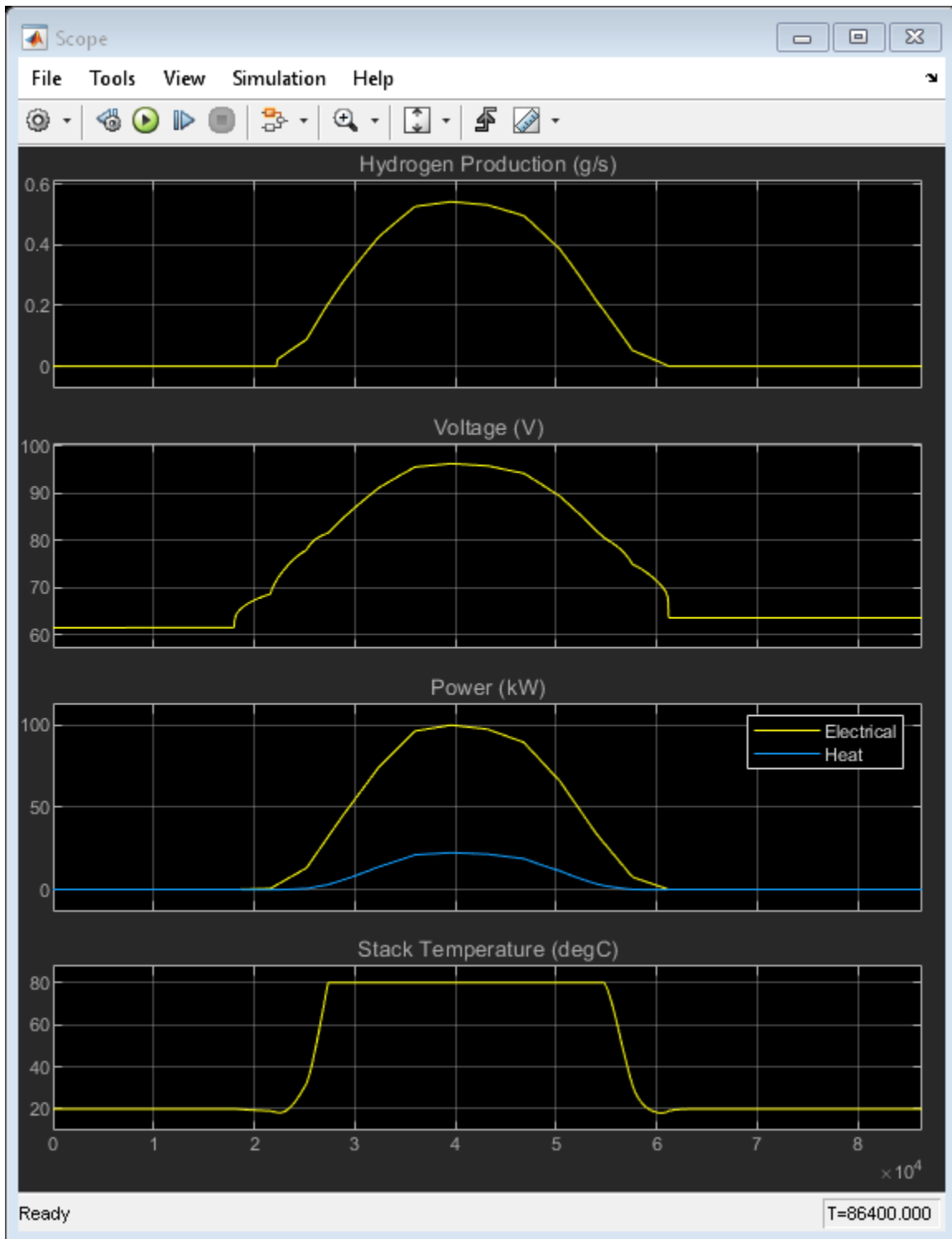
**Simulation Results from Simscape Logging**

This plot shows the current-voltage (i-v) curve and the power consumed by a cell in the stack. As the current ramps up, an initial rise in voltage occurs due to electrode activation losses, followed by a gradual increase in voltage due to Ohmic resistances. The cell voltage is about 1.71 V at a current density of 2 A/cm^2.
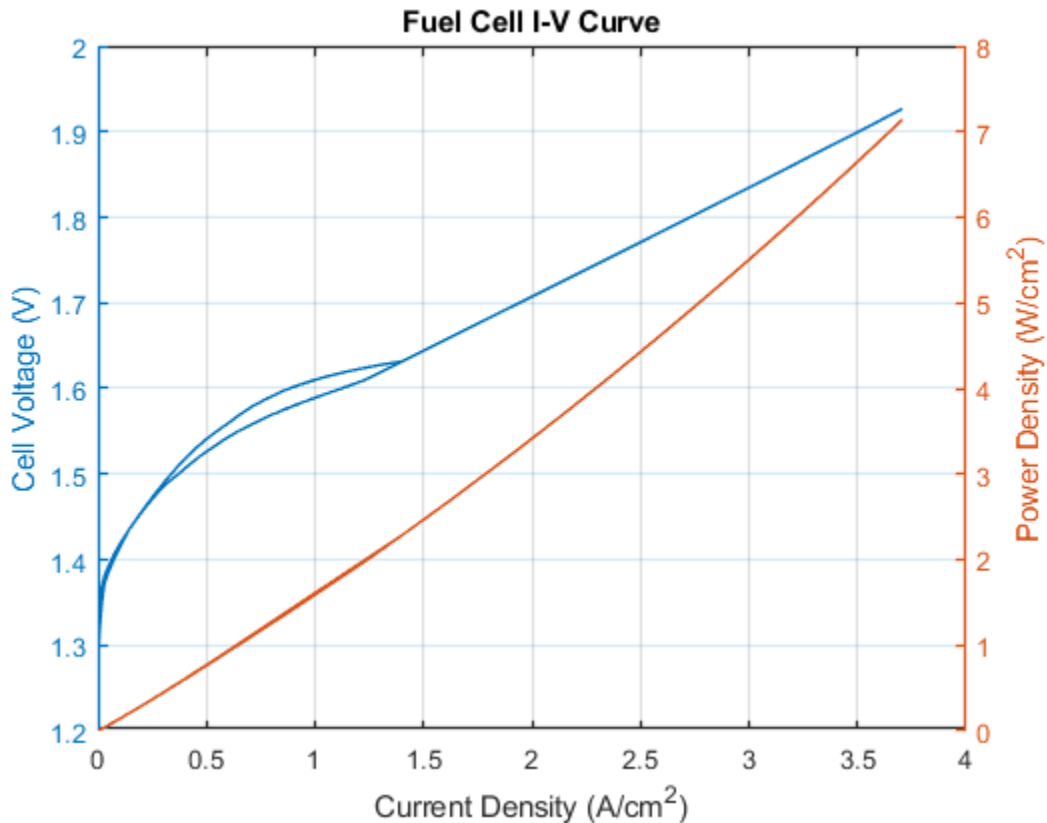


This plot shows the electrical power consumed by the electrolyzer. The electrical power is greater than the power needed to produce hydrogen due to various losses. The difference is the heat dissipated.

This plot also shows the thermal efficiency of the electrolyzer, which indicates the fraction of the electrical power used to generate hydrogen based on hydrogen's heating value. This electrolyzer is about 87% efficient at a current density of 2 A/cm^2.
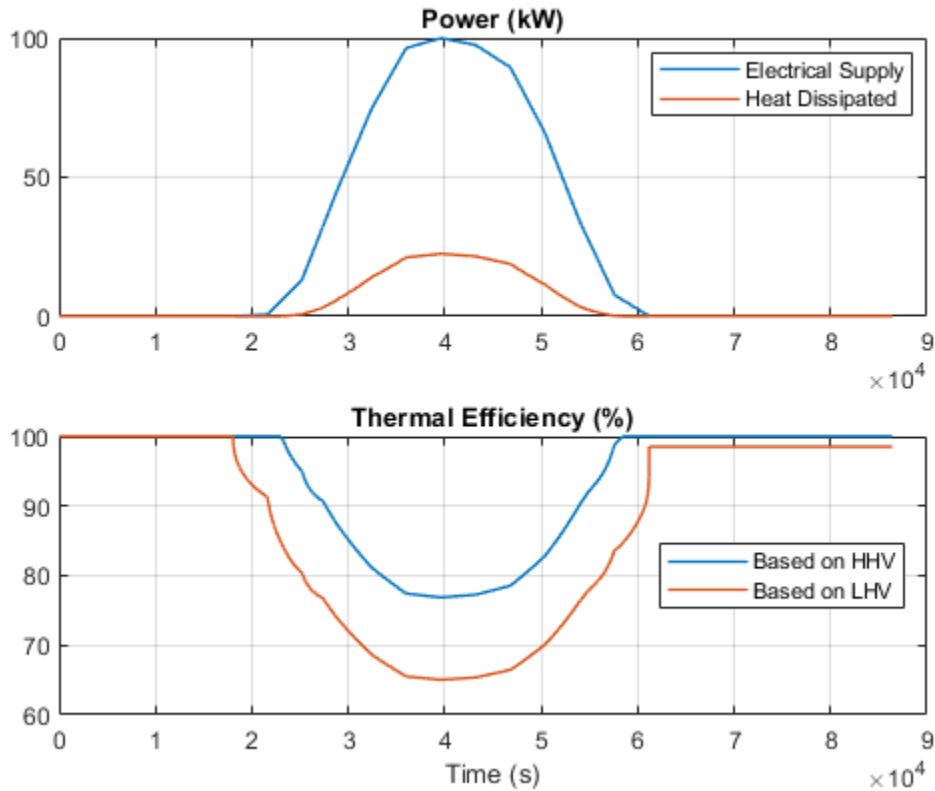
This plot shows the rate of hydrogen produced, the rate of water consumed at the anode, as well as the rate of water transported to the cathode due to diffusion, electro-osmosis drag, and hydraulic pressure difference. As a result, aadehumidificationa step is necessary to produce hydrogen at the desired purity.

This plot also shows the total mass of hydrogen produced and the equivalent energy based on its higher heating value. This provides an indication of the amount of energy available if the hydrogen is used to generate power in a fuel cell.
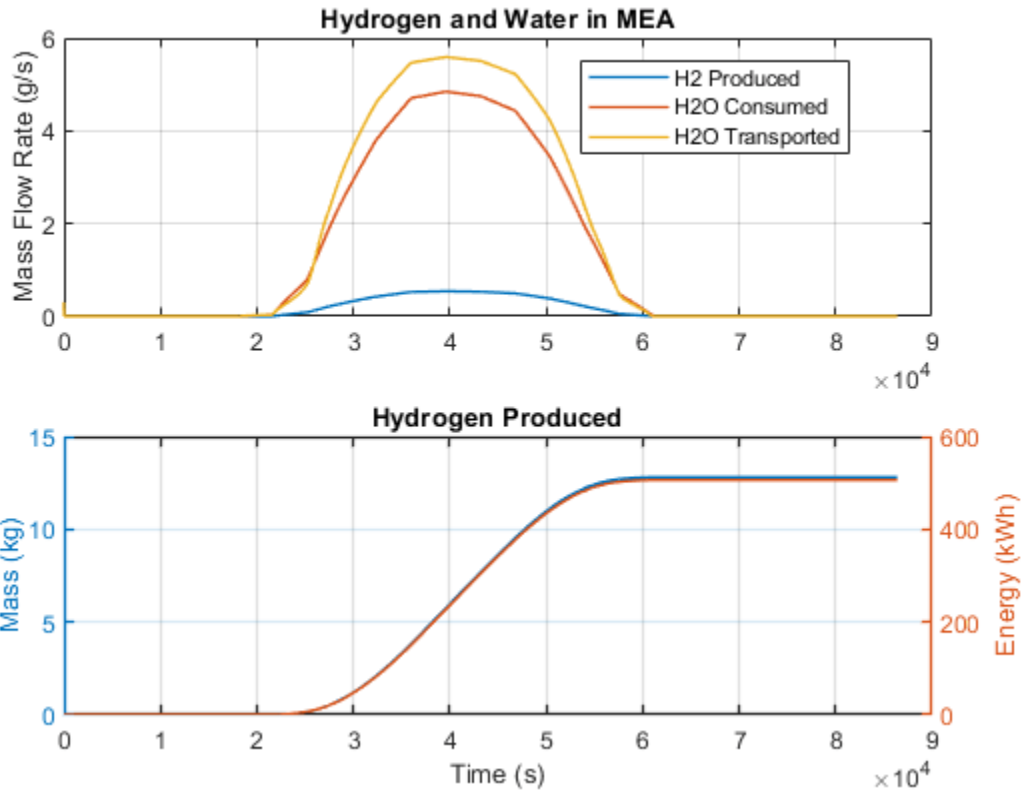
# Medical Ventilator with Lung Model

This example models a positive-pressure medical ventilator system. A preset flow rate is supplied to the patient. The lungs are modeled with the Translational Mechanical Converter (MA), which converts moist air pressure into translational motion. By setting the Interface cross-sectional area to unity, displacement in the mechanical translational network becomes a proxy for volume, force becomes a proxy for pressure, spring constant becomes a proxy for respiratory elastance, and damping coefficient becomes a proxy for respiratory resistance.

The exchange of oxygen and carbon dioxide in the moist air mixture is not currently modeled.

**Model**



**Medical Ventilator with Lung Model**

1. Plot temperature and humidity in ventilation tubes (see code)
2. Plot condensation in expiratory tube (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example
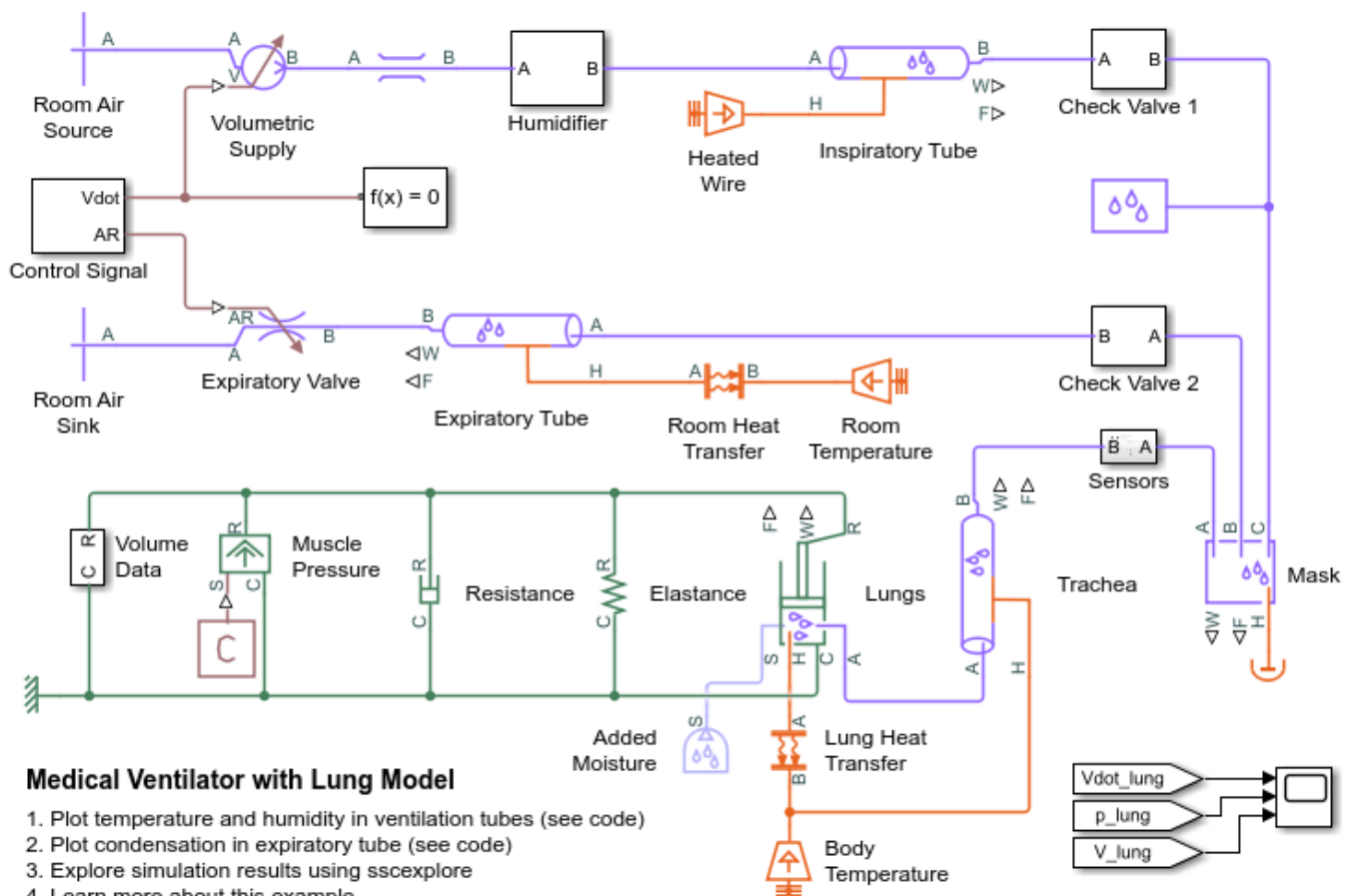
Copyright 2017-2021 The MathWorks, Inc.

**Check Valve 1 Subsystem**
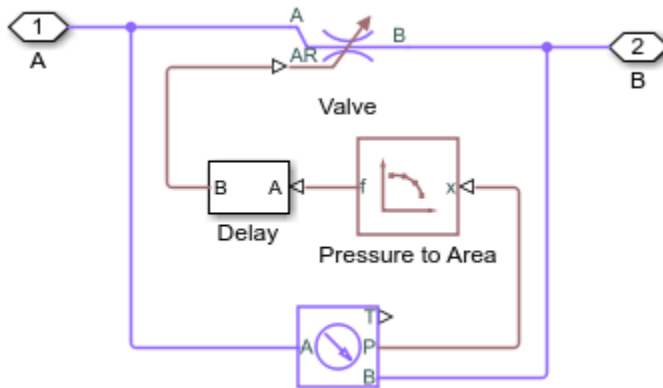


**Control Signal Subsystem**
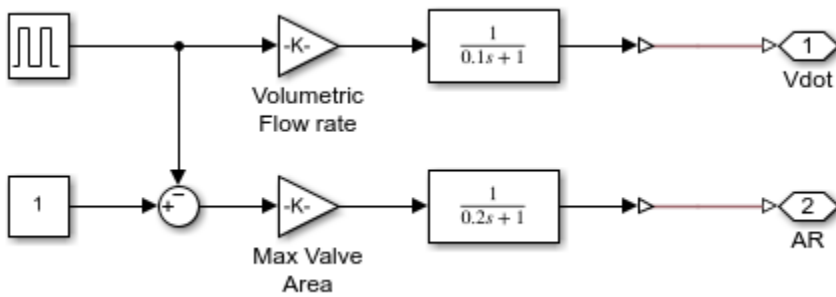
## Humidifier Subsystem
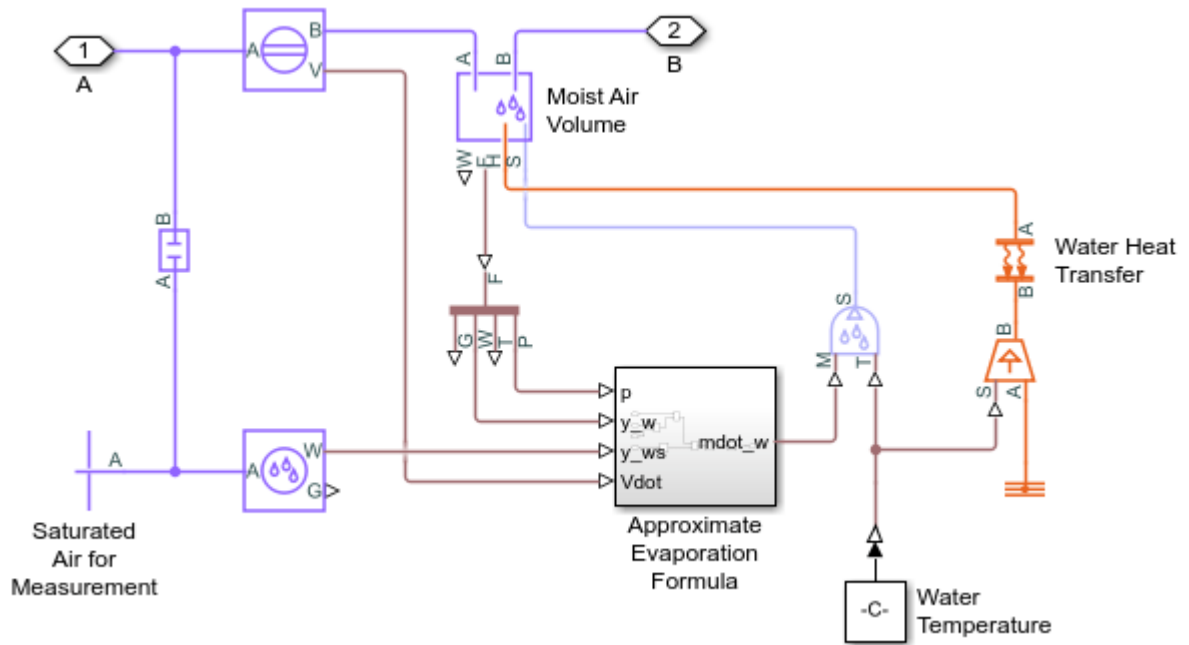
**Simulation Results from Scopes**
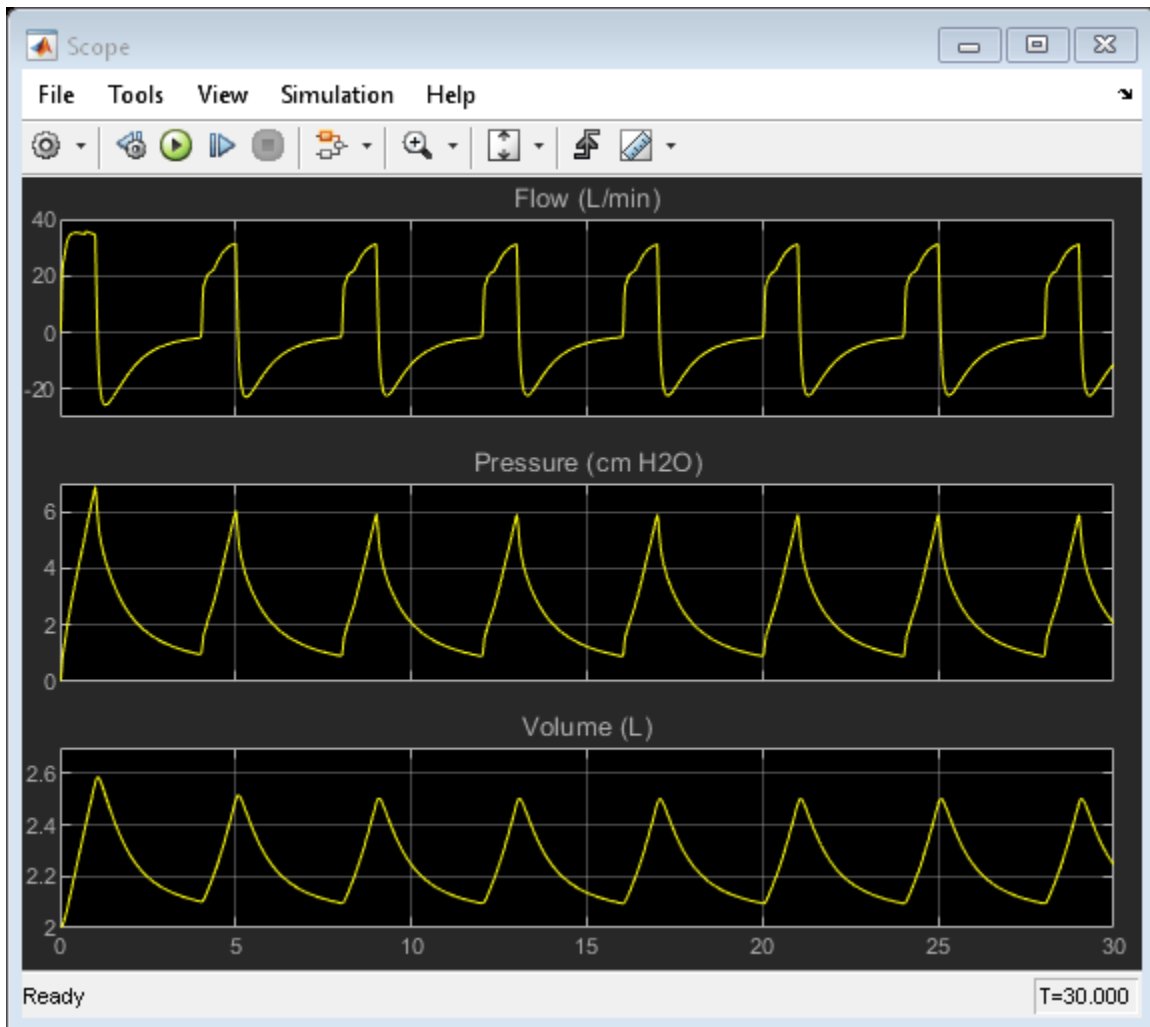


**Simulation Results from Simscape Logging**

This plot shows the temperature and relative humidity of air flowing through the inspiratory and expiratory tubes. Room air is heated and humidified before it is supplied to the patient.
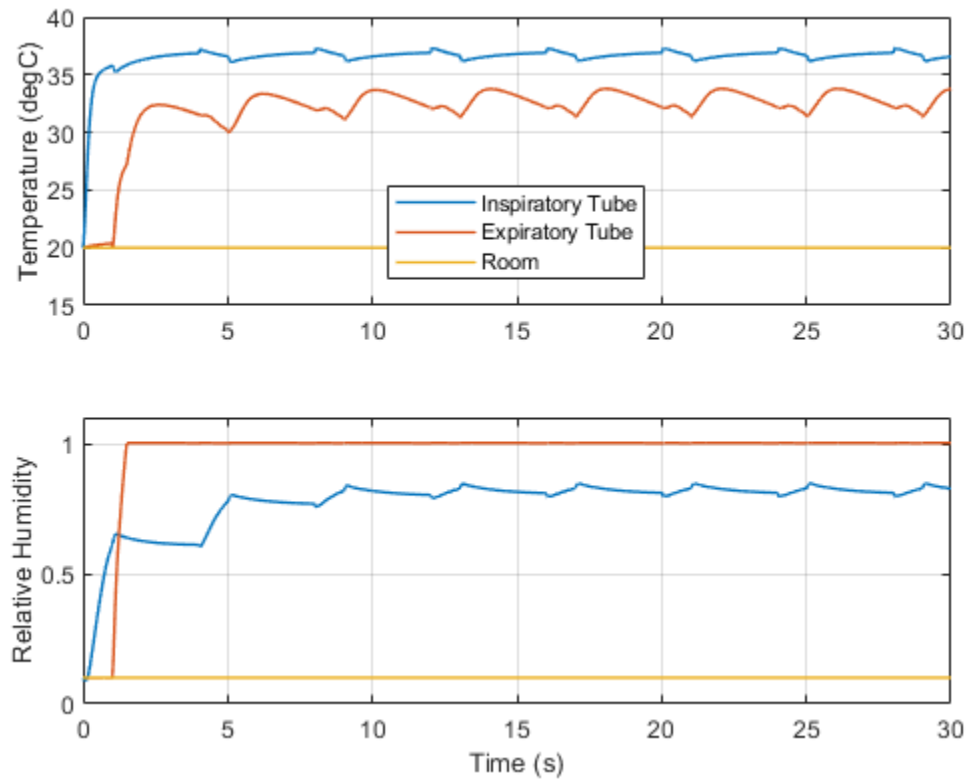
This plot shows the accumulation of condensed water in the expiratory tube, which should be drained periodically. The water comes from the humidifier and the patient's respiration.

Condensate Accumulation in Expiratory Tube
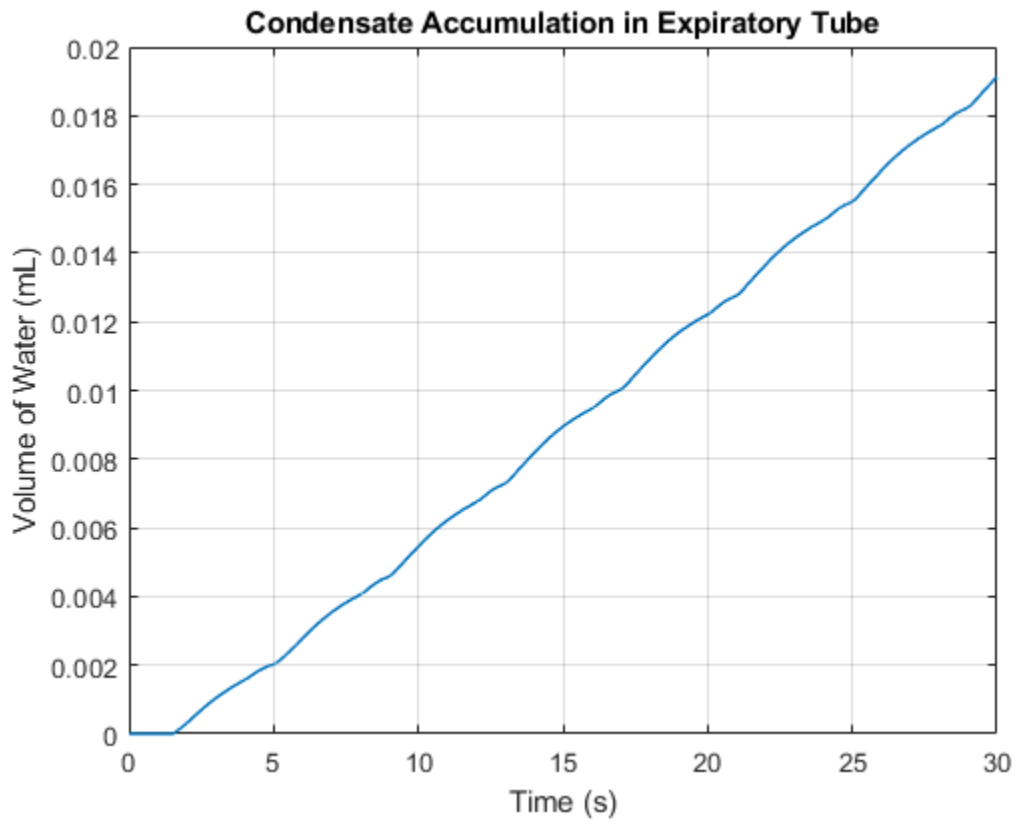
# Oxygen Concentrator

This example models an oxygen concentrator device coupled to a lung model. One of the two sieves filters out nitrogen from the air to produce concentrated oxygen in the product tank. The two sieves switches periodically so that while one sieve is filtering, the other can purge the adsorbed nitrogen. When the lung model inhales, some of the oxygen-rich gas from the product tank is mixed into the inspiratory flow.

This model shows one use case of modifying the default properties in the Moist Air Properties (MA). The default "dry air" has been replaced with oxygen and the default "trace gas" has been replaced with nitrogen. This way, the Controlled Trace Gas Source (MA) block can be used to remove "trace gas" (i.e., nitrogen) from the flow through the sieve.

The lungs are represented by a Translational Mechanical Converter (MA), which converts pressure into translational motion. By setting the Interface cross-sectional area to unity, displacement in the mechanical translational network becomes a proxy for volume changes, force becomes a proxy for pressure, spring constant becomes a proxy for respiratory elastance, and damping coefficient becomes a proxy for respiratory resistance.

The device has two modes of operation: continuous or pulsating. The simulation starts in continuous mode, which delivers constant oxygen-rich flow to the lung model. At t = 70 s, the simulation switches to pulsating mode, which synchronizes oxygen delivery with inspiration. State Flow™ is used to estimate the breaths per minute and to control the conserving valve in the device.

**Model**
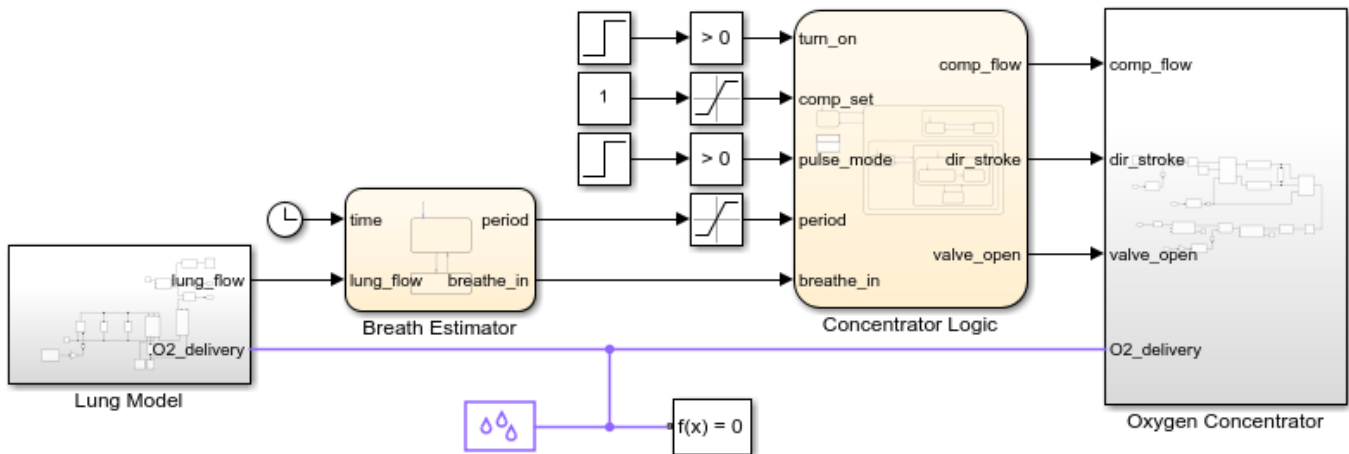


## Oxygen Concentrator

1. Plot average air volume delivered by the device (see code)
2. Plot pressure and volume of the lung model (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 2018-2021 The MathWorks, Inc.
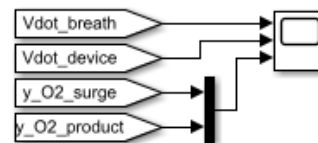
**Breath Estimator Subsystem**
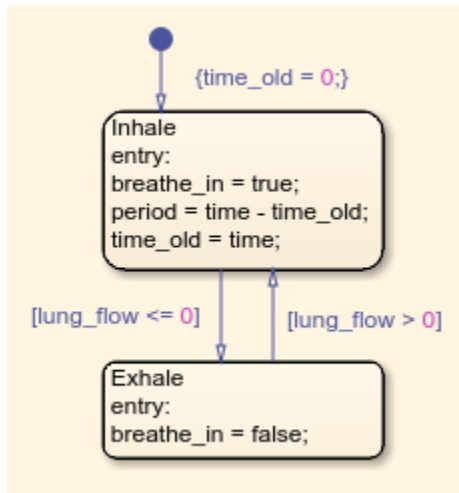


**Concentrator Logic Subsystem**
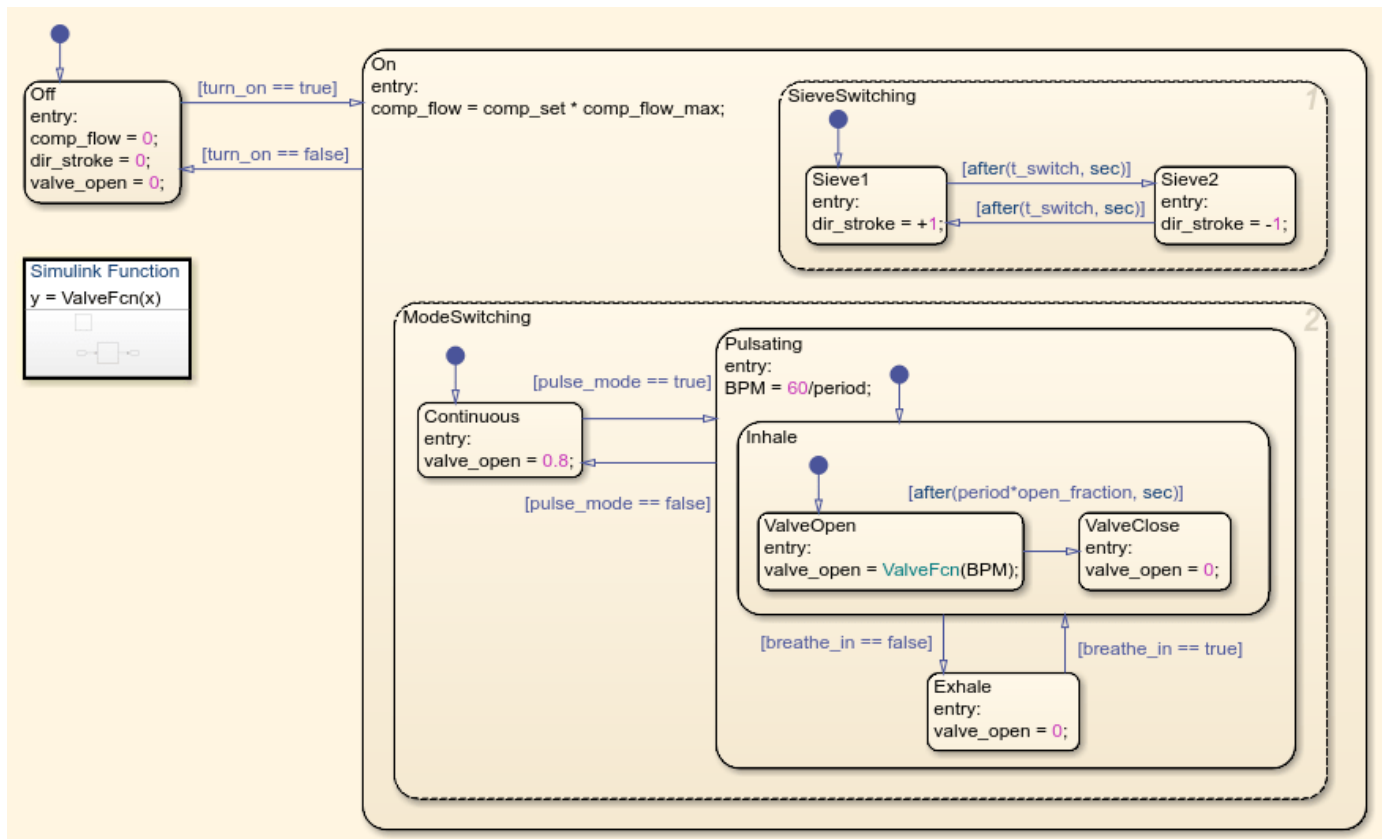
**Lung Model Subsystem**
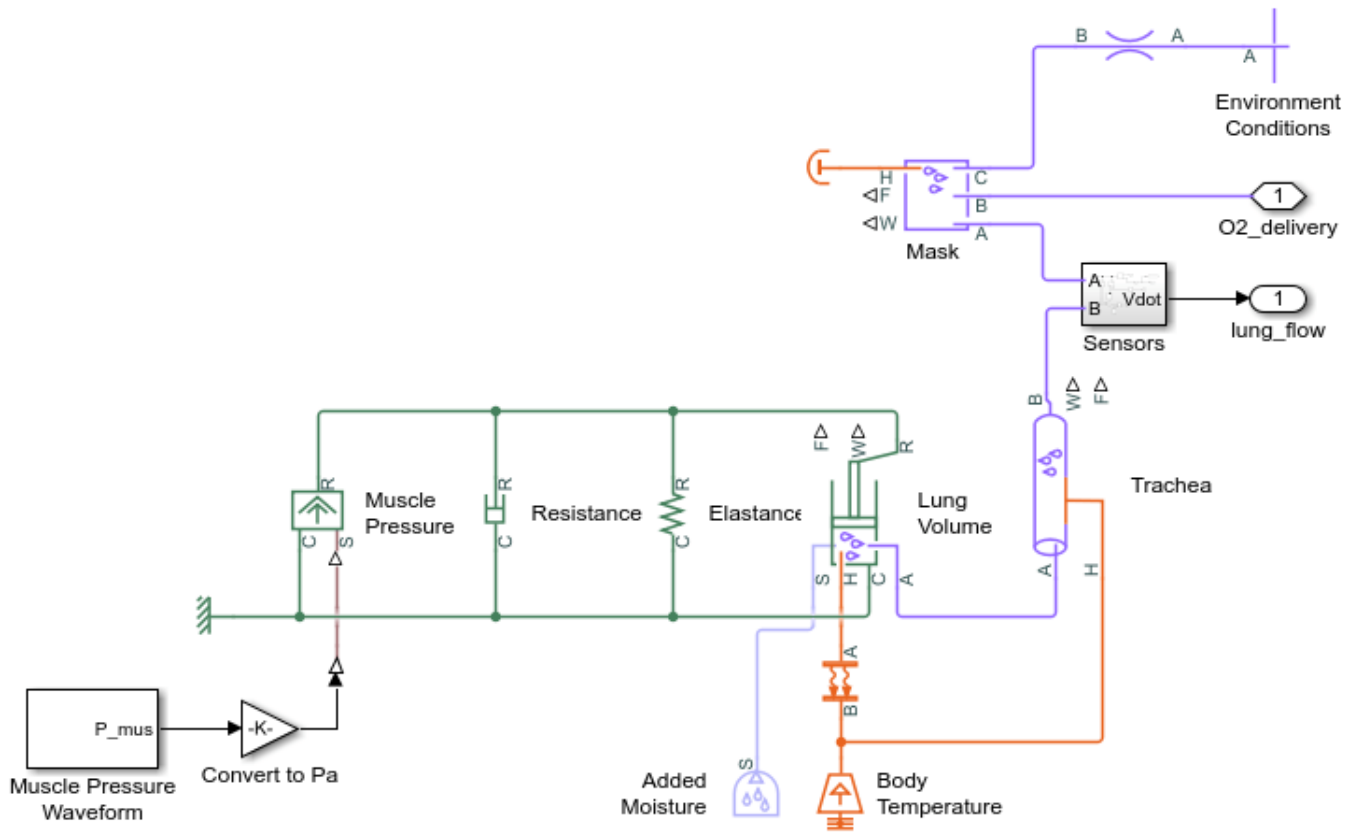
## Oxygen Concentrator Subsystem
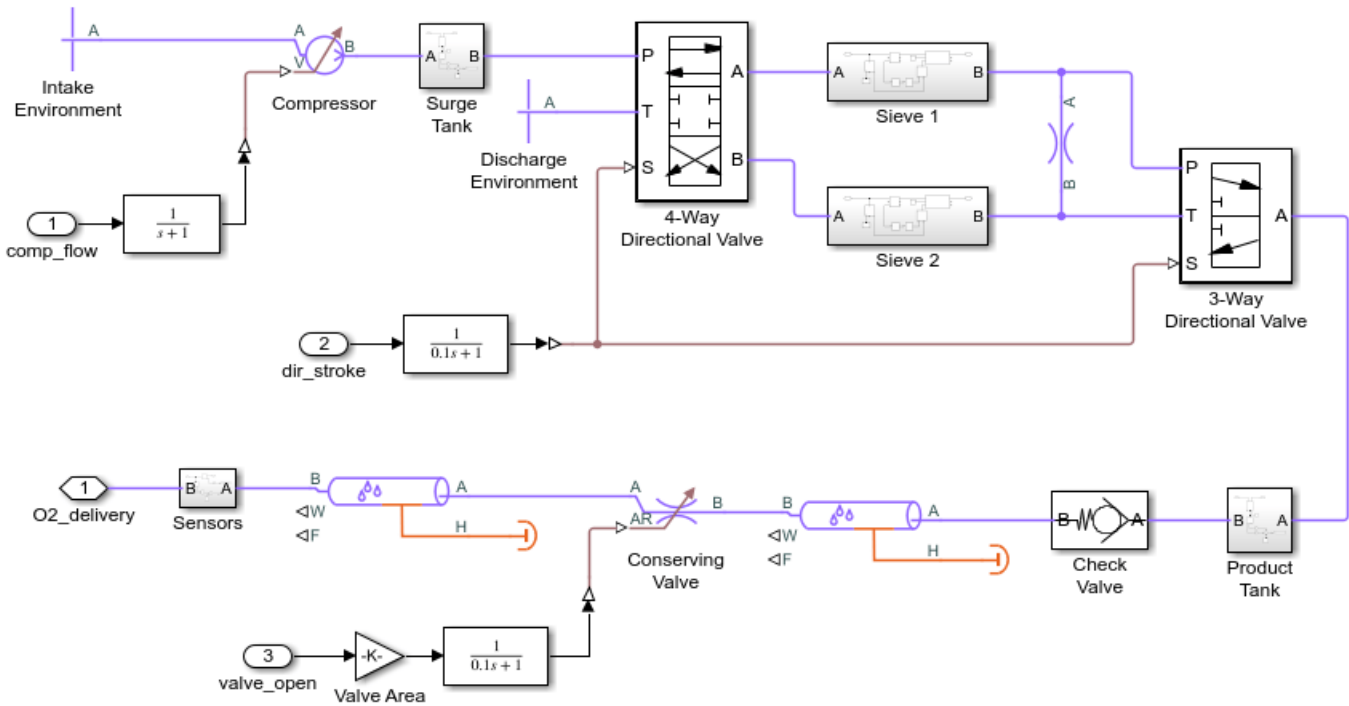


## Sieve 1 Subsystem
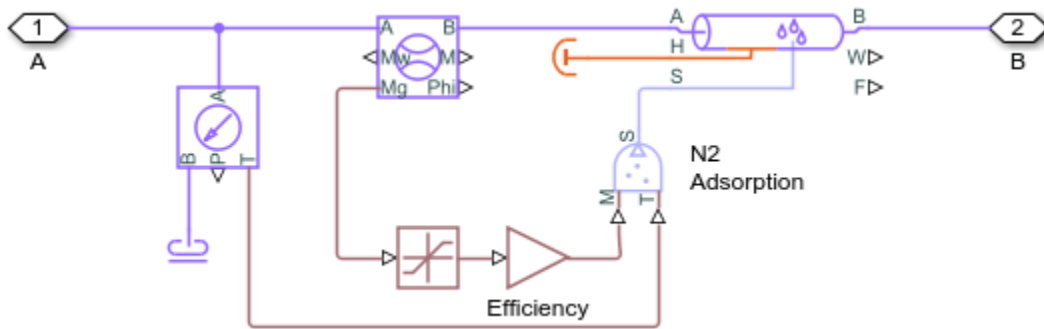
**Simulation Results from Scopes**
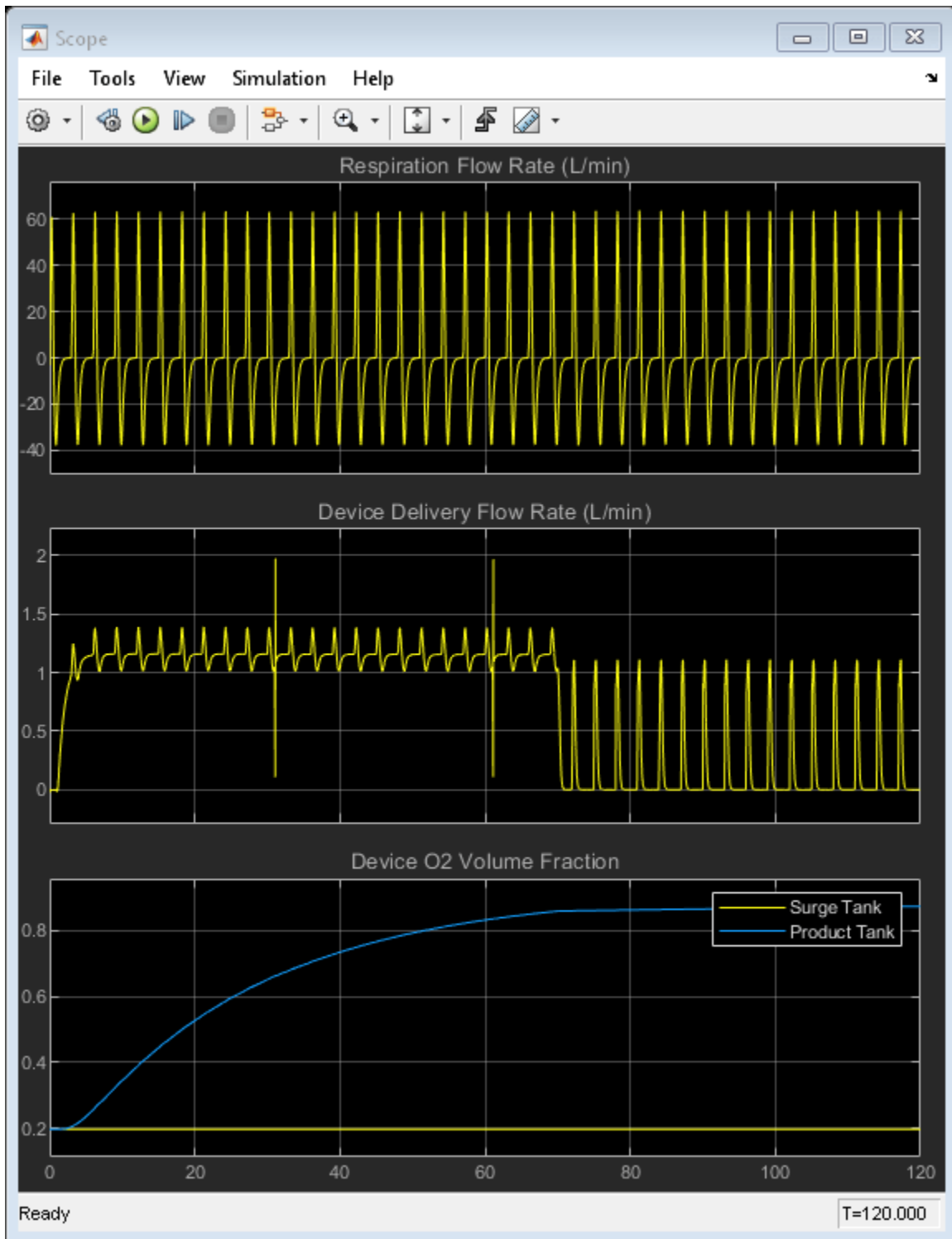
**Simulation Results from Simscape Logging**

This plot shows volume delivered by the device calculated based on a moving average flow rate. During continuous mode, the device delivers about 1.2 L/min. During pulsating mode, because flow is only delivered during inhalation, the device delivers about 0.2 L/min. The delivered oxygen-rich flow is mixed into mask along with air from the environment during inspiration.
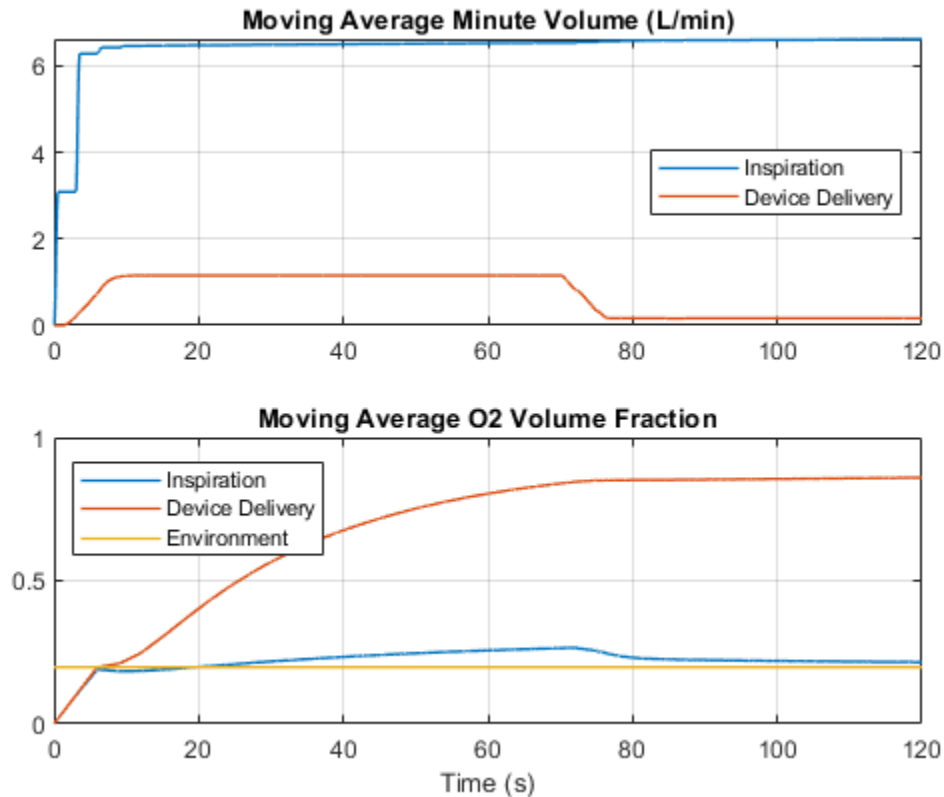


This plot shows the air pressure and volume in the lung. The lung is modeled with a simple mechanical elastance and resistance. The muscle pressure force causes the expansion of the lung volume, which lowers the lung pressure to draw in air from the mask.
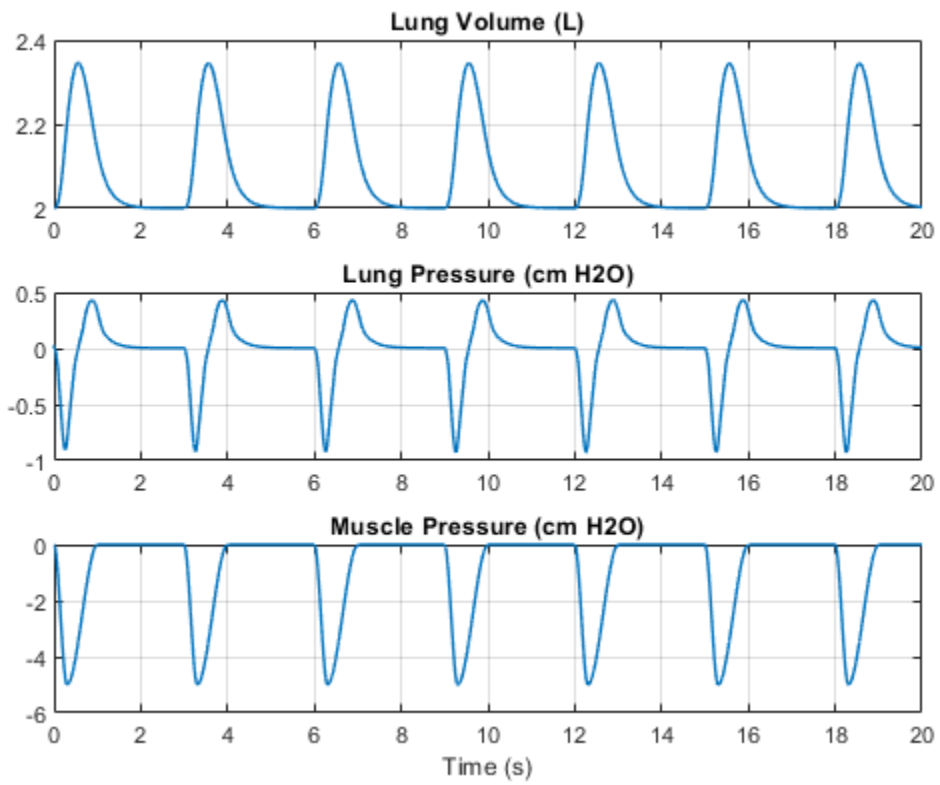
# Pneumatic Actuator with Humidity

This example shows how the Simscape™ Foundation Library moist air components can be used to model a pneumatic actuator operating in a humid environment. The Directional Valve is a subsystem composed of four Variable Local Restriction (MA) blocks, and the Double-Acting Actuator is a subsystem composed of two Translational Mechanical Converter (MA) blocks in opposite mechanical orientation.

Water condenses in the pipes and actuator due to the humidity in the environment. Accumulation of condensation over time may result in damage to the system.

**Model**
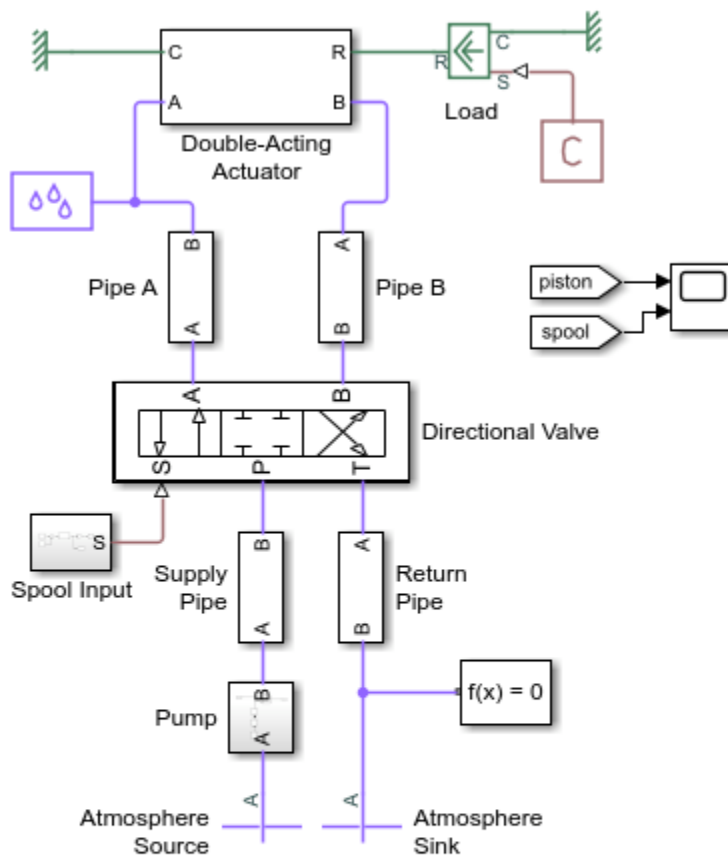


## Pneumatic Actuator with Humidity

1. Plot mass flow rates through valve (see code)
2. Plot pressures and temperatures in actuator (see code)
3. Plot condensation accumulated in pipes and actuator (see code)
4. Explore simulation results using sscexplore
5. Learn more about this example

Copyright 2017-2021 The MathWorks, Inc.

**Directional Valve Subsystem**
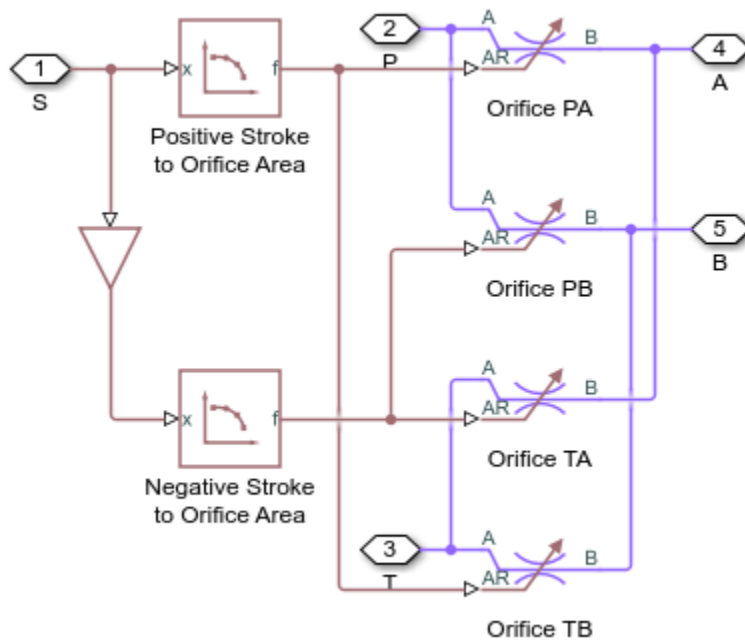
**Double-Acting Actuator Subsystem**
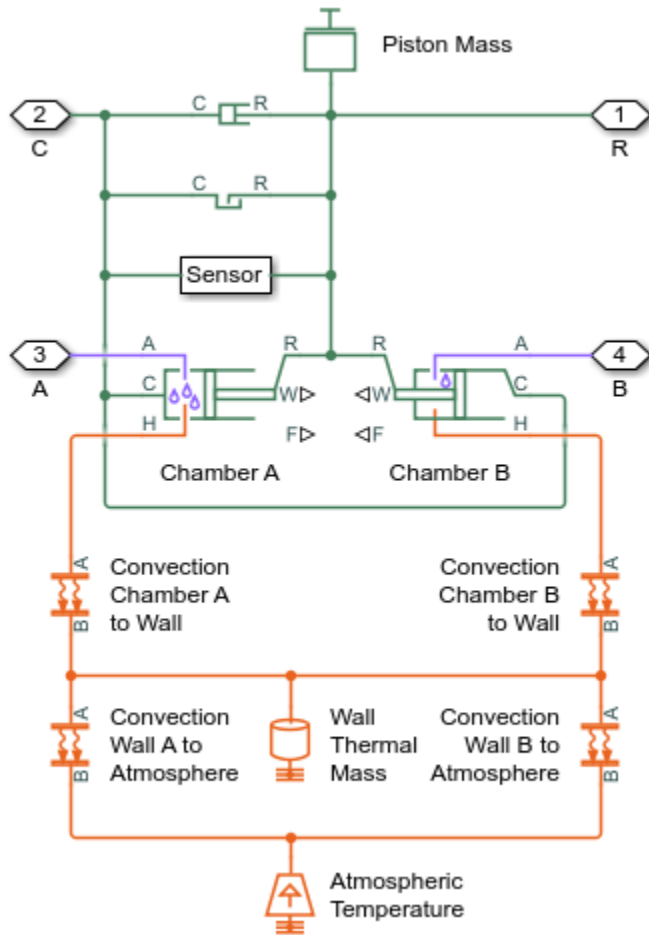


**Spool Input Subsystem**
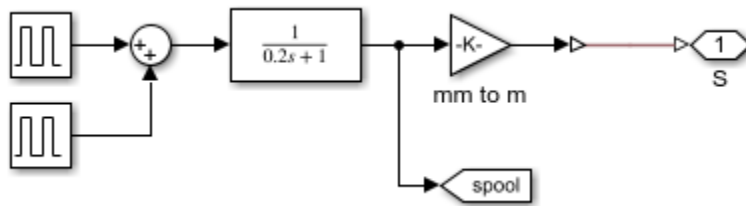
**Supply Pipe Subsystem**
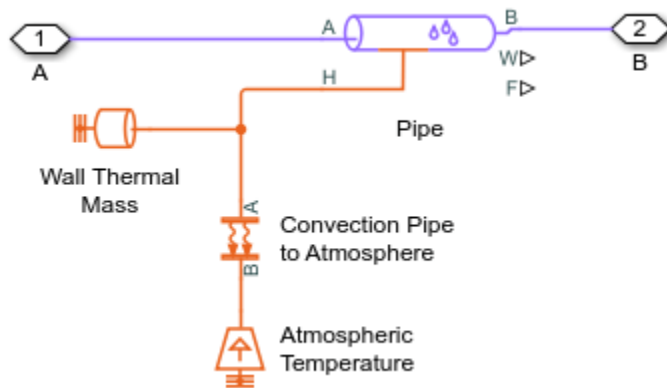


**Simulation Results from Scopes**
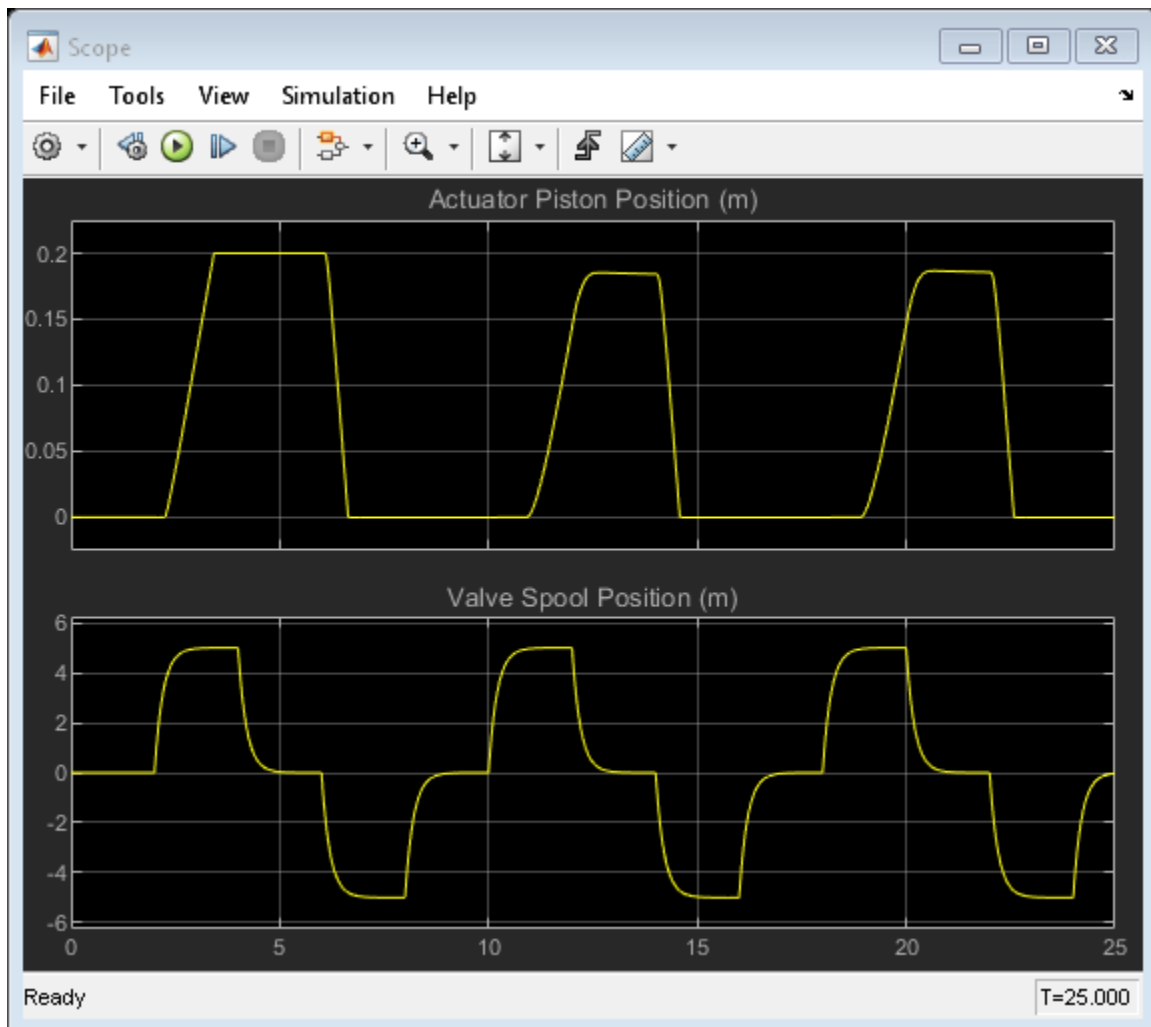
**Simulation Results from Simscape Logging**
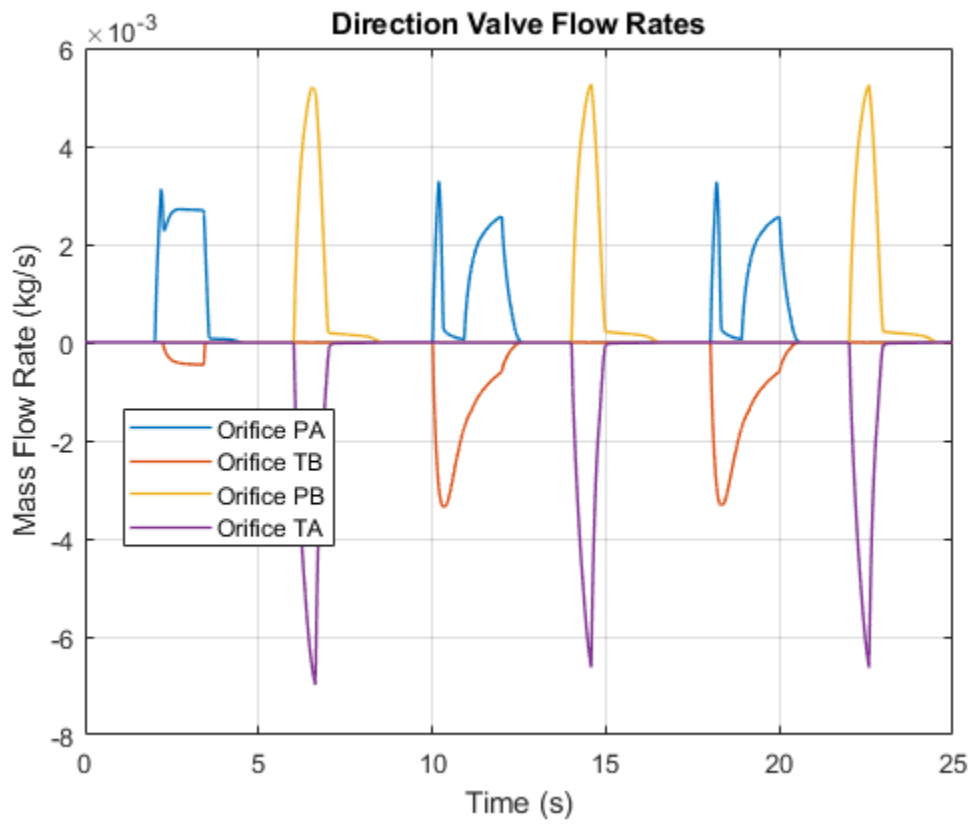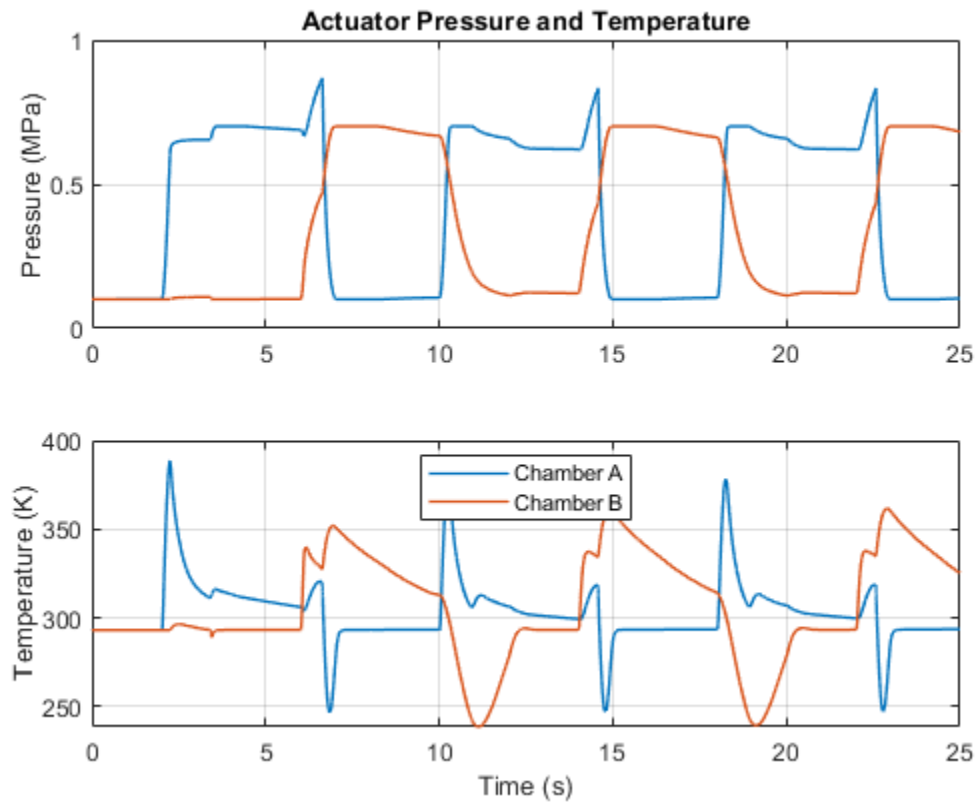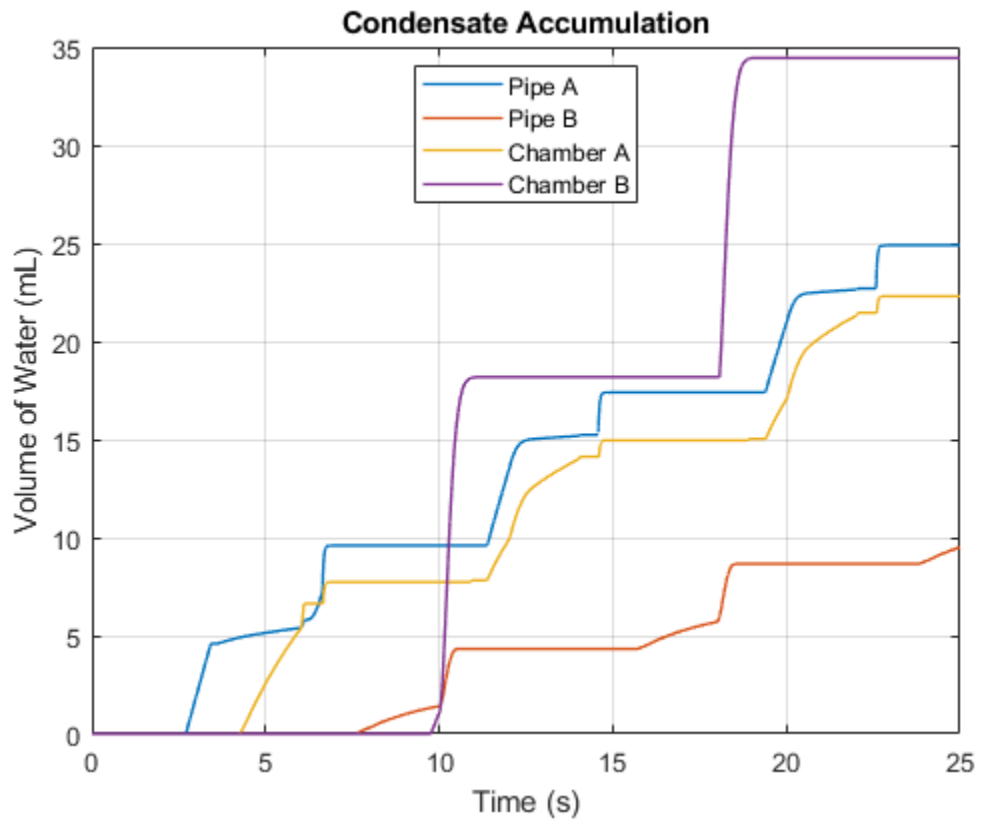
# Cavitation in Two-Phase Fluid

This example shows how two-phase fluid components can be used to simulate cavitation. The model is a translational mechanical converter driven by an oscillating pressure source. During the negative portion of the pressure source cycle, the fluid cavitates, reducing the force produced by the converter. As a result, the converter displacement drifts and does not return to the starting position.

**Model**
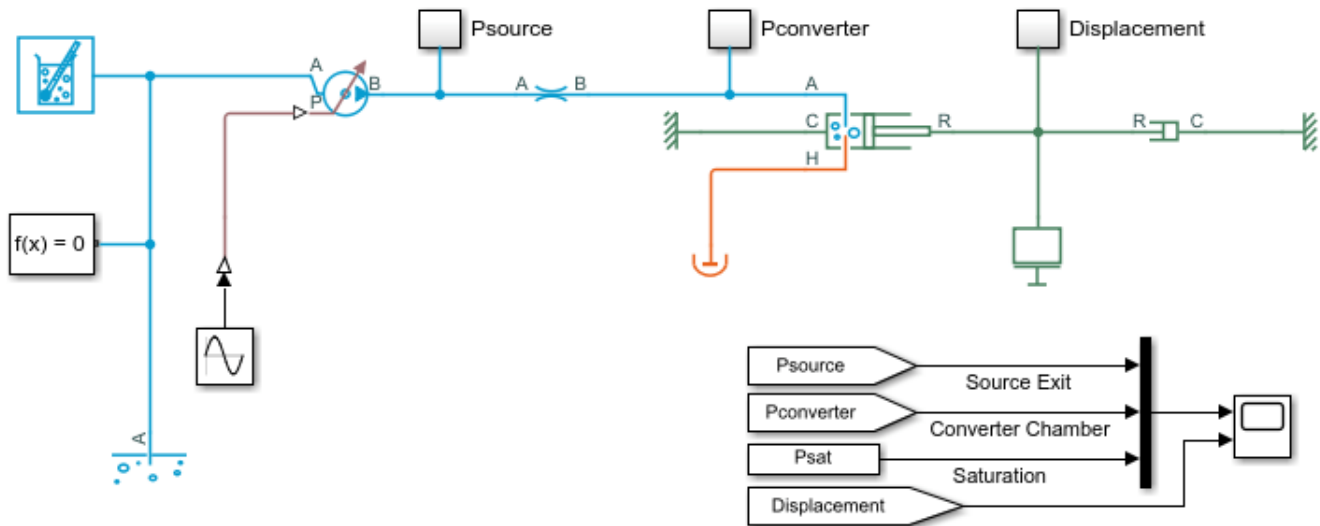


**Cavitation in Two-Phase Fluid**
1. Plot the normalized internal energy in Translational Mechanical Converter (2P) (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Scopes**
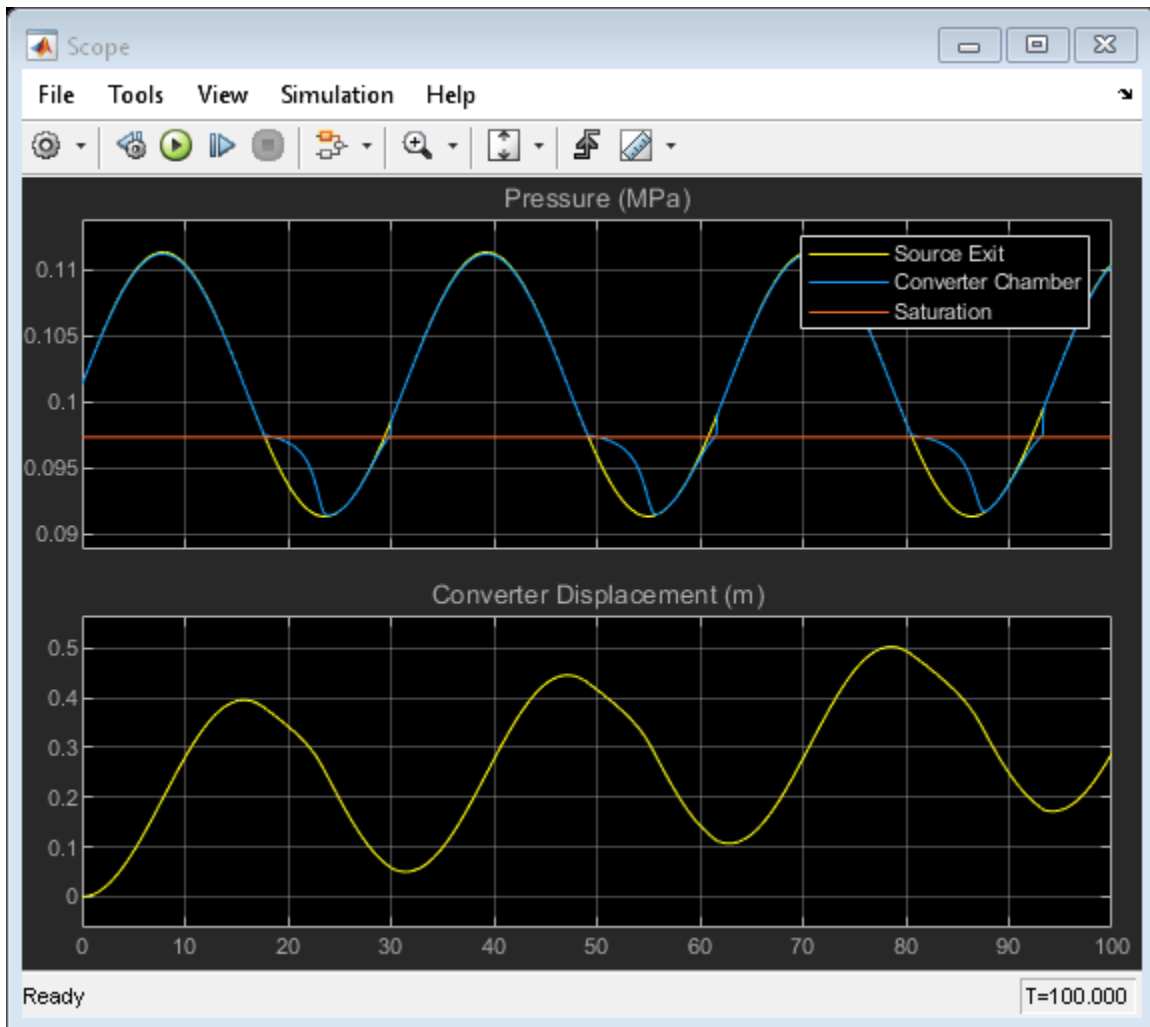


**Simulation Results from Simscape Logging**

This plot shows the normalized internal energy (unorm) of the fluid volume in the Translational Mechanical Converter (2P). The phase of the fluid can be determined by inspecting this plot: the fluid is a

- subcooled liquid when -1 <= unorm < 0;
- two-phase mixture when 0 <= unorm <= 1;
- superheated vapor when 1 < unorm <= 2.

When the fluid is in the two-phase region, the value of unorm corresponds to the vapor quality.
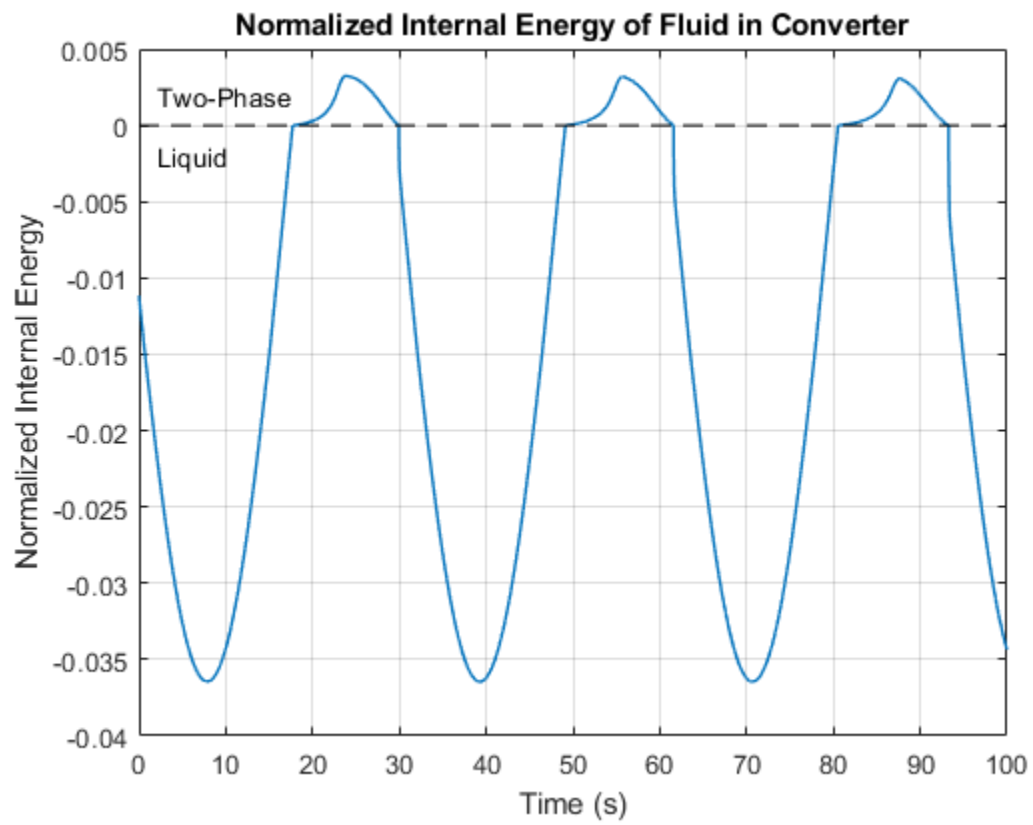
# Fluid Vaporization in Pipe

This example shows how to model the vaporization of water to generate steam. Liquid water enters the pipe at 370 K at a rate of 1 kg/s. The pipe is heated to 1000 K, causing the water flowing inside pipe to saturate.

When liquid in a large fluid volume saturates, the vaporization process can produce a surge in fluid pressure. Breaking a pipe into multiple segments allows a smaller fluid volume in each segment to saturate one at a time, reducing the strength of the pressure spike.

**Model**



**Fluid Vaporization in Pipe**
1. Plot the pressure of the fluid volume in each pipe segment (see code)
2. Plot the enthalpy of the fluid volume in each pipe segment (see code)
3. Plot fluid properties of water (see code)
4. Explore simulation results using sscexplore
5. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.
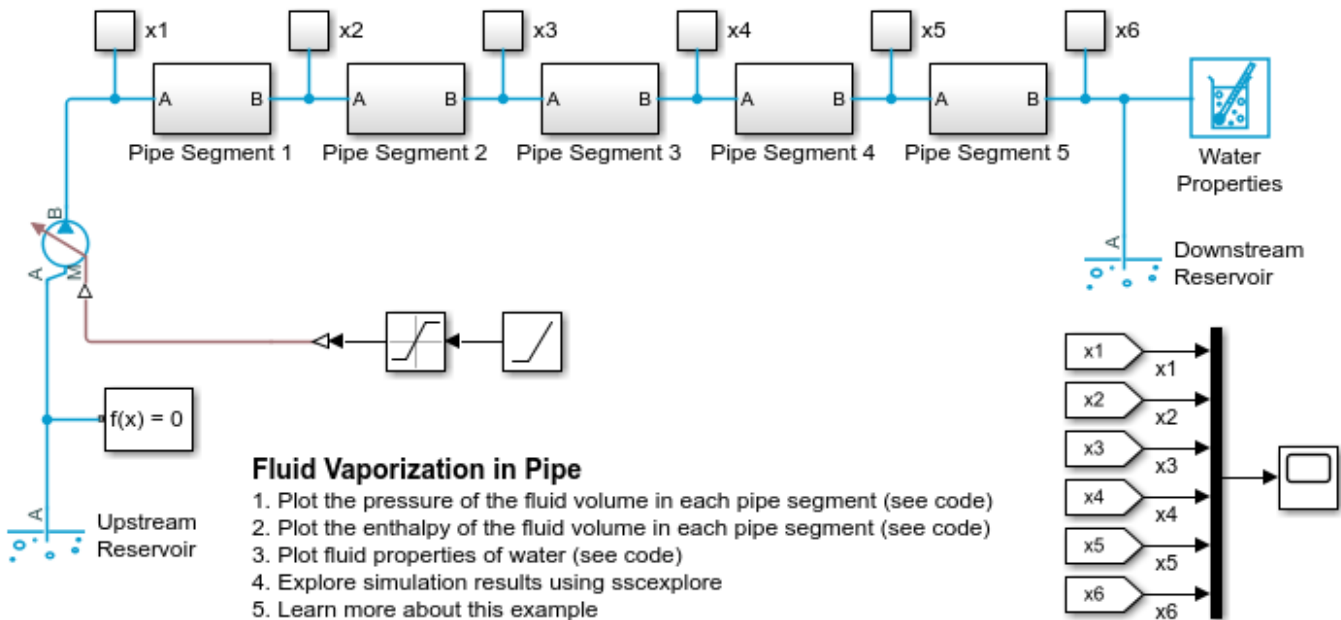
**Pipe Segment 1 Subsystem**
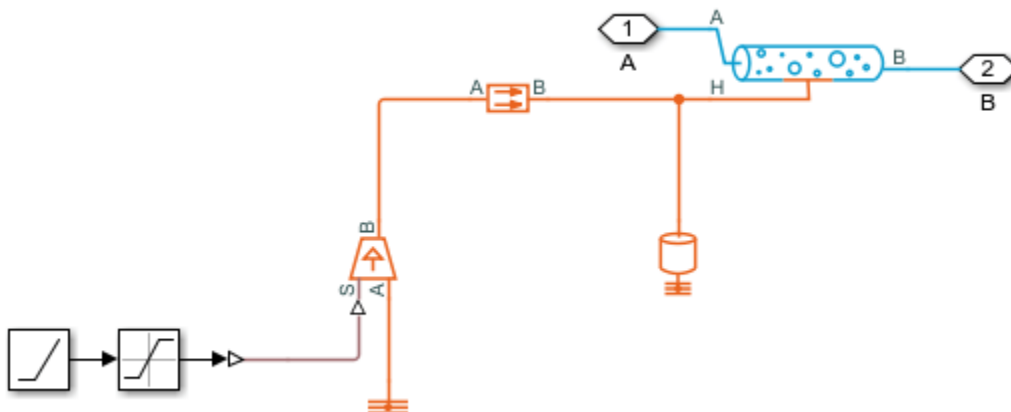
**Simulation Results from Scopes**
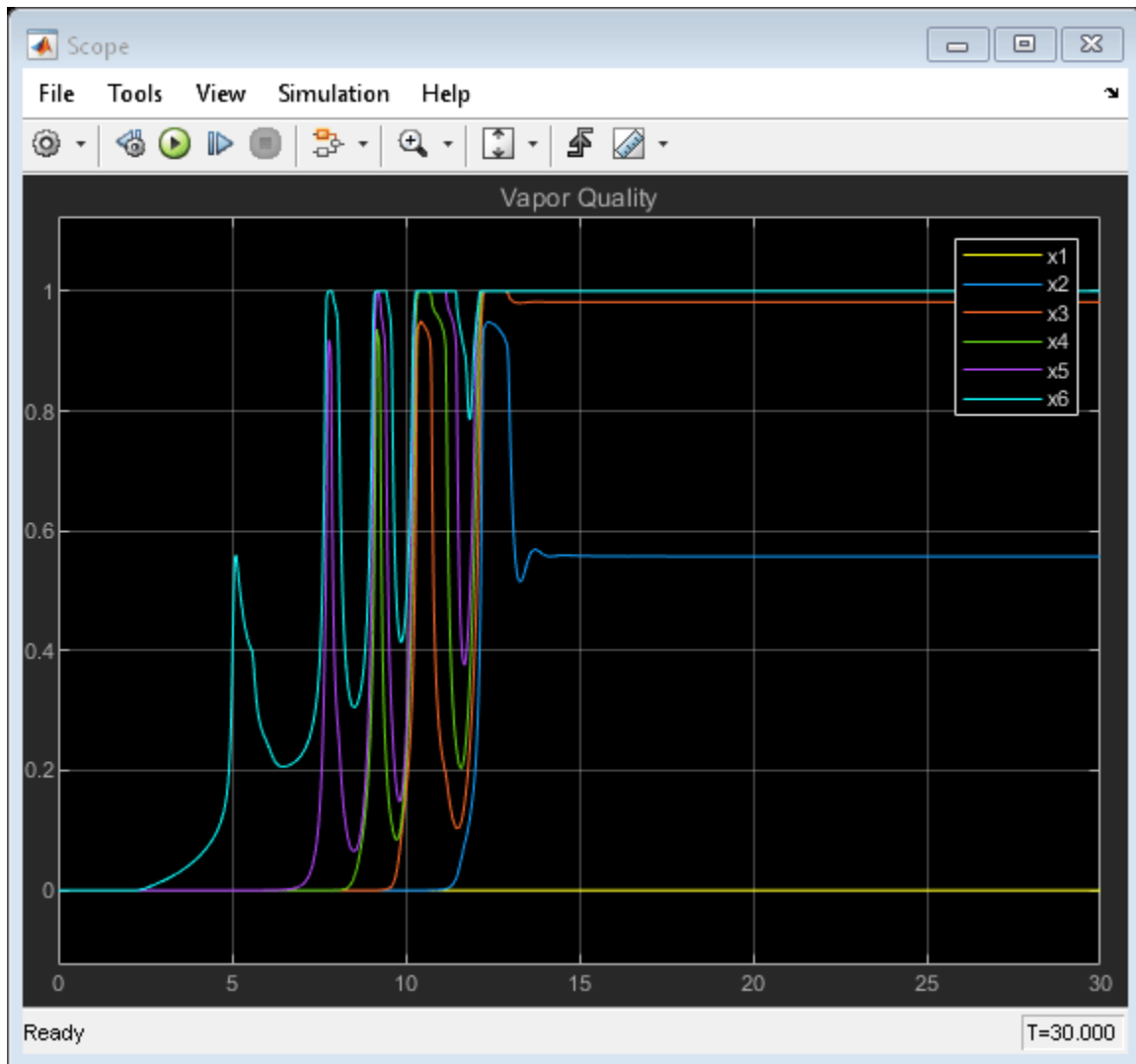


**Simulation Results from Simscape Logging**

This plot shows the pressure of the fluid volume in each pipe segment. The five pressure spikes correspond to the saturation of the fluid volume in each pipe segment, starting with the last segment and progressing upstream to the first segment. When a liquid crosses the saturation boundary, its specific volume increases rapidly. If the fluid cannot evacuate the volume quickly enough, then pressure builds up inside the volume. In the Simscape™ Two-Phase Fluid Library, components such as Pipe (2P) represent the fluid as a lumped parameter model. This means that the entire fluid volume inside the component saturates at once, resulting in the pressure spikes seen in the plot.

In some models, these pressure spikes can produce unexpected behavior such as a rapid surge of reversed flow upstream. One way to mitigate the pressure spikes is to break a single long pipe into multiple shorter pipe segments. This allows the smaller fluid volume in each pipe segment to saturate one a time instead of all at once. Another way to mitigate the pressure spikes is to increase the value of the Phase change time constant parameter. In this example, the 10 m pipe is broken into five 2 m pipe segments in order to simulate water vaporizing into steam.

**Pressure of Fluid Volume in Pipe Segments**
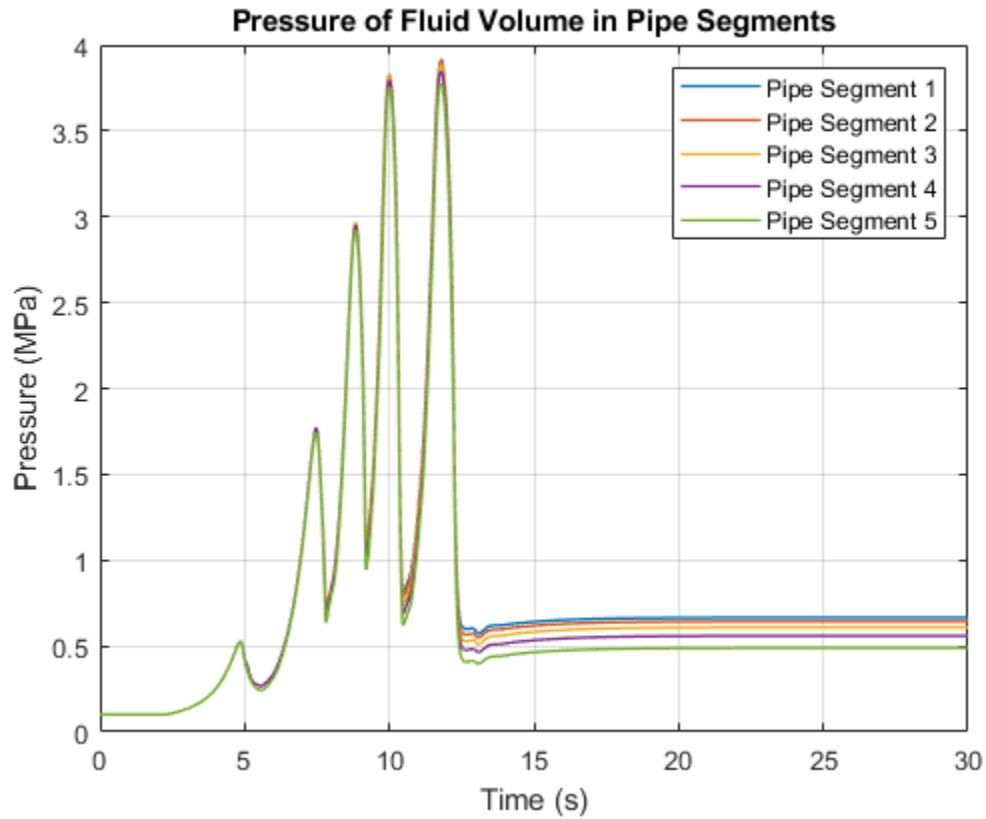
This plot shows the specific enthalpy of the fluid volume in each pipe segment. The specific enthalpy is not available directly from the logged simulation data. However, it can be computed from the formula: $h = u + p*v$ where $u$ is the specific internal energy, $p$ is the pressure, and $v$ is the specific volume.
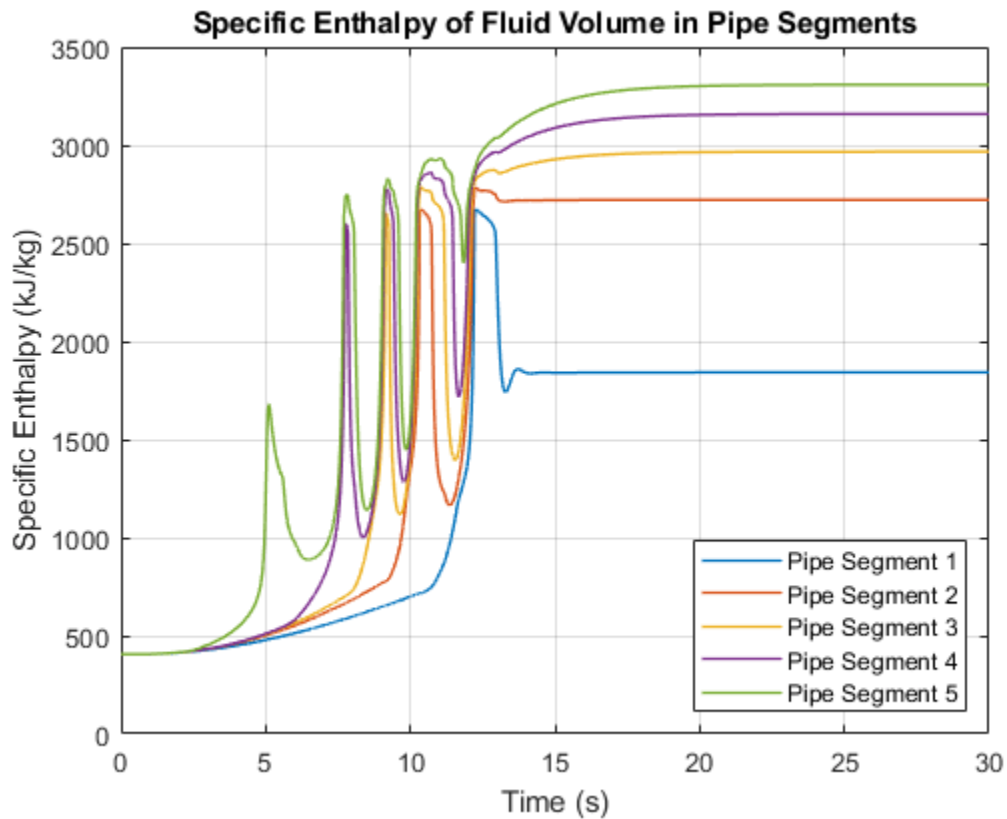
**Fluid Properties**

The following two figures plot the fluid properties of water as a function of pressure (p) and specific internal energy (u) and as a function of pressure (p) and normalized internal energy (unorm), respectively. The fluid is a

- subcooled liquid when -1 <= unorm < 0;
- two-phase mixture when 0 <= unorm <= 1;
- superheated vapor when 1 < unorm <= 2.

The fluid property data is provided as a rectangular grid in p and unorm. Therefore, the grid in terms of p and u is non-rectangular.

The water fluid property data can be found in `waterPropertyTables.mat`.
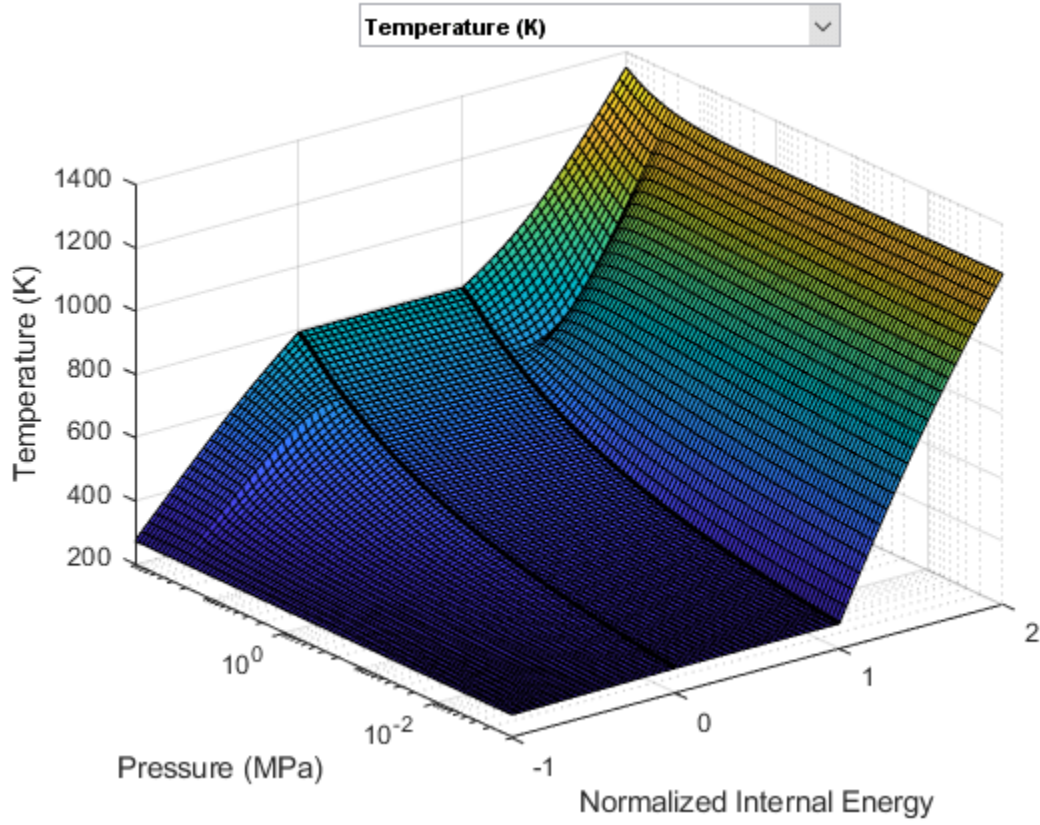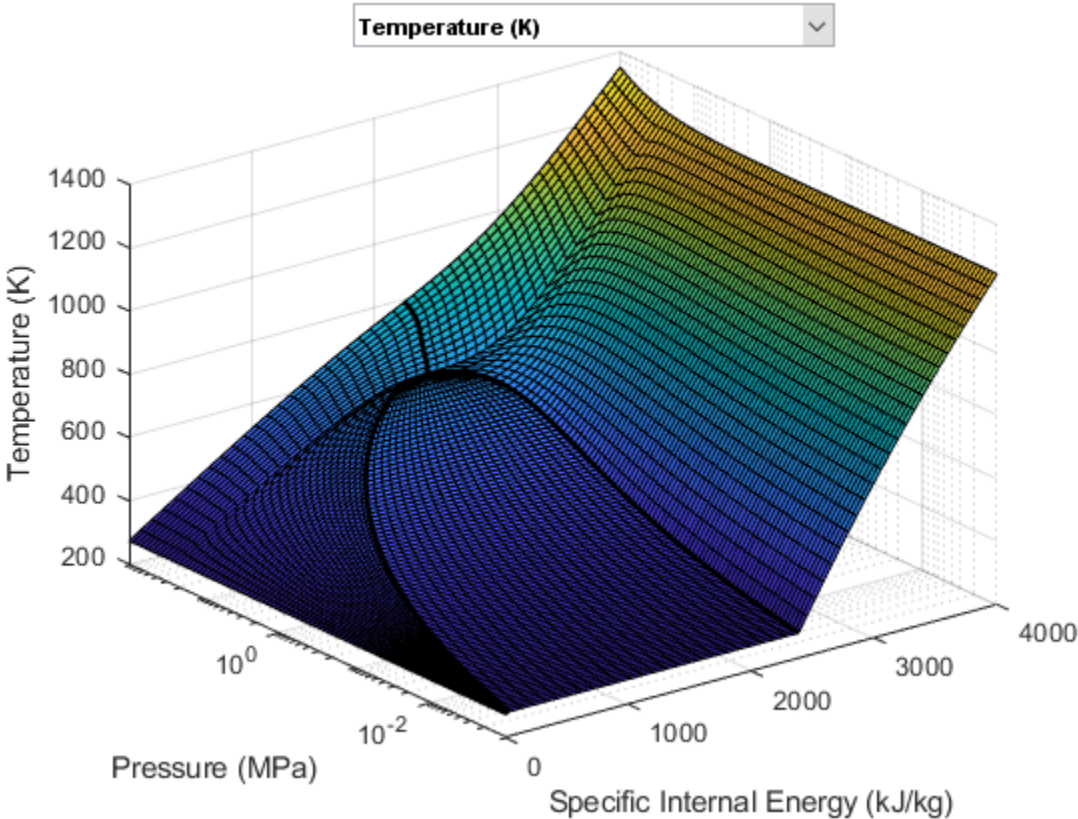
# Two-Phase Fluid Refrigeration

This example models a vapor-compression refrigeration cycle using two-phase fluid components. The compressor drives the R-134a refrigerant through a condenser, an expansion valve, and an evaporator. The hot gas leaving the compressor condenses in the condenser via heat transfer to the environment. The pressure drops as the refrigerant passes through the expansion valve. The drop in pressure lowers the saturation temperature of the refrigerant. This enables it to boil in the evaporator as it absorbs heat from the refrigerator compartment. The refrigerant then returns to the compressor to repeat the cycle. The controller turns the compressor on and off to maintain the refrigerator compartment temperature within a band around the desired temperature.

**Model**

**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

This figure plots the refrigeration cycle performance over time including the pressures, temperatures, energy flows, and mass flows. It shows that this refrigeration cycle operates at a compressor pressure ratio of about 5.5. The coefficient of performance, which is the ratio of the heat extracted to the compressor power input, is approximately 4.
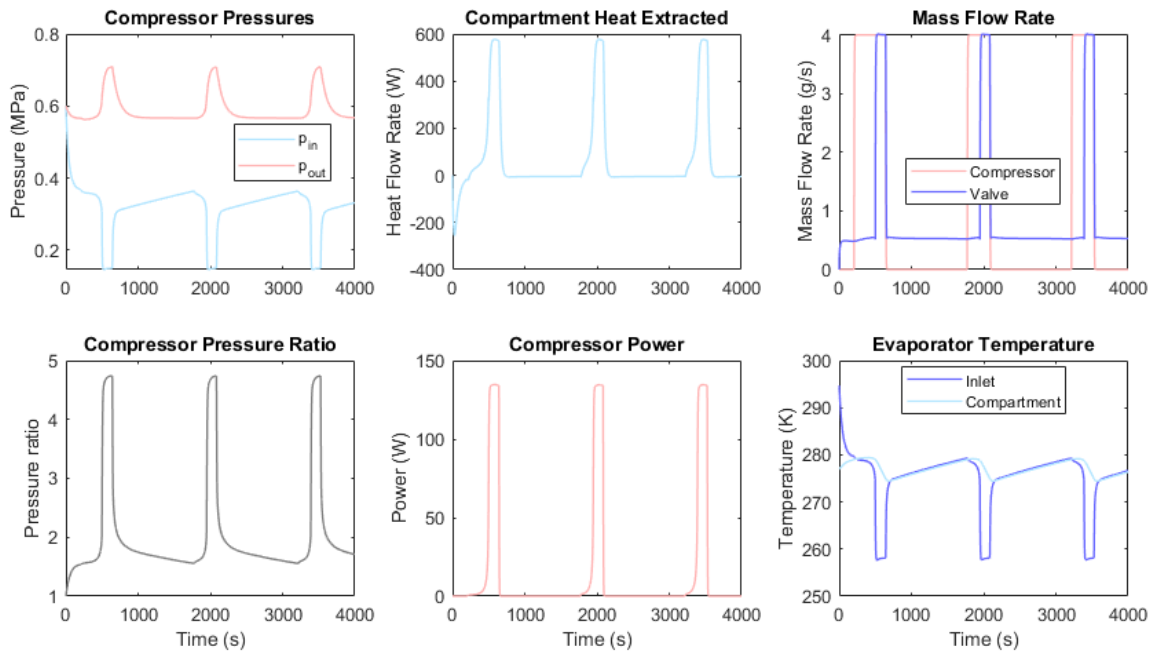
This figure plots the vapor quality at each of the four points on the refrigeration cycle. It shows that when the compressor is turned on, the evaporator absorbs enough heat from the refrigerator compartment to completely vaporize the refrigerant. The condenser then brings the vapor quality down to about 0.02. Flash evaporation occurs in the expansion valve such that the refrigerant enters the evaporator at a vapor quality of about 0.4.
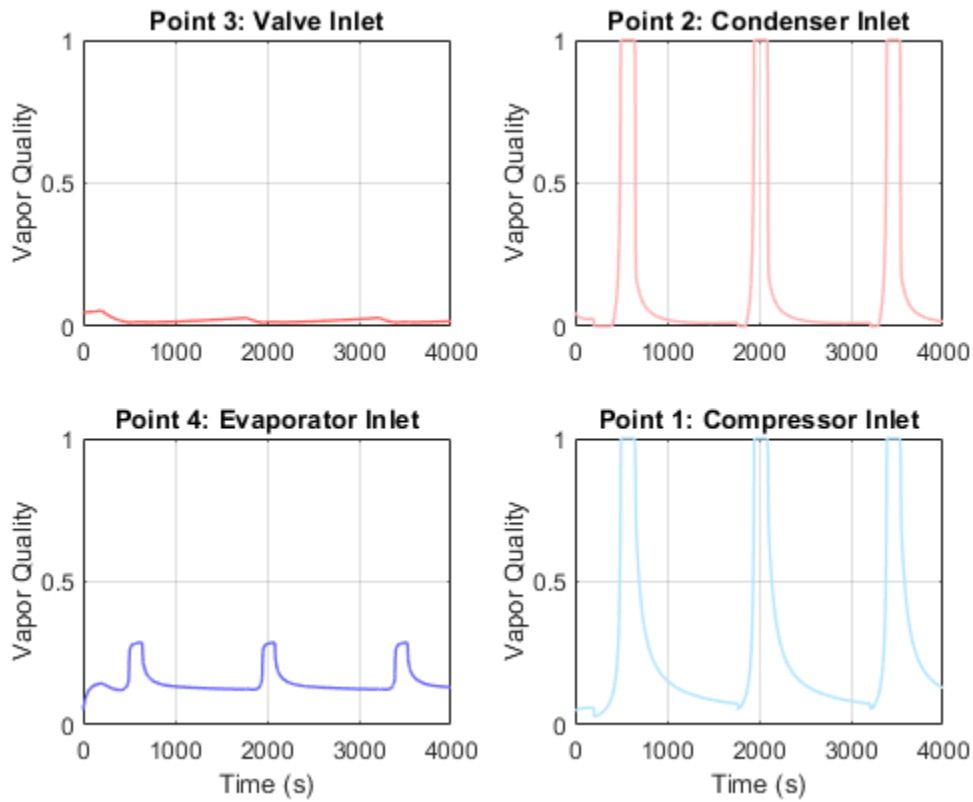
**Animation of Simscape Logging Results**

This figure shows the evolution of the fluid states in the refrigeration cycle over time. The four points on the refrigeration cycle (compressor inlet, condenser inlet, expansion valve inlet, and evaporator inlet) are plotted on the pressure-enthalpy diagram. The dotted contour lines indicates the temperature and the gray curve represents the saturation dome.
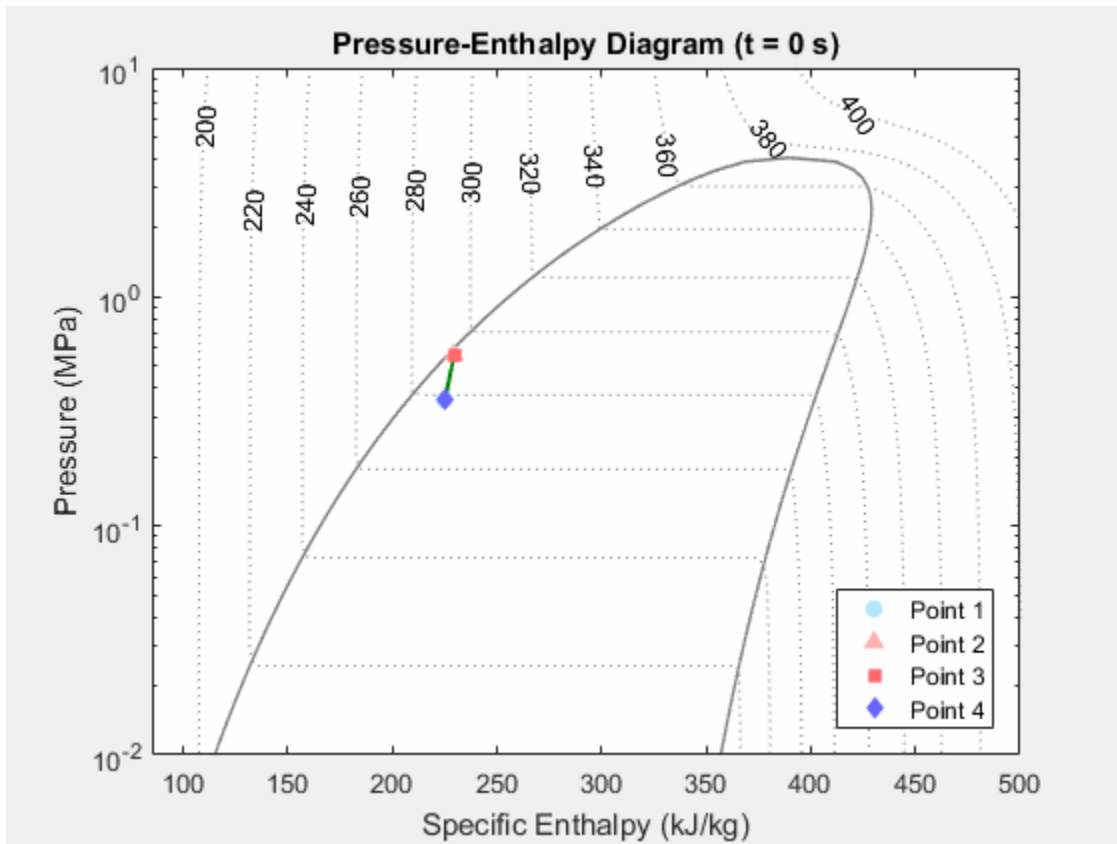
### Fluid Properties

The following two figures plot the fluid properties of refrigerant R-134a as a function of pressure (p) and specific internal energy (u) and as a function of pressure (p) and normalized internal energy (unorm), respectively. The fluid is a

- subcooled liquid when -1 <= unorm < 0;
- two-phase mixture when 0 <= unorm <= 1;
- superheated vapor when 1 < unorm <= 2.

The fluid property data is provided as a rectangular grid in p and unorm. Therefore, the grid in terms of p and u is non-rectangular.

The R-134a fluid property data can be found in `r134aPropertyTables.mat`.
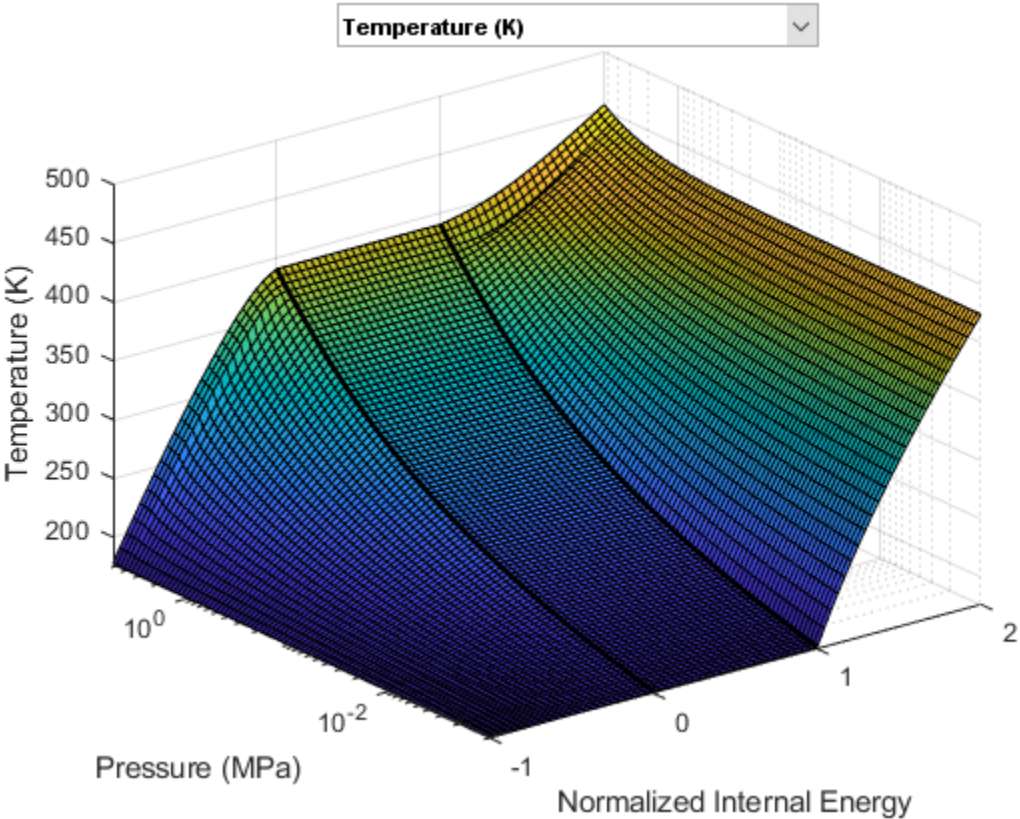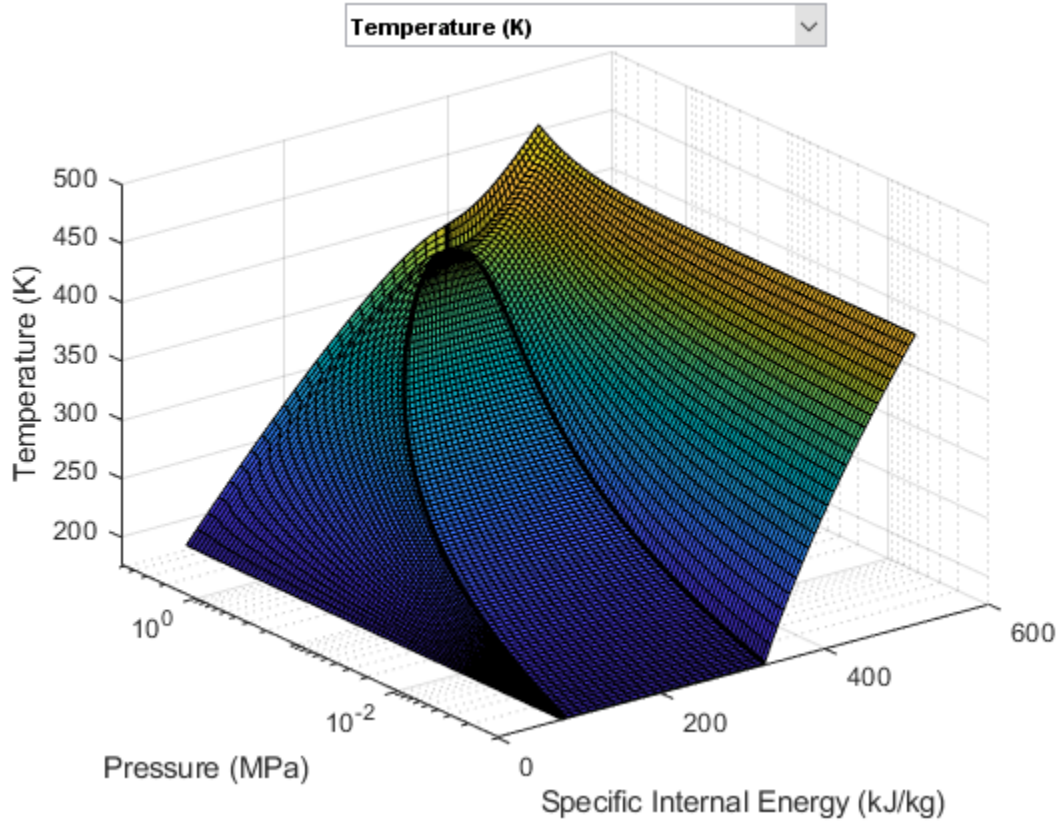
# Rankine Cycle (Steam Turbine)

This example models a steam turbine system based on the Rankine Cycle. The cycle includes superheating and reheating to prevent condensation at the high-pressure turbine and the low-pressure turbine, respectively. The cycle also has regeneration by passing extracted steam through closed feedwater heaters to warm up the water and improve cycle efficiency.

The Saturated Fluid Chamber block and the Turbine block are custom components based on the Simscape™ Foundation Two-Phase Fluid Library. The Saturated Fluid Chamber block models a separate saturated liquid volume and saturated vapor volume and is used to create the boiler and the condenser.

**Model**
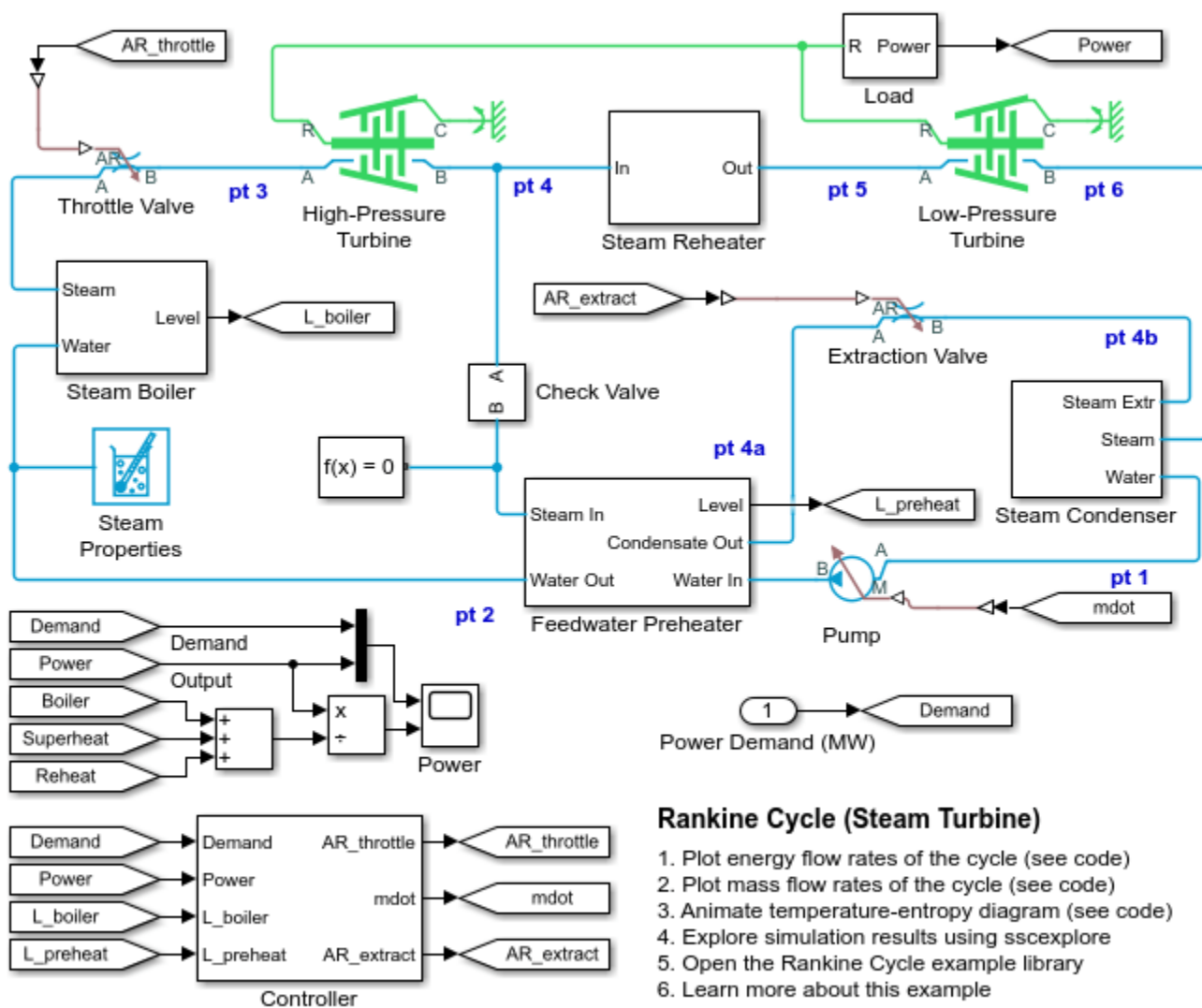


### Rankine Cycle (Steam Turbine)

1. Plot energy flow rates of the cycle (see code)
2. Plot mass flow rates of the cycle (see code)
3. Animate temperature-entropy diagram (see code)
4. Explore simulation results using sscexplore
5. Open the Rankine Cycle example library
6. Learn more about this example

Copyright 2016-2021 The MathWorks, Inc.

**Feedwater Preheater Subsystem**
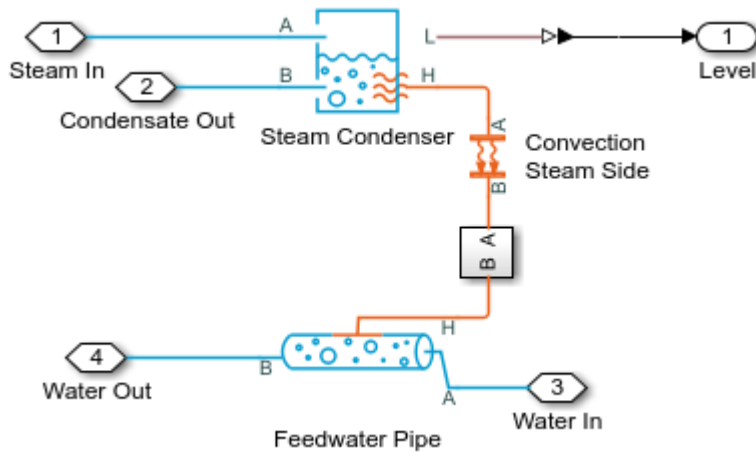


**Steam Boiler Subsystem**
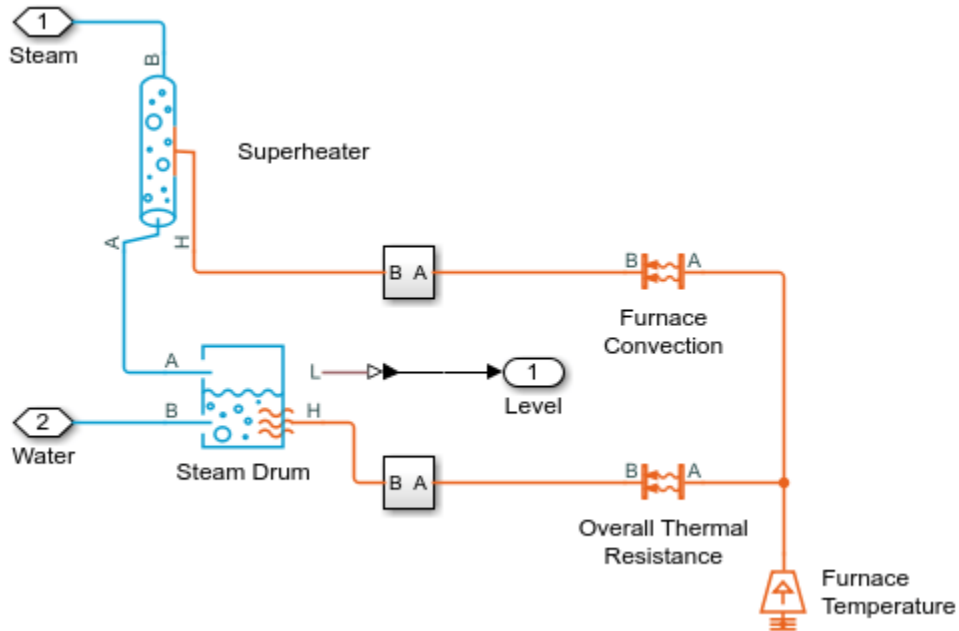
**Steam Condenser Subsystem**
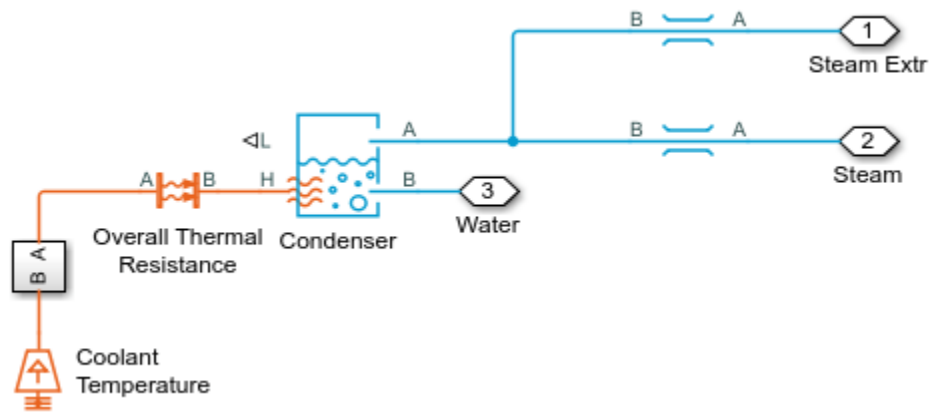


**Steam Reheater Subsystem**
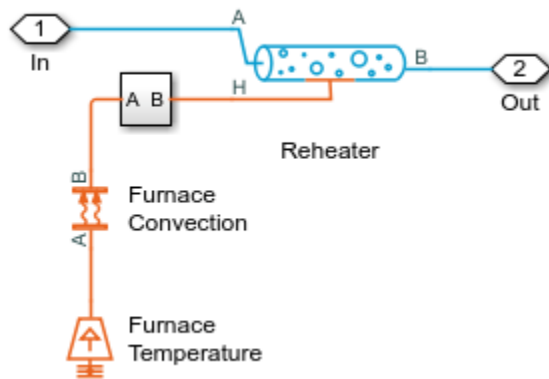
**Simulation Results from Scopes**
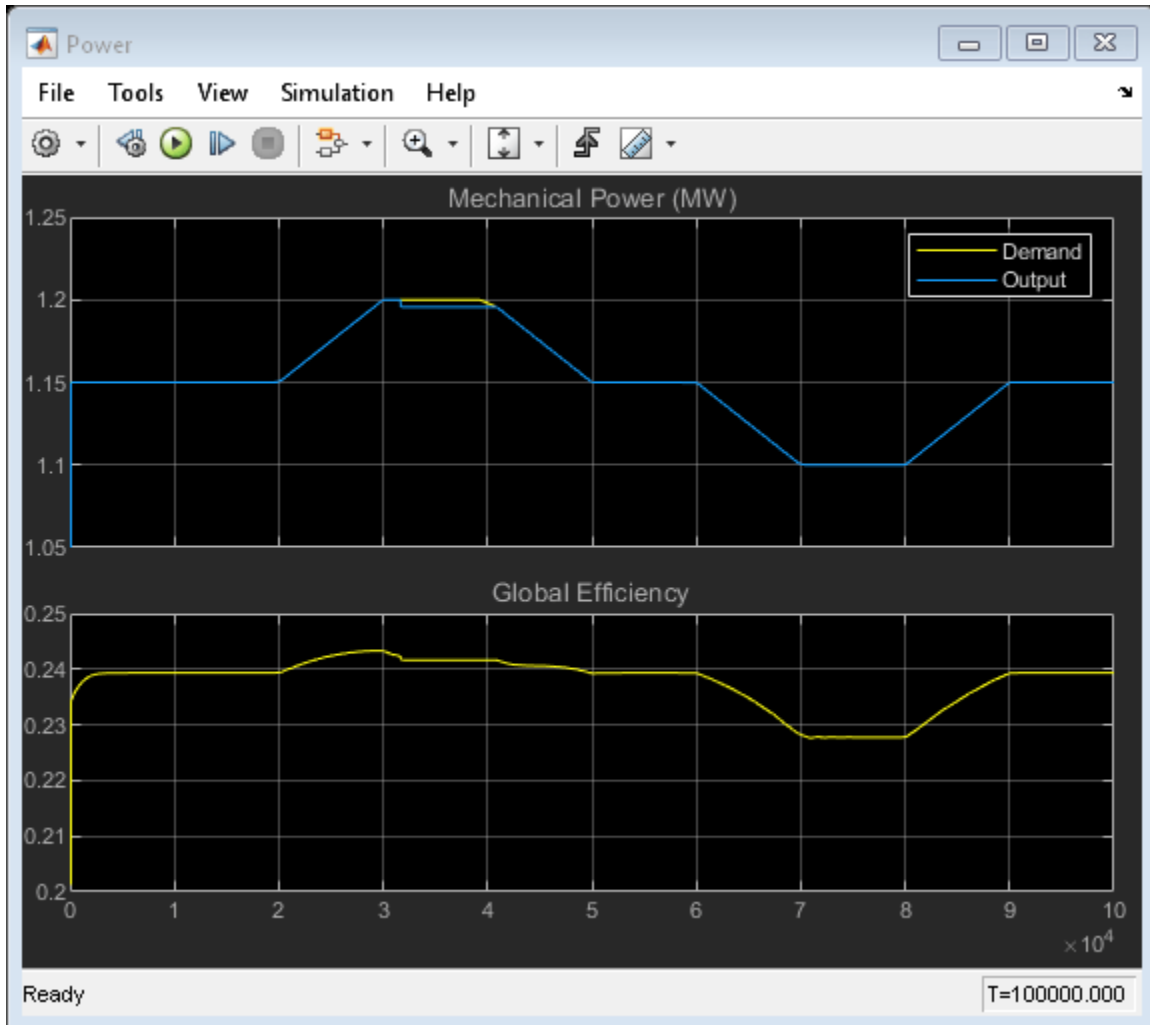


**Simulation Results from Simscape Logging**

This plot shows the energy exchanges in the system. Heat from the furnace is added by the boiler, the superheater, and the reheater. Useful work is extracted from the steam by the high-pressure turbine (HPT) and the low-pressure turbine (LPT). Waste heat is rejected to the coolant in the condenser. Heat is also transferred from the extracted steam to the feedwater to improve thermal efficiency.

## System Energy Flows
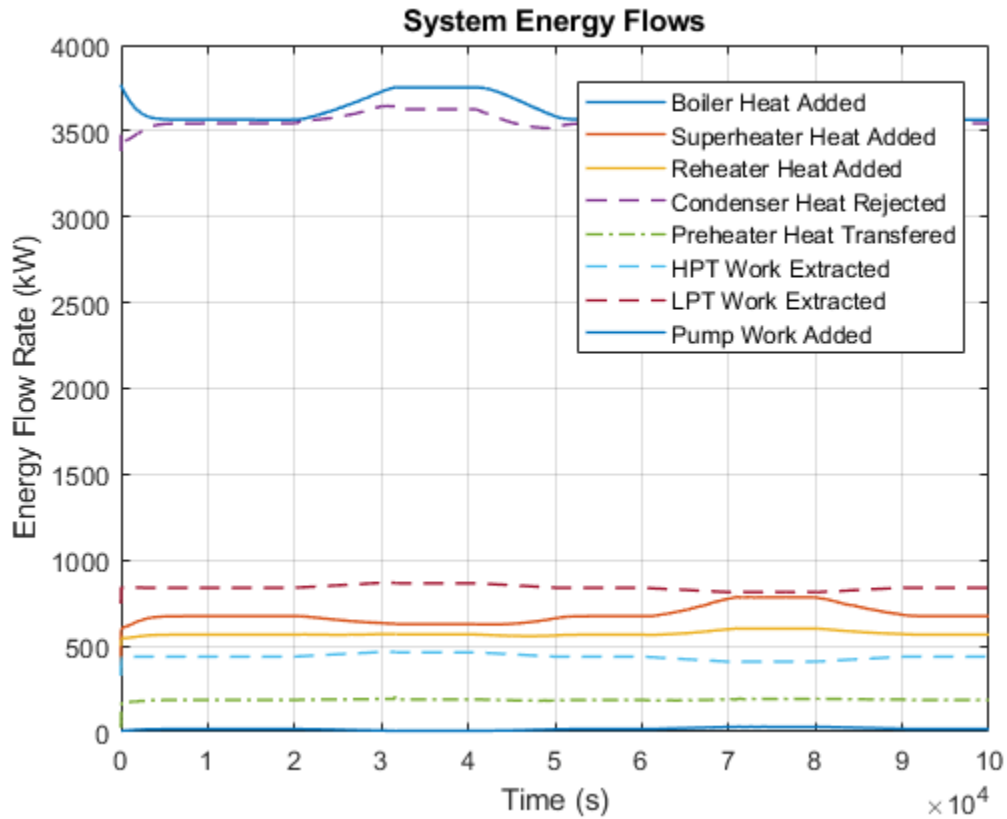


This plot shows the mass flow rates through the system. A portion of the steam is extracted between the high-pressure turbine (HPT) and the low-pressure turbine (LPT). The extracted steam is used to warm up the feedwater before rejoining the main flow at the condenser. The flow rates of the main flow and the extracted steam are regulated by the controllers to maintain the liquid level in the boiler and the preheater condenser, respectively.
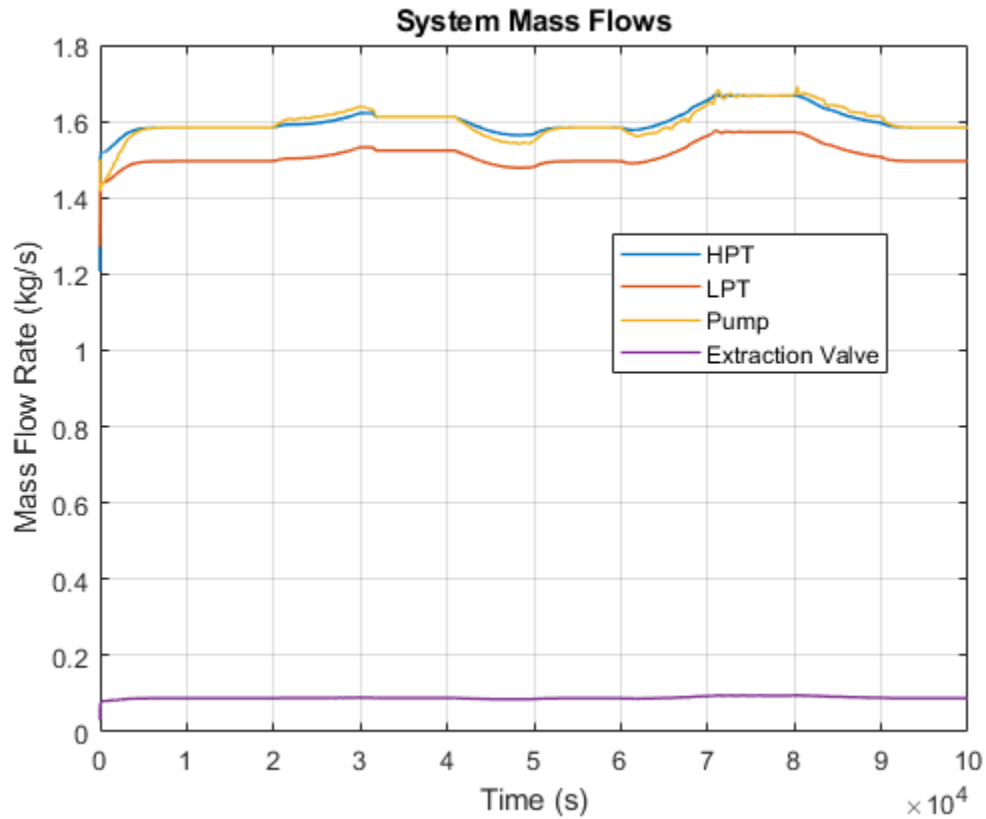
This figure shows an animation of the Rankine Cycle on a temperature-entropy diagram over time. The main steam flow corresponds to the loop from Cycle Points 1 to 6. The extracted steam flow corresponds to the dashed line from Cycle points 4 to 4b. In this model, the throttle valve before Cycle Point 3 provides a small amount of control over the power output.

Temperature-Entropy Diagram (t = 10.0x10⁴ s)
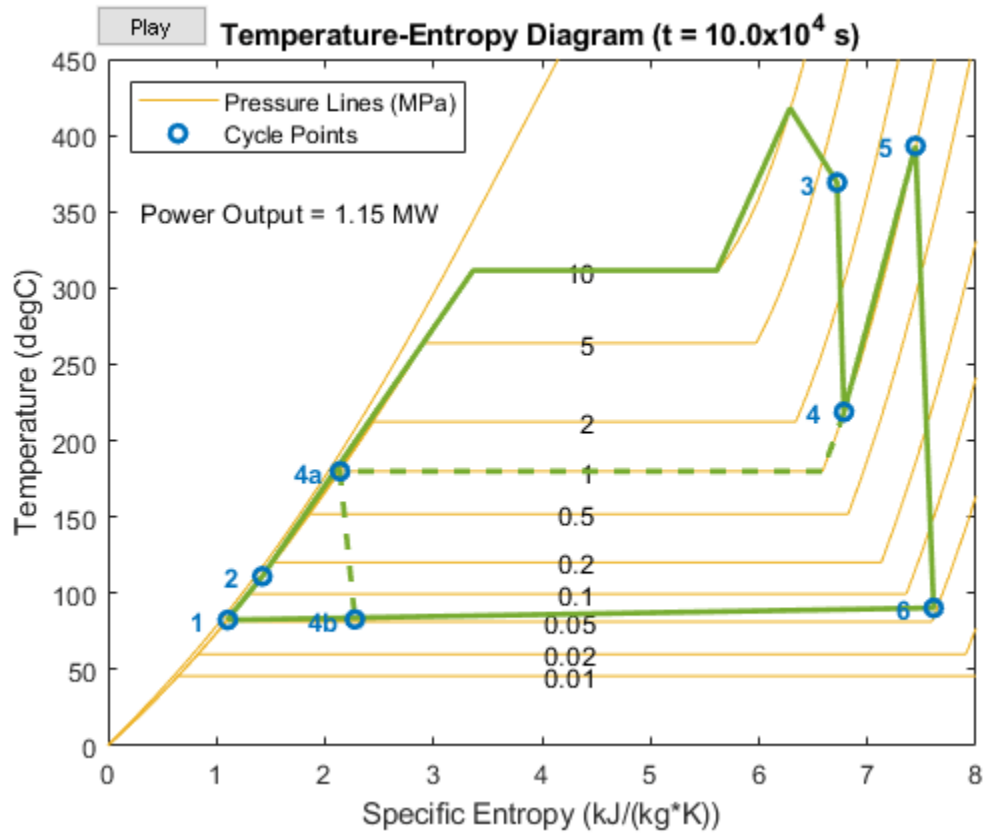
# Transcritical CO2 (R744) Refrigeration Cycle

This example models a vapor-compression refrigeration cycle in which the high pressure portion of the cycle operates in the supercritical fluid region. The refrigerant is carbon dioxide (CO2), also called R744 in this application.

The compressor drives the flow of CO2 through the cycle and raises the pressure above the critical pressure. The gas cooler rejects heat from the high pressure CO2 to the environment. Because CO2 is in a supercritical state, it does not condense and the temperature decreases. The expansion valve drops the pressure, causing some CO2 to vaporize. The two-phase mixture passes through the evaporator, absorbing heat from the compartment until it is superheated. The internal heat exchanger transfers some heat between the hot and cold side of the cycle to improve the efficiency of the cycle.

**Model**
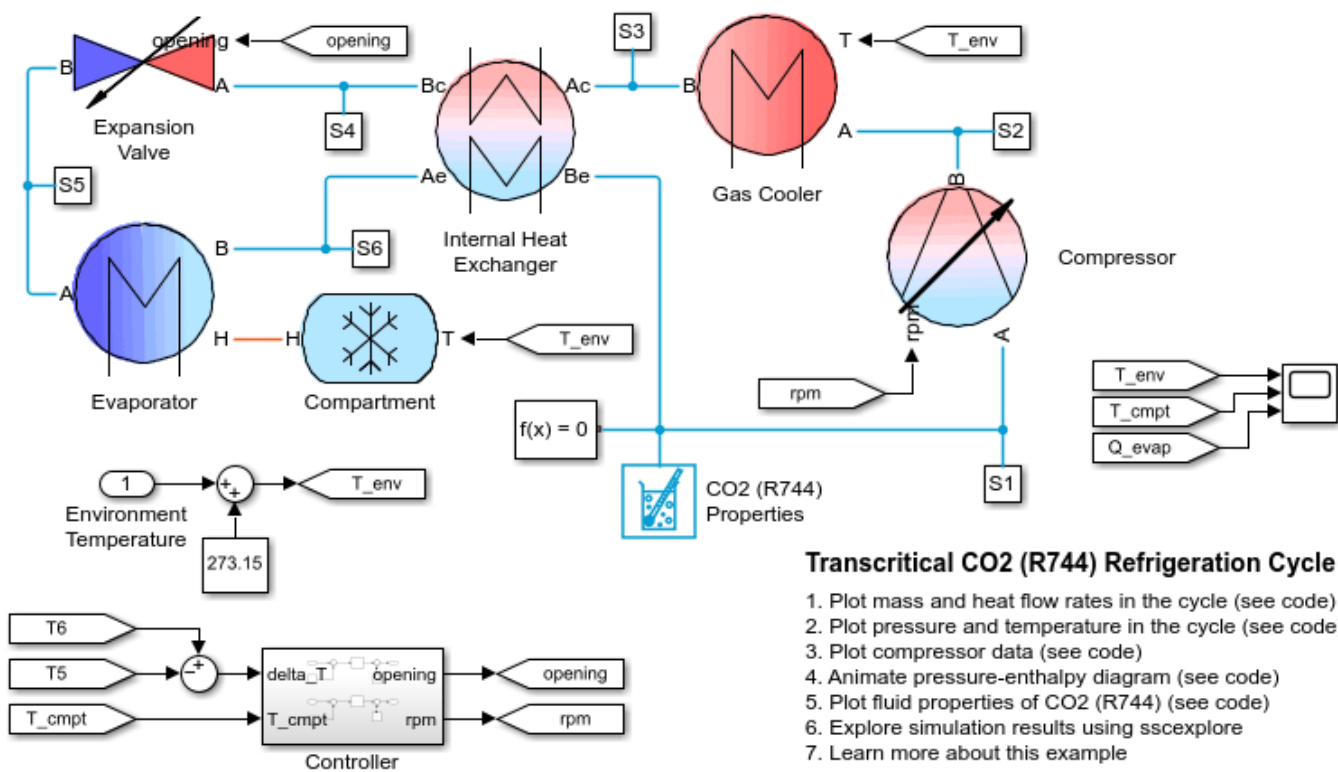


### Transcritical CO2 (R744) Refrigeration Cycle

1. Plot mass and heat flow rates in the cycle (see code)
2. Plot pressure and temperature in the cycle (see code)
3. Plot compressor data (see code)
4. Animate pressure-enthalpy diagram (see code)
5. Plot fluid properties of CO2 (R744) (see code)
6. Explore simulation results using sscexplore
7. Learn more about this example

Copyright 2018-2021 The MathWorks, Inc.

**Compartment Subsystem**
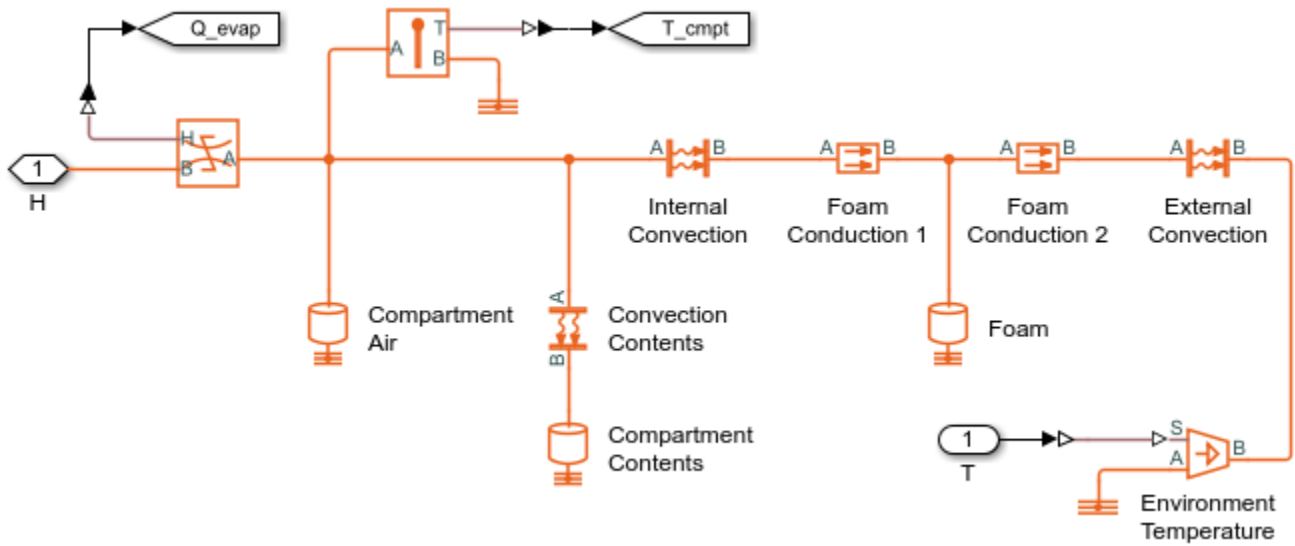


**Compressor Subsystem**
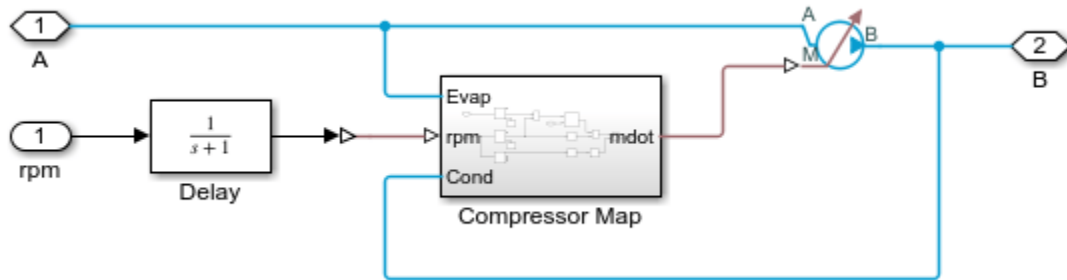
**Controller Subsystem**
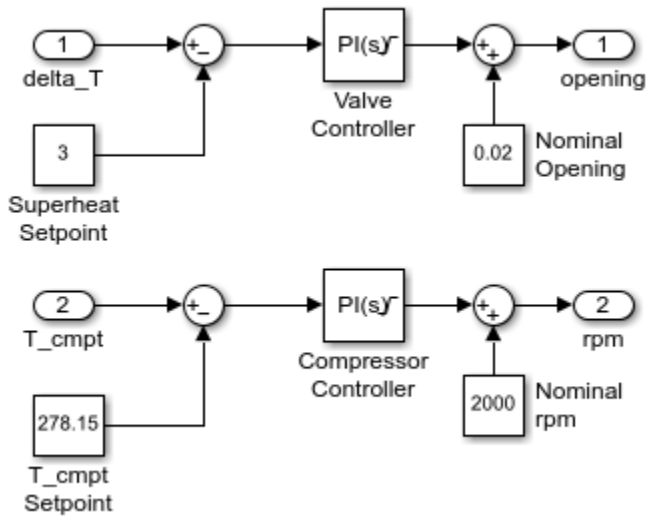


**Evaporator Subsystem**
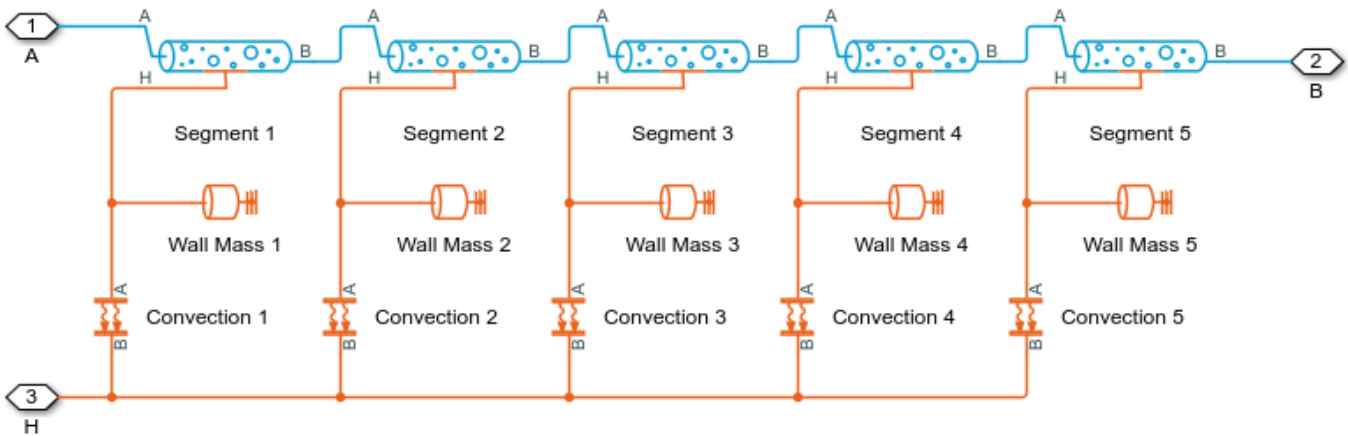


**Expansion Valve Subsystem**
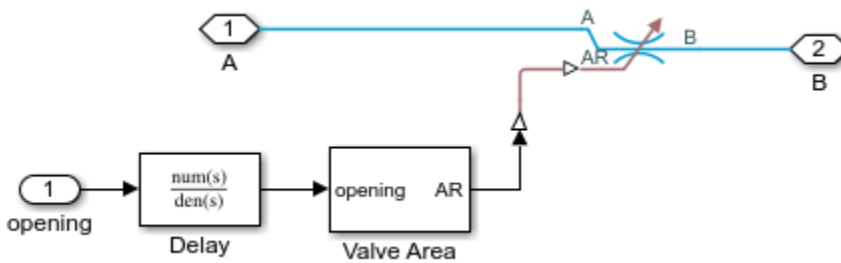
**Gas Cooler Subsystem**
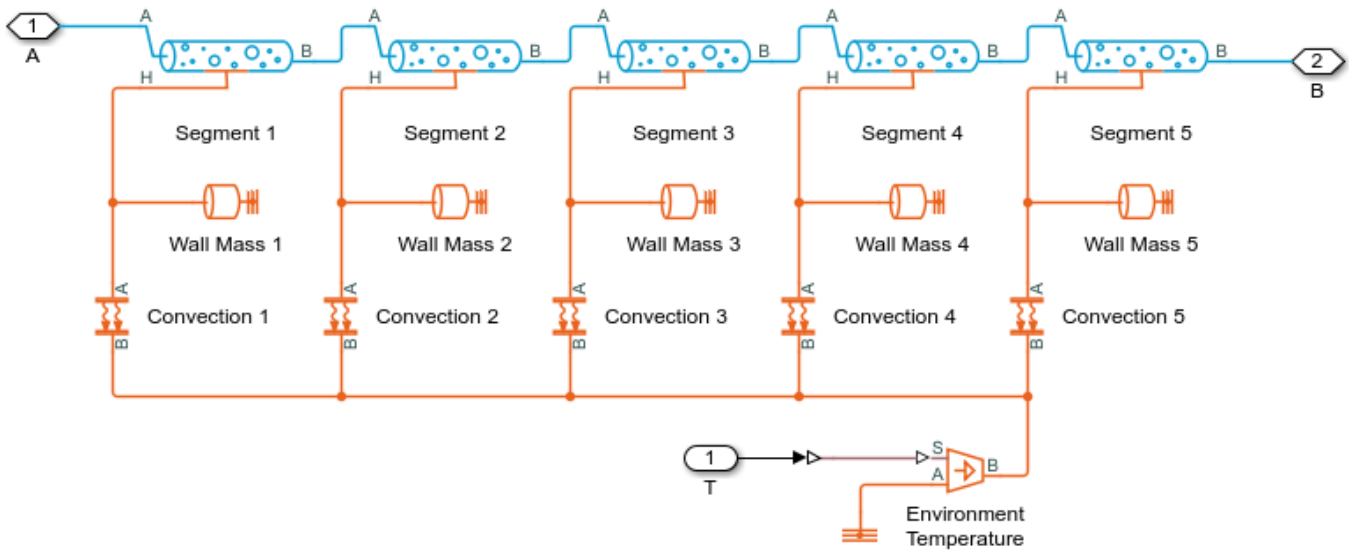


**Internal Heat Exchanger Subsystem**
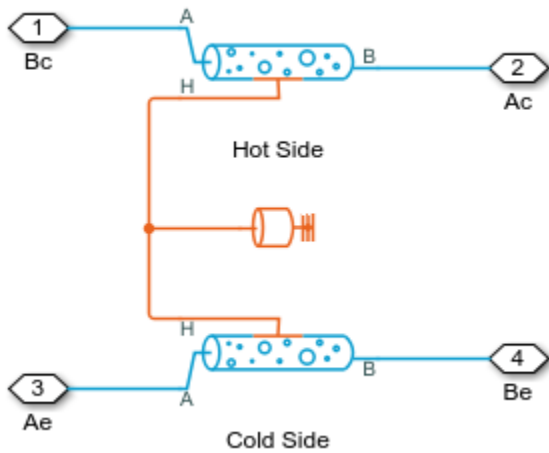
**Simulation Results from Scopes**
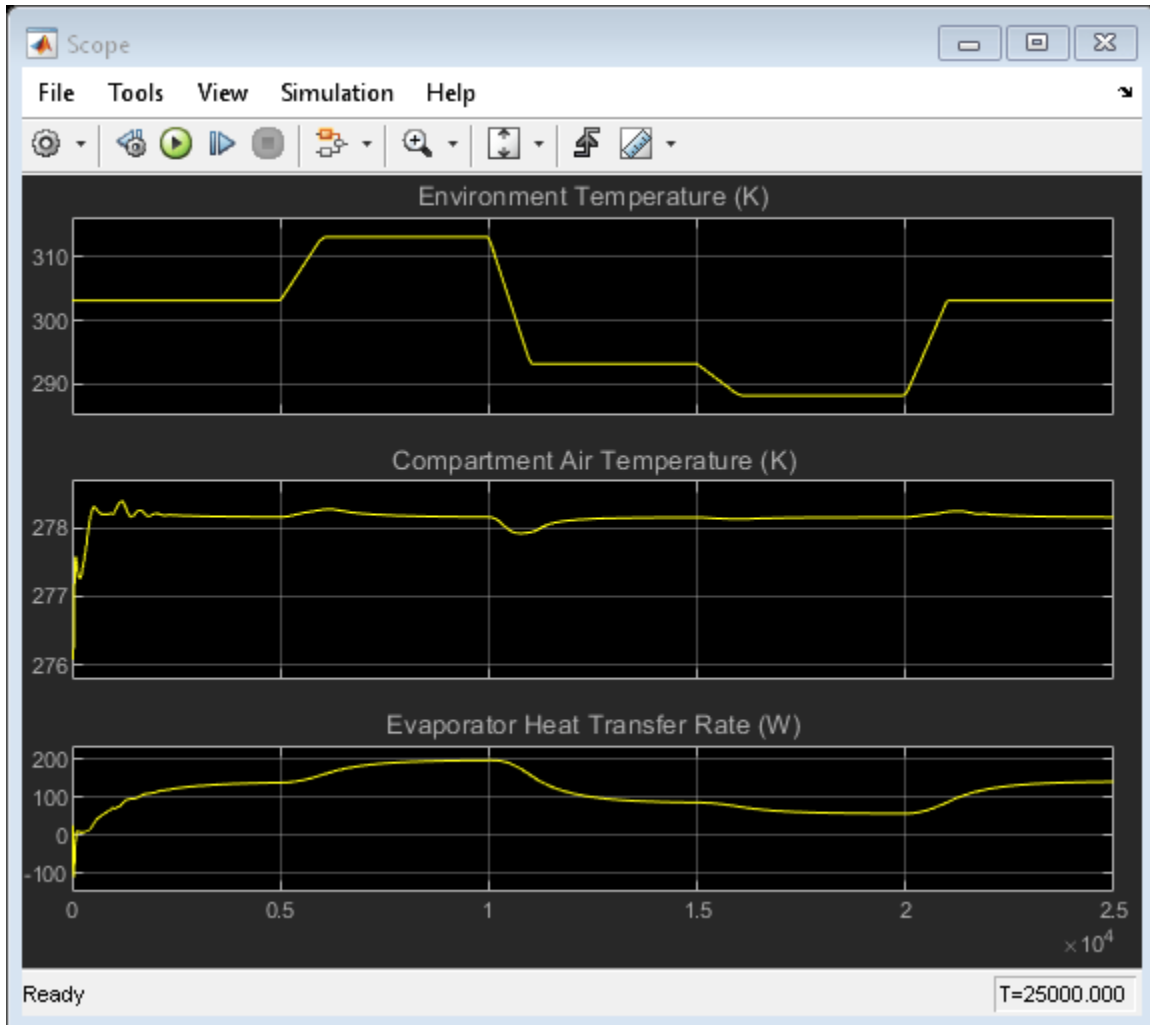


**Simulation Results from Simscape Logging**

This plot shows mass flow rate, isentropic compressor power input, and heat flow rates in the cycle. The Gas Cooler and Evaporator heat flow rates represent the heat rejection and heat absorption of the cycle, while the IHX heat flow rates are heat transfers within the cycle by the internal heat exchanger.
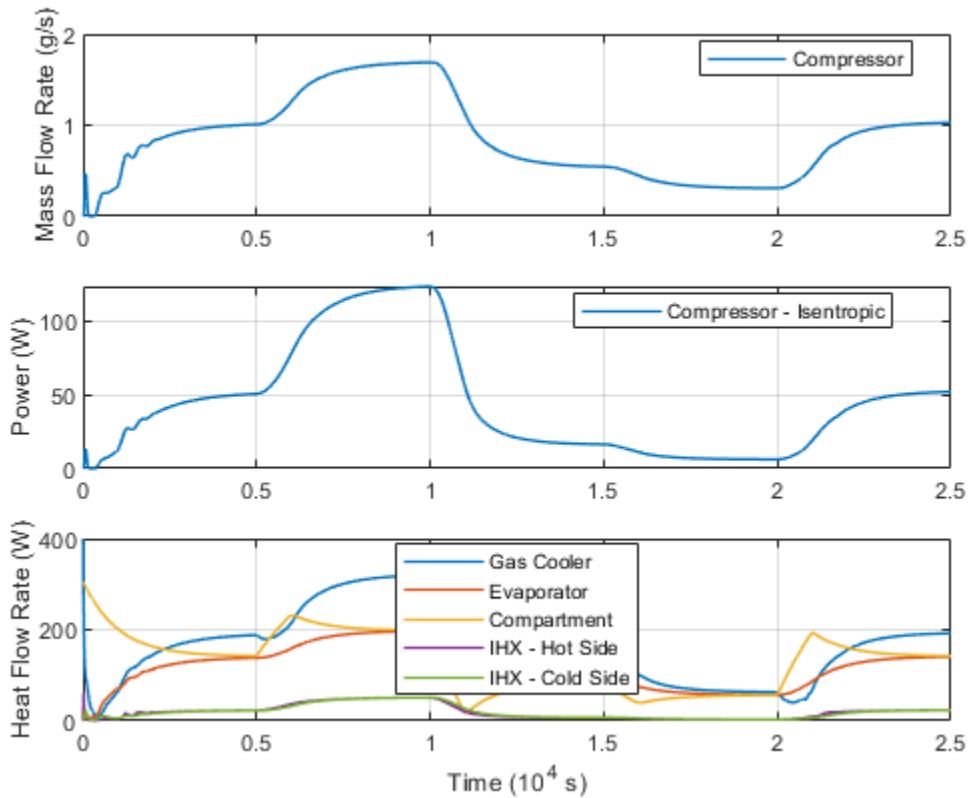
This plot shows the pressure and temperature at different points in the cycle. The evaporator pressure is at kept at around 3.5 MPa and the gas cooler pressure is nominally around 10 MPa, which is above the CO2 (R744) critical pressure of 7.4 MPa. Hence, this is a transcritical refrigeration cycle. The gas cooler pressure changes in response to changing environment temperature. At lower environment temperatures, the gas cooler pressure may drop to subcritical pressures.

Because a two-phase mixture enters the evaporator, the evaporator inlet temperature T5 is also the saturation temperature. Therefore, T6 - T5 represents the superheat in the evaporator, which is controlled by the expansion valve.
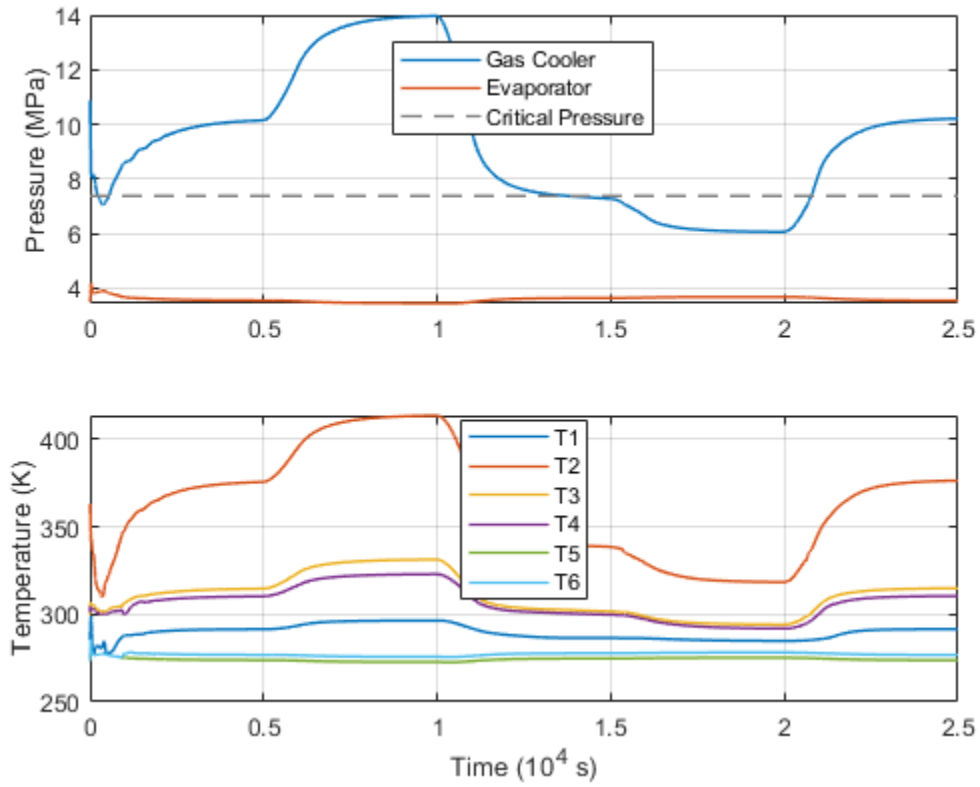
This plot shows the compressor pressure vs flow curves at different shaft speeds. The rotating shaft is not modeled here; the controller directly sets the shaft speed to produce the necessary flow rate.
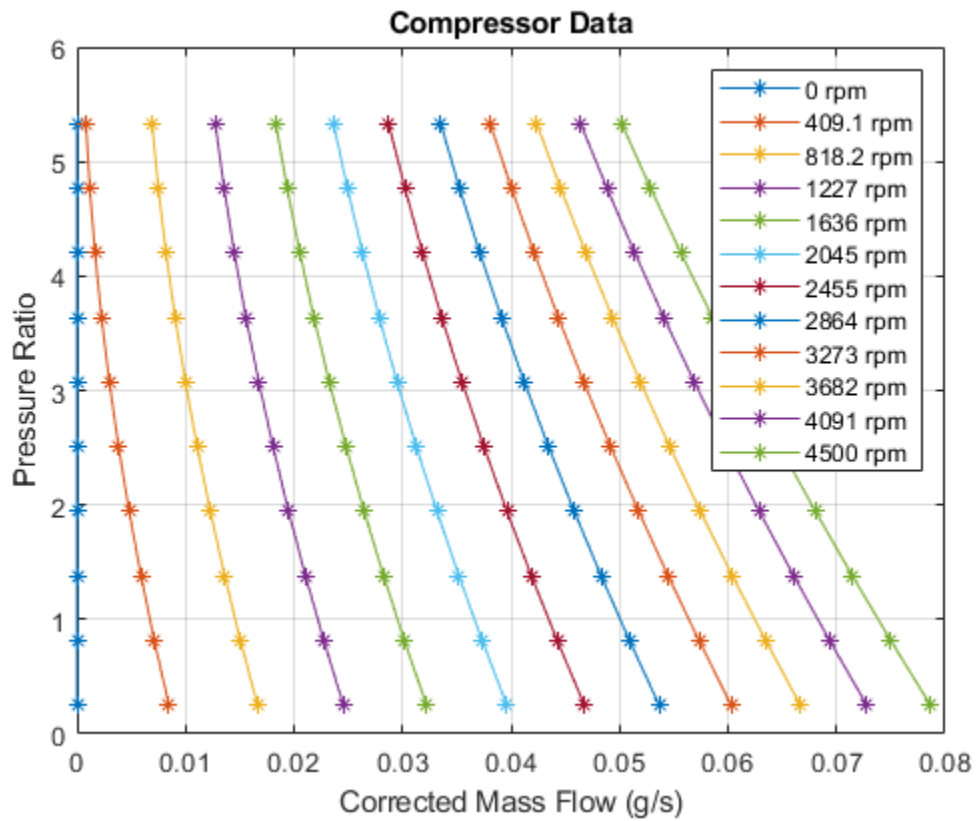
**Animation of Simscape Logging Results**

This figure shows the evolution of the fluid states in the transcritical refrigeration cycle over time. The 6 points in the cycle are the compressor inlet, condenser inlet, internal heat exchanger hot side inlet, expansion valve inlet, evaporator inlet, and internal heat exchanger cold side inlet, which are measured by sensors S1 to S6 in the model. They measurements are plotting on a pressure-enthalpy diagram. The contours are isotherms of CO2 (R744).
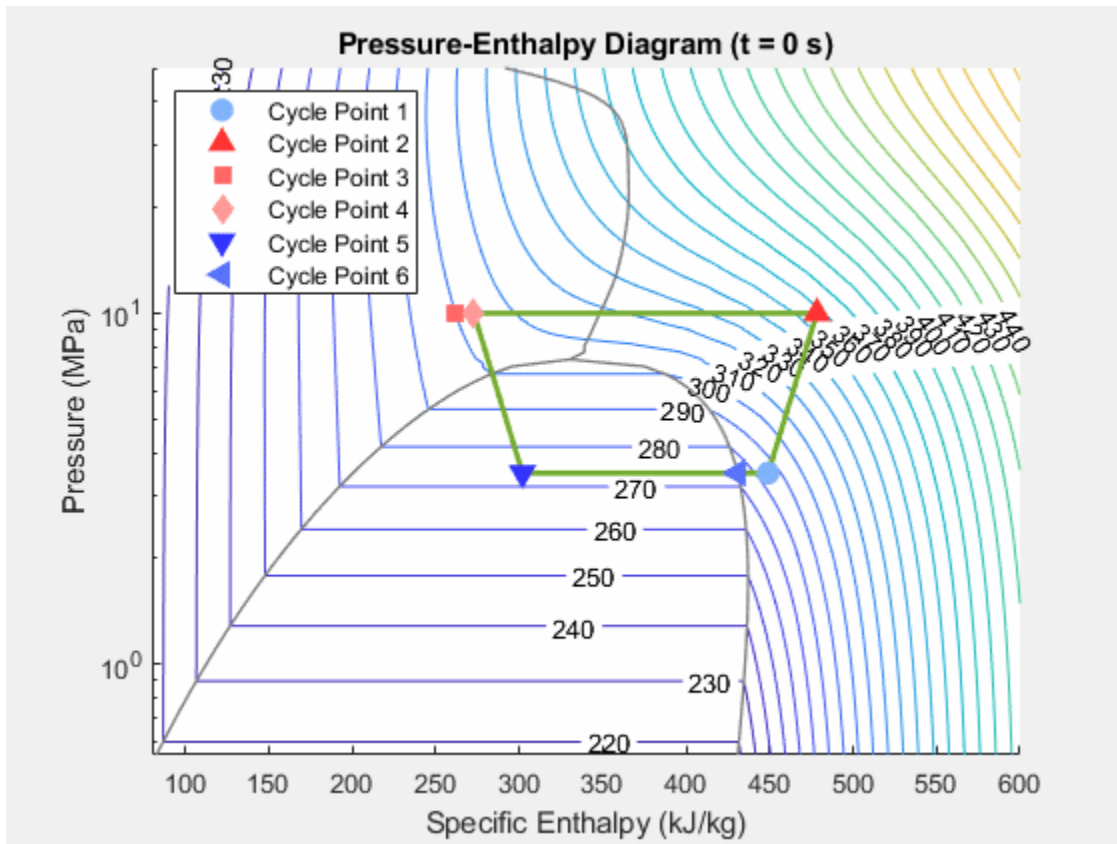
**Fluid Properties**

The following two figures plot the fluid properties of CO2 (R744) as a function of pressure (p) and normalized internal energy (unorm) and as a function of pressure (p) and specific internal energy (u), respectively. The fluid is a

- subcooled liquid when -1 <= unorm < 0;
- two-phase mixture when 0 <= unorm <= 1;
- superheated vapor when 1 < unorm <= 2.

The fluid property data is provided as a rectangular grid in p and unorm. Therefore, the grid in terms of p and u is non-rectangular.

The CO2 (R744) fluid property data can be found in `CO2PropertyTables.mat`.
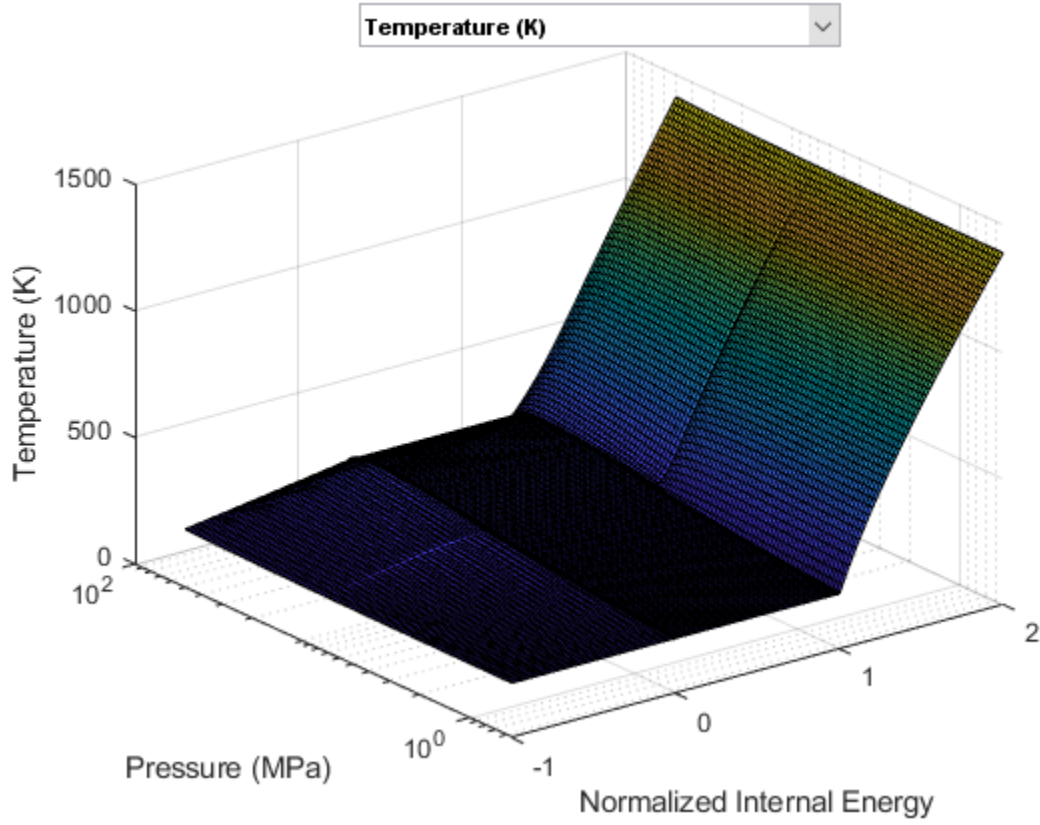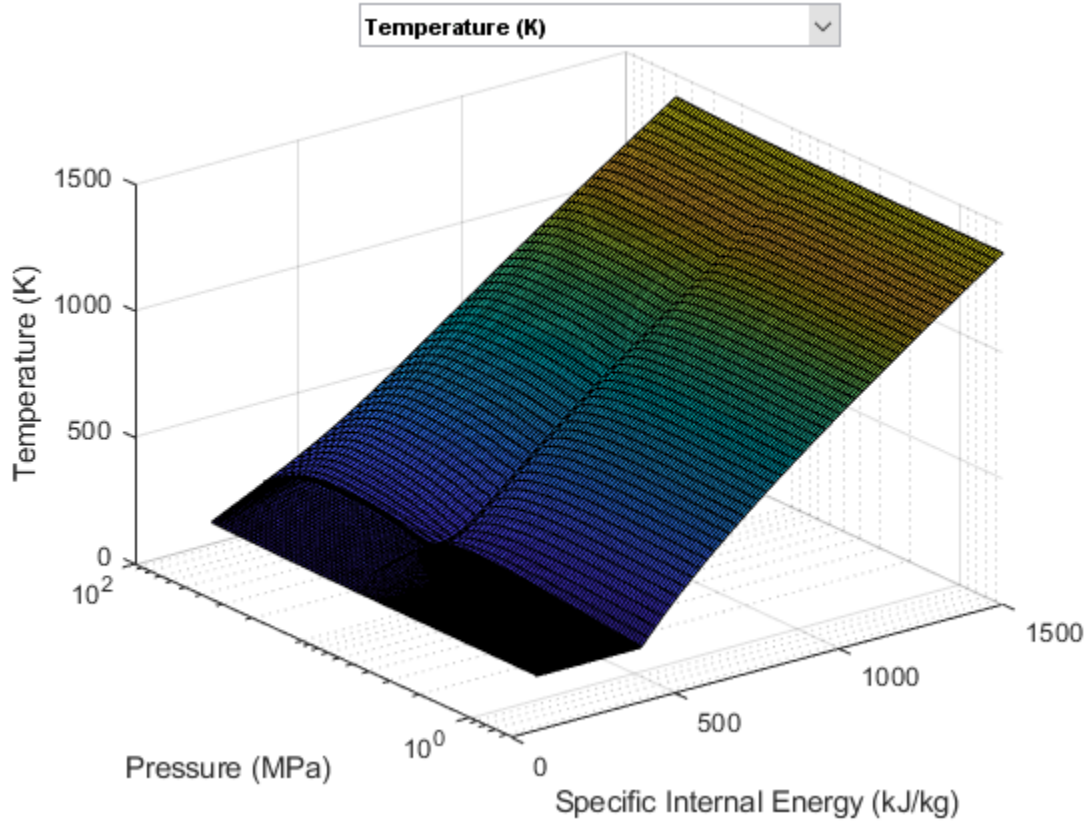
# House Heating System

This example shows how to model a simple house heating system. The model contains a heater, thermostat, and a house structure with four parts: inside air, house walls, windows, and roof.

The house exchanges heat with the environment through its walls, windows, and roof. Each path is simulated as a combination of a thermal convection, thermal conduction, and the thermal mass. The heater starts pumping hot air if room temperature falls below 18 degrees C and is turned off if the temperature exceeds 23 degrees C. The simulation calculates the heating cost and indoor temperatures.

The manual switch allows you to investigate system behavior with the heating system turned off.

**Model**
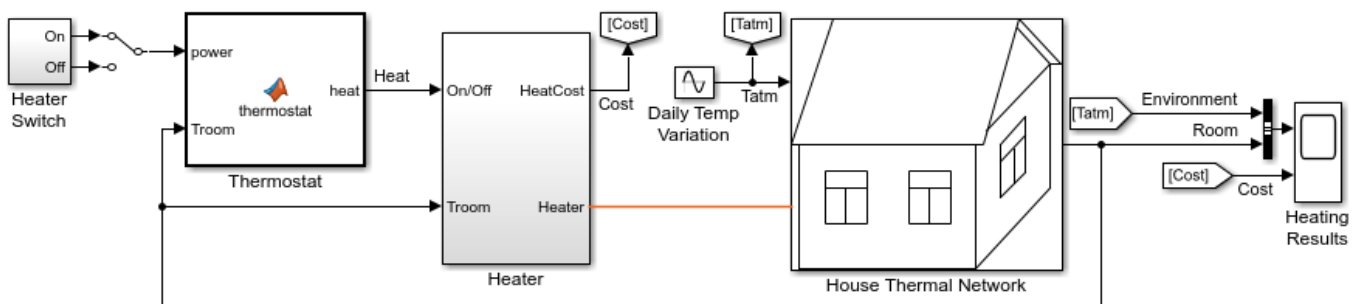


**House Heating System**

1. Plot temperature of wall, window, and roof (see code)
2. Plot heat flow through wall, window, and roof (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

Copyright 1998-2021 The MathWorks, Inc.

**House Thermal Network Subsystem**
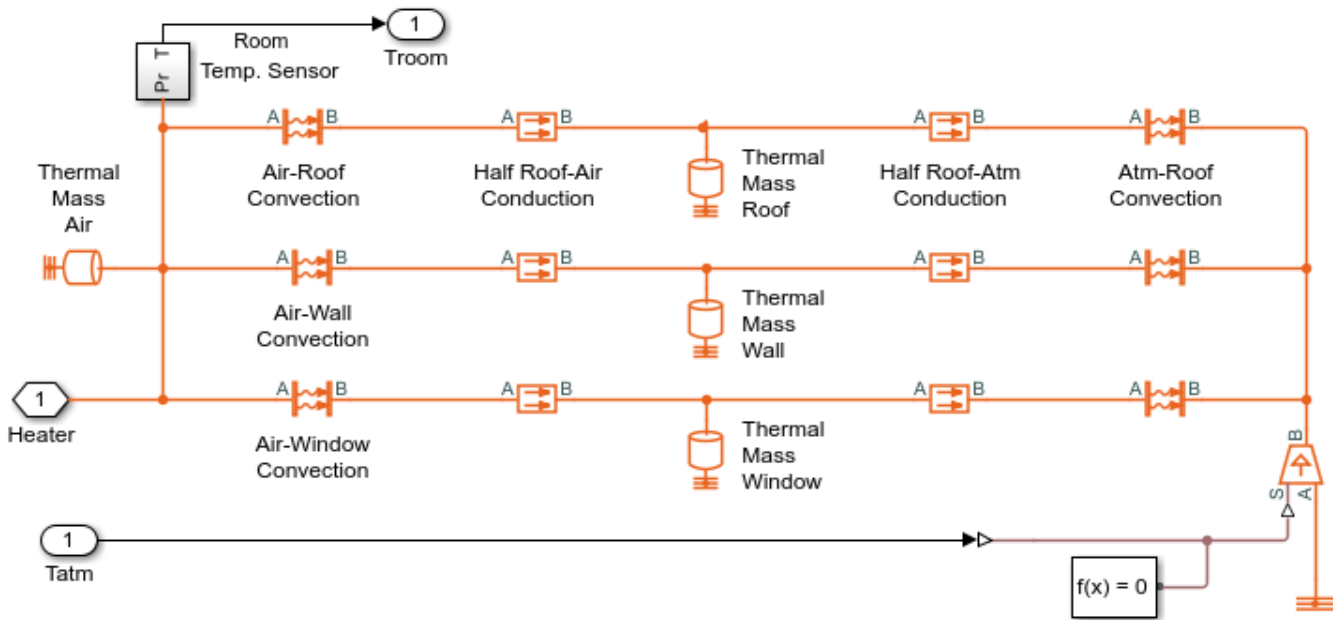


**Simulation Results from Scopes**
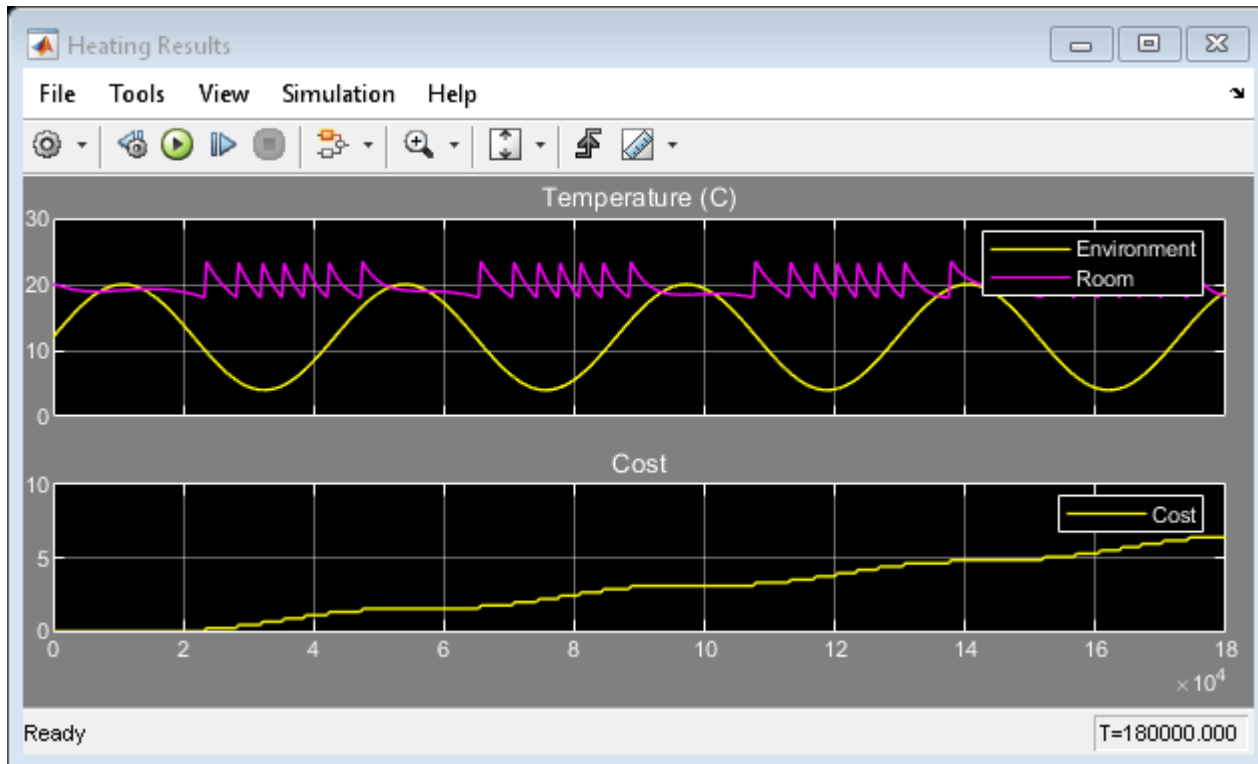
**Simulation Results from Simscape Logging**
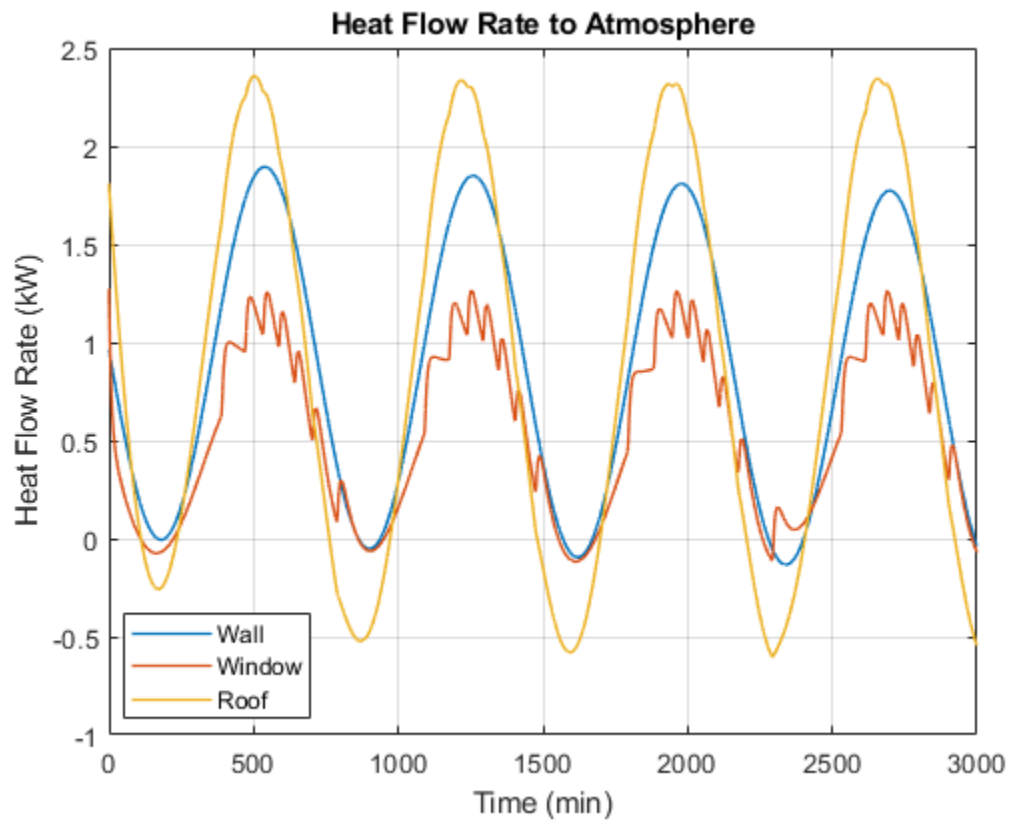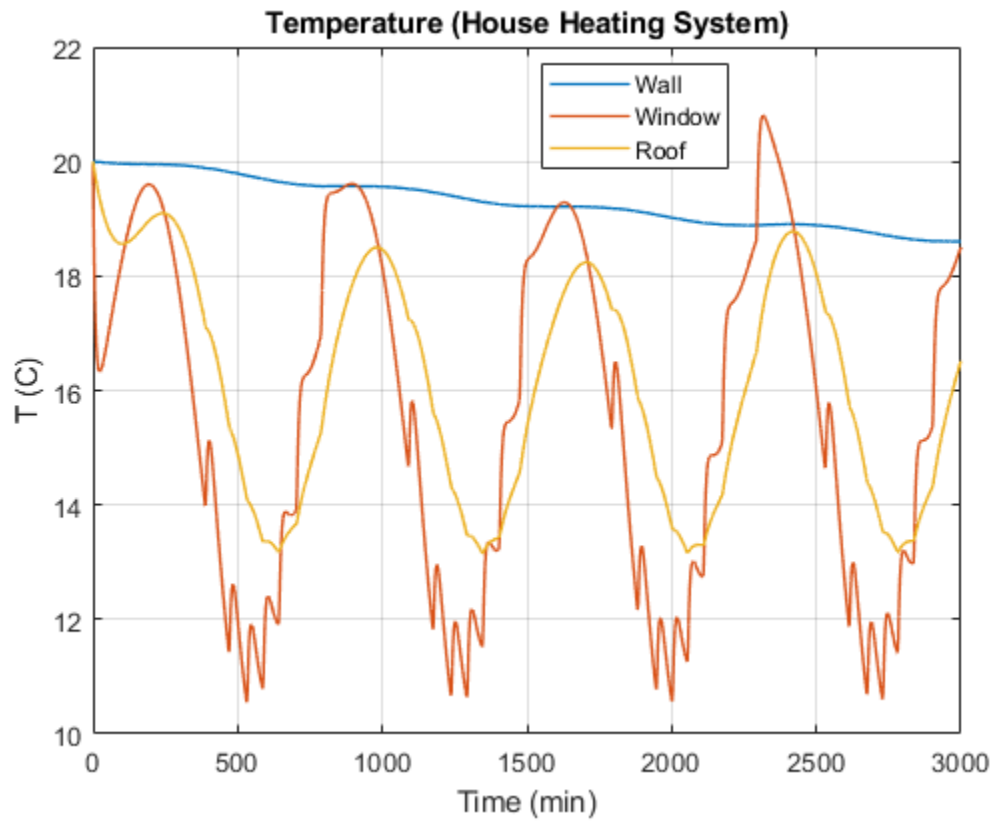
# Motor Thermal Circuit

This example shows how the thermal behavior of a brushless servomotor can be simulated using a lumped parameter model. Heat generated due to power losses in the stator iron stack, stator winding and rotor is represented by three heat flow sources: stator iron losses (Q_Iron), winding power losses (Q_Wind), and magnetizing and eddy current rotor losses (Q_Rotor). The losses were recorded during a motor typical cycle simulation and stored in the motor_losses.mat file. The motor thermal circuit is built of thermal conductances, thermal masses, and convective heat transfer blocks, which reproduce heat paths in the motor parts: winding, stator iron, motor case, rotor, front and rear bearing plates, and motor mounting flange. The motor exchanges heat with the atmosphere through the case-atmosphere, flange-atmosphere, and rear plate-atmosphere contacts. The ambient conditions are simulated with the ideal temperature source set to 300 K.

**Model**



**Motor Thermal Circuit**

1. Plot temperature of motor parts (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example
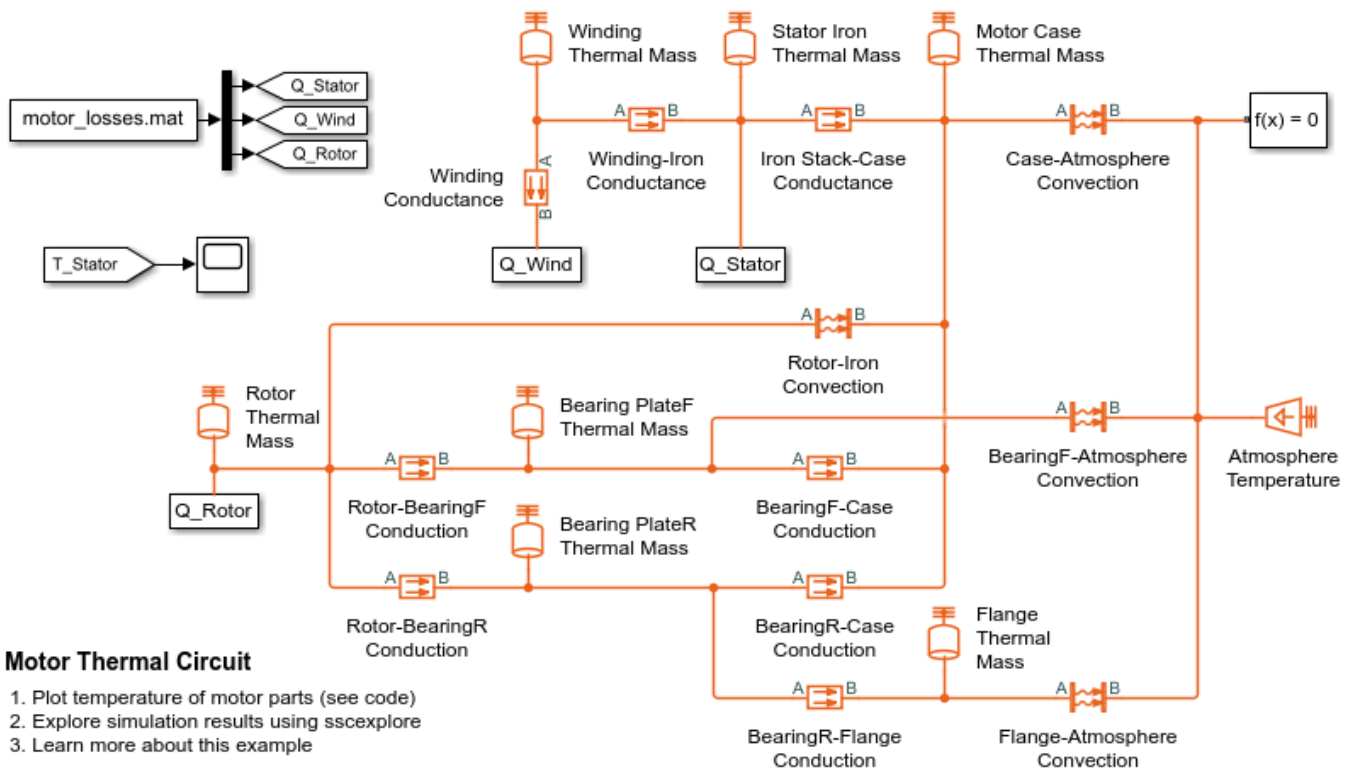
Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
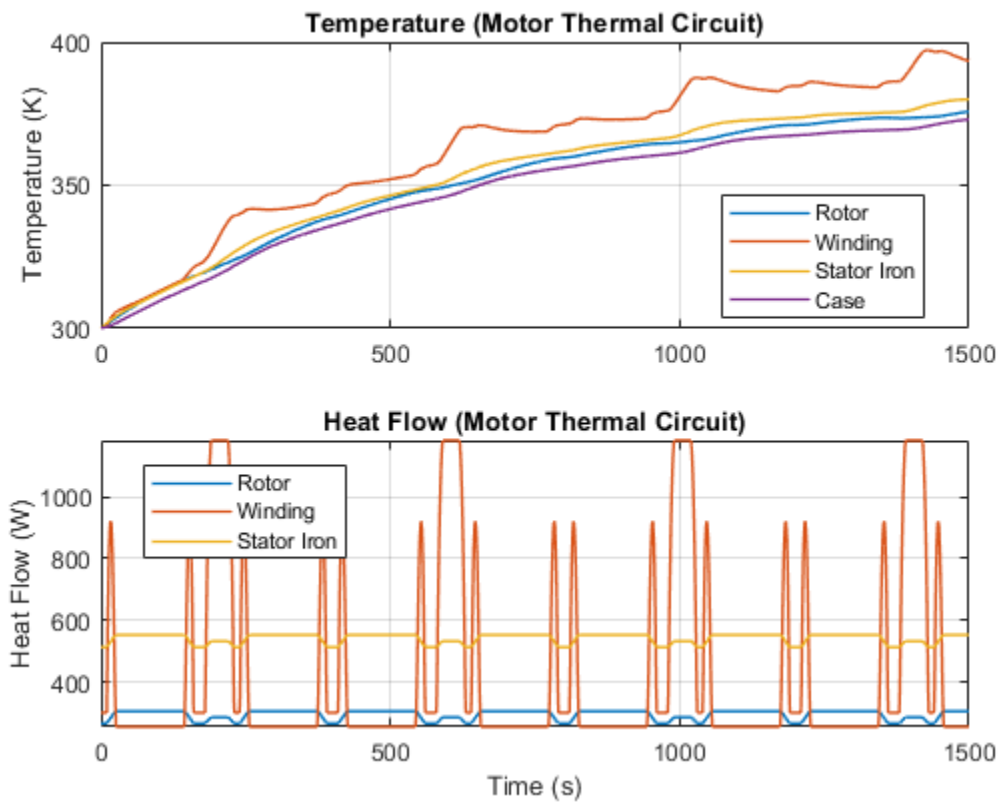
# Heat Conduction Through Iron Rod

This example shows the usage of thermal blocks for developing a model of a long iron rod that is heated with a heat source through face A. Face B and the outer cylindrical surface are open to atmosphere and subjected to forced heat convection. The rod is made of iron, is 20 cm in length and 2.25 cm in diameter. Two thermal models of the rod are considered.

In the Rod model, the rod is treated as a single thermal mass. There is one convective heat transfer element for the single cylindrical surface and two equally divided conductive heat transfer elements for the two halves of the rod.

The Segmented Rod model represents the rod as a set of 20 elements connected in series. Each element has the same structure as the Rod model, with 1/20-th of the overall thermal mass, one convective heat transfer element for the single cylindrical surface of the element, and two equally divided conductive heat transfer elements for the two halves of the element.

The temperature of every fifth element in the Segmented Rod is plotted with the temperature measured in the Rod model. The plot shows the temperature distribution along the Segmented Rod model, which can be compared to the temperature in the single element Rod model.

**Model**
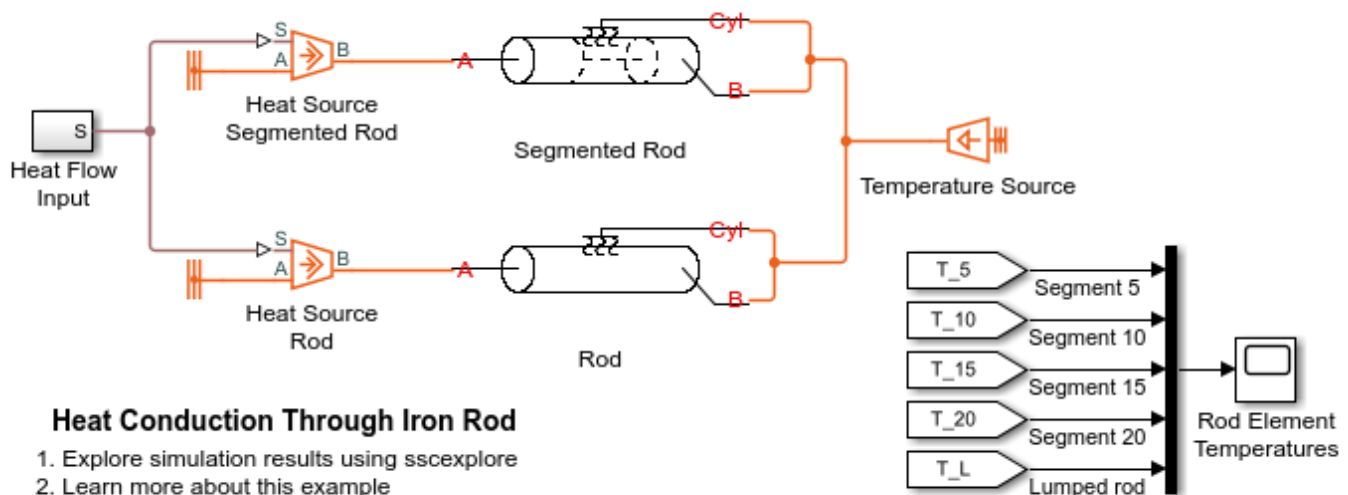


### Heat Conduction Through Iron Rod

1. Explore simulation results using sscexplore
2. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Segmented Rod Subsystem**
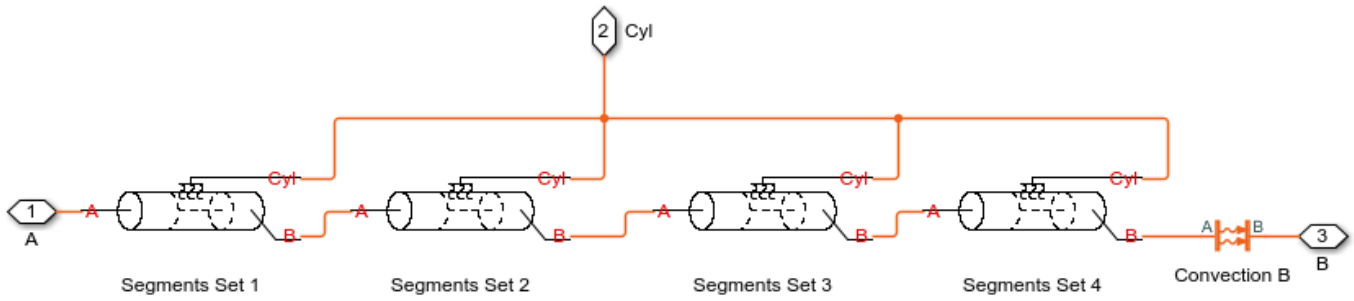


**Segments Set 1 Subsystem**
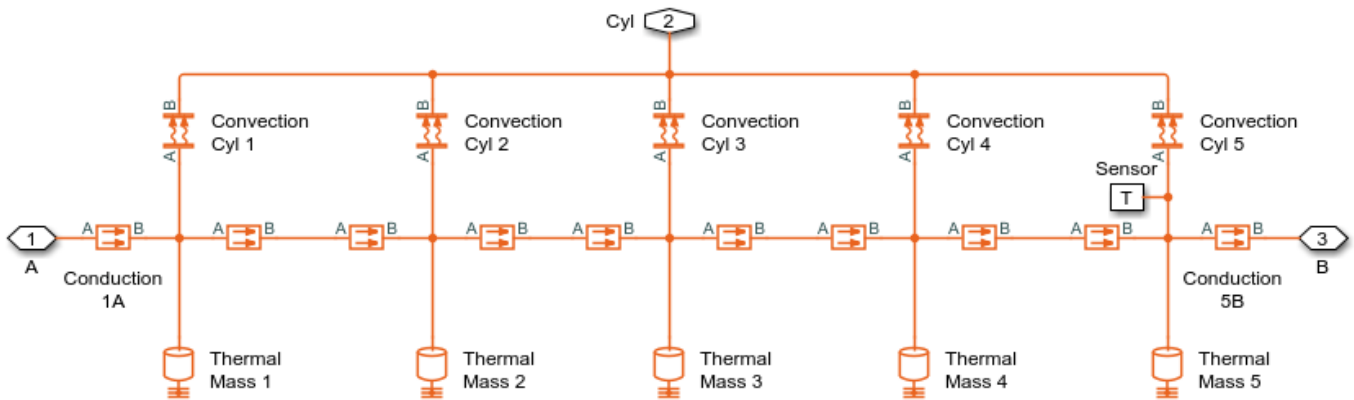
**Simulation Results from Scopes**
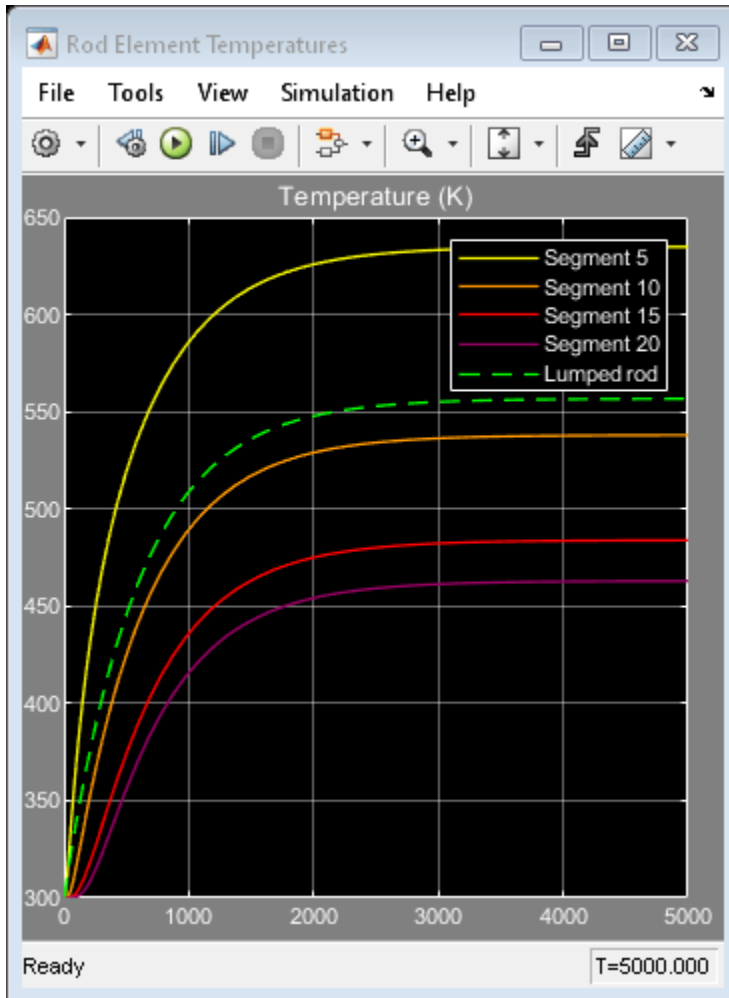
# Ultracapacitor Energy Storage with Custom Component

This example shows how to use the Simscape™ example library Capacitors_lib. The model is constructed using components from the example library. The circuit charges an ultracapacitor from a constant 0.05 amp current source, and then delivers a pulse of current to a load. The ultracapacitor enables a much higher current to be delivered than is possible directly from the current source. The library contains capacitor models with different levels of fidelity to allow exploration of the effect of losses and nonlinearity.

**Model**
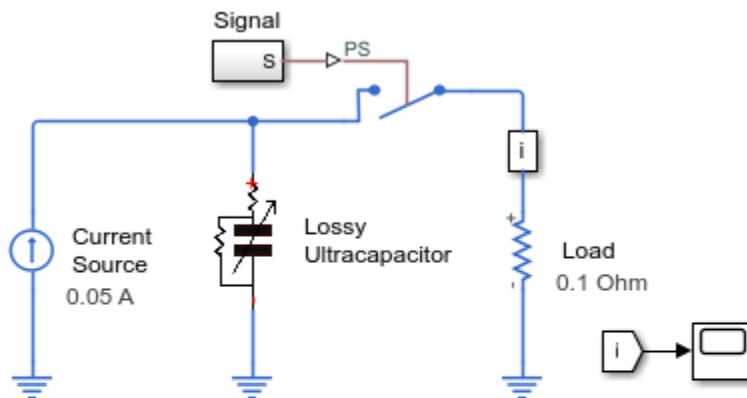


**Ultracapacitor Energy Storage with Custom Component**

1. Plot voltages and currents (see code)
2. Explore simulation results using sscexplore
3. Open the capacitor library
4. Learn more about this example

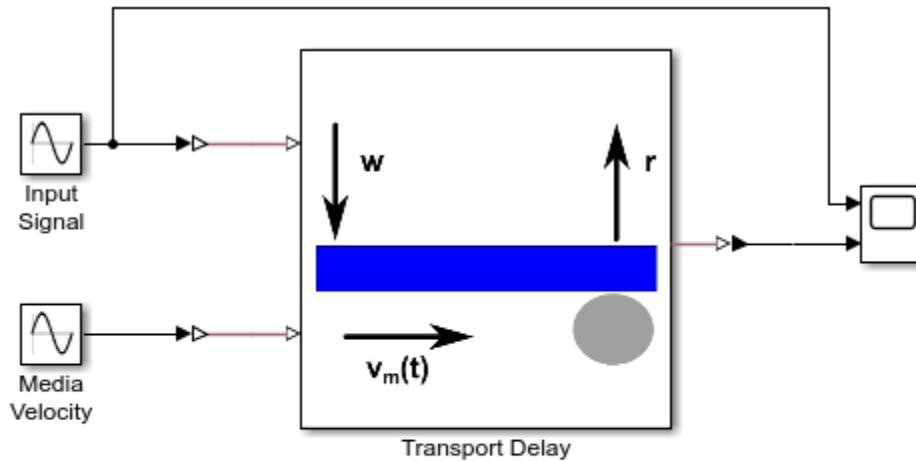Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
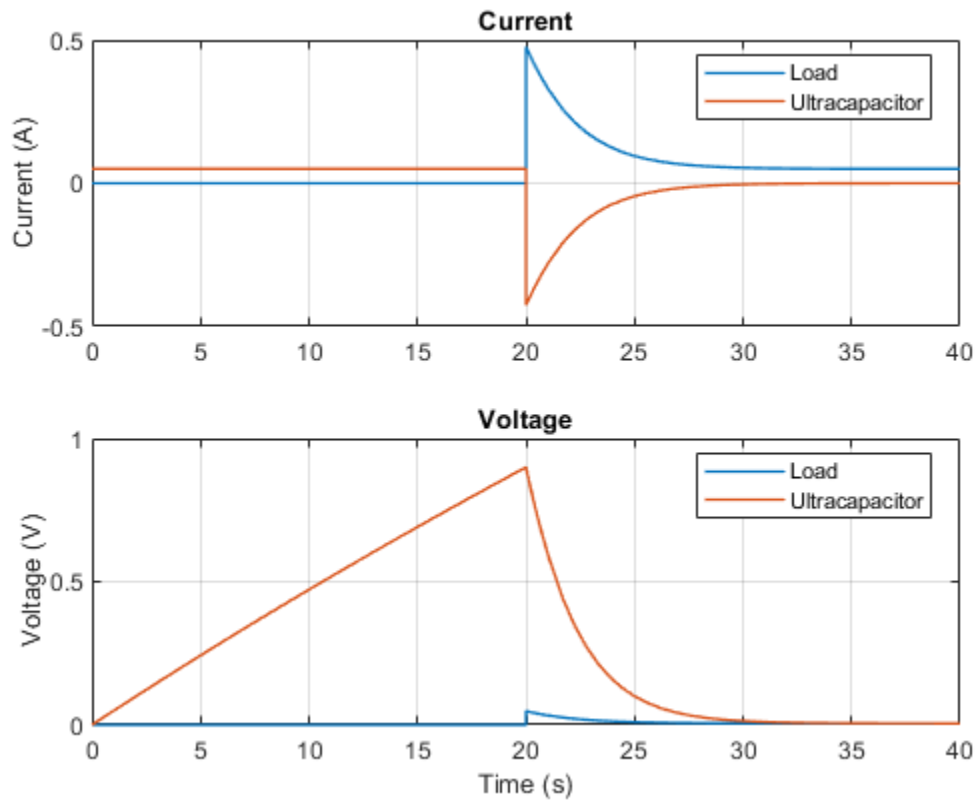
# Transmission Line

This example shows a lumped parameter transmission line model. It is built from a custom Simscape™ component that defines a single T-section segment. The model concatenates 50 segments, each of length 0.1m, thereby modeling a 5m length of coaxial cable. The transmission delay can be observed from the simulation results.

**Model**
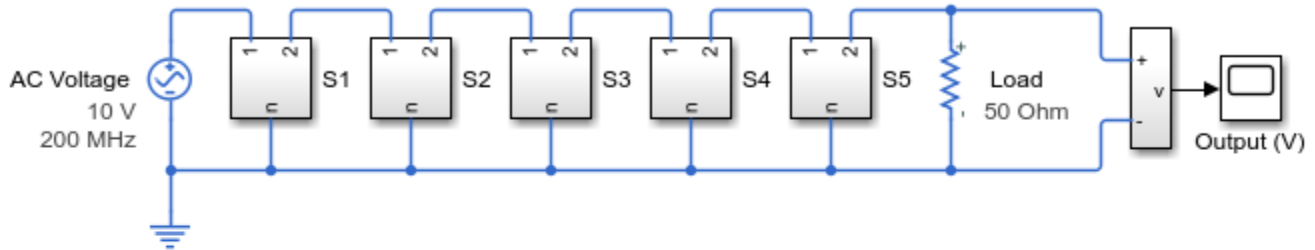


**Transmission Line**

1. Plot voltages along line (see code)
2. Explore simulation results using sscexplore
3. Open transmission line component library
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**S1 Subsystem**
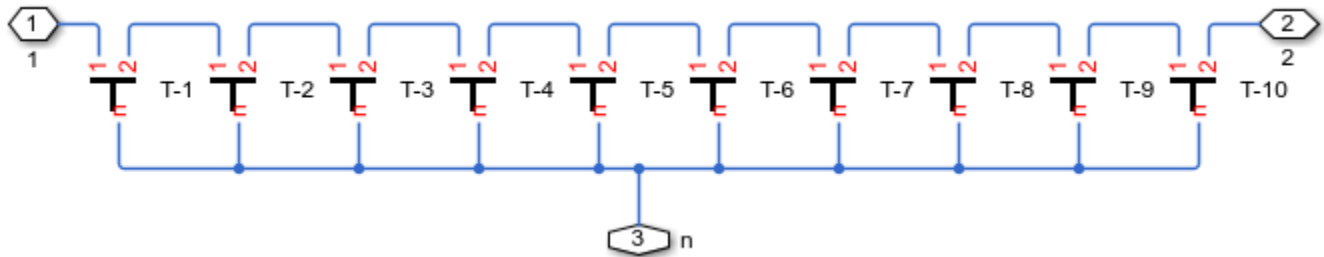
**Simulation Results from Simscape Logging**
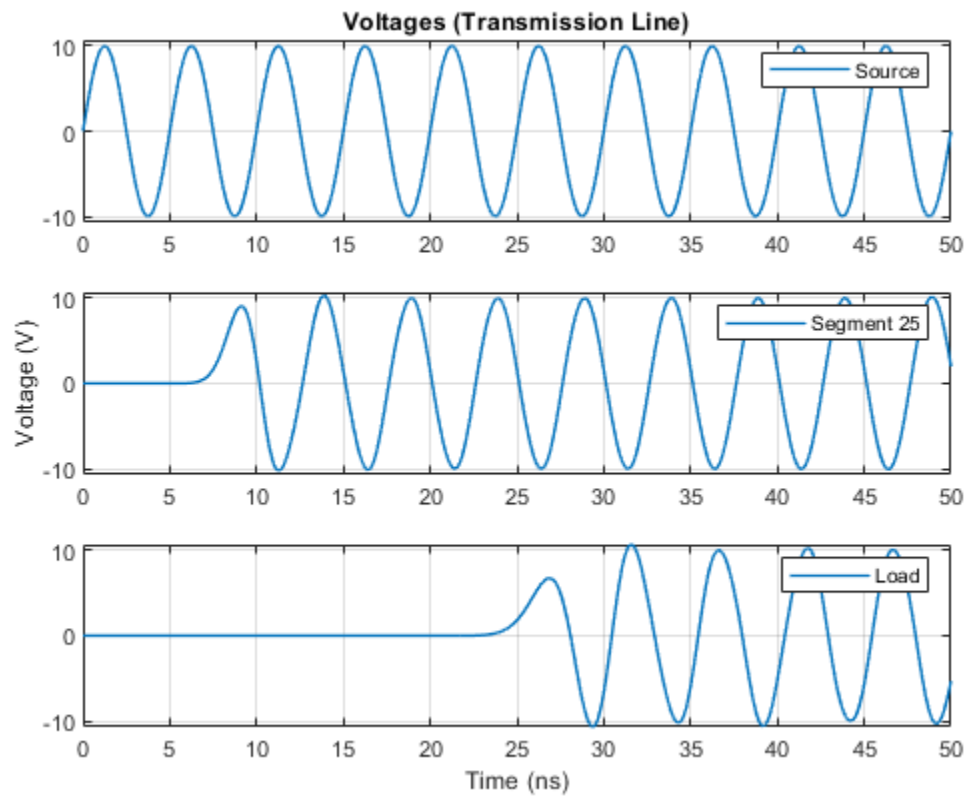
# Battery Cell with Custom Electrochemical Domain

This example shows how to use the Simscape™ example library ElectroChem_lib. In the model Fe3+ ions are reduced to Fe2+, and Pb is oxidized to Pb2+, thereby releasing chemical energy. The molar flow rate of lead ions is half that of the iron ions as two electrons are exchanged when Pb is oxidized to Pb2+. The chemical potential of the Pb source is by convention zero as it is a solid.

The electrochemical library defines the through variable as molar flow rate, and the across variable as chemical potential. This example is motivated by Pecheux, F., Allard, B., Lallement, C. & Vachoux, A. & Morel, H., "Modeling and Simulation of Multi-Discipline Systems using Bond Graphs and VHDL-AMS", International Conference on Bond Graph Modeling and Simulation (ICBGM), New Orleans, USA, 23-27 Jan. 2005

**Model**
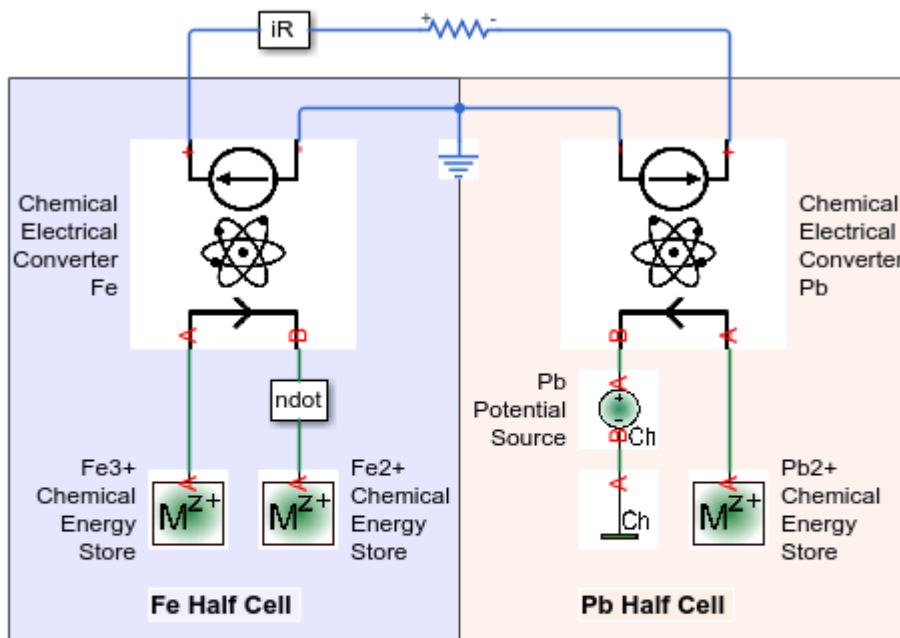


### Battery Cell with Custom Electrochemical Domain

1. Plot voltage and molar flow rates (see code)
2. Explore simulation results using sscexplore
3. Open the electrochemical library
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**
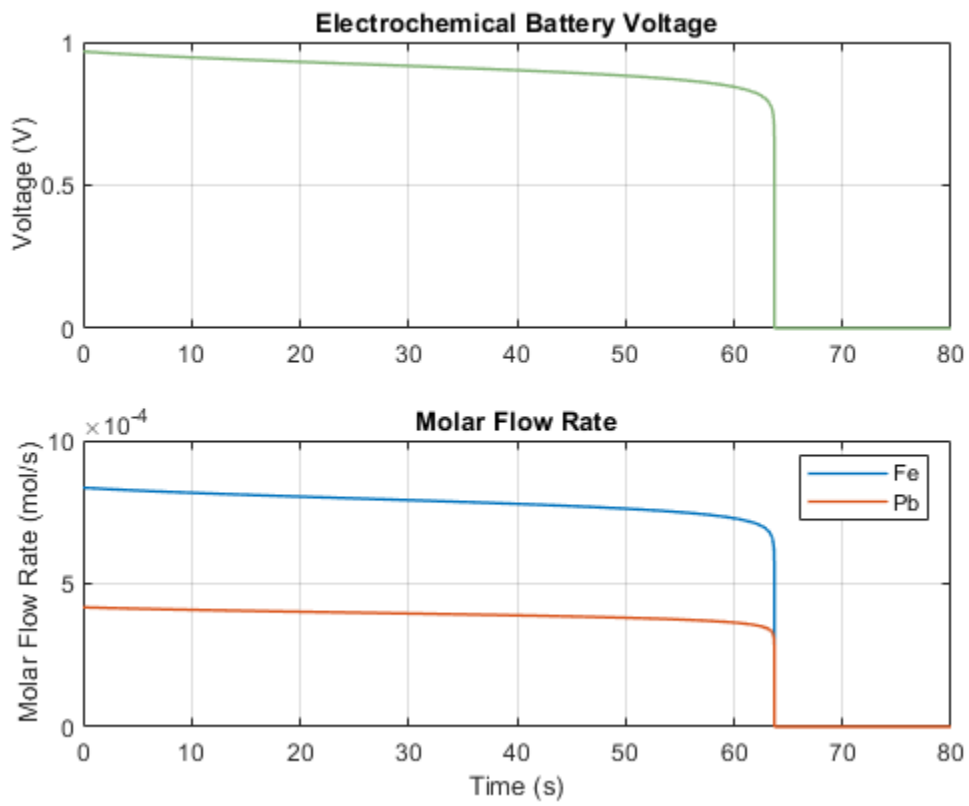
# Variable Transport Delay

This example shows how to use Simscape™ to model a variable transport delay. The Transport Delay block models signal propagation through media moving between the Input and the Output terminals. The media velocity may vary, thus it is specified through the block port. The distance between the terminals as well as the initial output are constant and they are specified as block parameters.

**Model**



**Variable Transport Delay**

1. Explore simulation results using sscexplore
2. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Scopes**
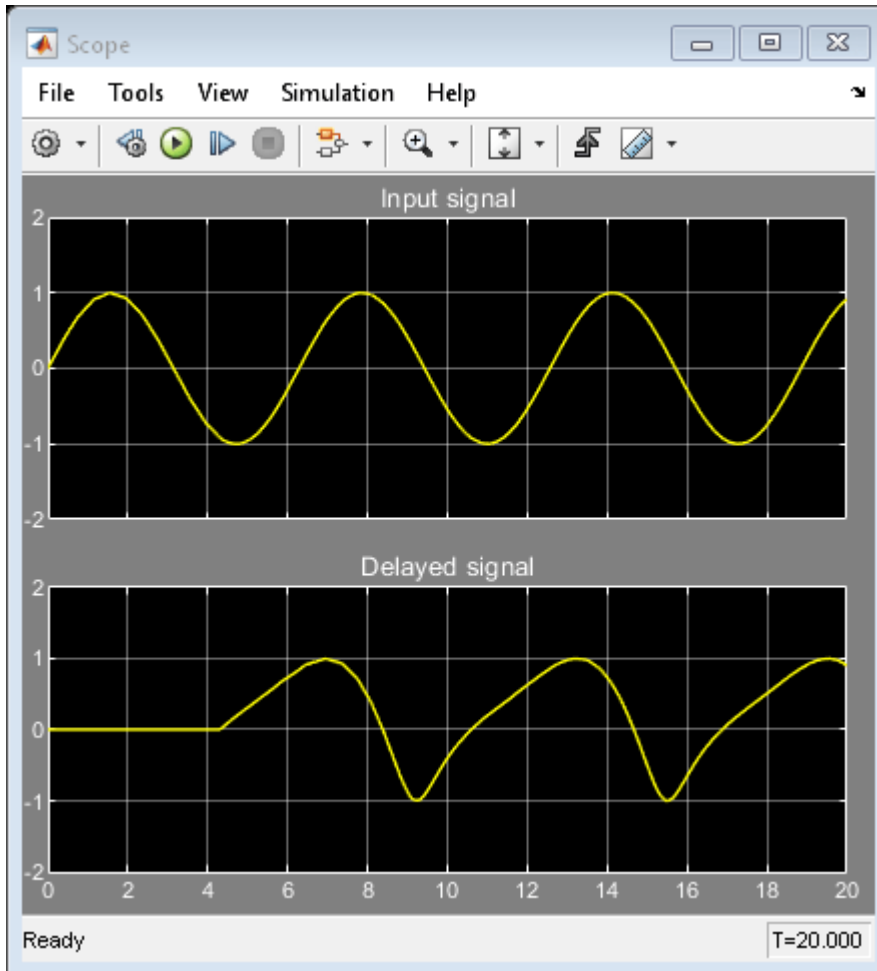
# Asynchronous PWM Voltage Source

This example shows how the Simscape™ Foundation Library PS Asynchronous Sample & Hold block can be used to build components with more complex behaviors. The model implements a controllable PWM voltage source where the PWM on-time (the duty cycle) is proportional to the physical signal input u.

The PS Asynchronous Sample & Hold block enables accurate capturing of the PWM signal edges and faster simulation speed than a discrete implementation. This model should be run using a variable-step solver so that equations are solved at exactly every PWM switching event. This gives perfect resolution of the desired duty cycle. Running fixed step would require running the entire simulation at a step size equal to the desired duty cycle resolution.

For an alternative discrete-time implementation, see the Discrete-Time PWM Voltage Source example, ssc_pwm_discrete. The discrete-time version is better suited to fixed-step solvers and hardware-in-the-loop applications.

**Model**
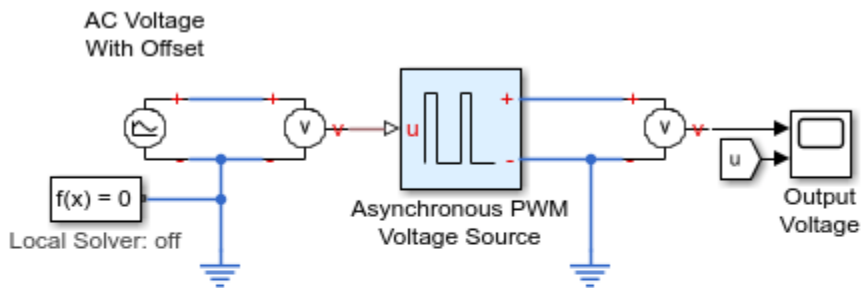


## Asynchronous PWM Voltage Source

1. Plot output voltage and step size (see code)
2. Explore simulation results using sscexplore
3. Compare with Discrete PWM Voltage Source example 4.
Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Asynchronous PWM Voltage Source Subsystem**
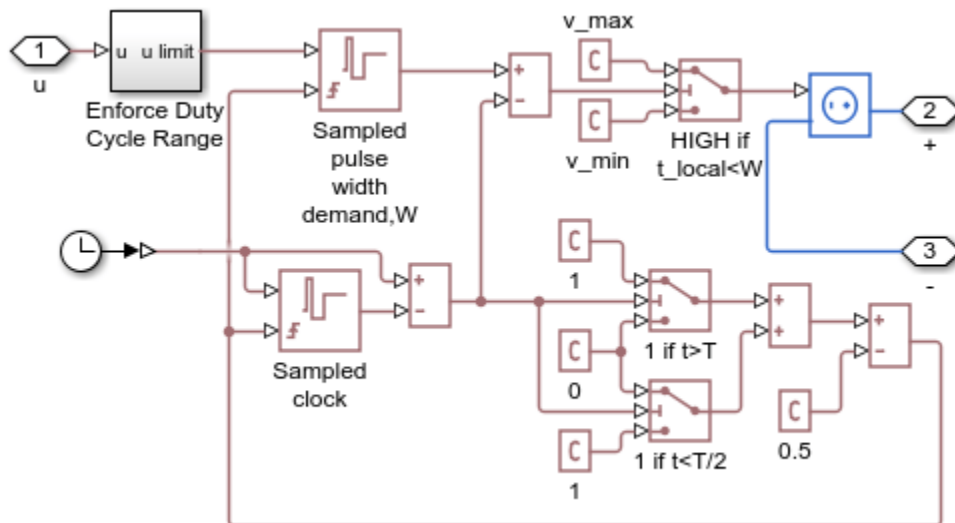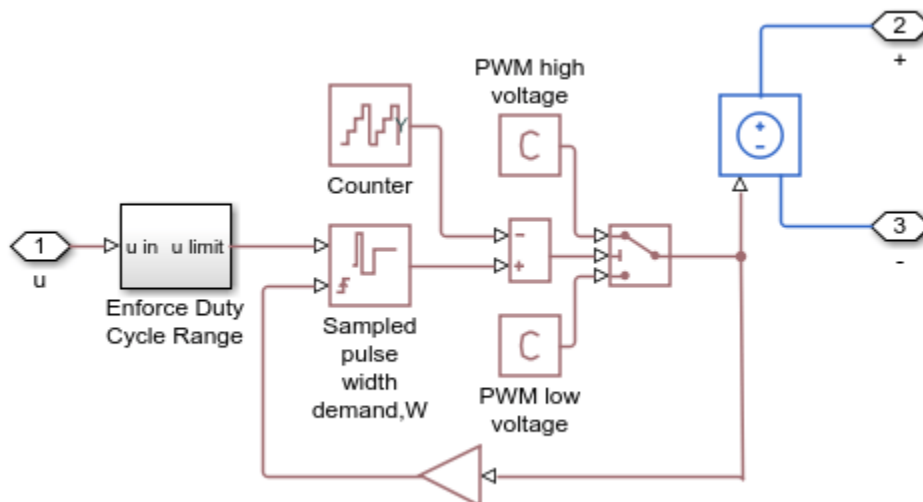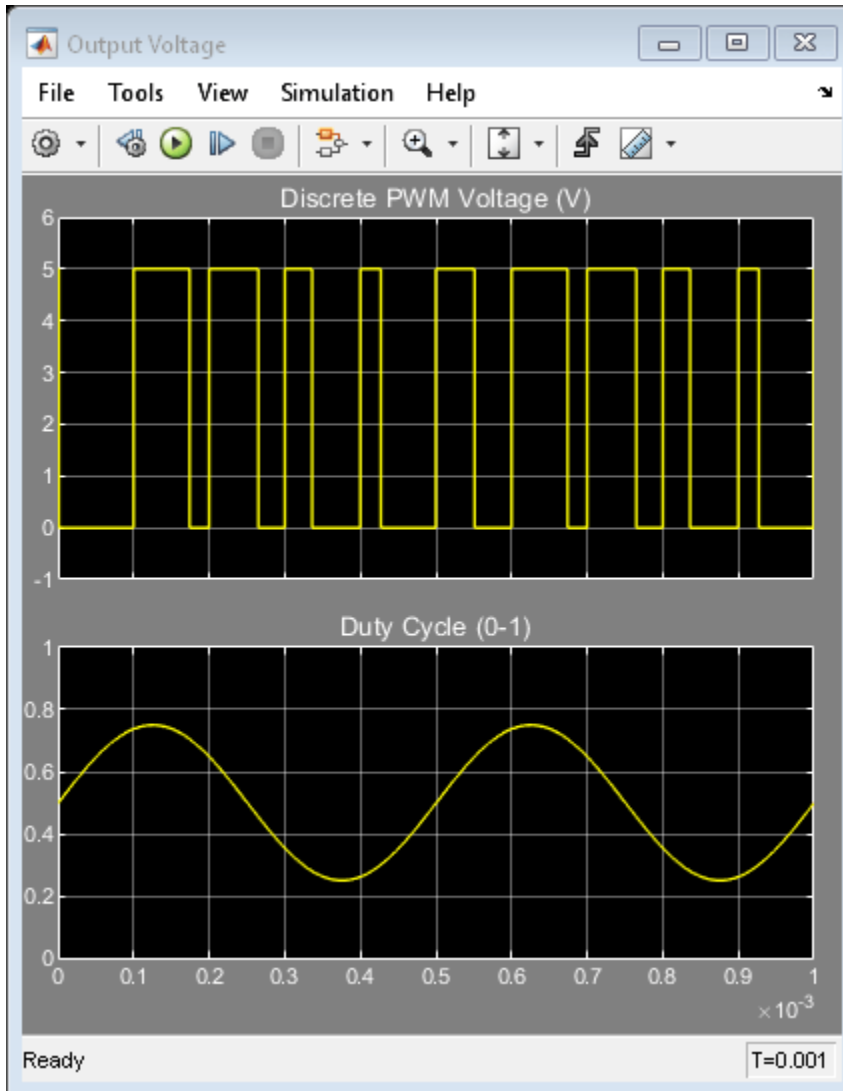
**Simulation Results from Scopes**
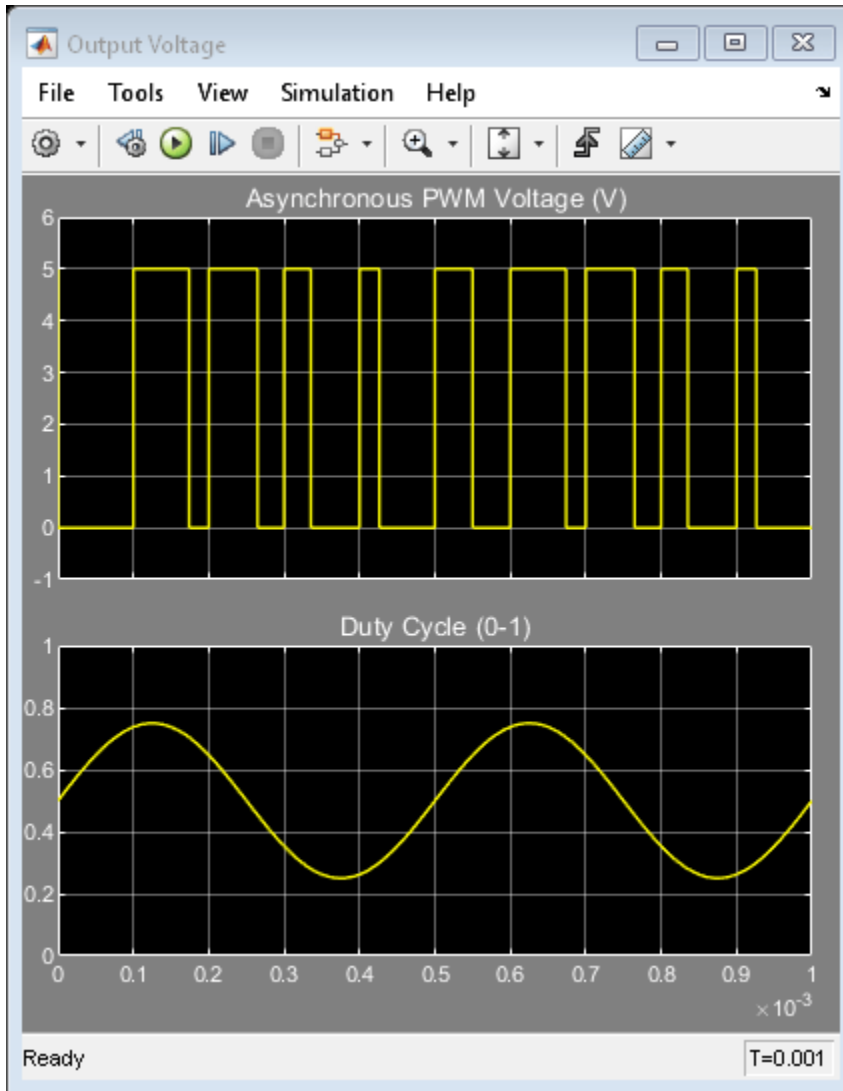


**Simulation Results from Simscape Logging**

These plots show the output voltage of the Asynchronous PWM Voltage Source as well as the step size used during simulation. Because a variable-step solver is used, large steps can be taken until the start or end of a PWM pulse is reached.
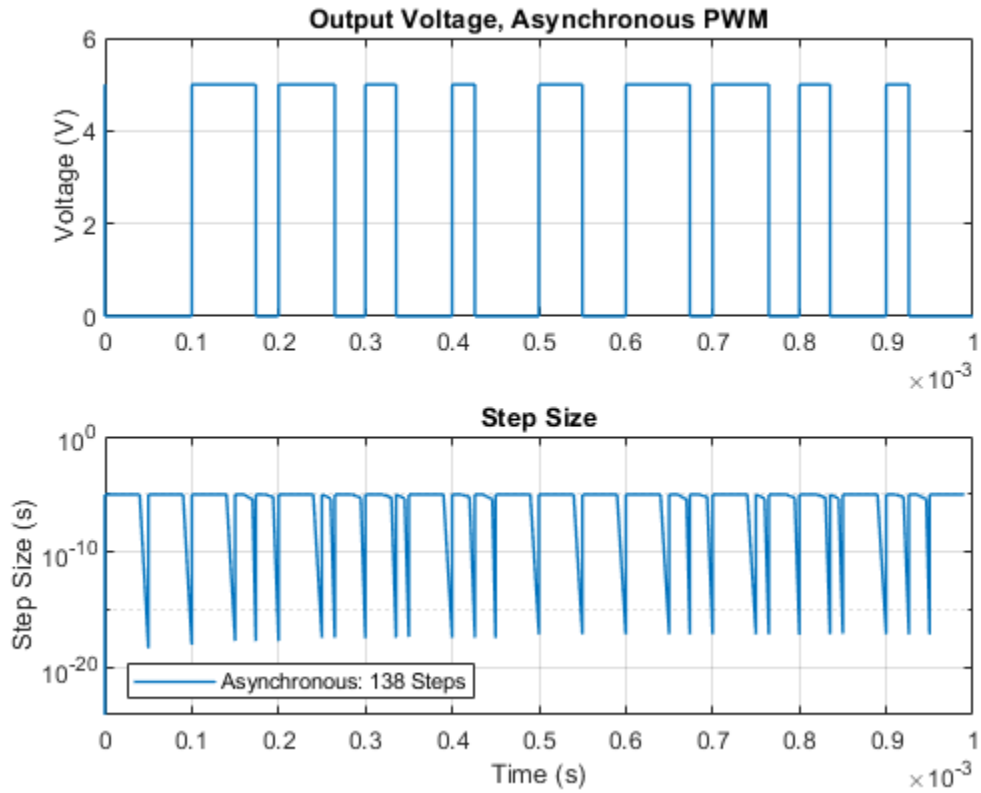
These plots compare the results of ssc_pwm_discrete and ssc_pwm_asynchronous. The models produce nearly the same voltage signal, but the asynchronous model can run with a variable-step solver and can take larger steps. This can result in faster simulations.
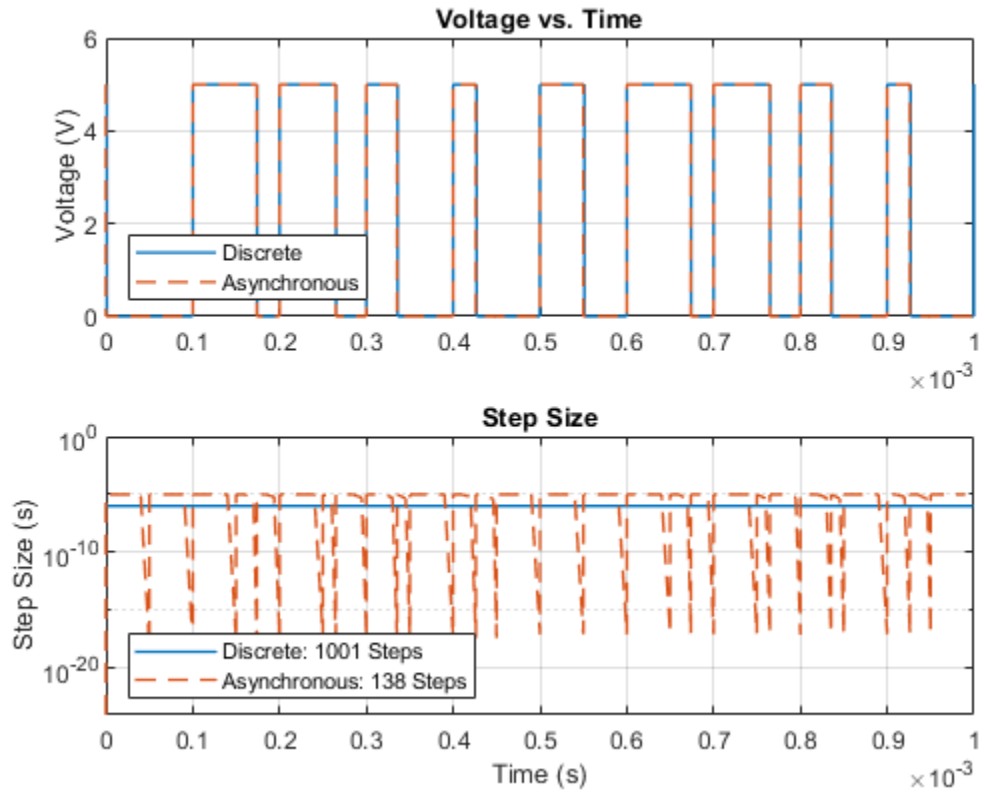
# Discrete-Time PWM Voltage Source

This example shows how the discrete-time Simscape™ Foundation Library PS Counter block can be used to build components with more complex behaviors. The model implements a controllable PWM voltage source where the PWM on-time (the duty cycle) is proportional to the physical signal input u.

For an alternative asynchronous implementation, see the Asynchronous PWM Voltage Source example, ssc_pwm_asynchronous. The discrete-time version is better suited to fixed-step solvers and hardware-in-the-loop applications, whereas the asynchronous implementation is better suited to fast desktop simulation using variable-step solvers.

**Model**
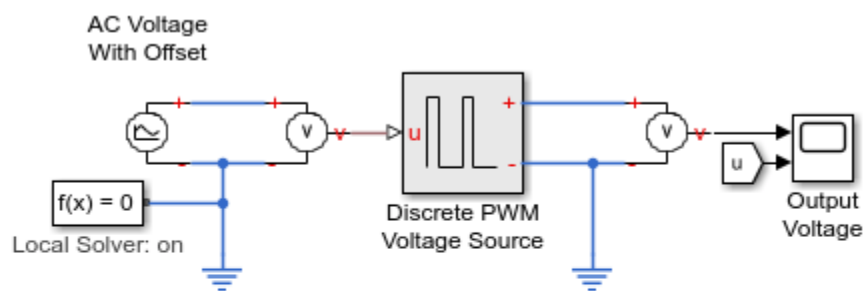


### Discrete-Time PWM Voltage Source

1. Plot output voltage and step size (see code)
2. Explore simulation results using sscexplore
3. Compare with Asynchronous PWM Voltage Source example
4. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Discrete PWM Voltage Source Subsystem**

**Simulation Results from Scopes**



**Simulation Results from Simscape Logging**

These plots show the output voltage of the Discrete PWM Voltage Source as well as the step size used during simulation. Because a fixed-step solver is used, the step size remains constant during the simulation. The step size used must be small enough to capture the duty cycle variation with an acceptable resolution.
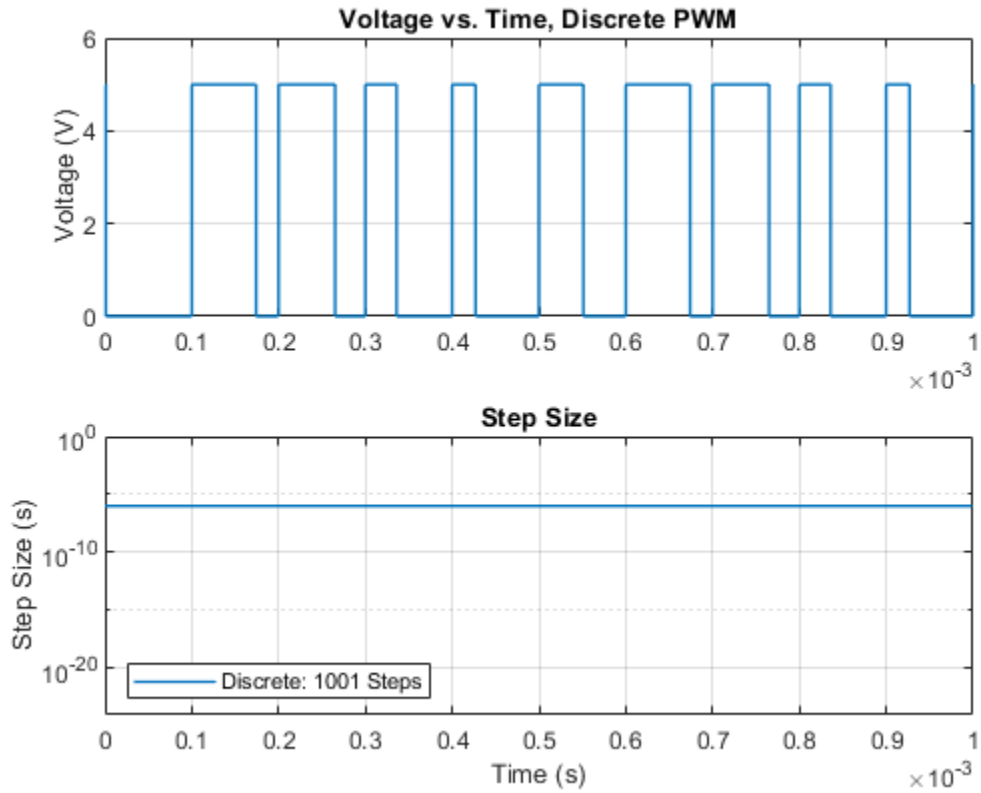
These plots compare the results of ssc_pwm_discrete and ssc_pwm_asynchronous. The models produce nearly the same voltage signal, but the asynchronous model can run with a variable-step solver and can take larger steps. This can result in faster simulations.
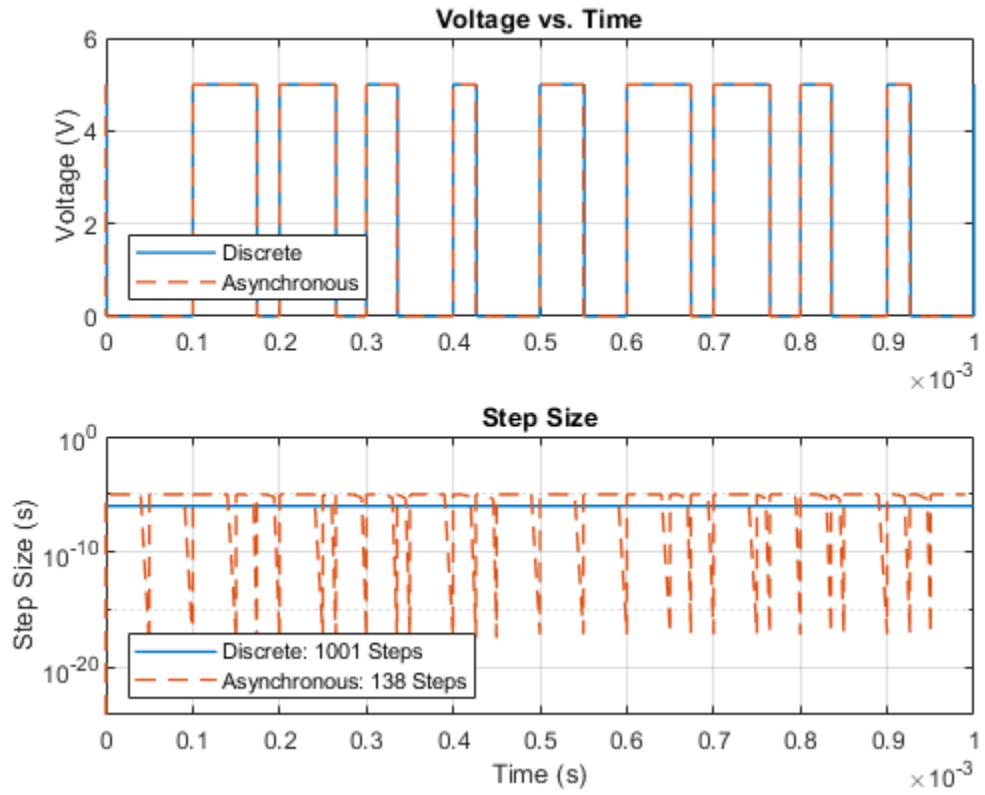
# Actuation Circuit with Custom Pneumatic Components

This example shows how to model a controlled actuator using simplified custom pneumatic components. There are two across variables, defined as pressure and temperature, and two through variables, defined as mass flow rate and heat flow rate. The simplified approach means that every node in the circuit must have a volume of gas associated with it. This physical volume of gas in the circuit is represented by the Constant Volume Pneumatic Chamber blocks, the Pneumatic Piston Chamber blocks, and the Pneumatic Atmospheric Reference block. Conversely, the Foundation Library gas components require no such connection rules at every node. See the "Pneumatic Actuation Circuit" on page 20-160 example for a more capable way of modeling pneumatic systems using Foundation Library gas components.

During the simulation, the spool of the directional valve is moved to its maximum positive displacement, and the actuator moves the load until it reaches the actuator end stop. The valve is then centered, and the load is held. Next the valve moves to its maximum negative displacement and the actuator retracts to its minimum length. The valve is then centered again, locking the load in position.

**Model**
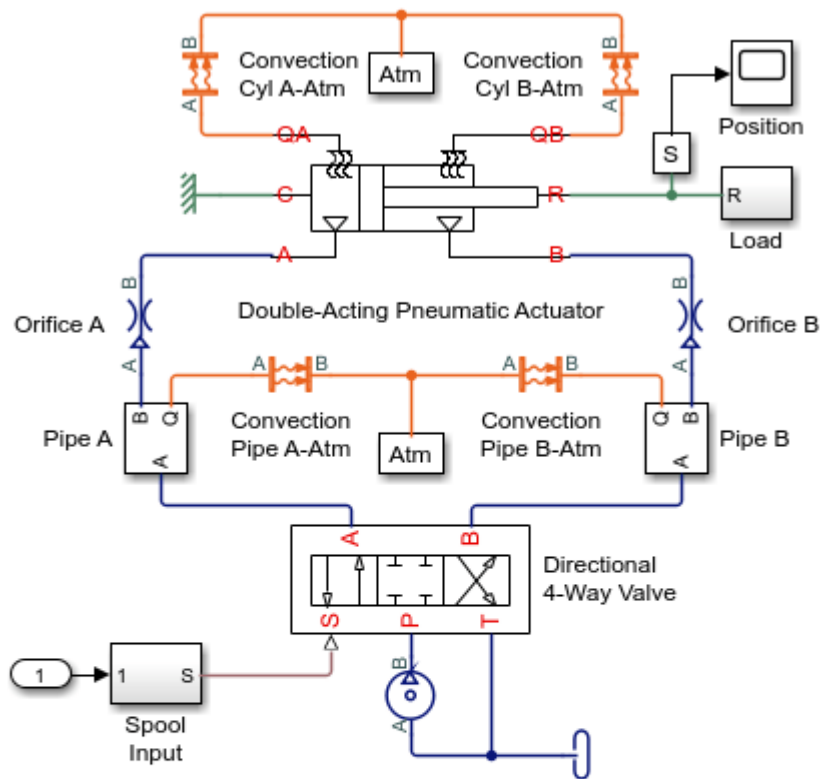


## Actuation Circuit with Custom Pneumatic Components

1. Plot positions for valve spool and actuator rod (see code)
2. Plot mass flow rates through valve (see code)
3. Plot pressures and temperatures for actuator (see code)
4. Explore simulation results using sscexplore
5. Open the custom pneumatic library
6. Learn more about this example
7. See Pneumatic Actuation Circuit for a more capable way of modeling pneumatic systems using Foundation Library gas components.

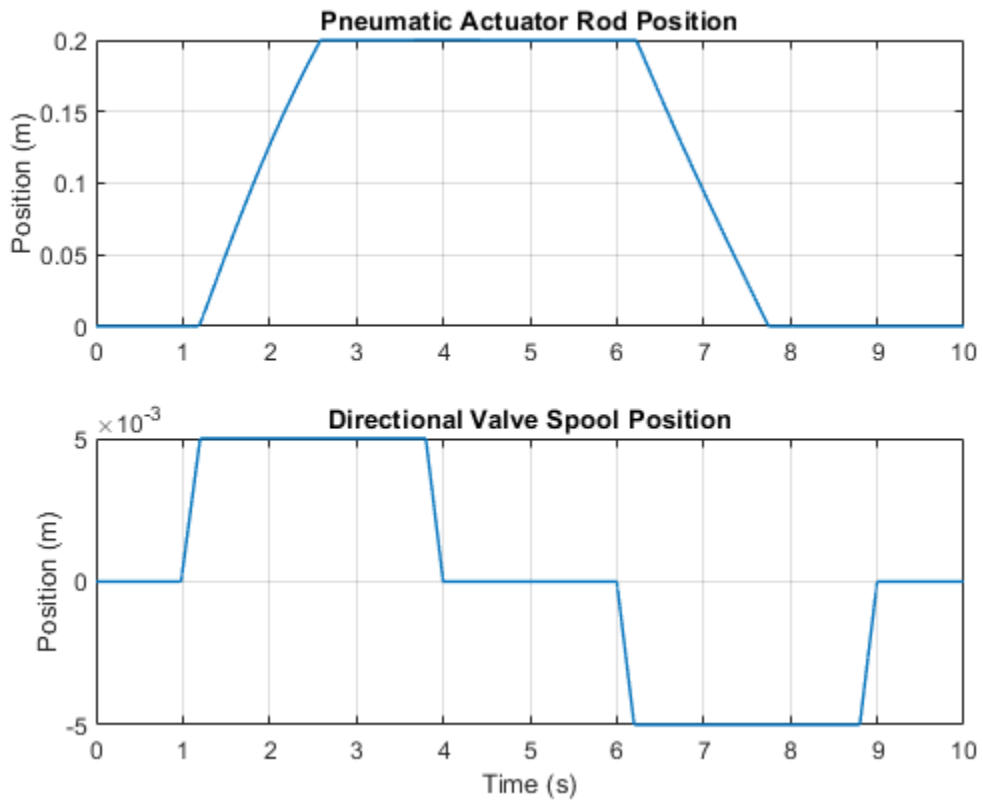Copyright 2008-2021 The MathWorks, Inc.

**Directional 4-Way Valve Subsystem**
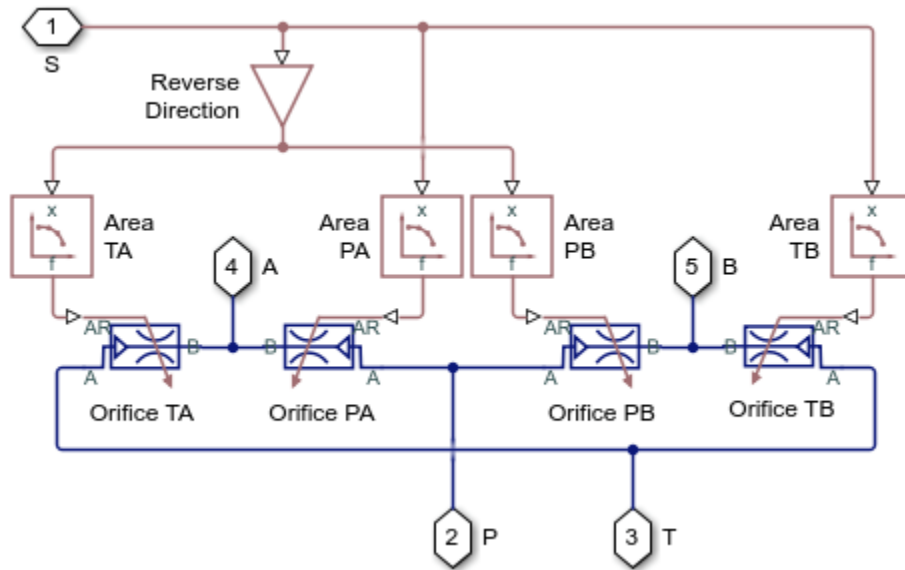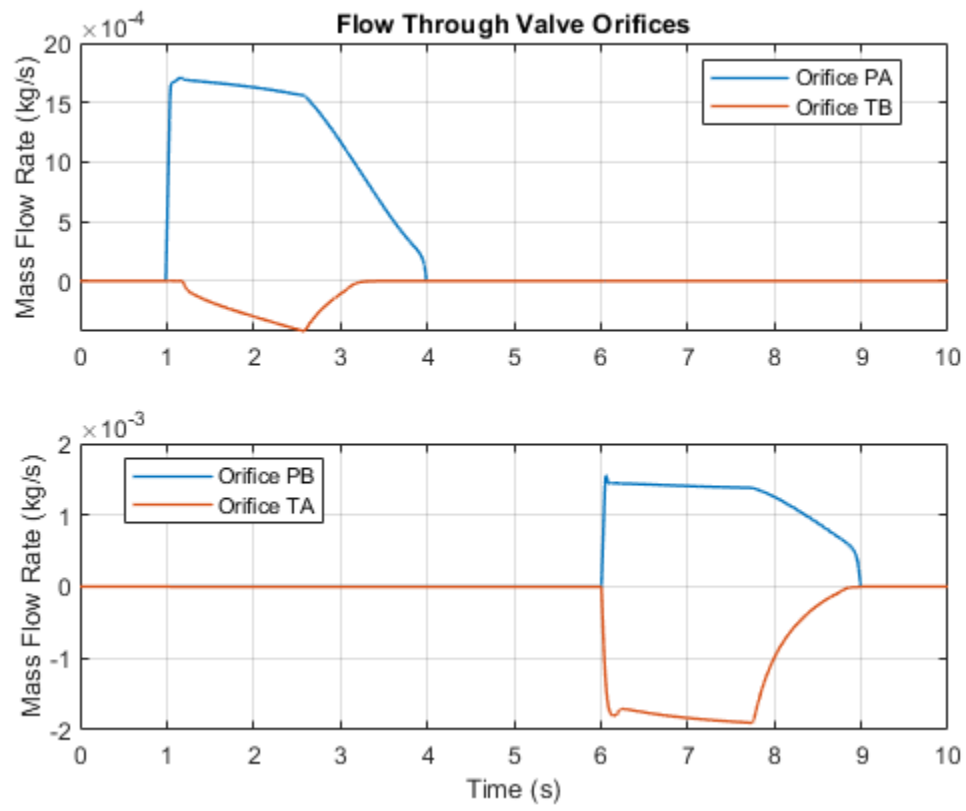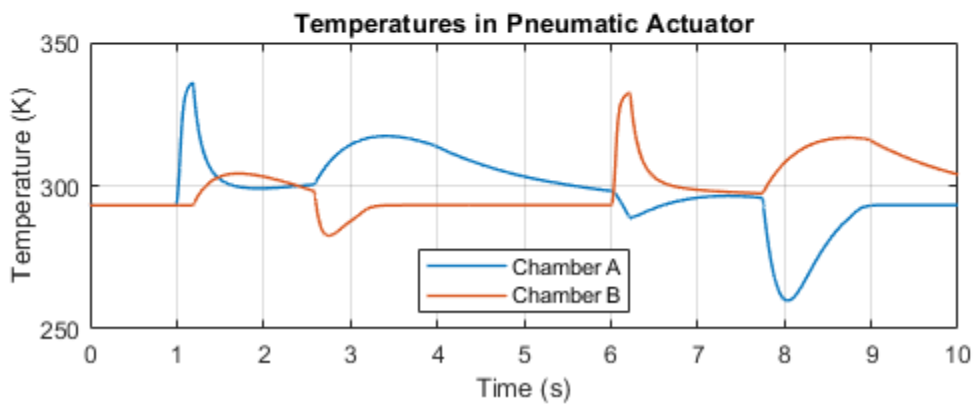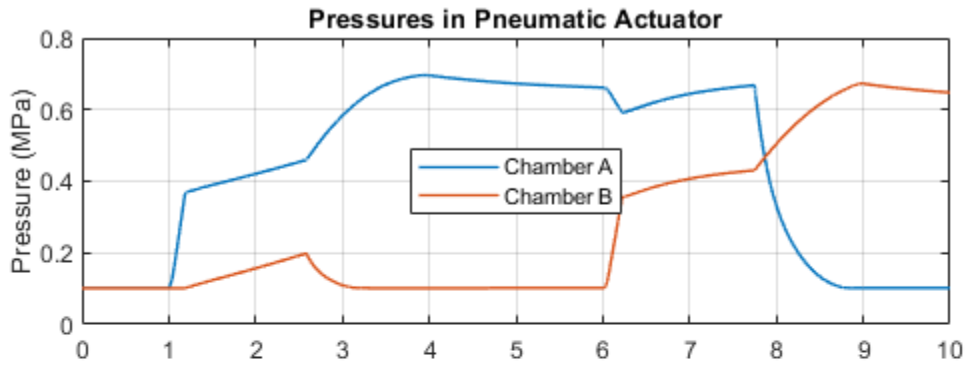


**Double-Acting Pneumatic Actuator Subsystem**

**Pipe A Subsystem**



**Simulation Results from Simscape Logging**
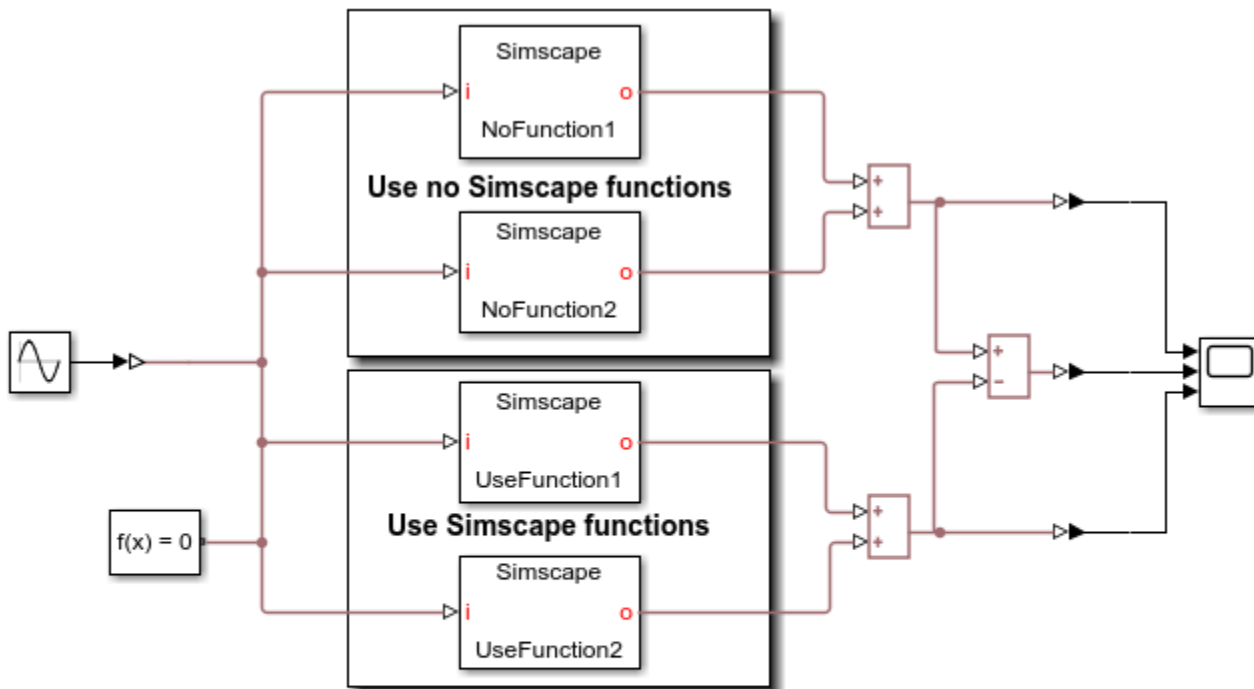
Flow Through Valve Orifices
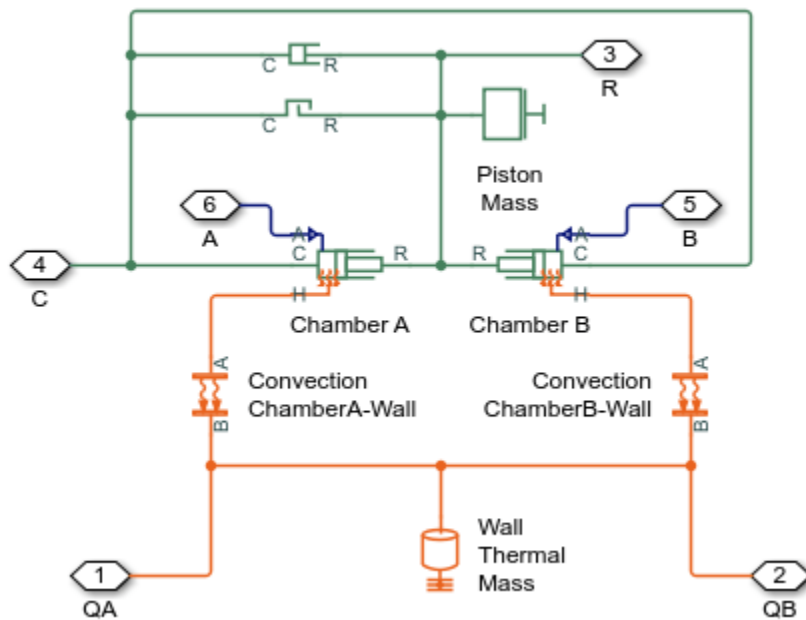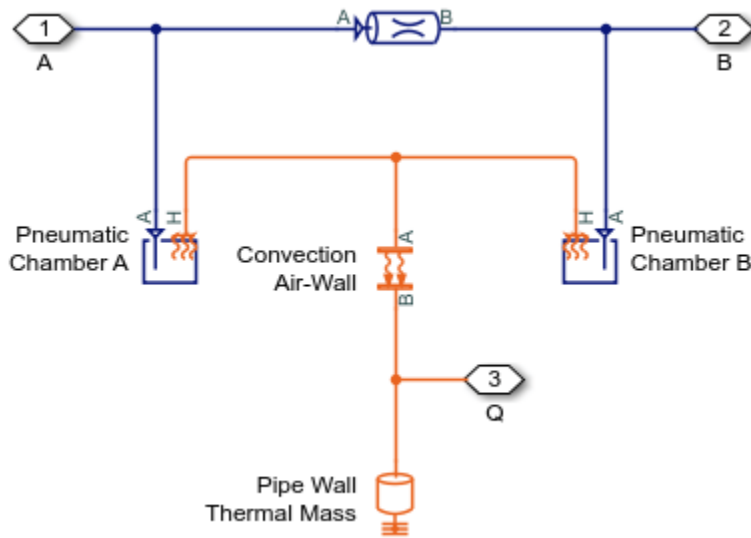
# Simscape Functions

This example shows how to write Simscape™ functions to compute numerical values with Simscape expressions and how to use Simscape functions to improve code reuse across components. The top two Simscape component blocks ( inside the "Use no Simscape functions" box ) are respectively created using two Simscape component files. Comparing these two component files, similar Simscape expressions can be observed on the right hand side of the equation to compute numerical values, which is essentially a modification of exp(i) to provide protection for large magnitude of i. Such expressions are common in standard diode modeling. Using Simscape functions, such expressions are abstracted out into a Simscape function file, and their usages inside the component files are replaced by calls to such Simscape functions. The bottom two Simscape component blocks ( inside the "Use Simscape functions" box ) are created using component files using Simscape functions.
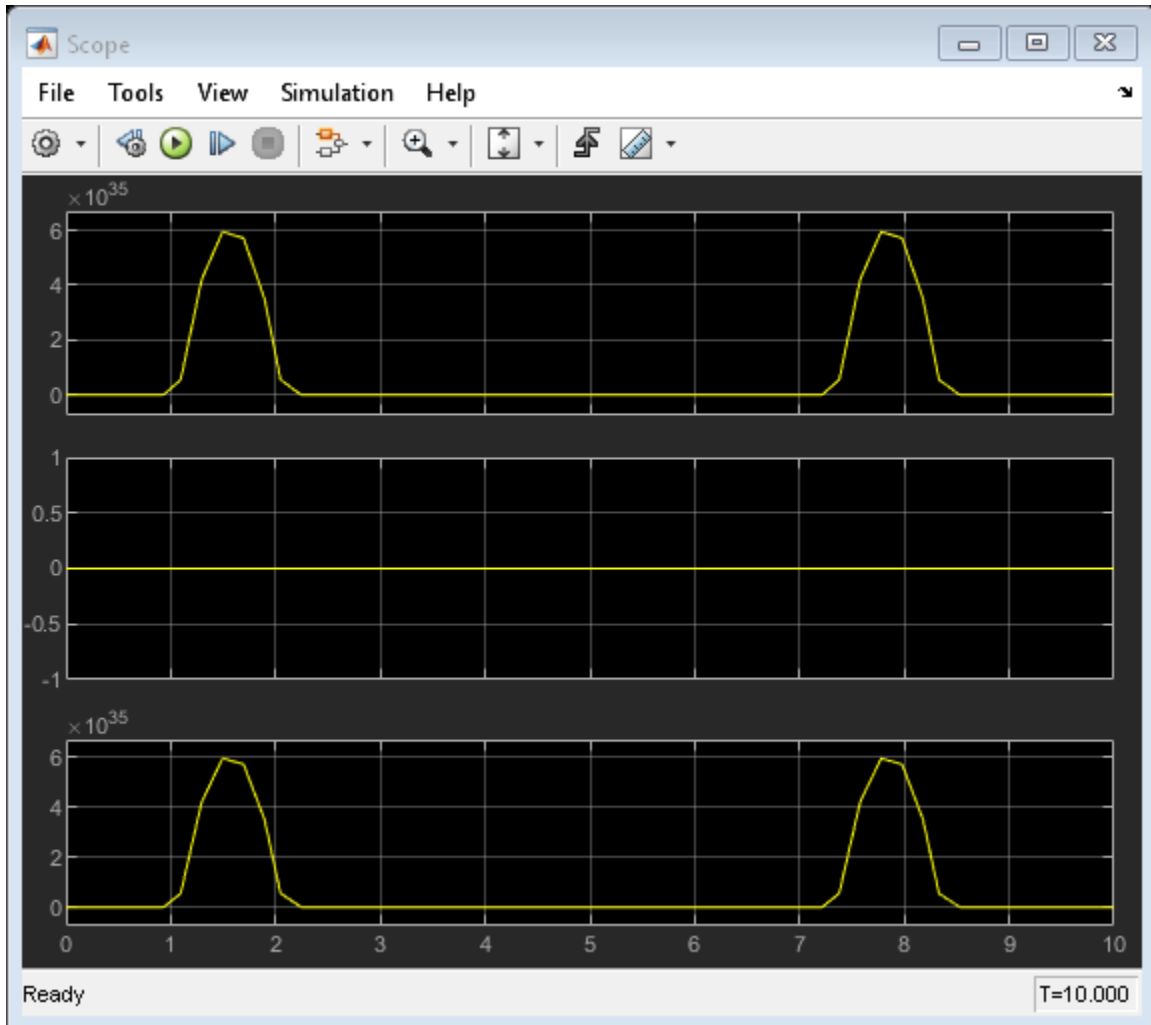
**Model**



## Simscape Functions

1. Write Simscape function to compute numerical values using Simscape expressions (see UserFunction)
2. Use Simscape functions to reuse code across components (see code when using Simscape function (UseFunction1 and UseFunction2) and code when not using Simscape function (NoFunction1 and NoFunction2)
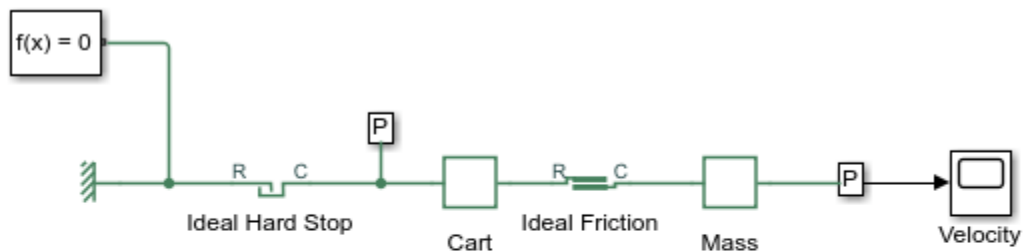3. Learn more about this example

Copyright 2017-2021 The MathWorks, Inc.

**Simulation Results from Scopes**

# Mass on Cart Using an Ideal Hard Stop

This example shows a cart bouncing between the two ends of an ideal hard stop, while a mass slides freely on top of it. The friction between the mass and cart is modeled using an ideal, modechart-based friction block, while the hard stop is modeled using instantaneous modes and entry actions. When the cart hits the bounds of the hard stop, the impulsive force is propagated to the mass above, causing it to be displaced as it transitions from static to dynamic friction modes.
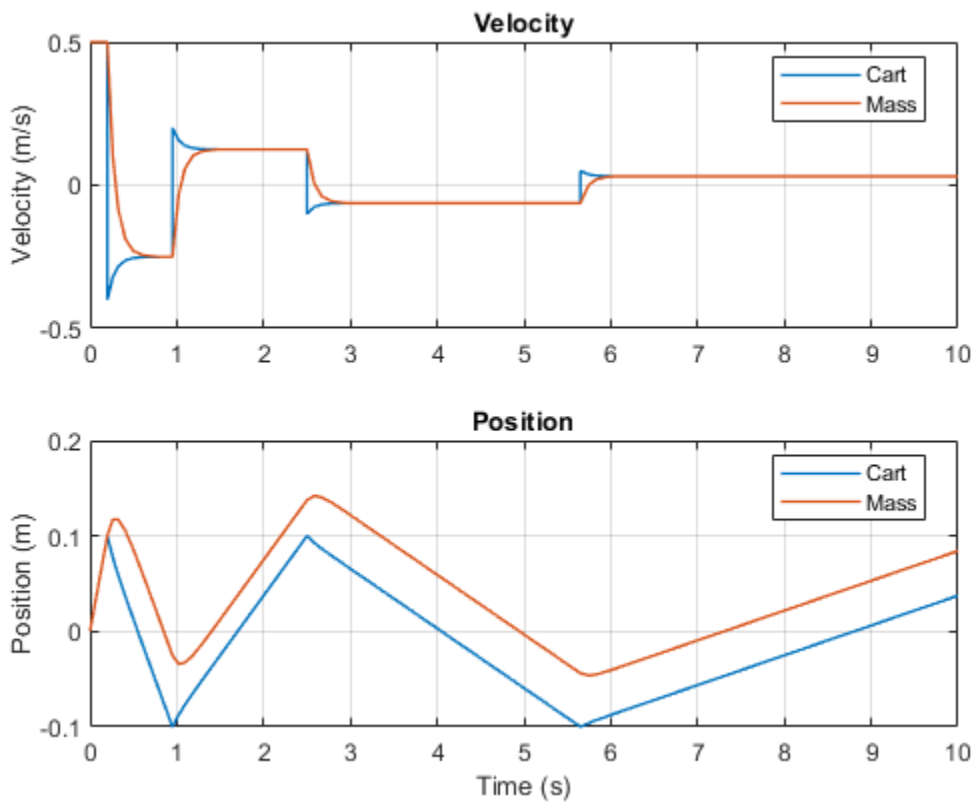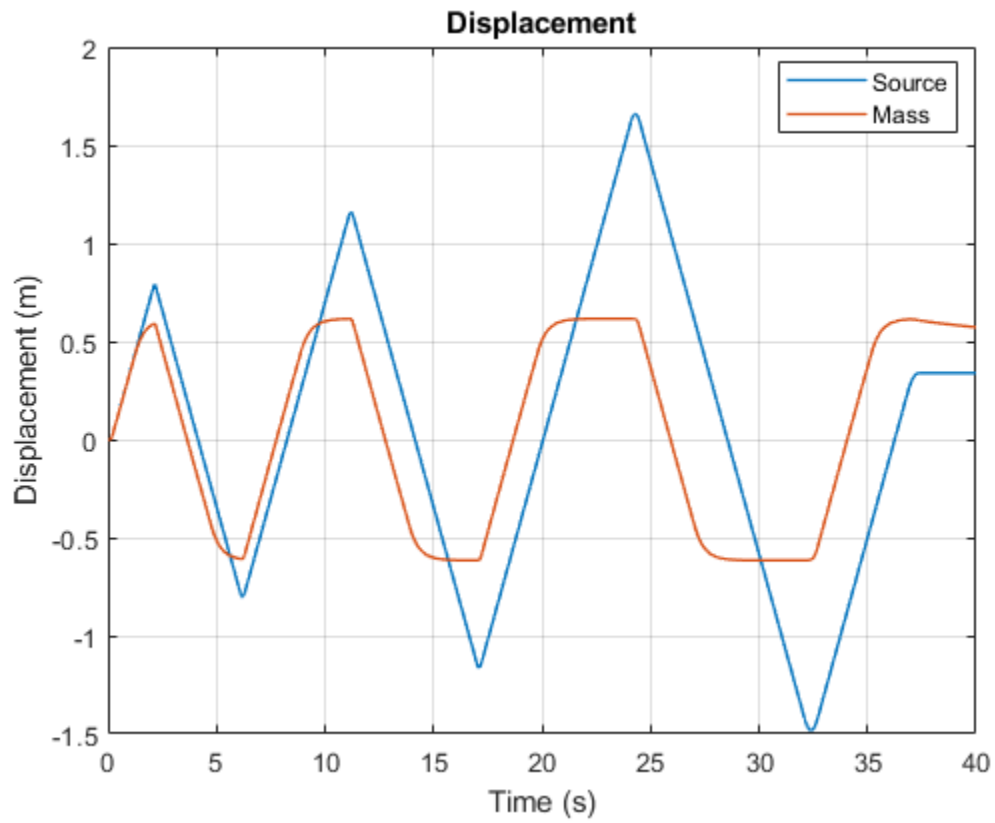
**Model**



**Mass on Cart Using an Ideal Hard Stop**

1. Plot velocities and positions of the two masses (see code)
2. Explore simulation results using sscexplore
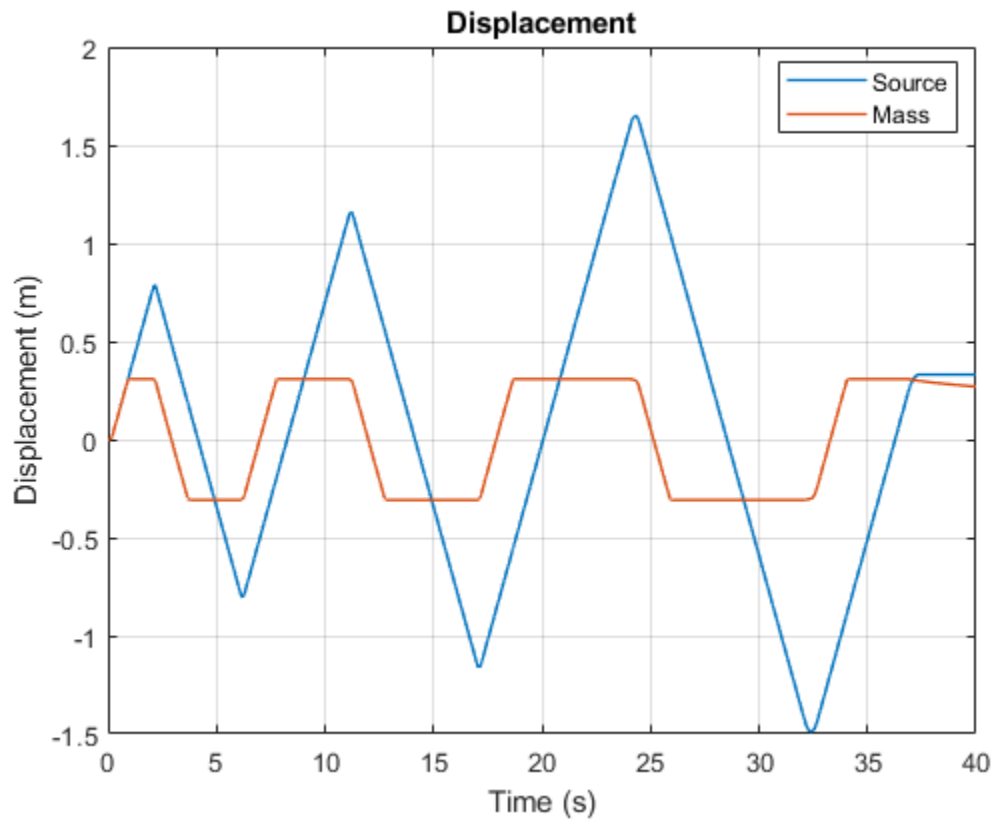3. Learn more about this example

Copyright 2020-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**

# Variant Leaf Region in Mechanical System

This example shows how to simulate the displacement of velocity sources and mass by using the leaf type of the Variant Connector block. The leaf Variant Connector block forms a leaf region that allows you to vary the displacement during simulation.

During simulation, Simulink® computes the variant conditions associated with each leaf Variant Connector block in the model. If the variant condition of a leaf Variant Connector block evaluates to `true`, all the physical components that are connected to the **Condition** port of the connector block become active. In this example, when A==1 evaluates to `true`, the components of Leaf Region_1 become active and the components of Leaf Region_2 remain inactive.

**Model**



### Variant Leaf Region in Mechanical System

1. Plot displacement of source and mass for different variant configuration (see code)

|  | Leaf Region 1 (A==1) | Leaf Region 2 (B==1) |
|---|---|---|
| Plot | True | True |
| Plot | True | False |
| Plot | False | True |
| Plot | False | False |

2. Learn more about this example

Copyright 2015-2021 The MathWorks, Inc.

**Simulation Results from Simscape Logging**

**Case 1: Leaf Region_1 Is Active and Leaf Region_2 Is Inactive**

**Case 2: Leaf Region_1 Is Inactive and Leaf Region_2 is Active**

# Variant Bounded Region in Electrical Circuit

This example shows how to activate or deactivate the flow of a current to a set of components in an electrical circuit by using the primary and nonprimary type Variant Connector blocks. The primary and nonprimary Variant Connector blocks form a bounded region that allows you to vary the electrical circuit during simulation.

During simulation, Simulink® computes the variant conditions associated with each bounded region in the model. If the variant condition of a bounded region evaluates to `true`, all the physical components that are located inside the region become active. In this example, when `A == 1` evaluates to `true`, the components of bounded region, BoundedRegion_1, become active and the components of BoundedRegion_2 remain inactive.
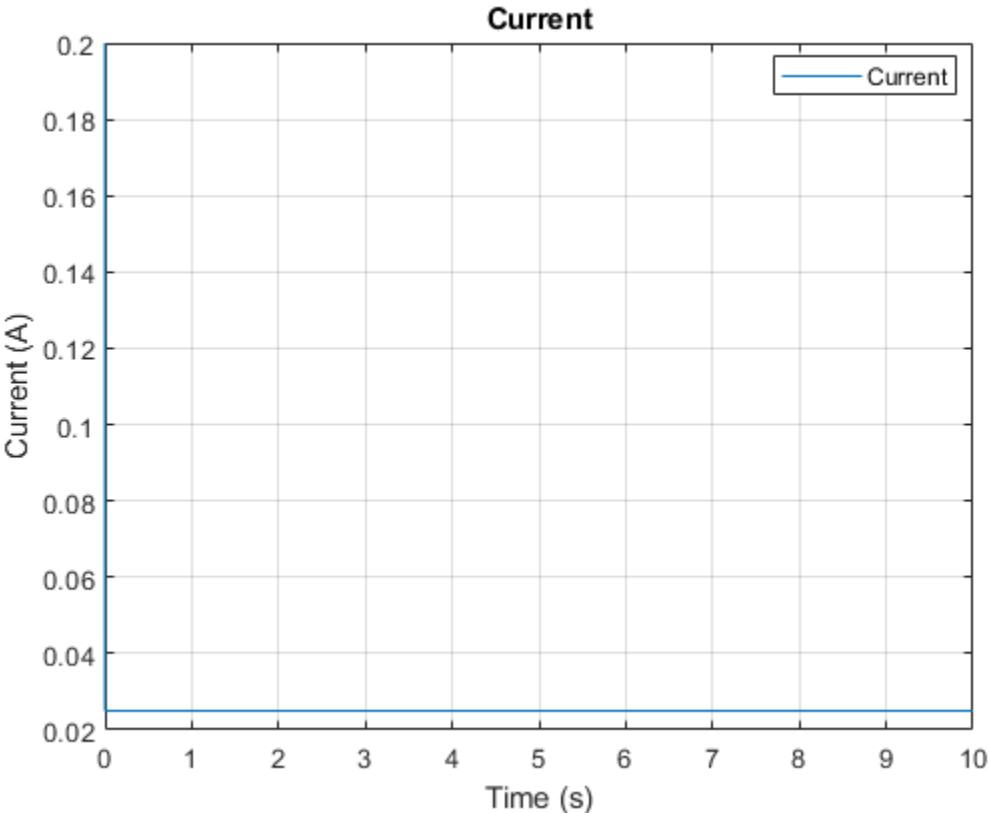
**Model**



**Simulation Results from Simscape Logging**

**Case 1: BoundedRegion_1 Is Active and BoundedRegion_2 Is Inactive**

**Case 2: BoundedRegion_1 Is Inactive and BoundedRegion_2 Is Active**

# Variant Connector Block with Simscape Bus Block

This example shows how to use Variant Connector blocks with the Simscape Bus block.

In this example, the mechanical and electrical network lines are bundled together using the Simscape Bus block. The components of the mechanical network are in the leaf region formed by the leaf Variant Connector block VC1. The components of the electrical network are in the leaf region formed by the leaf Variant Connector block VC2. During simulation, VC1 and VC2 are aware of the Simscape Bus hierarchy and its connectivity. This means that if the variant condition A==1 evaluates to `true`, only the components in the mechanical network become active. The components in the electrical network remain inactive.

**Model**
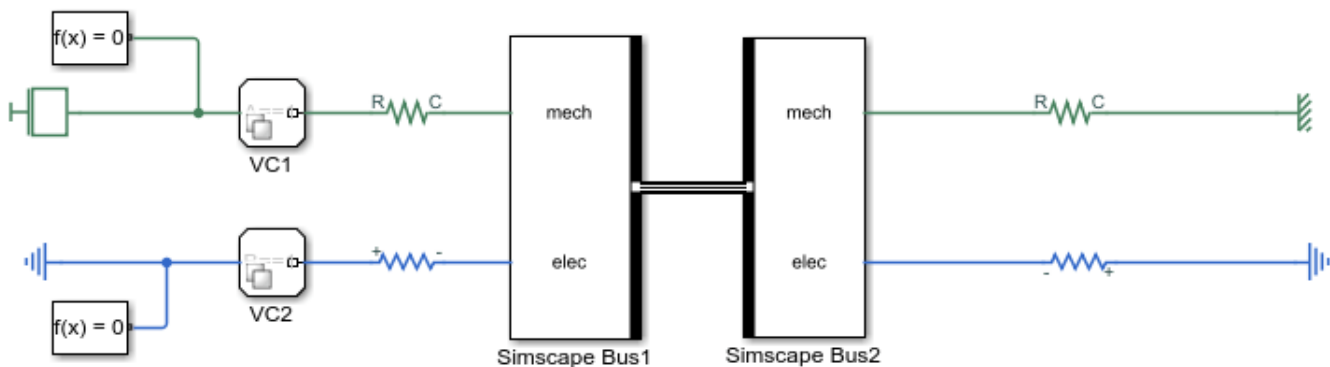


Variant Connector Block with Simscape Bus Block

1. Simulate the model with different variant configurations

| | Mechanical Network (A == 1) | Electrical Network (B == 1) |
|---|---|---|
| simulate | true | true |
| simulate | true | false |
| simulate | false | true |
| simulate | false | false |

2. Learn more about this example

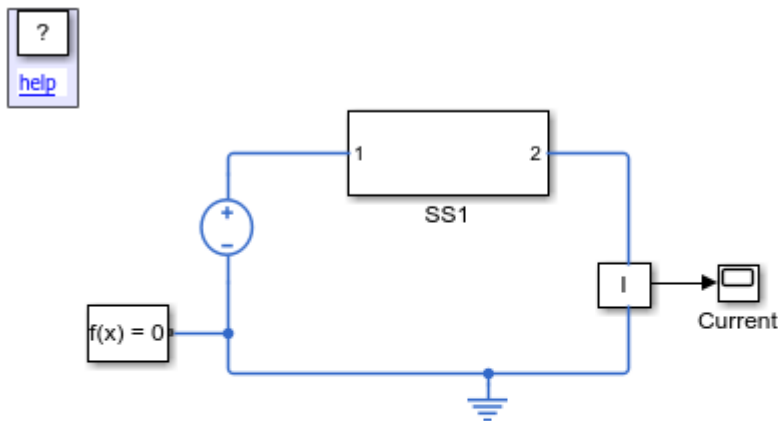Copyright 2019-2021 The MathWorks, Inc.

# Mask Workspace Variable in Variant Connector Block

This example shows how to use a mask workspace variable as a variant control variable in a Variant Connector block.

The mask workspace variables are the variables that you define in the mask workspace of the given block. These variables have a limited scope. They can be used only by the given block and the underlying layers of that block, so you can use same name for the variables in different masks. If you use a mask workspace variable as a variant control variable of a Variant Connector block, you can use the same variable name to set different active choices for multiple instances of the block in different masks. Using same name reduces the number of variables in the base workspace. When you select a value in the Block Parameters dialog box, the index of that value is mapped to the underlying mask workspace variable during simulation. The variable is then used to evaluate the variant control expression of the Variant Connector block. Depending on the variant control expression that evaluates to `true`, the blocks in the bounded region formed by the Variant Connector block remain active or inactive.

In this example, the mask workspace variable A is used as a variant control variable in the Variant Connector block. The scope of A is limited to the SS1 subsystem, so only SS1 and its underlying blocks can access A. If you select `Resistor` in the Block Parameters dialog box of SS1, its index is mapped during simulation to the underlying mask variable A, which then evaluates the variant control expression A==1 in the Variant Connector block to `true`. The Resistor block R1 becomes active.
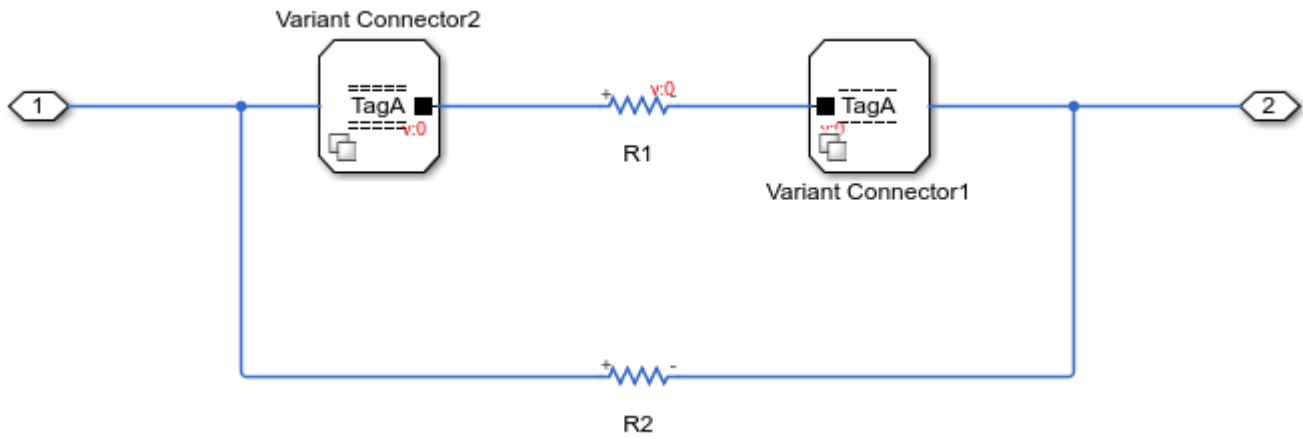
**Model**



**Mask Workspace Variable in Variant Connector Block**

1. Plot current with resistor (R1) (see code)
2. Plot current without resistor (R1) (see code)
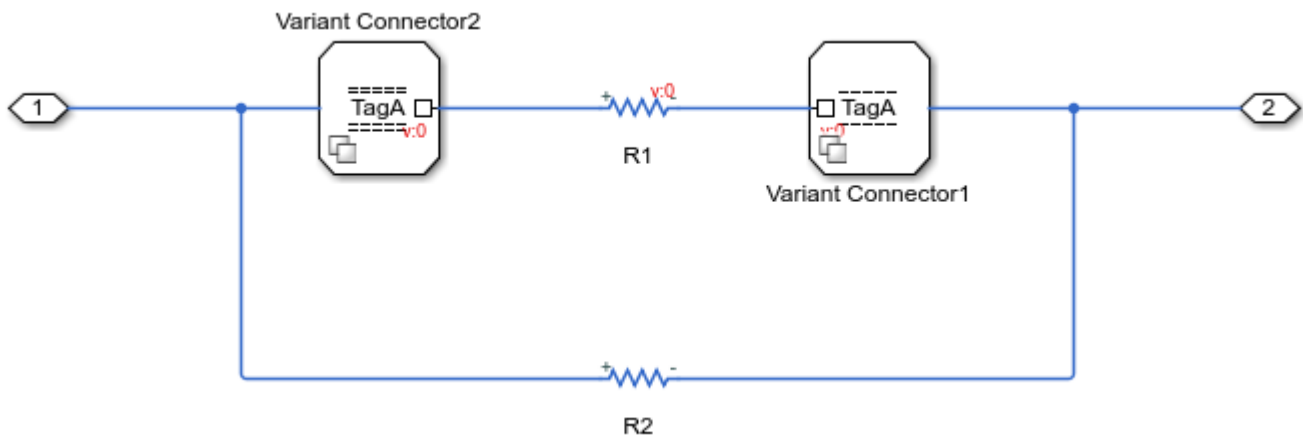3. Learn more about this example

Copyright 2020-2021 The MathWorks, Inc.

**Simulation Results**

**Case 1: If you select `Resistor` in the SS1 Block Parameters dialog box, A==1 evaluates to `true` and the R1 block becomes active**

**Case 2: If you select** `No resistor` **in the SS1 Block Parameters dialog box,** `A==1` **evaluates to** `false` **and the R1 block becomes inactive**

# Nonlinear Electromechanical Circuit with Partitioning Solver

This example models a DC Motor controlled by a ramp input. The resulting system of equations contains switched linear and nonlinear elements brought about by the Diode and Rotational Friction blocks respectively. However, the Partitioning solver is able to convert this system into several smaller sets of linear time-invariant and switched linear equations connected by nonlinear functions. This helps in reducing computational cost, which in turn yields faster simulation.
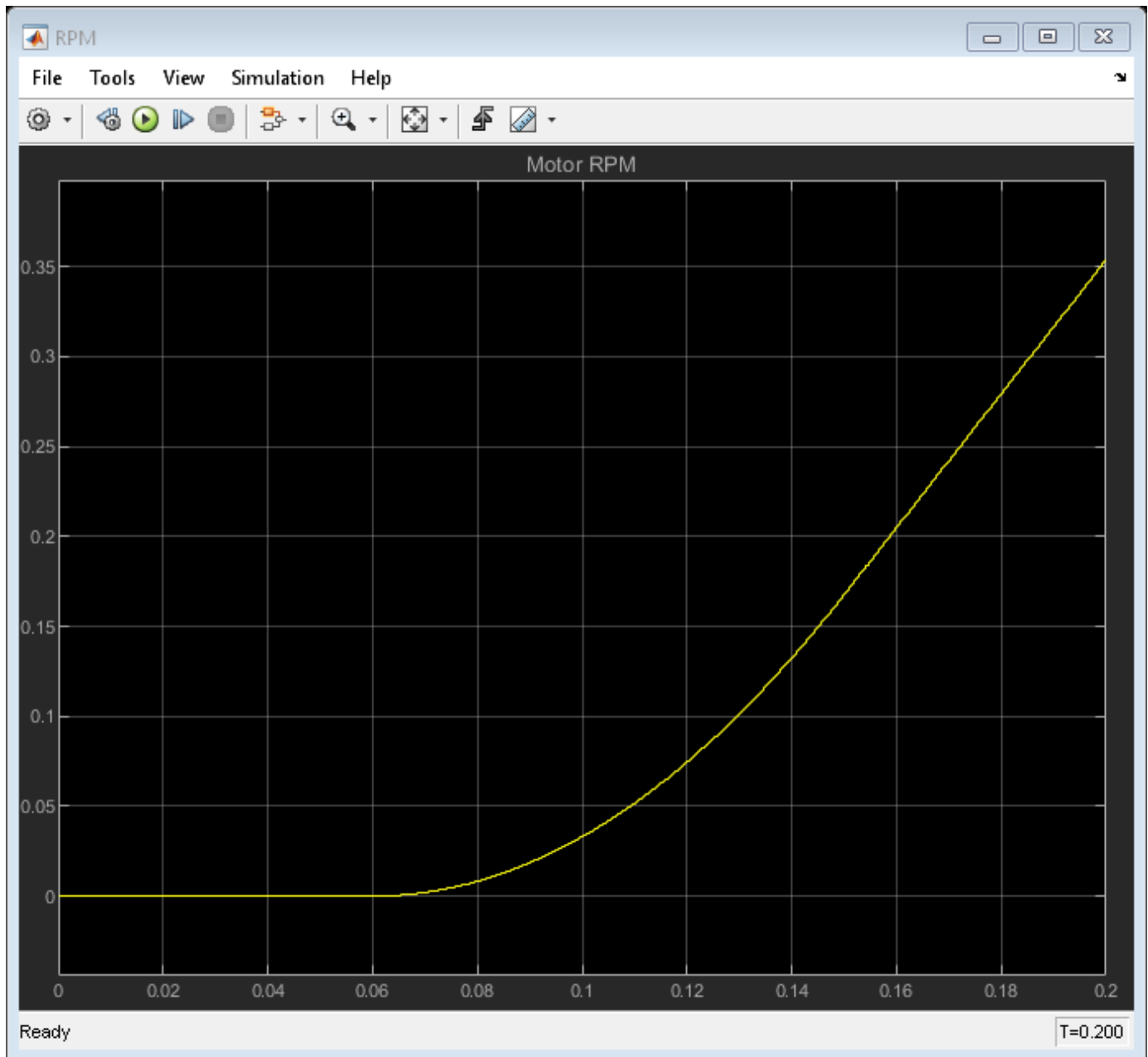
**Model**
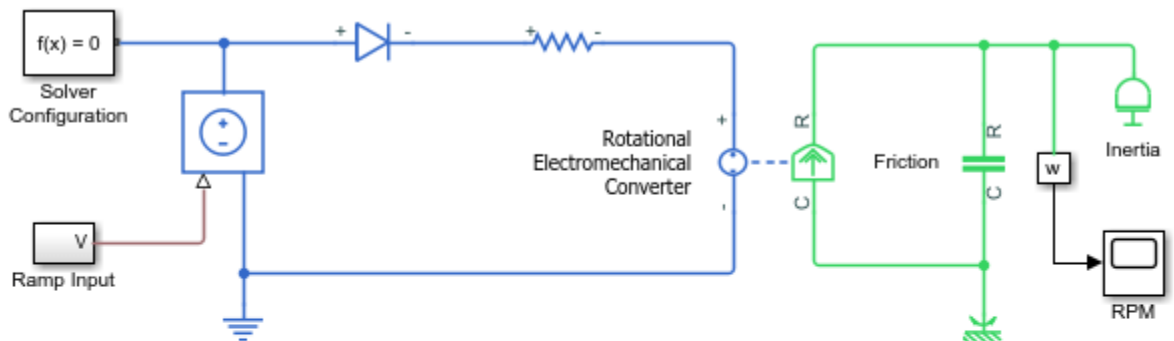


### Nonlinear Electromechanical Circuit with Partitioning Solver

1. Double click the Solver Configuration block to view solver settings
2. Open the Simscape Statistics Viewer to view the partitioned system
3. Learn more about this example

Copyright 2020-2021 The MathWorks, Inc.

**Simulation Results from Scopes**

# Configuring an EV Simulation for Multirate HIL

This example shows an electric vehicle model suitable for multirate Hardware-In-the-Loop (HIL) deployment. The example uses the Simscape™ Network Couplers Library to split the model into separate Simulink® subsystems that can be deployed at different sample rates. This allows you to run parts of the system (for example thermal components) with a slower sample time thereby reducing overall computational cost.

**Step 1: Design Model**

The top-level design model is shown below. The Vehicle Control and the Driver and Environment blocks are discrete-time Simulink® subsystems running at 100Hz. The remainder of the model can be run either fixed or variable step by changing the Solver Configuration block settings.

In the design model, three thermal masses have deliberately been implemented at the top-level so that their associated time constants can be used to separate out the thermal network without changing simulation behavior.
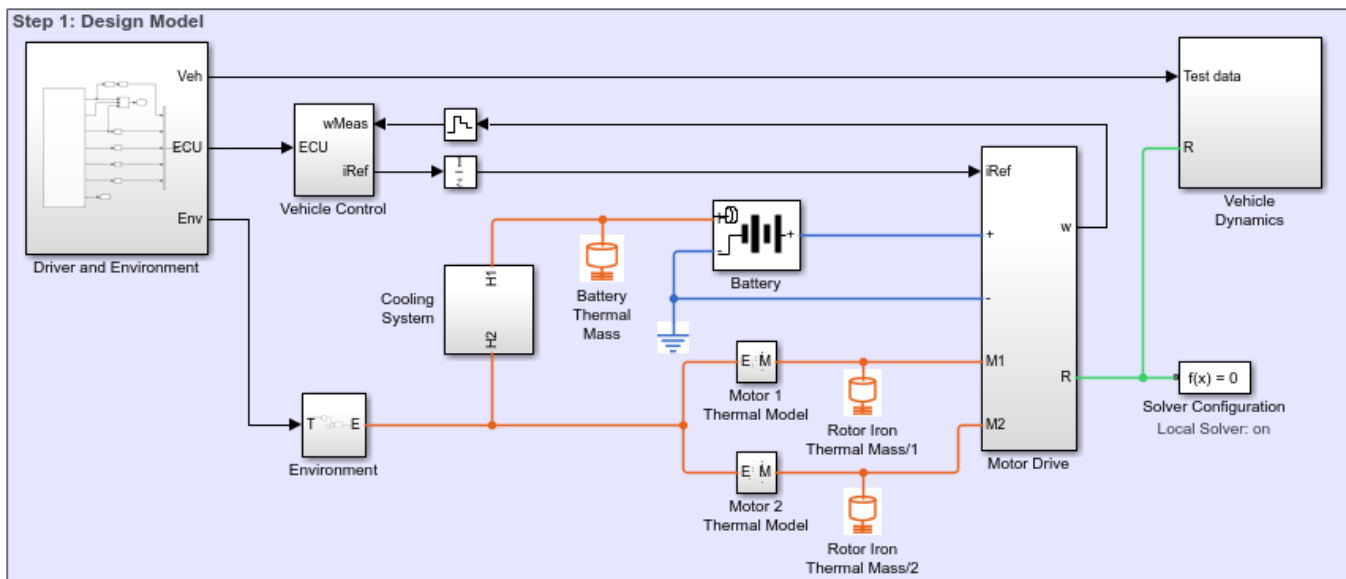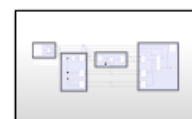


The subsystems labeled Step 1, Step 2, and Step 3 show successive steps to map from this top-level model to an implementation suitable for deployment to multirate HIL.

**Step 2: Multirate Model**

In this step the Simscape Network Coupler Library blocks are used to interactively experiment with good places to divide the model. From the diagram below it can be seen that the three thermal masses have been replaced with the Simscape Network Coupler Library thermal mass block. These new thermal mass blocks break the Simscape network thermal connection, replacing it with a

behaviorally equivalent Simulink connection. The thermal part of the model is now a separate Simscape network, and hence another solver block is added. Because the thermal time constants are longer, the solver block is configured such that the thermal network runs five times slower than other parts of the model.
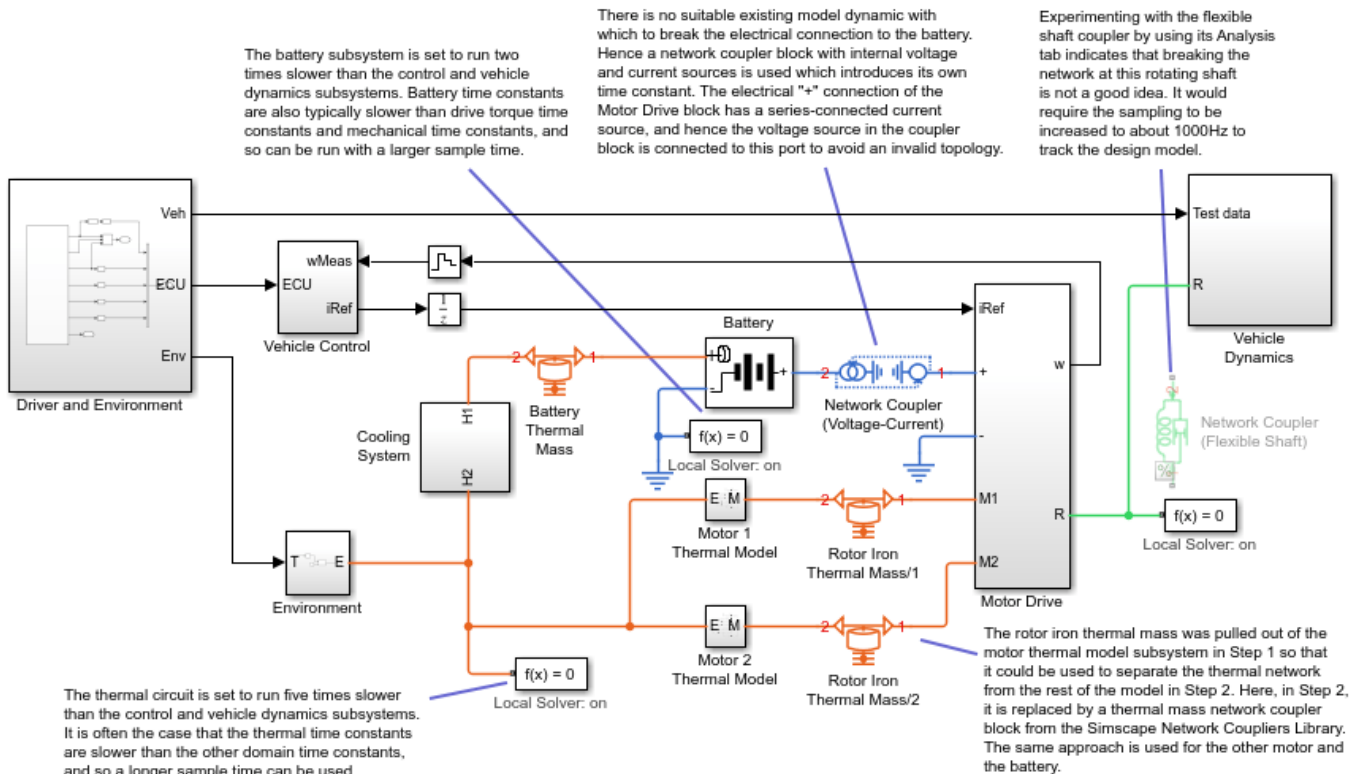


A Network Coupler (Voltage-Current) block is used to enable simulation of the battery as a separate Simscape network. The battery also does not need to be sampled at the highest rate, here being sampled at half the fastest rate. Because there is no suitable dynamic with which to break the battery connection to the Motor Drive subsystem, a numerical time constant is introduced by the Network Coupler (Voltage-Current) block. This time constant is automatically calculated based on the port sample times you specify for the block. Sample times can be iterated on until a suitable compromise between accuracy and simulation overhead is achieved.

The commented-out Network Coupler (Flexible Shaft) shows that some experimentation was done with separating out the vehicle dynamics from the motor drive dynamics. This coupler block breaks the networks with a rotational compliance, and automatically calculates a suitable stiffness based on the inertias and sample times of the two connected networks. With this it is easy to discover that sampling at about 1000Hz at the motor drive connection would be needed for reasonable accuracy, ruling this out as a place to split the model.

### Step 3: Ungroup and Redraw

In this step all of the Simscape Network Coupler Library blocks are first unmasked and then their contents moved up a level using the Subsystem & Model Reference -> Expand Subsystem option from the right-click context menu. This makes the Simulink connections visible at the top level, along with the rate transition blocks. The blocks and subsystems can then be grouped together by sampling time. Here Simulink areas have been used to improve readability of the model.
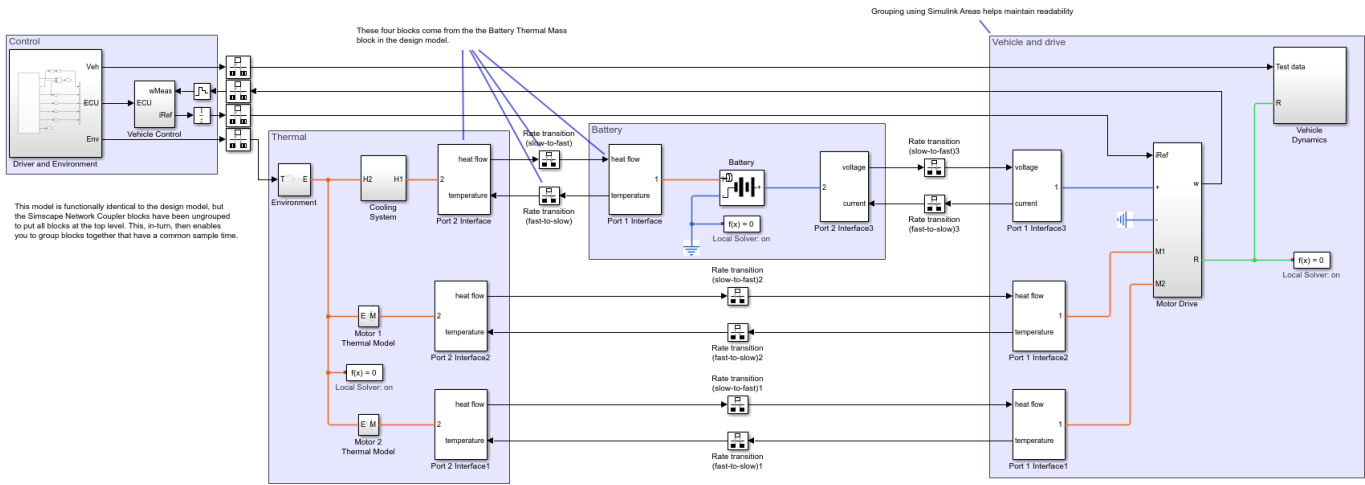
## Step 4: Deployment Model

In this final step each group of blocks and subsystems with a common sampling time is moved into its own subsystem. Sample time allocations can be checked by turning on sample time colors. The model is now ready for deployment to a multirate HIL platform, just in the same way as for any other Simulink model.
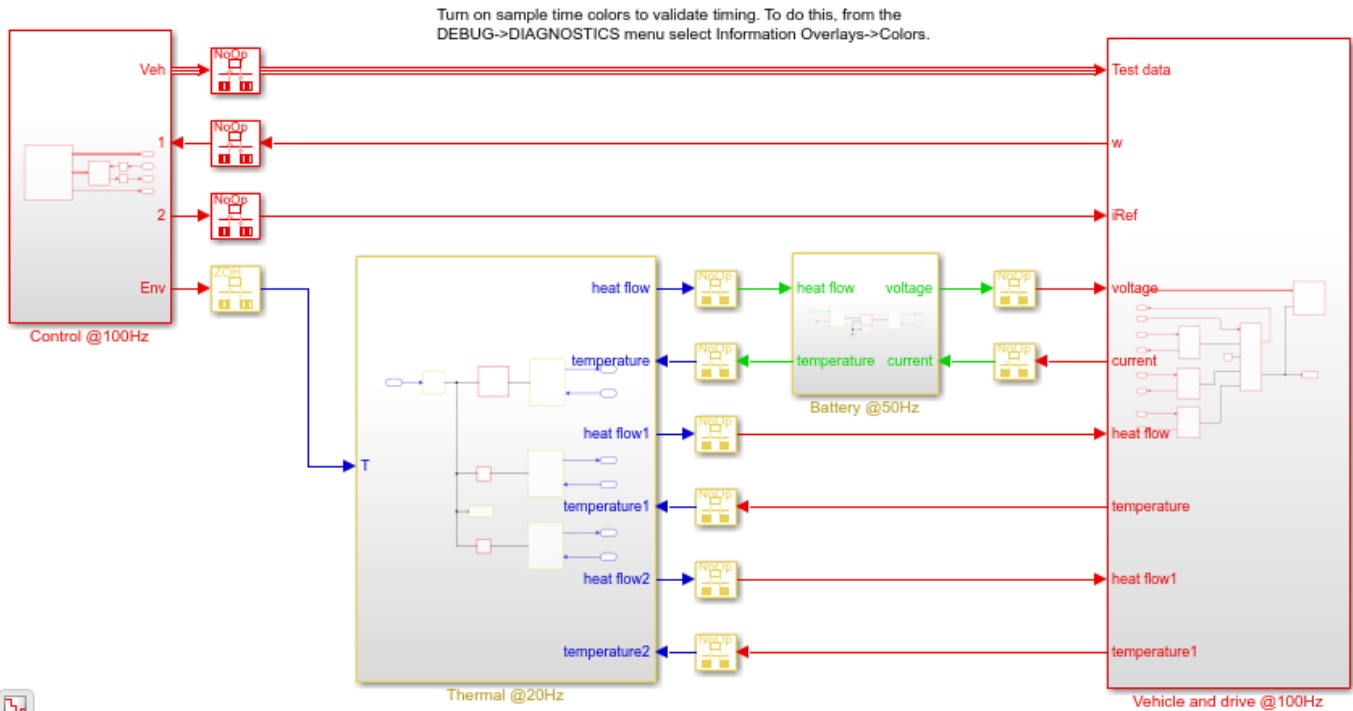


## Simulation Results from Simscape Logging

The test run shows the vehicle accelerating to a steady speed up an incline followed by a period of descent during which electrical power is returned to the battery. It is used to show that the mapping from design model to deployment model does not introduce any significant loss of accuracy.