

Simscape™
User's Guide



MATLAB® & SIMULINK®

R2015b



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.
3 Apple Hill Drive
Natick, MA 01760-2098

Simscape™ User's Guide

© COPYRIGHT 2007–2015 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

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.

Revision History

March 2007	Online only	New for Version 1.0 (Release 2007a)
September 2007	Online only	Revised for Version 2.0 (Release 2007b)
March 2008	Online only	Revised for Version 2.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.0 (Release 2008b)
March 2009	Online only	Revised for Version 3.1 (Release 2009a)
September 2009	Online only	Revised for Version 3.2 (Release 2009b)
March 2010	Online only	Revised for Version 3.3 (Release 2010a)
September 2010	Online only	Revised for Version 3.4 (Release 2010b)
April 2011	Online only	Revised for Version 3.5 (Release 2011a)
September 2011	Online only	Revised for Version 3.6 (Release 2011b)
March 2012	Online only	Revised for Version 3.7 (Release 2012a)
September 2012	Online only	Revised for Version 3.8 (Release 2012b)
March 2013	Online only	Revised for Version 3.9 (Release 2013a)
September 2013	Online only	Revised for Version 3.10 (Release 2013b)
March 2014	Online only	Revised for Version 3.11 (Release 2014a)
October 2014	Online only	Revised for Version 3.12 (Release 2014b)
March 2015	Online only	Revised for Version 3.13 (Release 2015a)
September 2015	Online only	Revised for Version 3.14 (Release 2015b)

Model Construction

Basic Principles of Modeling Physical Networks	1-2
Overview of the Physical Network Approach to Modeling	
Physical Systems	1-2
Variable Types	1-4
Building the Mathematical Model	1-5
Direction of Variables	1-6
Connector Ports and Connection Lines	1-8
Connecting Simscape Diagrams to Simulink Sources and Scopes	1-9
Simscape Block Libraries	1-11
Library Structure Overview	1-11
Using the Simulink Library Browser to Access the Block Libraries	1-12
Using the Command Prompt to Access the Block Libraries .	1-13
Essential Physical Modeling Techniques	1-15
Building Your Model	1-15
Using the Conserving Ports	1-16
Using the Physical Signal Ports	1-16
Creating and Simulating a Simple Model	1-18
Building a Simscape Diagram	1-18
Modifying Initial Settings	1-26
Running the Simulation	1-28
Adjusting the Parameters	1-30
Modeling Best Practices	1-36
Grounding Rules	1-36
Avoiding Numerical Simulation Issues	1-40
Domain-Specific Line Styles	1-43

Modeling Pneumatic Systems	1-46
Intended Applications	1-46
Assumptions and Limitations	1-46
Fundamental Equations	1-47
Network Variables	1-48
Connection Constraints	1-48
References	1-49

Thermal Liquid Models

2

Modeling Thermal Liquid Systems	2-2
When to Use Thermal Liquid Blocks	2-2
Modeling Workflow	2-3
Establish Model Requirements	2-3
Model Physical Components	2-4
Prepare Model for Analysis	2-5
Run Simulation	2-5
 Thermal Liquid Library	 2-6
Why Use Thermal Liquid Blocks?	2-6
Representing Thermal Liquid Components	2-6
Specifying Thermal Liquid Medium	2-8
Modeling Multidomain Systems	2-8
 Thermal Liquid Modeling Framework	 2-10
How Blocks Represent Components	2-10
How Ports Represent Interfaces	2-11
Full Flux Scheme	2-12
 Heat Transfer in Insulated Oil Pipeline	 2-14
Oil Pipelines	2-14
Modeling Considerations	2-15
Simscape Model	2-17
Run Simulation	2-18
Run Optimization Script	2-25

3

Manually Generate Fluid Property Tables	3-2
Fluid Property Tables	3-2
Steps for Generating Property Tables	3-3
Before Generating Property Tables	3-3
Create Fluid Property Functions	3-3
Set Property Table Criteria	3-4
Create Pressure-Normalized Internal Energy Grids	3-5
Map Grids Onto Pressure-Specific Internal Energy Space ...	3-5
Obtain Fluid Properties at Grid Points	3-6
Visualize Grids	3-7

Model Simulation

4

How Simscape Models Represent Physical Systems	4-2
Representations of Physical Systems	4-2
Differential, Differential-Algebraic, and Algebraic Systems ..	4-2
Stiffness	4-3
Events and Zero Crossings	4-3
Working with Simscape Representation	4-3
How Simscape Simulation Works	4-5
Simscape Simulation Phases	4-5
Model Validation	4-7
Network Construction	4-7
Equation Construction	4-8
Initial Conditions Computation	4-8
Transient Initialization	4-9
Transient Solve	4-10
Setting Up Solvers for Physical Models	4-11
About Simulink and Simscape Solvers	4-11
Choosing Simulink and Simscape Solvers	4-11
Harmonizing Simulink and Simscape Solvers	4-13

Important Concepts and Choices in Physical Simulation . .	4-17
Variable-Step and Fixed-Step Solvers	4-17
Explicit and Implicit Solvers	4-18
Full and Sparse Linear Algebra	4-18
Event Detection and Location	4-18
Unbounded, Bounded, and Fixed-Cost Simulation	4-19
Global and Local Solvers	4-19
Making Optimal Solver Choices for Physical Simulation . .	4-21
Simulating with Variable Time Step	4-21
Simulating with Fixed Time Step — Local and Global Fixed- Step Solvers	4-21
Simulating with Fixed Cost	4-22
Troubleshooting and Improving Solver Performance	4-23
Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System	4-24
Troubleshooting Simulation Errors	4-26
Troubleshooting Tips and Techniques	4-26
System Configuration Errors	4-27
Numerical Simulation Issues	4-29
Initial Conditions Solve Failure	4-30
Transient Simulation Issues	4-30
Limitations	4-32
Sample Time and Solver Restrictions	4-32
Algebraic Loops	4-32
Restricted Simulink Tools	4-33
Unsupported Simulink Tools and Features	4-35
Simulink Tools Not Compatible with Simscape Blocks	4-35
Code Generation	4-35
References	4-37

Variable Initialization and State Viewer

5

About Variable Initialization	5-2
Initializing Block Variables for Model Simulation	5-2
Variable Initialization Priority	5-3

Suggested Workflow	5-4
Set Priority and Initial Target for Block Variables	5-5
Initialize Variables for a Mass-Spring-Damper System	5-7
Variable Viewer	5-23
About Variable Viewer	5-23
Advanced Configuration	5-25
Switching Between Tree View and Flat View	5-27
Useful Filtering Techniques	5-29
Link to Block Diagram	5-30
Interaction with Model Updates and Simulation	5-31

Linearization and Trimming

6

Finding an Operating Point	6-2
What Is an Operating Point?	6-2
Finding Operating Points in Physical Models	6-3
Linearizing at an Operating Point	6-7
What Is Linearization?	6-7
Linearizing a Physical Model	6-9
Linearize an Electronic Circuit	6-13
Explore the Model	6-13
Linearize with Steady-State Solver and linmod Function ...	6-17
Linearize with Simulink Control Design Software	6-19
Use Control System Toolbox Software for Bode Analysis ...	6-20
Linearize a Plant Model for Use in Feedback Control	
Design	6-23
Explore the Model	6-23
Trim Using the Controller and Linearize with Simulink linmod	
Function	6-26
Linearize with Simulink Control Design Software	6-27

Model Preparation Objectives	7-2
Obtain Reference Results	7-2
Determine Step Size	7-2
Adjust Model Fidelity or Scope	7-2
Real-Time Model Preparation Workflow	7-5
Prepare Your Model for Real-Time Simulation	7-6
Insufficient Computational Capability for Workflow Completion	7-8
Improving Speed and Accuracy	7-10
Why Speed and Accuracy Matter for Real-Time Simulation ..	7-10
Balancing Speed and Accuracy	7-11
Eliminating Effects That Require Intensive Computation ..	7-12
Optimizing Local and Global Solver Configurations	7-13
Upgrading Target Hardware	7-13
Simulating Parts of the System in Parallel	7-13
Determine Step Size	7-15
Reduce Computation Costs	7-25
Data Logging and Monitoring Guidelines	7-25
Improve Data Logging and Monitoring Efficiency	7-25
Additional Methods for Reducing Computational Cost	7-29
Reduce Numerical Stiffness	7-31
Why Reduce Stiffness?	7-31
Review Reference Results	7-31
Identify and Modify a Stiff Element	7-34
Analyze Results	7-36
Reduce Zero Crossings	7-41
Why Reduce Zero Crossings?	7-41
Obtain Reference Results and Plot Simulation Step Size ...	7-41
Identify and Modify Elements That Cause Zero Crossings ..	7-45
Analyze the Results of the Modified Model	7-50
Fixed-Cost Simulation for Real-Time Viability	7-55

Real-Time Simulation Workflow	7-57
Make Your Model Real-Time Viable	7-60
Insufficient Computational Capability for Real-Time Viability	7-62
Solvers for Real-Time Simulation	7-63
Choosing Between Discrete and Continuous Solvers	7-64
Computational Cost for Continuous Solvers	7-64
How Numerical Stiffness Affects Solver Choice	7-65
Using Simscape Local Fixed-Step Solvers	7-66
Determine System Stiffness	7-68
Obtain Reference Results	7-68
Simulate with an Implicit Fixed-Step Solver	7-69
Simulate with an Explicit Fixed-Step Solver	7-71
Analyze the Results	7-73
Estimate Computation Costs	7-76
Choose Step Size and Number of Iterations	7-79
Obtain Reference Results	7-80
Determine Maximum Step Size for Accurate Results	7-82
Parameterize Global and Local Solver Settings	7-84
Perform Fixed-Step, Fixed-Cost Simulation	7-85
Adjust Solver Settings to Improve Accuracy	7-89
What Is Hardware-in-the-Loop Simulation?	7-96
Why Perform Hardware-in-the-Loop Simulation?	7-97
Hardware-in-the-Loop Simulation Workflow	7-100
Perform Hardware-in-the-Loop Simulation	7-104
Insufficient Computational Capability for Hardware-in-the- Loop Simulation	7-105
Code Generation Requirements	7-106
Hardware Requirements	7-106
Software Requirements	7-107
Generate, Download, and Execute Code	7-108
Requirements for Building and Executing Simulink Real-Time Applications	7-108
Create, Build, and Download a Real-Time Application	7-108
Execute Real-Time Application	7-109

Requirements for Using Alternative Platforms	7-111
Hardware Requirements	7-111
Software Requirements	7-111

Code Generation

8

About Code Generation from Simscape Models	8-2
Reasons for Generating Code	8-3
Using Code-Related Products and Features	8-4
How Simscape Code Generation Differs from Simulink	8-5
Simscape and Simulink Code Generated Separately	8-5
Compiler and Processor Architecture Requirements	8-5
Precompiled Libraries Provided for Selected Compilers	8-5
Simscape Code Reuse Not Supported	8-6
Tunable Parameters Not Supported	8-6
Simscape Run-Time Parameter Inlining Override of Global Exceptions	8-6

Data Logging

9

About Simulation Data Logging	9-2
Suggested Workflows	9-2
Limitations	9-3
Enable Data Logging for the Whole Model	9-4
Log Data for Selected Blocks Only	9-5
Data Logging Options	9-6
Log and Plot Simulation Data	9-8

Log Simulation Statistics	9-13
Log and View Simulation Data for Selected Blocks	9-17
Log, Navigate, and Plot Simulation Data	9-21
About the Simscape Results Explorer	9-26
Use Custom Units to Plot Simulation Data	9-27
View Sparkline Plots of Simulation Data	9-31

Model Statistics

10

Simscape Model Statistics	10-2
1-D Physical System Statistics	10-4
3-D Multibody System Statistics	10-7
1-D/3-D Interface Statistics	10-10
View Model Statistics	10-11
Access Block Variables Using Statistics Viewer	10-16

Physical Units

11

How to Work with Physical Units	11-2
Unit Definitions	11-4
How to Specify Units in Block Dialogs	11-9

Thermal Unit Conversions	11-11
About Affine Units	11-11
When to Apply Affine Conversion	11-11
How to Apply Affine Conversion	11-12
Angular Units	11-15
References	11-15
Units for Angular Velocity and Frequency	11-16

Add-On Product License Management

12

About the Simscape Editing Mode	12-2
Suggested Workflows	12-2
What You Can Do in Restricted Mode	12-3
What You Can Do in Full Mode	12-4
Switching Between Modes	12-4
Working with Block Libraries	12-7
Set the Model Loading Preference	12-9
Save a Model in Restricted Mode	12-11
Example of Saving a Model in Restricted Mode	12-12
Work with a Model in Restricted Mode	12-14
How to Simulate and Fine-Tune a Model in Restricted Mode	12-14
How to Add and Delete Simulink Blocks in Restricted Mode	12-19
Performing an Operation Disallowed in Restricted Mode ..	12-24
Switch from Restricted to Full Mode	12-28
Editing Mode Information	12-30
What Is the Current Mode?	12-30
Which Licenses Are Checked Out?	12-30

Model Construction

- “Basic Principles of Modeling Physical Networks” on page 1-2
- “Simscape Block Libraries” on page 1-11
- “Essential Physical Modeling Techniques” on page 1-15
- “Creating and Simulating a Simple Model” on page 1-18
- “Modeling Best Practices” on page 1-36
- “Domain-Specific Line Styles” on page 1-43
- “Modeling Pneumatic Systems” on page 1-46

Basic Principles of Modeling Physical Networks

In this section...

“Overview of the Physical Network Approach to Modeling Physical Systems” on page 1-2

“Variable Types” on page 1-4

“Building the Mathematical Model” on page 1-5

“Direction of Variables ” on page 1-6

“Connector Ports and Connection Lines” on page 1-8

“Connecting Simscape Diagrams to Simulink Sources and Scopes” on page 1-9

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.

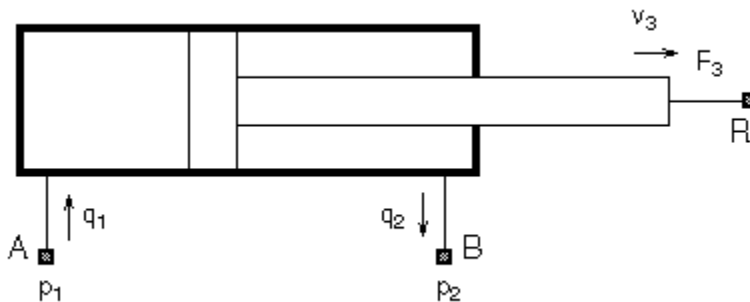
Simulink blocks represent basic mathematical operations. When you connect Simulink blocks together, the resulting diagram is equivalent to the mathematical model, or representation, of the system under design. Simscape technology lets you create a network representation of the system under design, based 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 1-4 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 1-8 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:

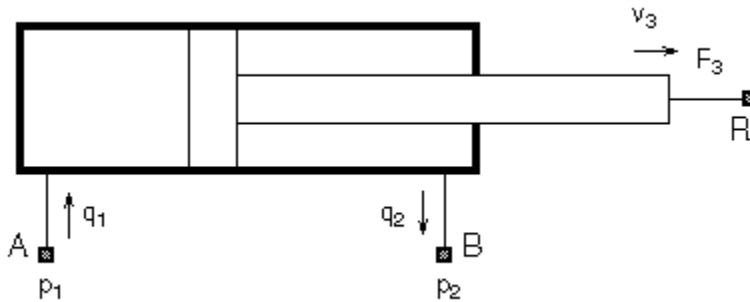
Physical Domain	Across Variable	Through Variable
Electrical	Voltage	Current
Hydraulic	Pressure	Flow rate
Magnetic	Magnetomotive force (mmf)	Flux
Mechanical rotational	Angular velocity	Torque
Mechanical translational	Translational velocity	Force
Pneumatic	Pressure and temperature	Mass flow rate and heat flow
Thermal	Temperature	Heat flow
Thermal liquid	Pressure and temperature	Mass flow rate and thermal flux
Two-phase fluid	Pressure and specific internal energy	Mass flow rate and heat flow rate

Note Generally, the product of each pair of Across and Through variables associated with a domain is power (energy flow in watts). The exceptions are pneumatic domain (where the product of pressure and mass flow rate is not power), magnetic domain (where the product of mmf and flux is not power, but energy), and the two thermodynamic domains,

thermal liquid and two-phase fluid (where products of both variable pairs are not power). These result in a pseudo-bond graph.

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

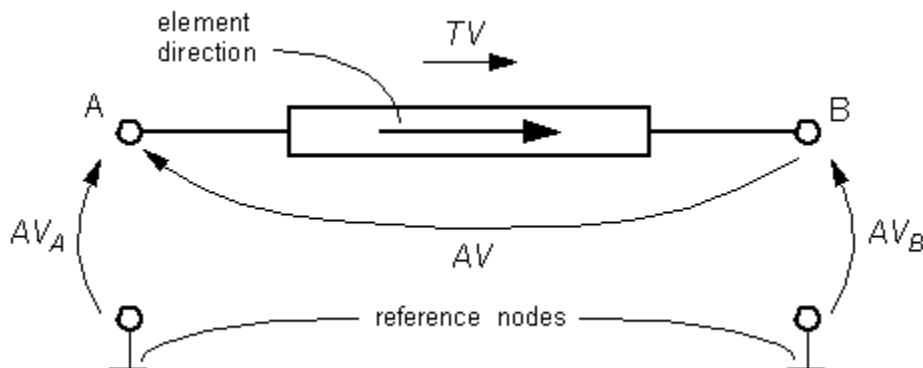
q_1, q_2	Flow rates through ports A and B, respectively (Through variables)
p_1, p_2	Gauge pressures at ports A and B, respectively (Across variables)
A_1, A_2	Piston effective areas
F_3	Rod force (Through variable)
v_3	Rod velocity (Across variable)

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 SimHydraulics® 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.

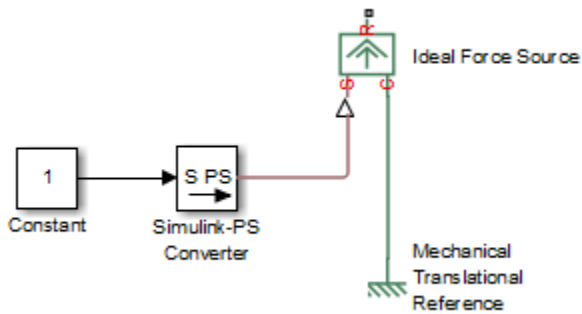
Elements with only two ports are characterized with one pair of variables, a Through variable and an Across variable. Since these variables are closely related, their orientation is defined with one direction. For example, if an element is oriented from port A to port B, it implies that the Through variable (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 $A \rightarrow B$ 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 1-4.

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 1-43.

Physical Signal Ports

Physical signal ports \triangleright 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. Unlike Simulink signals, which are essentially unitless, physical signals can have units associated with them. You specify the units along with the parameter values in the block dialogs, 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 1-43.

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 Physical Network diagram. 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 Physical Network diagram 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 an example of using converter blocks to connect Simscape diagrams to Simulink sources and scopes, see “Creating and Simulating a Simple Model” on page 1-18.

Simscape Block Libraries

In this section...

“Library Structure Overview” on page 1-11

“Using the Simulink Library Browser to Access the Block Libraries” on page 1-12

“Using the Command Prompt to Access the Block Libraries” on page 1-13

Library Structure Overview

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

- Foundation library — Contains basic hydraulic, pneumatic, mechanical, electrical, magnetic, thermal, thermal liquid, and physical signal blocks, 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 of the Physical Modeling family, you will see the corresponding libraries under the main Simscape library.

Simscape Foundation libraries contain a comprehensive set of basic elements and building blocks, such as:

- Mechanical building blocks for representing one-dimensional translational and rotational motion
- Electrical building blocks for representing electrical components and circuits
- Magnetic building blocks that represent electromagnetic components
- Hydraulic building blocks that model fundamental hydraulic effects and can be combined to create more complex hydraulic components
- Pneumatic building blocks that model fundamental pneumatic effects based on the ideal gas law
- Thermal building blocks that model fundamental thermal effects
- Thermal liquid building blocks that model fundamental thermodynamic effects in liquids
- Two-phase fluid building blocks that model fundamental thermodynamic effects in systems where the working agent is part liquid and part vapor

- Physical Signals block library that lets you perform math operations on physical signals, and graphically enter equations inside the physical network

Using the elements contained in these 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 outputs to Physical Signal inports. Use the PS-Simulink Converter block to connect Physical Signal outputs to Simulink inports.

For examples of using these blocks in a Simscape model, see the tutorial “Creating and Simulating a Simple Model” on page 1-18.

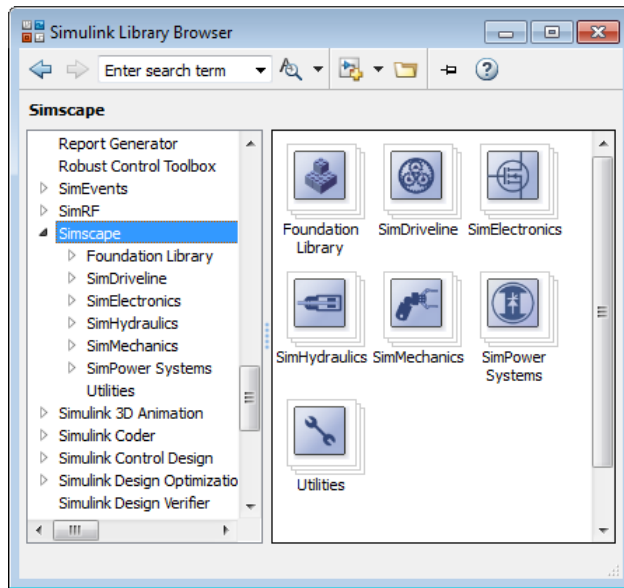
You can combine all these blocks in your Simscape diagrams to model physical systems. You can also use the basic Simulink blocks in your diagrams, such as sources or scopes. See “Connecting Simscape Diagrams to Simulink Sources and Scopes” on page 1-9 for more information on how to do this.

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”.

Using the Simulink Library Browser to Access the Block Libraries

You can access the blocks through the Simulink Library Browser. To display the Library Browser, click the **Simulink Library** button in the toolbar of the MATLAB® desktop.

Alternatively, you can type `simulink` in the MATLAB Command Window. Then expand the **Simscape** entry in the contents tree.



Using the Command Prompt to Access the Block Libraries

To access individual block libraries by using the command prompt:

- To open the Simscape library, type `simscape` in the MATLAB Command Window.
- To open the main Simulink library (to access generic Simulink blocks), type `simulink` in the MATLAB Command 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.
- If you have hydraulic elements in your model, the working fluid used in the hydraulic circuit defines their global parameters, such as fluid density, fluid kinematic viscosity, fluid bulk modulus, and so on. To specify the working fluid, attach a Custom Hydraulic Fluid block (or a Hydraulic Fluid block, available with SimHydraulics block libraries) to each topologically distinct hydraulic circuit. If no Hydraulic Fluid block or Custom Hydraulic Fluid block is attached to a circuit, the hydraulic blocks use the default fluid, which is equivalent to fluid defined by a Custom Hydraulic Fluid block with the default parameter values.
- If you have pneumatic elements in your model, default gas properties are for dry air and ambient conditions of 101325 Pa and 20 degrees Celsius. Attach a Gas Properties block to each topologically distinct pneumatic circuit to change gas properties and ambient conditions.
- 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 1-16.
- 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 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 SimHydraulics block libraries, or, depending on your application, even with a Segmented Pipeline block if you also need to account for fluid inertia. 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, pneumatic, 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 1-4.
- 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 1-43.
- 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 outputs to Physical Signal inputs. Use the PS-Simulink Converter block to connect Physical Signal outputs to Simulink inputs.
- Unlike Simulink signals, which are essentially unitless, Physical Signals can have units associated with them. Simscape block dialogs 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 1-18.

Creating and Simulating a Simple Model

In this section...

“Building a Simscape Diagram” on page 1-18

“Modifying Initial Settings” on page 1-26

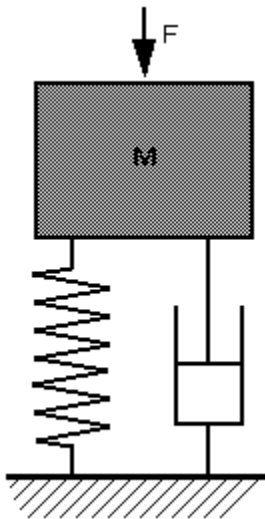
“Running the Simulation” on page 1-28

“Adjusting the Parameters” on page 1-30

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 and makes you familiar with using the basic Simscape blocks.

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.

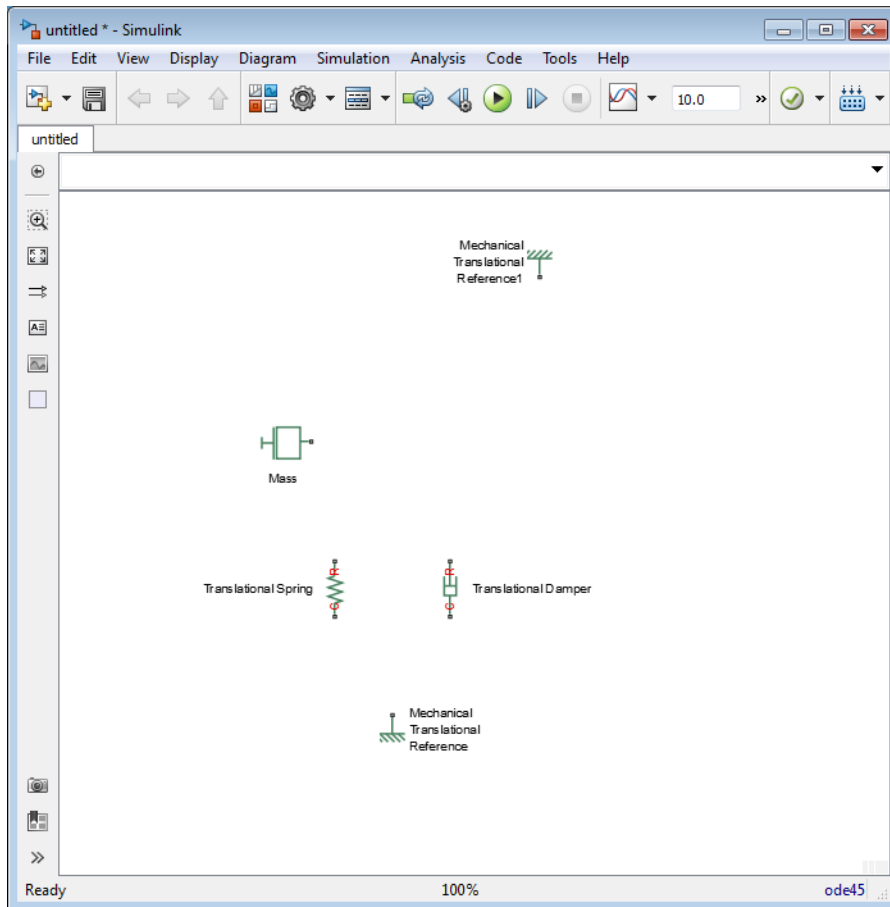


To create an equivalent Simscape diagram, follow these steps:

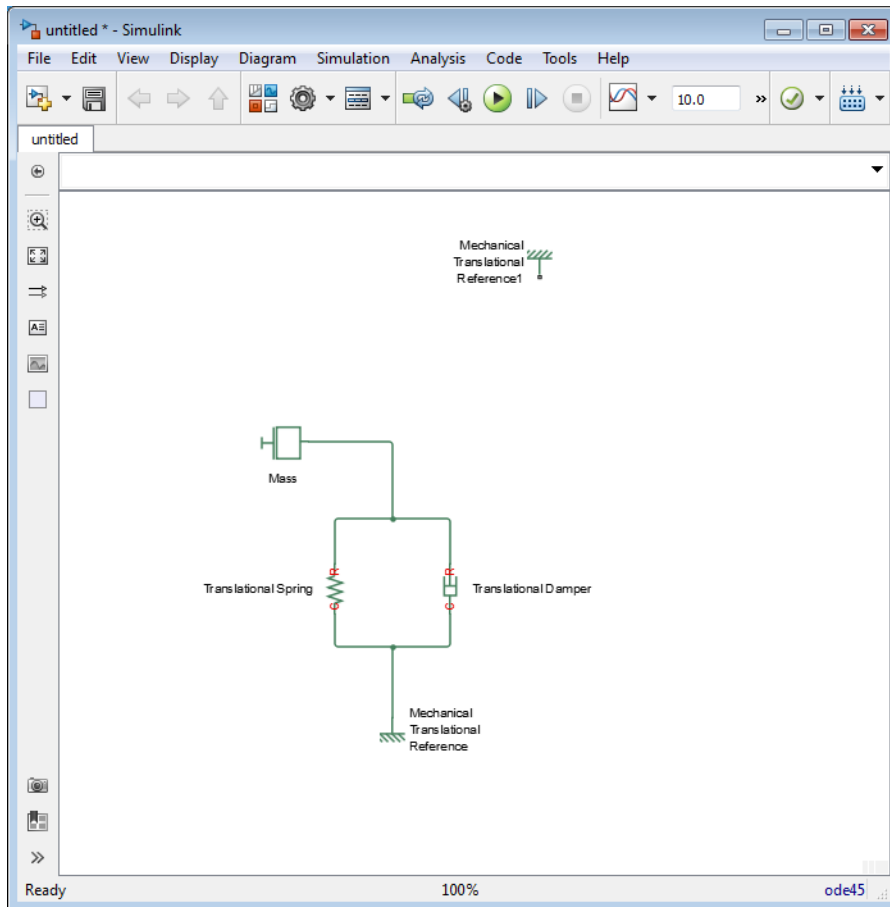
- 1 Open the Simulink Library Browser, as described in “Simscape Block Libraries” on page 1-11.
- 2 Create a new model. To do this, from the top menu bar of the Library Browser, select **File > New > Model**. The software creates an empty model in memory and displays it in a new model editor window.

Note Alternately, 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 Open the Simscape > Foundation Library > Mechanical > Translational Elements library.
- 4 Drag the Mass, Translational Spring, Translational Damper, and two Mechanical Translational Reference blocks into the model window.
- 5 Orient the blocks as shown in the following illustration. To rotate a block, select it and press **Ctrl+R**.

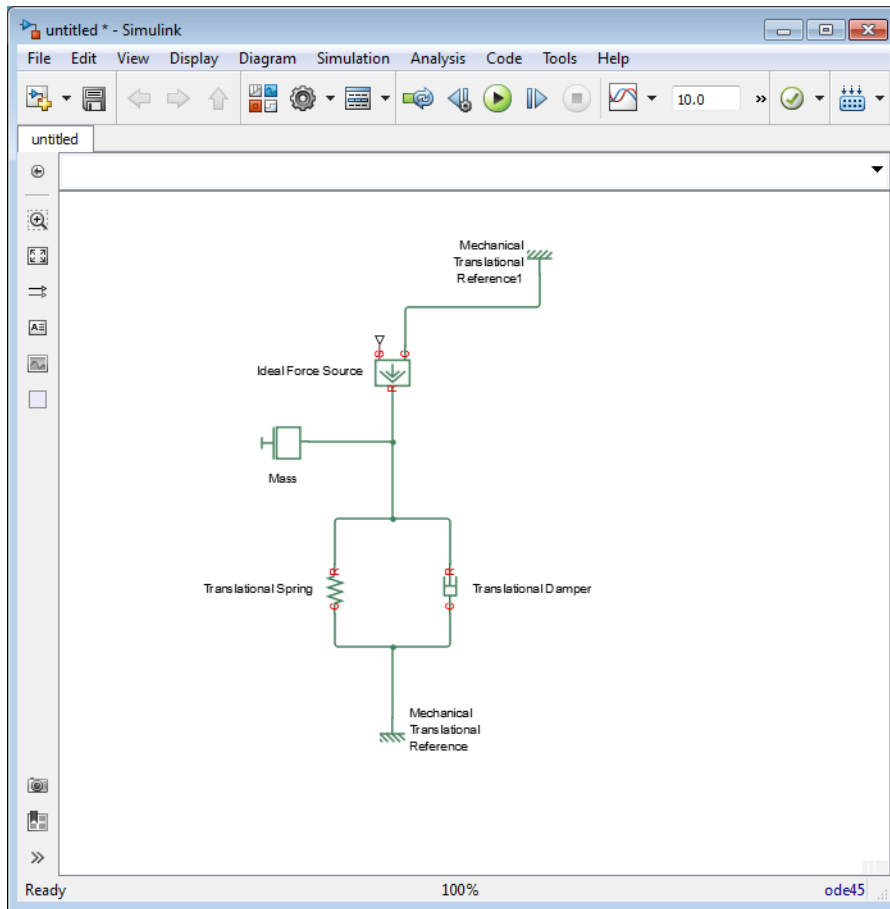


- 6 Connect the Translational Spring, Translational Damper, and Mass blocks to one of the Mechanical Translational Reference blocks as shown in the next illustration.

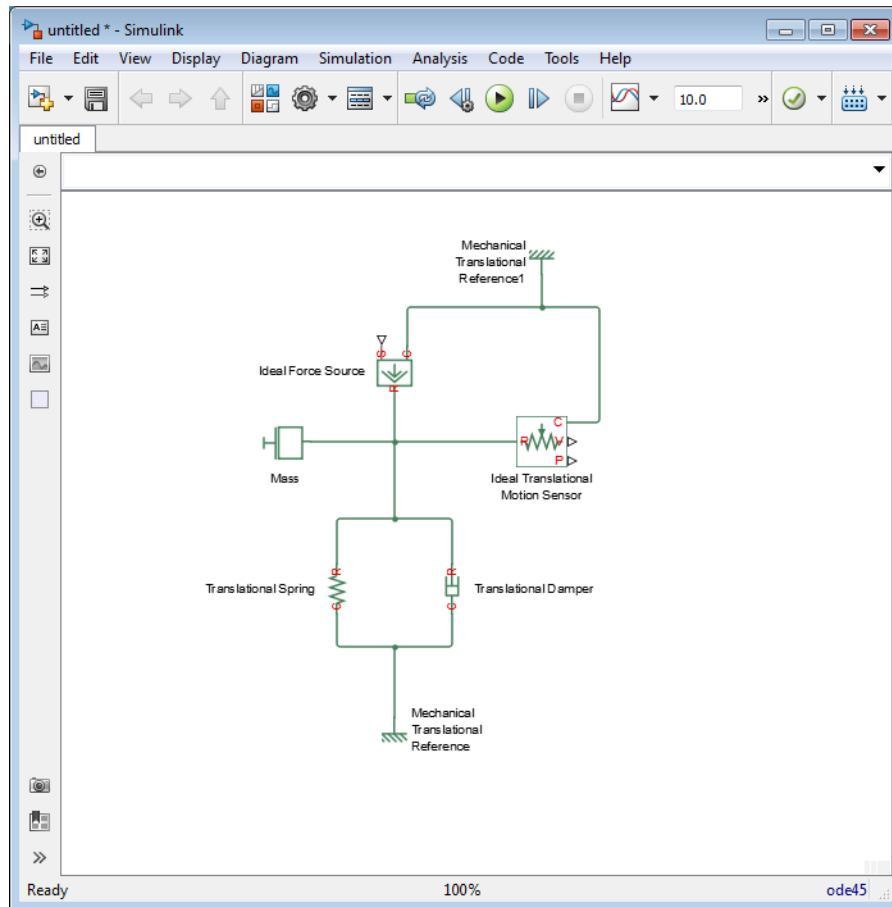


- 7 To add the representation of the force acting on the mass, open the Simscape > Foundation Library > Mechanical > Mechanical Sources library and add the Ideal Force Source block to your diagram.

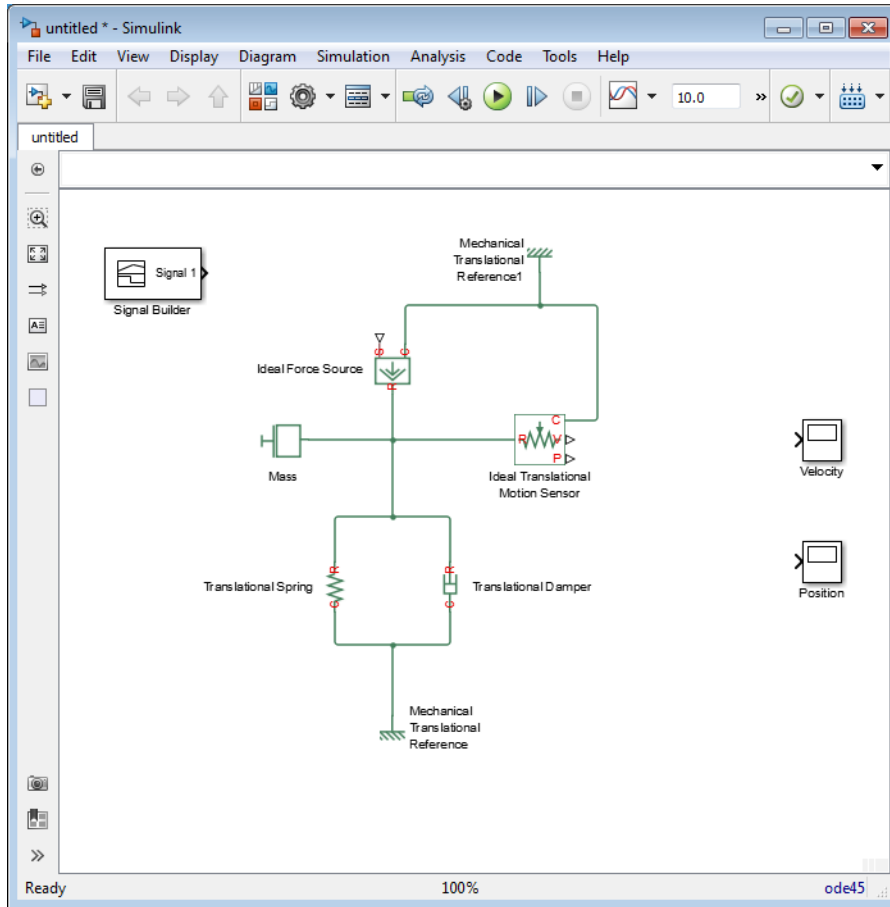
To reflect the correct direction of the force shown in the original schematic, flip the block by selecting **Diagram > Rotate & Flip > Flip Block > Up-Down** from the top menu bar of the model window. Connect the block's port C (for “case”) to the second Mechanical Translational Reference block, and its port R (for “rod”) to the Mass block, as shown below.



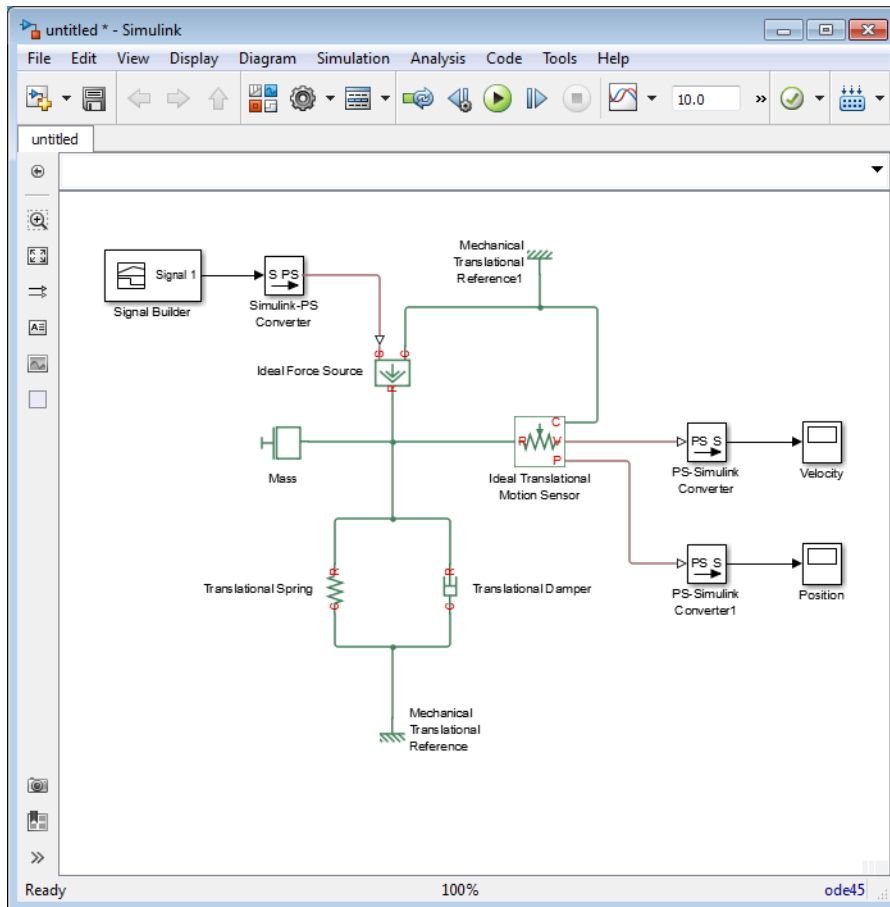
- 8 Add the sensor to measure speed and position of the mass. Place the Ideal Translational Motion Sensor block from the Mechanical Sensors library into your diagram and connect it as shown below.



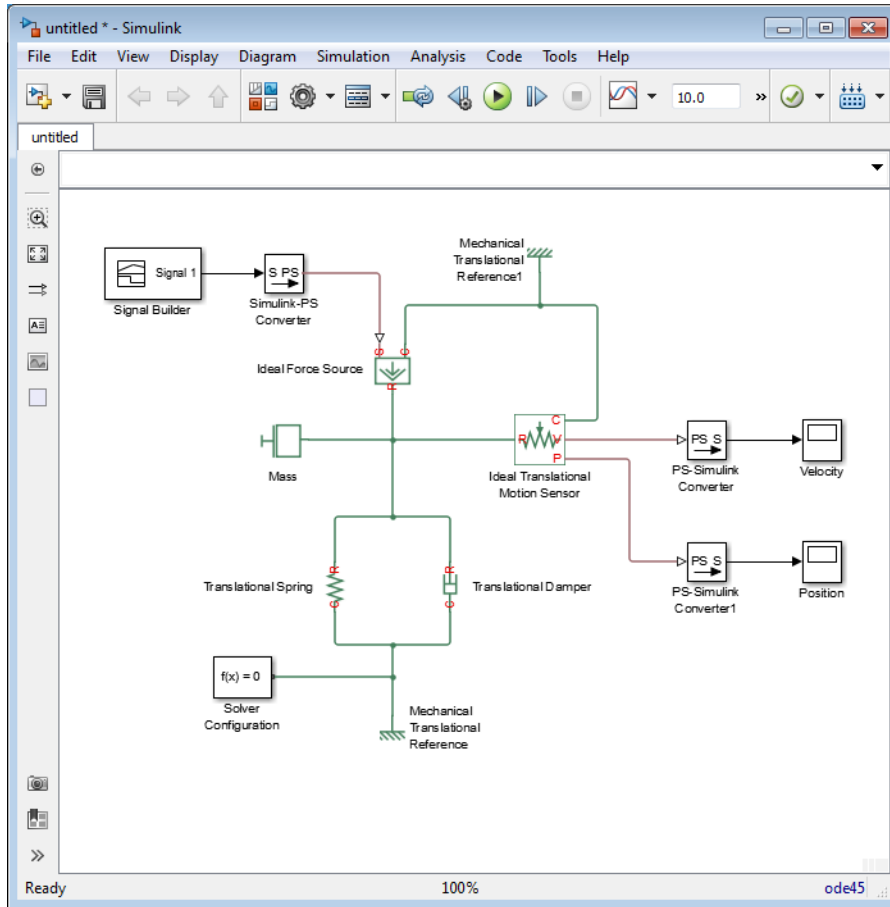
- 9 Now you need to add the sources and scopes. They are found in the regular Simulink libraries. Open the Simulink > Sources library and copy the Signal Builder block into the model. Then open the Simulink > Sinks library and copy two Scope blocks. Rename one of the Scope blocks to **Velocity** and the other to **Position**.



- 10 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.



- 11 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.



12 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, you need to select a solver and provide the correct values for configuration parameters.

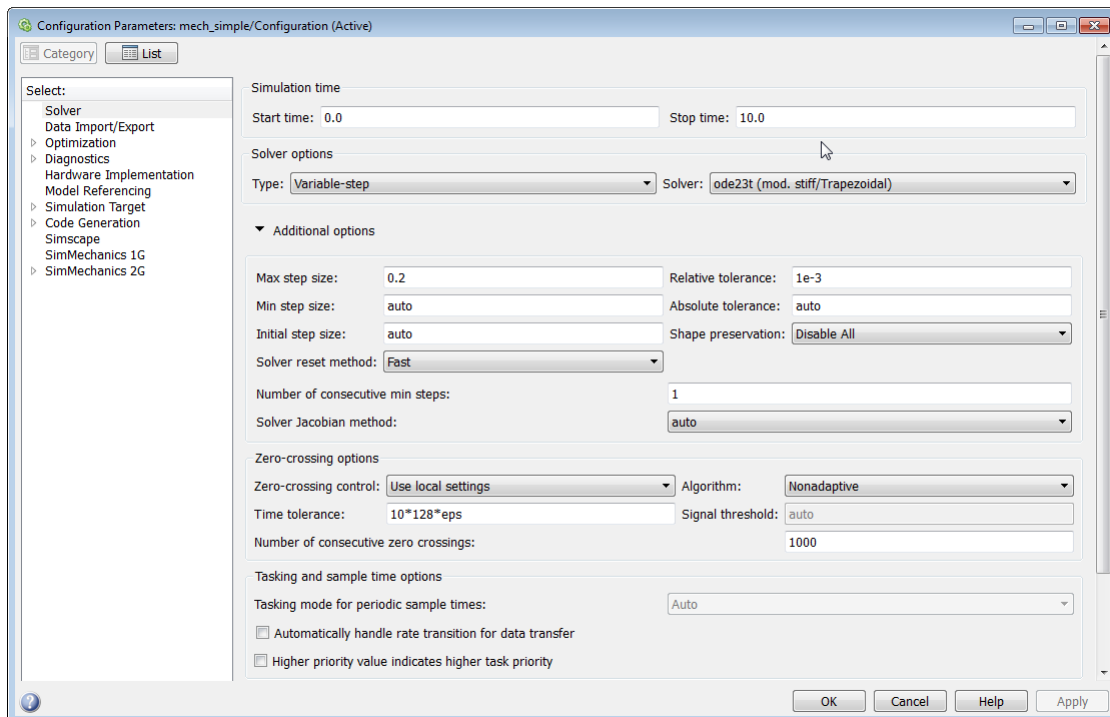
To prepare for simulating the model, follow these steps:

- 1 Select a Simulink solver. On the top menu bar of the model window, select **Simulation > Model Configuration Parameters**. The Configuration Parameters dialog box opens, showing the **Solver** node.

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

Expand **Additional options** 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.



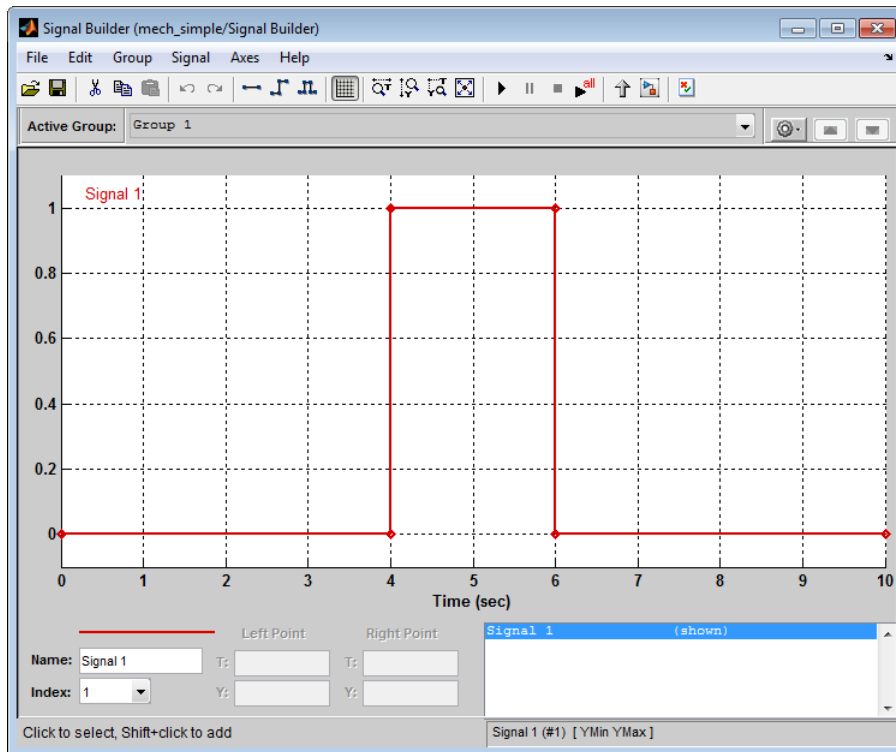
Click **OK** to close the Configuration Parameters dialog box.

- 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 Builder block. The signal profile is shown in the illustration below. It 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. This is the default profile.



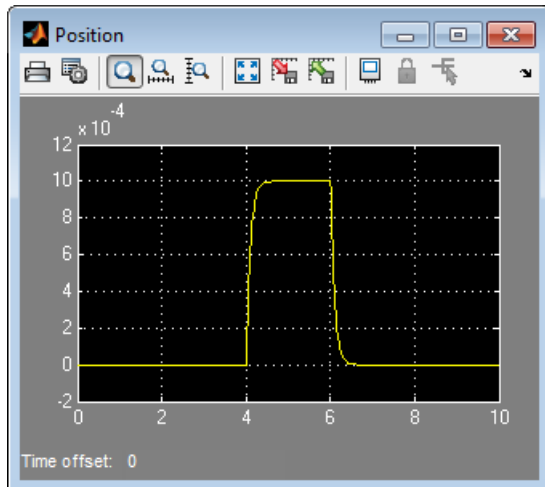
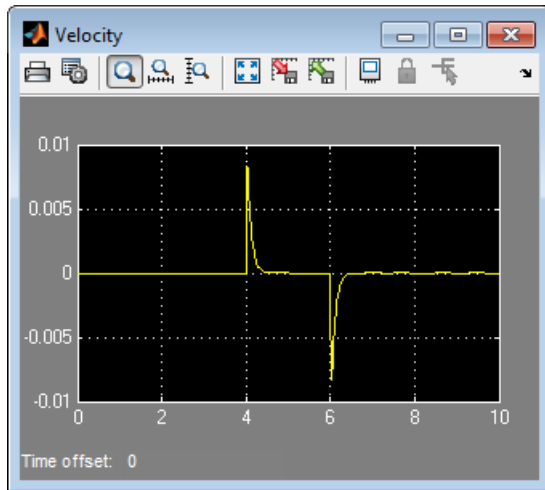
The Velocity scope outputs the mass velocity, and the Position scope outputs the mass displacement as a function of time. Double-click both scopes to open them.

2

To run the simulation, click  in the model window toolbar. 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 4-5. 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.

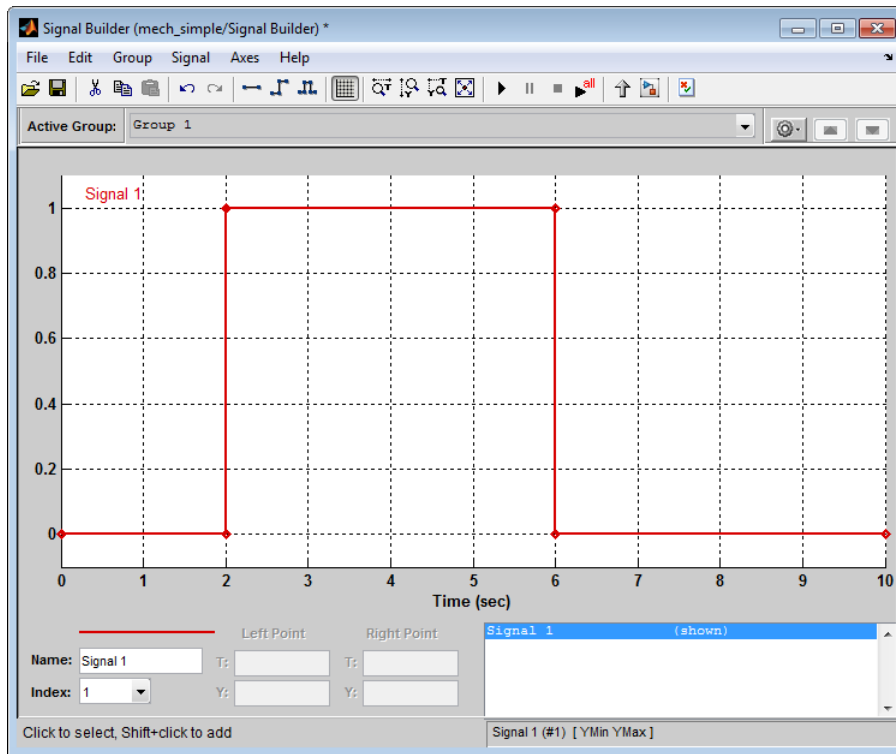
Try the following adjustments:

- 1 Change the force profile.
- 2 Change the model parameters.
- 3 Change the mass position output units.

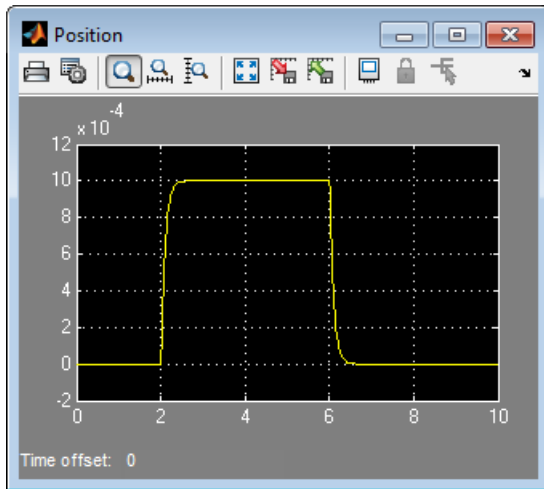
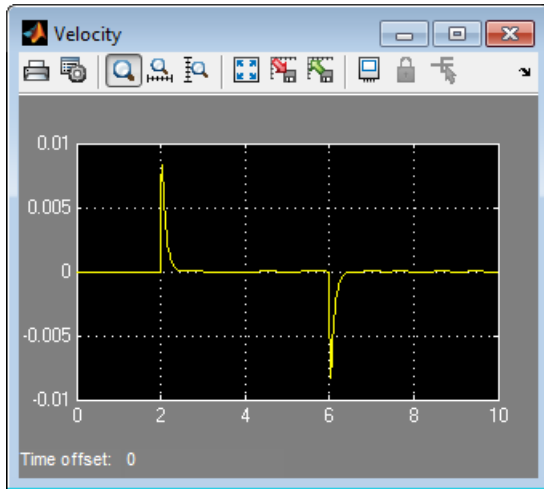
Changing the Force Profile

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

- 1 Double-click the Signal Builder block to open it.
- 2 Click the first vertical segment of the signal profile and drag it 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.

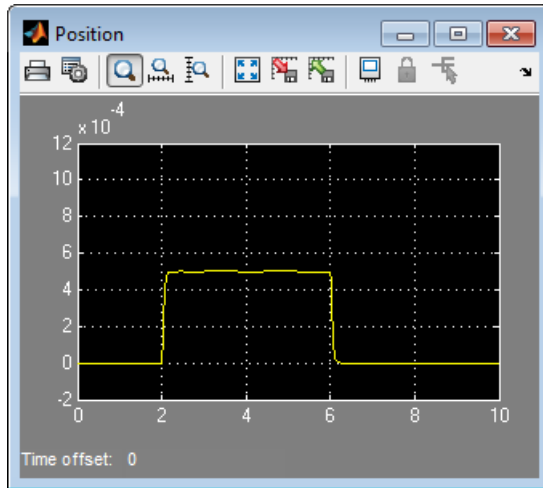


Changing the Model Parameters

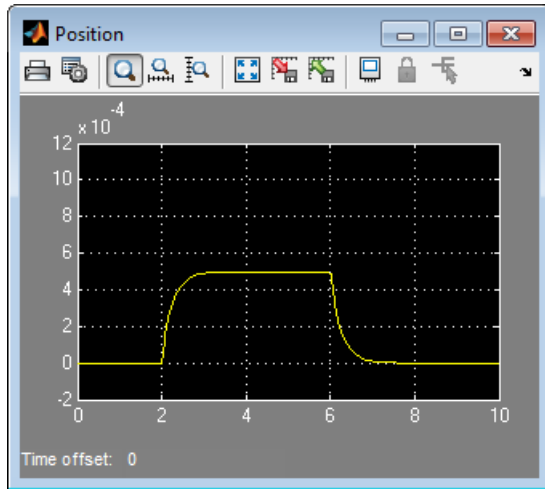
In our model, the force acts on a mass against a translational spring and damper, connected in parallel. This example shows how changes in the spring stiffness and damper viscosity affect the mass displacement.

- 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.




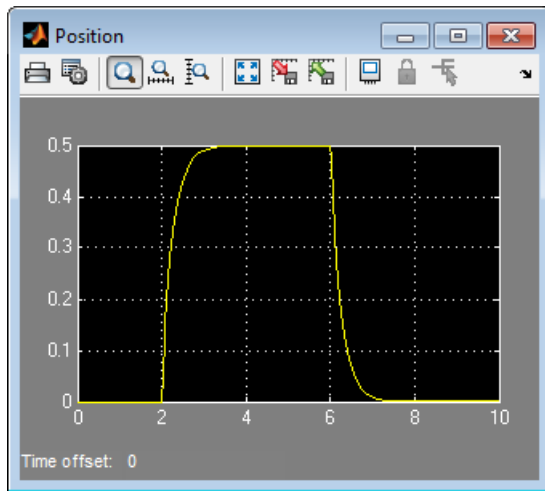
- 3 Next, double-click the Translational Damper block. Set its **Damping coefficient** to 500 N/(m/s).
- 4 Run the simulation. Because of the increase in viscosity, the mass is slower both in reaching its maximum displacement and in returning to the initial position, as shown in the following illustration.



Changing the Mass Position Output Units

In our model, we have used the PS-Simulink Converter block in its default parameter configuration, which does not specify units. 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 axes. The mass displacement is now output in millimeters, as shown in the following illustration.



More About

- “Basic Principles of Modeling Physical Networks” on page 1-2
- “Modeling Best Practices” on page 1-36

Modeling Best Practices

In this section...
“Grounding Rules” on page 1-36
“Avoiding Numerical Simulation Issues” on page 1-40

Grounding Rules

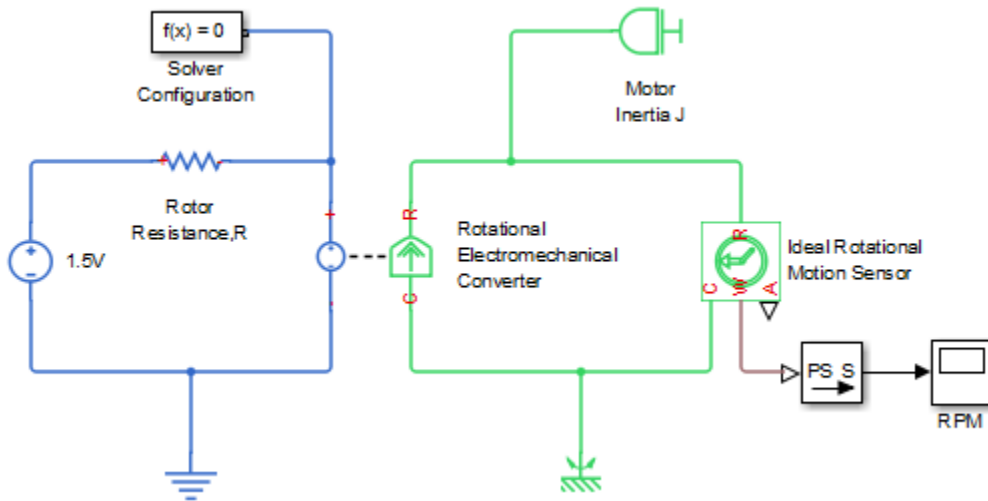
This section contains guidelines for using domain-specific reference blocks (such as Electrical Reference, Mechanical Translational Reference, and so on) in Simscape diagrams, along with examples of correct and incorrect configurations.

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

- “Each Domain Requires at Least One Reference Block” on page 1-36
- “Each Circuit Requires at Least One Reference Block” on page 1-37
- “Multiple Connections to the Domain Reference Are Allowed Within a Circuit” on page 1-39

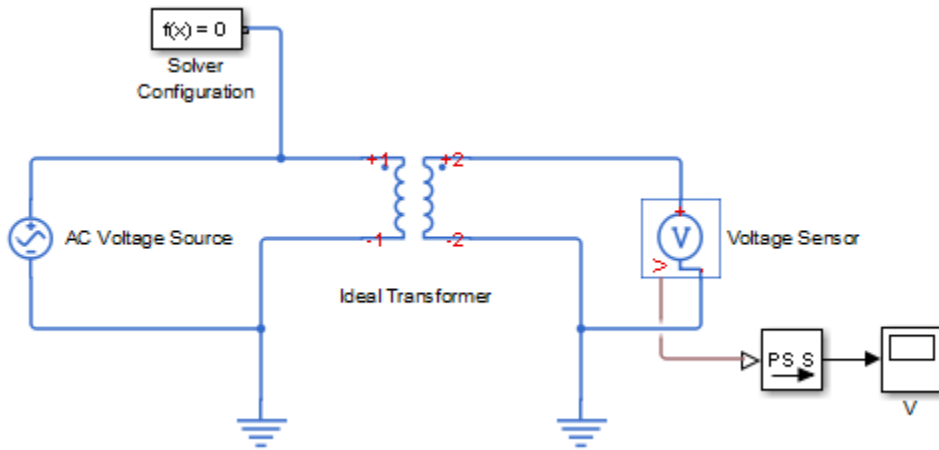
Each Domain Requires at Least One Reference Block

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 Rotational Reference blocks attached to the appropriate circuits.

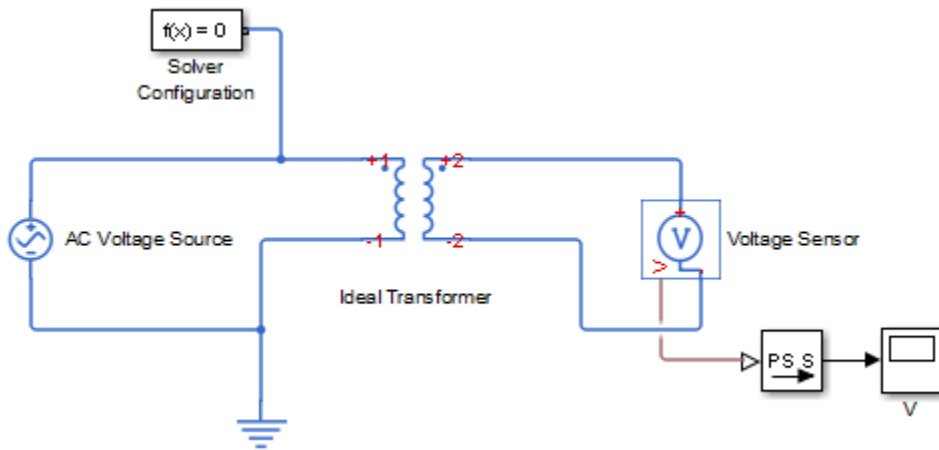


Each Circuit Requires at Least One Reference Block

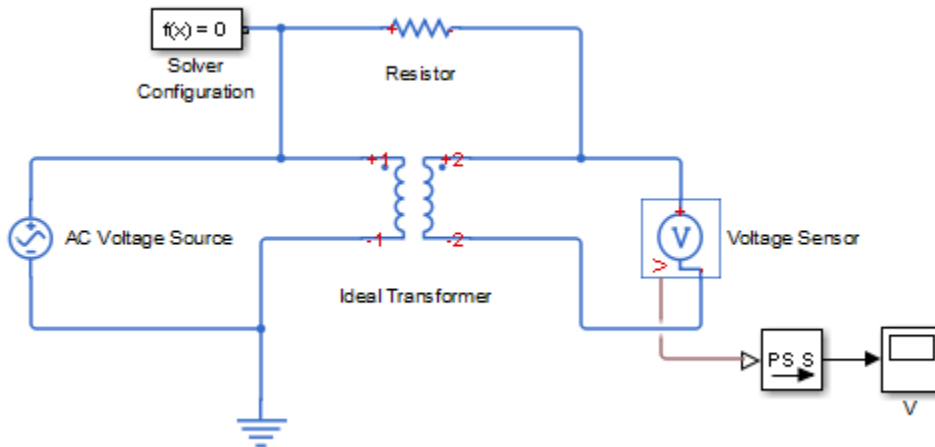
Each topologically distinct circuit within a domain must contain at least one reference block. Some blocks, such as an Ideal Transformer, interface two parts of the network but do not convey information about signal levels relative to the reference block. In the following diagram, there are two separate 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.

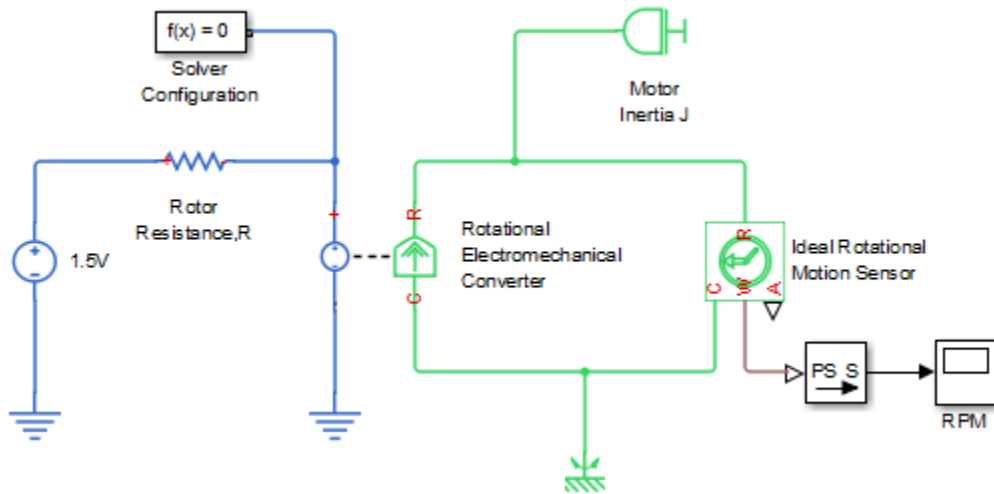


Multiple Connections to the Domain Reference Are Allowed Within a Circuit

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.
- 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.

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.

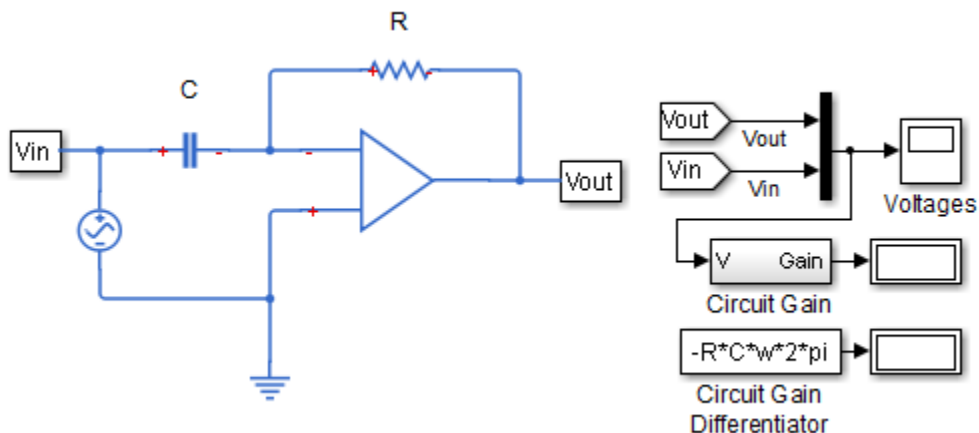
Note Mathematically, these topologies result in *Index-2 differential algebraic equations* (DAEs). Their solution requires two differentiations of the constraint equations and, as such, it is numerically better to avoid these component topologies where possible.

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.

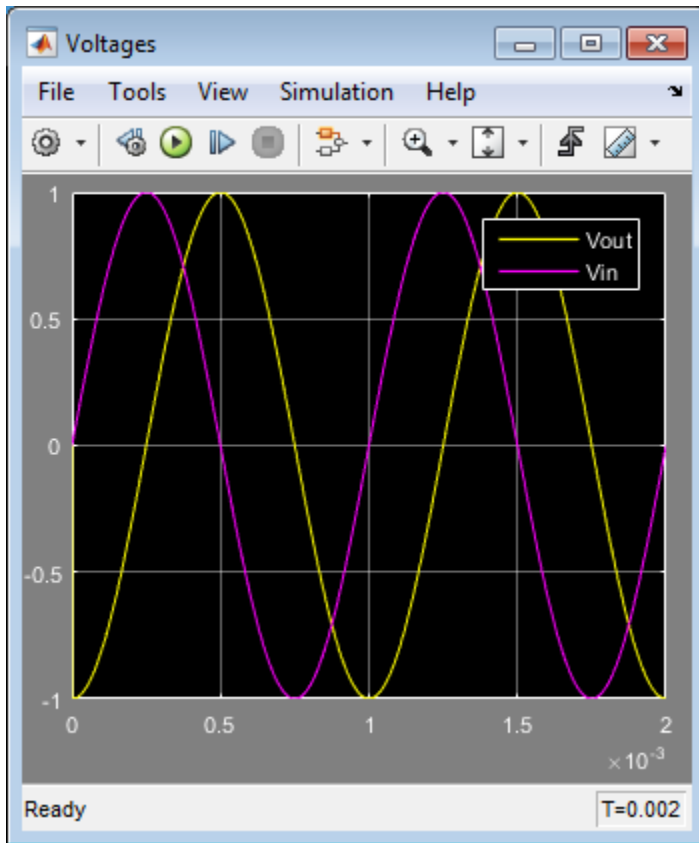
The second approach is to include small parasitic resistances in the component. In the Simscape Foundation library, the Capacitor and Inductor blocks include such parasitic terms, so that you can connect capacitances in parallel with voltage sources and inductors in series with current sources. If your 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.



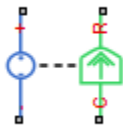
Now open the capacitor C block dialog, and set the series resistance to zero. The model now runs very slowly and issues warnings about problems with transient initialization and step size control for transient solve.

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.

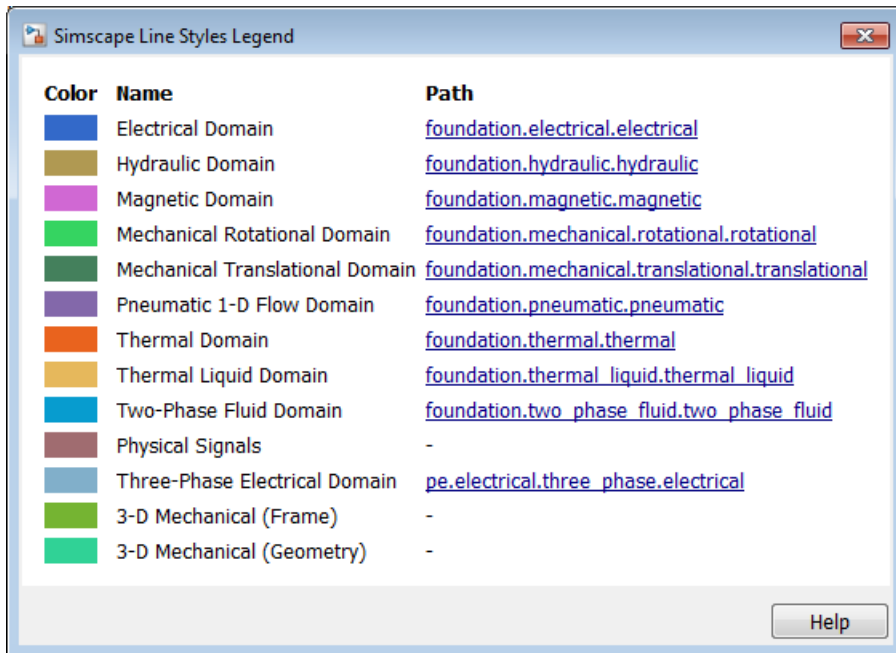
Domain-Specific Line Styles

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.

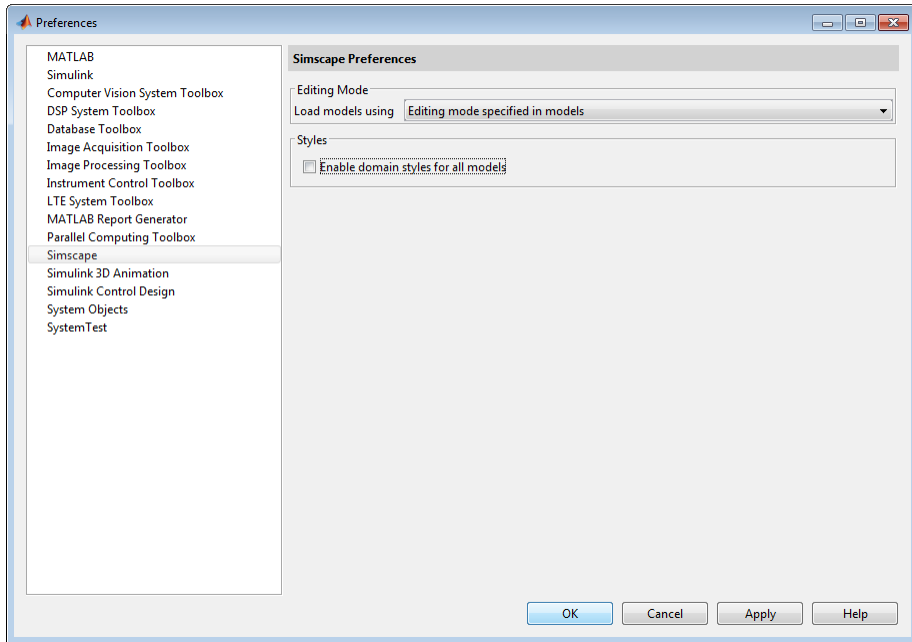


To view the line styles assigned to each domain, in a model window, from the top menu bar, select **Display > Simscape > Legend**. The Simscape Line Styles Legend window opens, listing the line color assigned to each registered domain, the domain name, and the domain path. If you click a domain path link, the Simscape file for the corresponding domain opens in MATLAB Editor. For more information on domain paths and files, see “Foundation Domains”.



To turn off domain-specific line styles for a particular model, in the model window, from the top menu bar, select **Display > Simscape > Domain Styles**. This action clears the check mark next to the **Domain Styles** menu option, and the block diagram display changes to black connection lines and block icons, with physical ports visible at connection points. Repeatedly selecting the **Domain Styles** menu option toggles the domain-specific line styles for this model on or off, as indicated by the check mark.

To turn off domain-specific line styles for all models, on 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.



Modeling Pneumatic Systems

In this section...
“Intended Applications” on page 1-46
“Assumptions and Limitations” on page 1-46
“Fundamental Equations” on page 1-47
“Network Variables” on page 1-48
“Connection Constraints” on page 1-48
“References” on page 1-49

Intended Applications

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

- Factory automation — basic pneumatic linear/rotational actuators, valves (variable orifices), and air supply
- Robotics — robotic arms and haptic interfaces
- Gaseous transportation systems and pipelines

You can also use these blocks to model dry air and low-pressure flows, for example, for HVAC applications.

Assumptions and Limitations

Pneumatic block models are based on the following assumptions:

- Working fluid is an ideal gas satisfying the ideal gas law.
- Specific heats at constant pressure and constant volume, c_p and c_v , are constant.
- Processes are adiabatic, that is, there is no heat transfer between components and the environment (except for components with a separate thermal port).
- Gravitational effects can be neglected, that is, underlying equations contain no head pressures due to gravity.

Fundamental Equations

The energy balance for a control volume [1] is

$$\frac{dE_{cv}}{dt} = Q_{cv} - W_{cv} + \sum_i \left(m_i \left(h_i + \frac{v_i^2}{2} + gz_i \right) \right) - \sum_o \left(m_o \left(h_o + \frac{v_o^2}{2} + gz_o \right) \right)$$

where

E_{cv}	Control volume total energy
Q_{cv}	Heat energy per second added to the gas through the boundary
W_{cv}	Mechanical work per second performed by the gas
h_i, h_o	Inlet and outlet enthalpies
v_i, v_o	Gas inlet and outlet velocities
g	Acceleration due to gravity
z_i, z_o	Elevations at inlet and outlet ports
m_i, m_o	Mass flow rates in and out of the control volume

The equation is an accounting balance for the energy of the control volume. It states that the rate of energy increase or decrease within the control volume equals the difference between the rates of energy transfer in and out across the boundary. The mechanisms of energy transfer are heat and work, as for closed systems, and the energy that accompanies the mass entering and exiting.

Pneumatic block models make several simplifying assumptions, as described previously.

The ideal gas law relates pressure, density, and temperature:

$$p = \rho RT$$

where

p	Absolute pressure
ρ	Gas density
R	Specific gas constant

T	Absolute gas temperature
-----	--------------------------

Also, the specific enthalpies for an ideal gas at temperature T and constant pressure and constant volume are given by:

$$h = c_p T$$

$$h = c_v T$$

The pneumatic components also use the mass continuity equation:

$$\frac{d\rho}{dt} = \frac{1}{V}(m_i - m_o)$$

where ρ is the density of the gas within the component. For components with no internal mass of gas, the equation simplifies to:

$$G = m_i = m_o$$

where G is the mass flow rate through the component.

For specific equations used in each block, see the block reference pages.

Network Variables

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

Connection Constraints

Every node in a pneumatic network must have a defined temperature as well as pressure. This rule places some constraints on how you connect the pneumatic elements. In effect, every node should have a volume of fluid associated with it. When the ideal gas law is applied, this volume of fluid determines the relationship between temperature and pressure. Some elements already have a volume of fluid associated with them, and therefore having just one of these components connected to a node satisfies this

condition. Such blocks include Constant Volume Pneumatic Chamber, Pneumatic Piston Chamber, Rotary Pneumatic Piston Chamber, and Pneumatic Atmospheric Reference.

An exception to the above rule (that every node must have a volume of fluid associated with it) occurs when two nodes are connected by a component for which the heat equation says that the temperatures are equal. In this case, just one of the nodes needs to be connected to a component with associated volume of fluid. Such components include the pressure and flow rate sources.

For models that represent an actual pneumatic network, these constraints should have no impact. For example, connecting two orifices in series makes no physical sense because the underlying assumption of the orifice equation is that gas is discharged into a volume of fluid. Therefore, modeling actual physical systems should automatically satisfy these constraints.

References

[1] Moran M.J. and Shapiro H.N. *Fundamentals of Engineering Thermodynamics*. Second edition. New York: John Wiley & Sons, 1992.

Thermal Liquid Models

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-6
- “Thermal Liquid Modeling Framework” on page 2-10
- “Heat Transfer in Insulated Oil Pipeline” on page 2-14

Modeling Thermal Liquid Systems

In this section...

“When to Use Thermal Liquid Blocks” on page 2-2

“Modeling Workflow” on page 2-3

“Establish Model Requirements” on page 2-3

“Model Physical Components” on page 2-4

“Prepare Model for Analysis” on page 2-5

“Run Simulation” on page 2-5

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—e.g. Hydraulic or Pneumatic—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. For gaseous systems, use Pneumatic blocks instead. For isothermal liquid systems, use Hydraulic blocks.

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. This block is not strictly required, but without it the liquid properties are reset to their default values, given for water. In the block dialog box, 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 custom, requiring multiple blocks typically within a **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 custom 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 custom 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 flow rate and heat flux), 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 9-2. For an example of how to plot logged data, see “Log and Plot Simulation Data” on page 9-8.

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 **Model Configuration Parameters** dialog box.

For physical models, variable-step solvers such as `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.

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-14

More About

- “Thermal Liquid Library” on page 2-6
- “Thermal Liquid Modeling Framework” on page 2-10

Thermal Liquid Library

In this section...
“Why Use Thermal Liquid Blocks?” on page 2-6
“Representing Thermal Liquid Components” on page 2-6
“Specifying Thermal Liquid Medium” on page 2-8
“Modeling Multidomain Systems” on page 2-8

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), Variable 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	Collect measurement data for analysis of parameters,	Pressure & Temperature Sensor (TL), Mass Flow

Component Type	Description	Thermal Liquid Blocks
	such as mass flow rate, thermal flux, pressure, and temperature.	Rate & Thermal Flux 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)	✓	✓

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-14

More About

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Modeling Framework” on page 2-10

Thermal Liquid Modeling Framework

In this section...

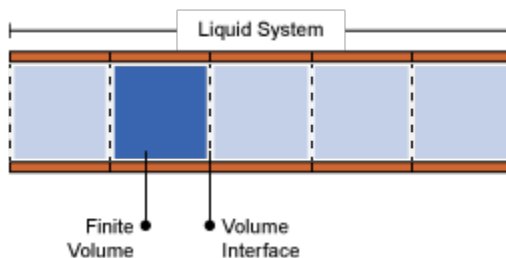
“How Blocks Represent Components” on page 2-10

“How Ports Represent Interfaces” on page 2-11

“Full Flux Scheme” on page 2-12

How Blocks Represent Components

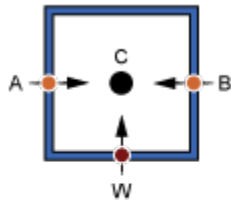
Thermal Liquid models are based on the finite volume method. This method discretizes a thermal liquid system into multiple control volumes that interact via shared interfaces. An oil pipeline system is one example: you can model this system as a set of pipeline segments that connect serially along the 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 9-2.



- 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.

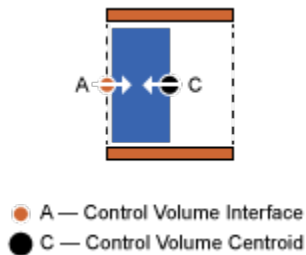
How Ports Represent Interfaces

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.



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.

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-14

More About

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-6

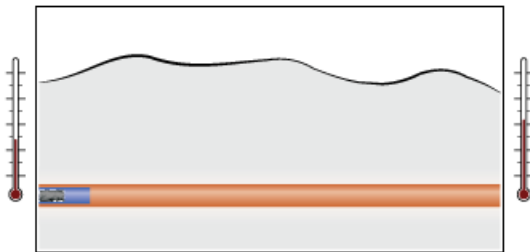
Heat Transfer in Insulated Oil Pipeline

In this section...

“Oil Pipelines” on page 2-14
“Modeling Considerations” on page 2-15
“Simscape Model” on page 2-17
“Run Simulation” on page 2-18
“Run Optimization Script” on page 2-25

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.

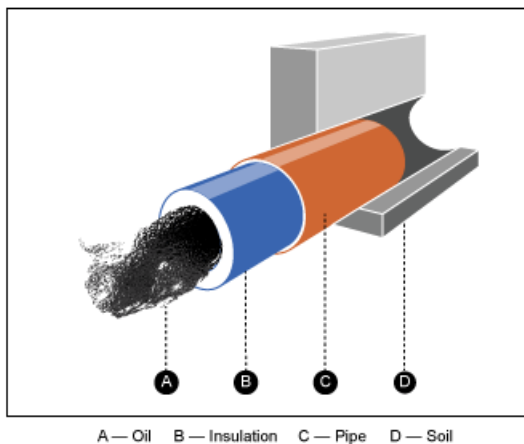
Viscous dissipation provides an additional heat source. As adjacent parcels of oil flow against each other, they experience energy losses that appear in the form of heat. The warming effect is small, but sufficient to at least partially offset the conductive heat losses that occur through the insulant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



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, $T_{Upstream}$, with a fixed mass flow rate, $Vdot * rho_0$, where:

- $Vdot$ is the volumetric flow rate of oil through the pipe.

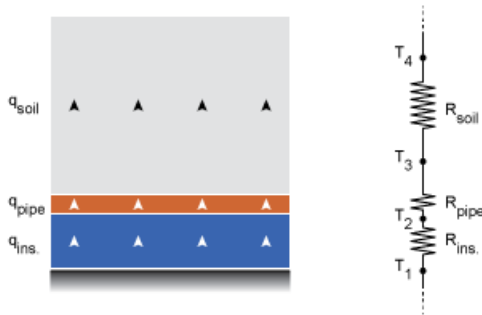
- ρ_0 is the mass density of oil entering the pipeline segment.

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:

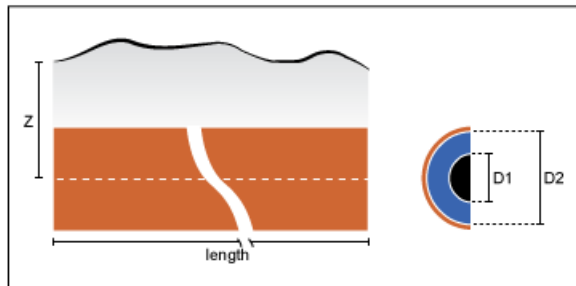
$$R_{combined} = R_{wall} + R_{ins.} + R_{soil}$$



Assuming the pipe wall is thin and its material 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, $R_{ins.}$ and R_{soil} .

The thermal resistance of the insulation layer is directly proportional to its thickness, $(D_2 - D_1)/2$, and inversely proportional to its thermal conductivity, $k_{Insulant}$. Likewise, the thermal resistance of the soil layer is directly proportional to its thickness, z , and inversely proportional to its thermal conductivity, k_{Soil} .

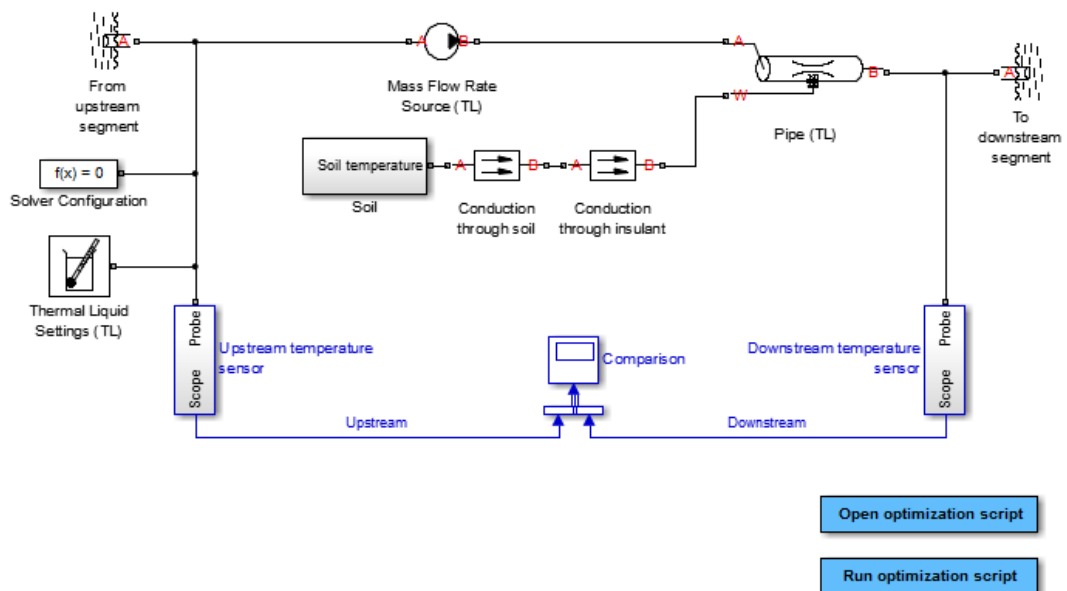
The figure shows the relevant dimensions of the pipeline segment. Variable names match those specified in the model. The inner insulation diameter, D_1 , is also the hydraulic diameter of the pipeline segment.



z — Soil layer thickness
 length — Pipeline segment length
 D1 — Inner insulation diameter
 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.



The Pipe (TL) block represents the physical system in this example, i.e., the oil pipeline segment. Port A represents its inlet and port B its outlet. Port W 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 From upstream segment block acts as a temperature source for the pipe inlet, while the To downstream segment block acts as a temperature sink at the pipe outlet.

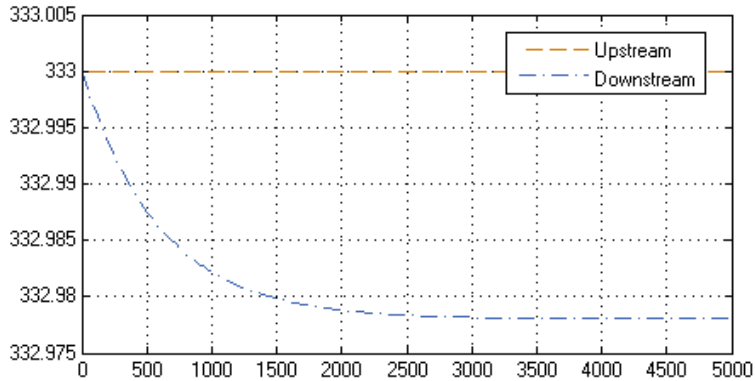
The Conduction through insulant and Conduction through soil blocks represent thermal conduction through insulant and soil layers, respectively. These blocks appear in the Simscape Thermal library as **Conductive Heat Transfer**. The Soil subsystem block provides the temperature boundary condition at the soil surface.

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

Block	Description
Pipe (TL)	Pipeline segment
Conduction through insulant	Insulant thermal conduction
Conduction through soil	Soil thermal conduction
Soil (Subsystem)	Soil temperature
From upstream segment	Pipe inlet temperature sink
To downstream segment	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 Comparison 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

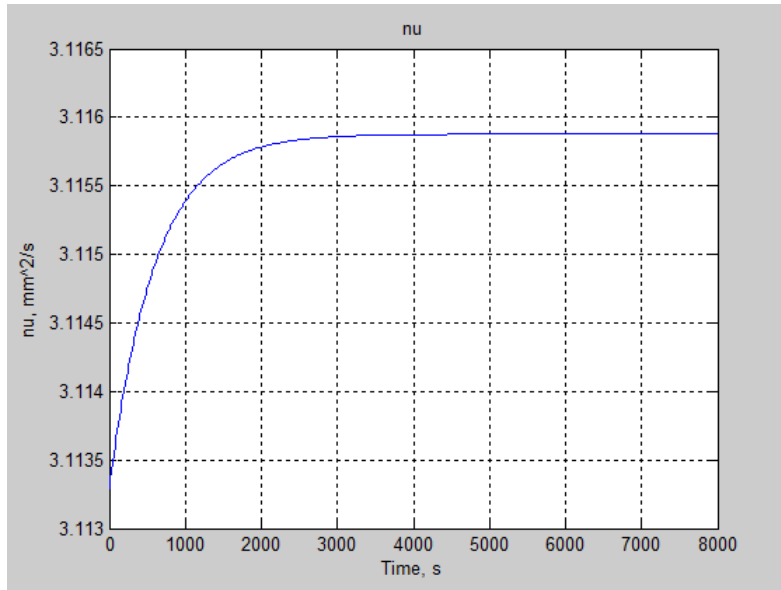
By using Simscape data logging, you can plot the physical properties of the oil as a function of simulation time. Such a plot clearly shows any variability in the value of a physical property. One example is the kinematic viscosity of oil in the pipeline segment, represented by the Pipe (TL) block.

- 1 At the MATLAB command line, enter `simlog.Pipe_TL.print`.

In the data tree, the kinematic viscosity `nu` appears under the node `pipe_model`, which itself appears under the node `simlog.Pipe_TL`. The logging object for the kinematic viscosity of oil in the pipe, then, is `simlog.Pipe_TL.pipe_model.nu`.

```
>> simlog.Pipe_TL.print
Pipe_TL
+-A
| +-T
| +-p
+-B
| +-T
| +-p
+-W
| +-T
+-pipe_model
+-A
| +-T
| +-p
+-B
| +-T
| +-p
+-Phi_A
+-Phi_B
+-Phi_W
+-Phi_convection_A
+-Phi_convection_B
+-T
+-W
| +-T
+-alpha
+-beta
+-cp
+-k
+-mdot_A
+-mdot_B
+-nu
+-p
+-rho
+-rho_A
+-rho_B
+-u
+-u_A
+-u_B
+-viscous_friction_A
+-viscous_friction_B
```

- At the MATLAB command line, enter `simscape.logging.plot({simlog.Pipe_TL.pipe_model.nu})`.



As expected, the kinematic viscosity remains approximately constant throughout the simulation, reflecting the minimal temperature changes that occur in the oil.

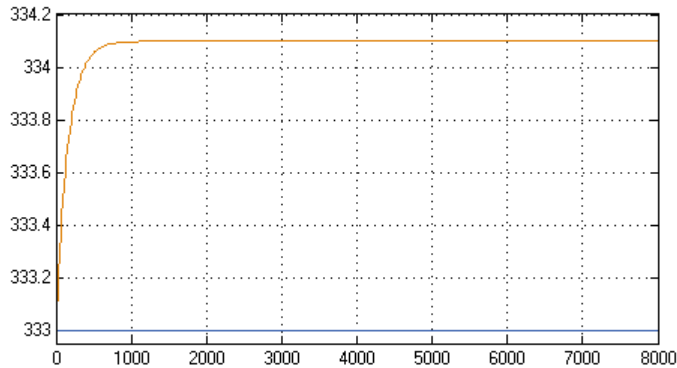
Note: For more information about Simscape logging, see “About Simulation Data Logging” on page 9-2.

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.

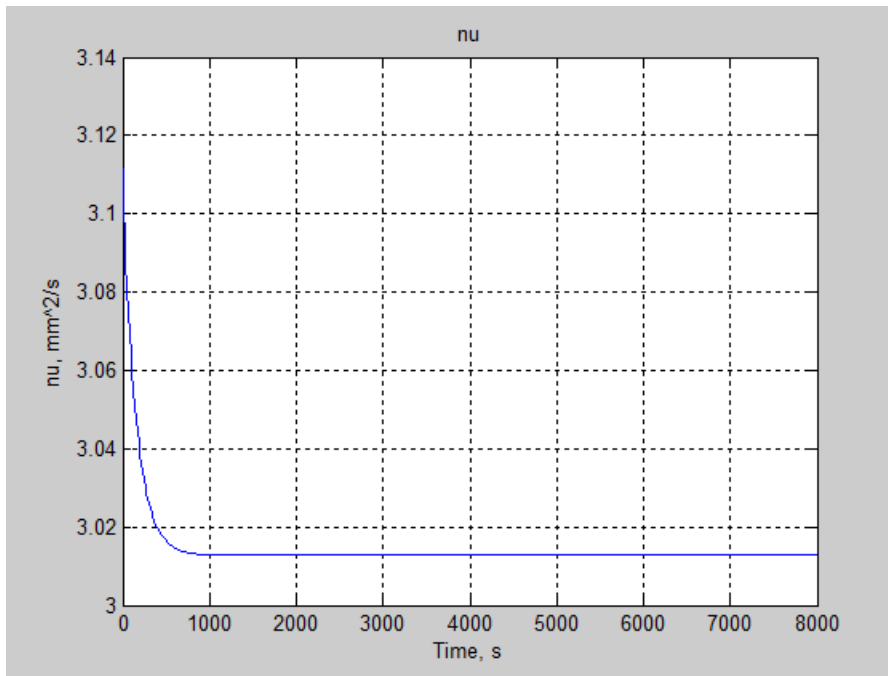
- Open Model Explorer.
- In the **Model Hierarchy** pane, select **Base Workspace**.
- In the **Contents** pane, click the value of parameter D1.
- Enter 0.20.

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 Comparison scope and autoscale to view full plot.

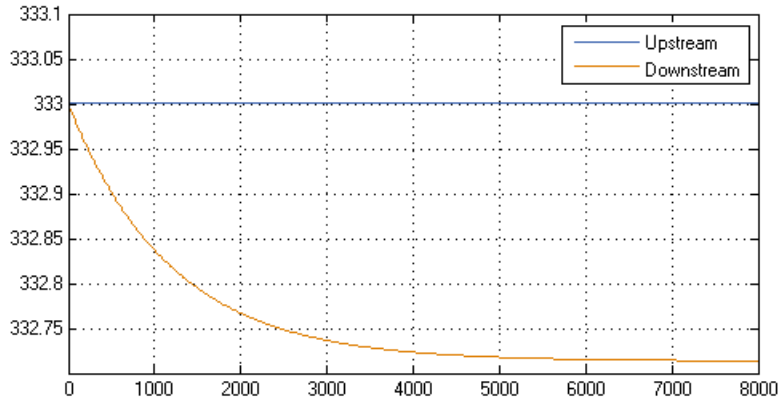


The new plot shows an oil temperature at the pipe outlet (top curve) that significantly exceeds that at the pipe inlet (bottom line). Viscous dissipation now dominates the thermal energy balance in the pipeline segment. The new insulation thickness poses a design problem: in a long pipeline, a 1.1 K/km heating rate can raise the oil temperature substantially at the receiving end of the pipeline.

Plotting the kinematic viscosity as a function of time shows that its variability is now quite significant also. At the MATLAB command line, enter the logging command:
`simscape.logging.plot({simlog.Pipe_TL.pipe_model.nu})`.

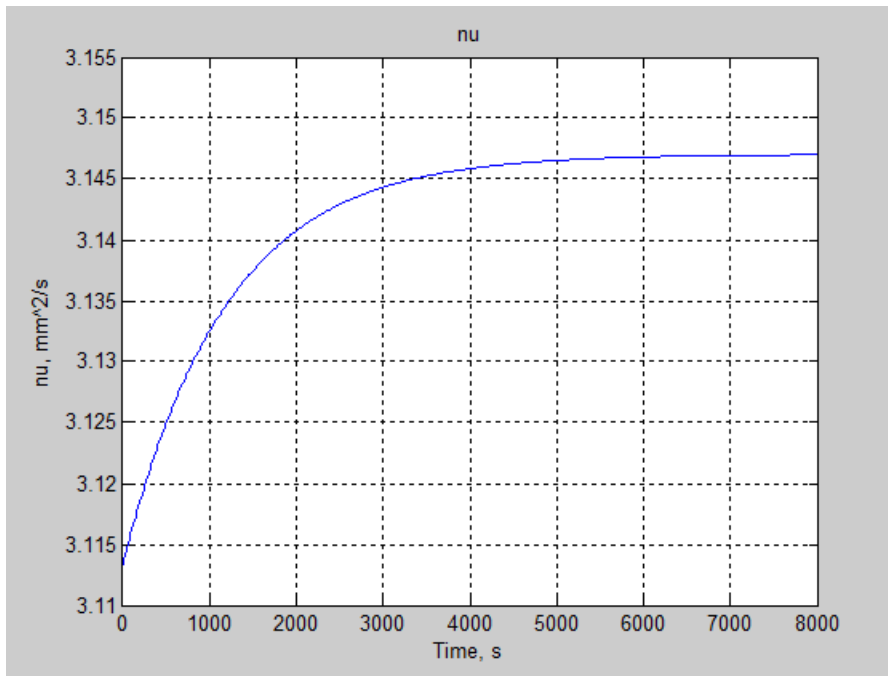


Try increasing the inner diameter of the insulation layer, D_1 , to 0.55. By increasing this value, you decrease the insulation thickness, accelerating heat loss through the pipe wall via thermal conduction. Then, run the simulation. Open the Comparison scope and autoscale to view the full plot.



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.

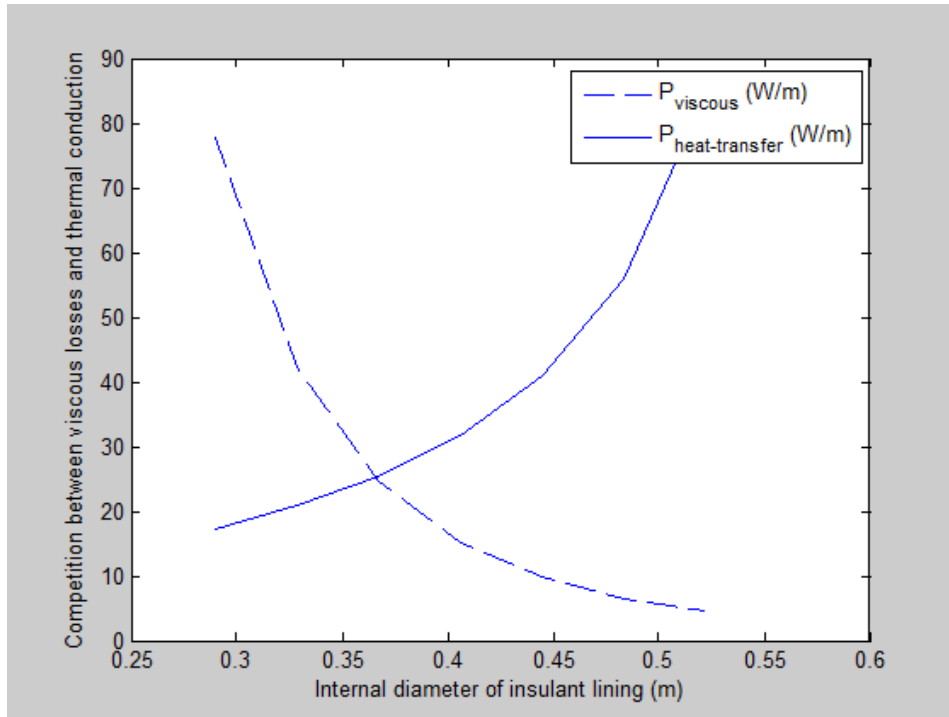
Plot the kinematic viscosity as a function of time using Simscape logging. Because the temperature change is now more modest, changes in viscosity are less significant.



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, double-click **Run optimization script**.

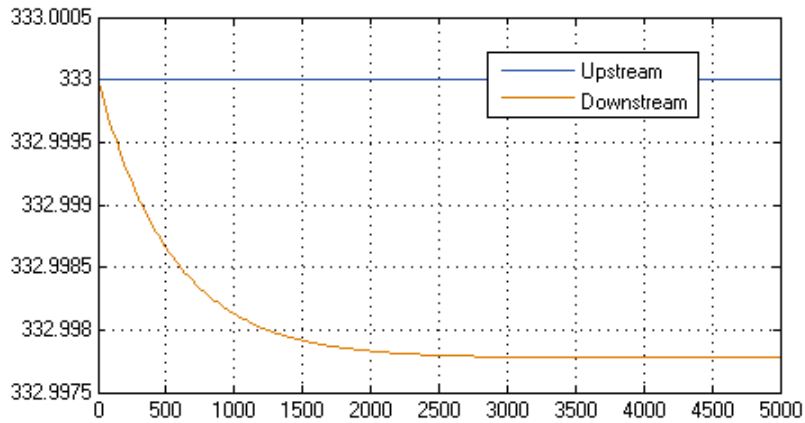


- 2 In the plot that opens, visually determine the horizontal-axis value for the intersection point between the two curves.

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 Comparison scope and autoscale to view the full plot. The temperature difference between the inlet and the outlet is negligible.



More About

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-6
- “Thermal Liquid Modeling Framework” on page 2-10

Two-Phase Fluid Models

Manually Generate Fluid Property Tables

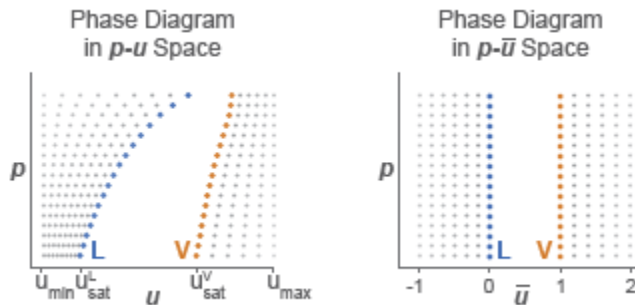
In this section...
“Fluid Property Tables” on page 3-2
“Steps for Generating Property Tables” on page 3-3
“Before Generating Property Tables” on page 3-3
“Create Fluid Property Functions” on page 3-3
“Set Property Table Criteria” on page 3-4
“Create Pressure-Normalized Internal Energy Grids” on page 3-5
“Map Grids Onto Pressure-Specific Internal Energy Space” on page 3-5
“Obtain Fluid Properties at Grid Points” on page 3-6
“Visualize Grids” on page 3-7

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 installed, you can automatically generate these tables using the `twoPhaseFluidTable` function. If you obtain the fluid properties from a different source, such as CoolProp software, 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 = 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 — 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 `liquids` 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-');

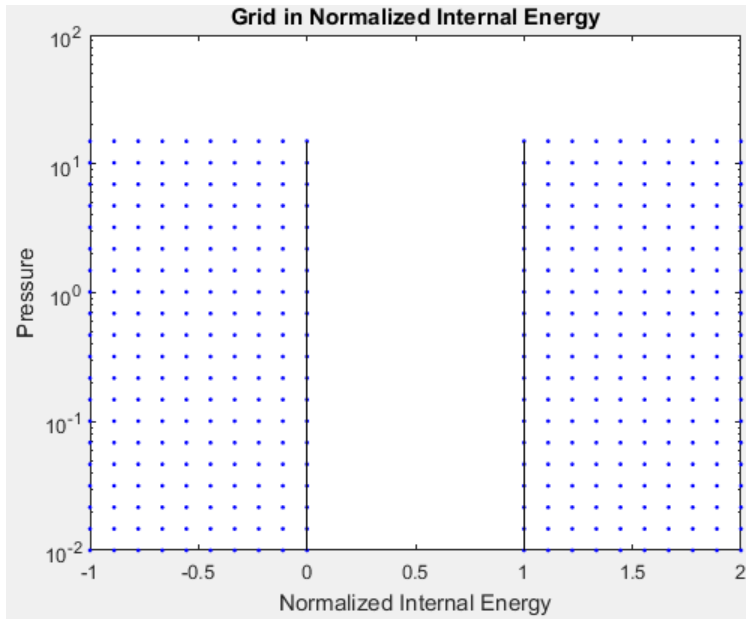
plot(zeros(1, n), fluidTables.p, 'k-');
plot(ones(1, n), fluidTables.p, 'k-');

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.');
```

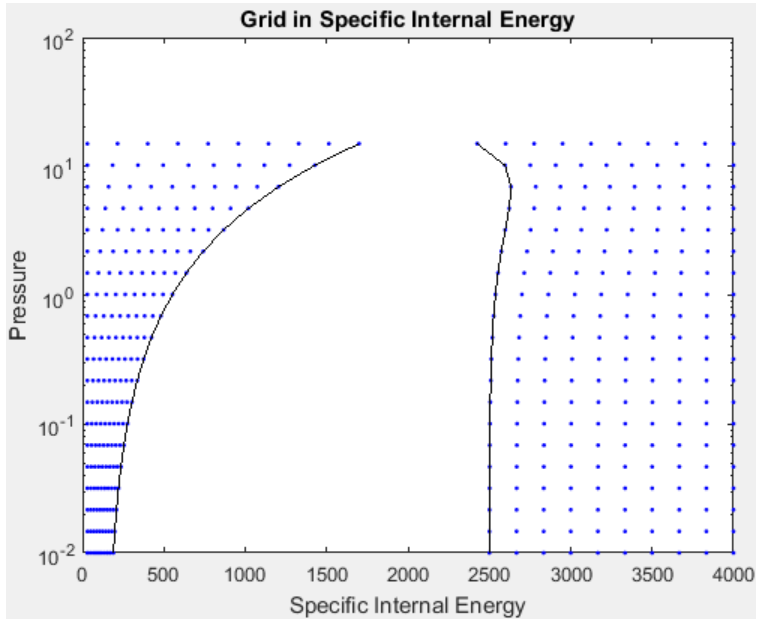
plot(fluidTables.vapor.u, pVapor, 'b.');

```
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.



Model Simulation

- “How Simscape Models Represent Physical Systems” on page 4-2
- “How Simscape Simulation Works” on page 4-5
- “Setting Up Solvers for Physical Models” on page 4-11
- “Important Concepts and Choices in Physical Simulation” on page 4-17
- “Making Optimal Solver Choices for Physical Simulation” on page 4-21
- “Troubleshooting Simulation Errors” on page 4-26
- “Limitations” on page 4-32
- “References” on page 4-37

How Simscape Models Represent Physical Systems

In this section...
“Representations of Physical Systems” on page 4-2
“Differential, Differential-Algebraic, and Algebraic Systems” on page 4-2
“Stiffness” on page 4-3
“Events and Zero Crossings” on page 4-3
“Working with Simscape Representation” on page 4-3

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 Simscape model or its components with 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 4-5.

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.

A *zero crossing* is a specific event type, represented by the value of a mathematical function changing sign.

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: “Differential, Differential-Algebraic, and Algebraic Systems” on page 4-2.

Remember that some physical networks are represented by ODEs only.

- Physical networks may contain “Stiffness” on page 4-3.
- Identify discrete and continuous components that might “Events and Zero Crossings” on page 4-3 during a simulation.

Creating and Detecting Zero Crossings in Simscape Models

Simulink and Simscape software have specific methods for detecting and locating zero-crossing events. For general information, see “Zero-Crossing Detection” in the Simulink documentation.

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

- Simscape and normal Simulink blocks copied from their respective block libraries
- Expressions programmed in the Simscape language

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.

Enabling and Disabling Zero-Crossing Conditions in Simscape Language

In the Simscape language, you can create or avoid Simulink 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.

How Simscape Simulation Works

In this section...

“Simscape Simulation Phases” on page 4-5

“Model Validation” on page 4-7

“Network Construction” on page 4-7

“Equation Construction” on page 4-8

“Initial Conditions Computation” on page 4-8

“Transient Initialization” on page 4-9

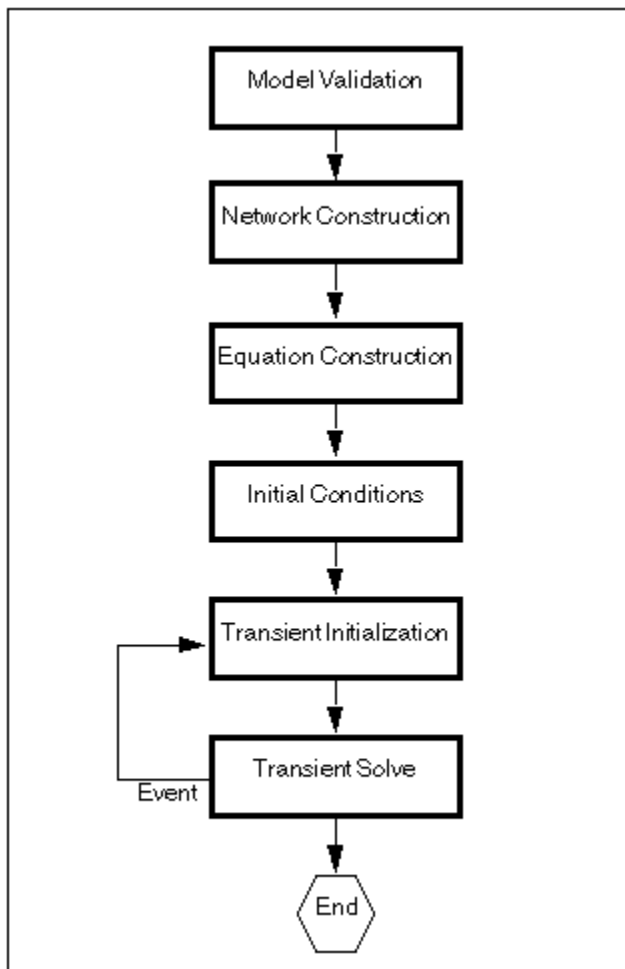
“Transient Solve” on page 4-10

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 4-2.

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 4-7
- 2 “Network Construction” on page 4-7
- 3 “Equation Construction” on page 4-8
- 4 “Initial Conditions Computation” on page 4-8
- 5 “Transient Initialization” on page 4-9

6 “Transient Solve” on page 4-10

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. Unconnected Conserving ports are not allowed.
- Each topologically distinct physical network in a diagram requires exactly one Solver Configuration block.
- If your model contains hydraulic elements, each topologically distinct hydraulic circuit in a diagram must connect to a Custom Hydraulic Fluid block (or Hydraulic Fluid block, available with SimHydraulics block libraries). These blocks define the fluid properties that act as global parameters for all the blocks that connect 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, if your model contains pneumatic elements, default gas properties for a pneumatic network are for dry air and ambient conditions of 101325 Pa and 20 degrees Celsius. If you attach a Gas Properties block to a pneumatic circuit, you can change gas properties and ambient conditions for all the blocks connected to the circuit. However, more than one Gas Properties block in a pneumatic circuit 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 output.

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 dialog box, 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 4-9.

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 on the **Variables** tab of the 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 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 block 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, 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 just described. 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. Steady state means that the system variables are no longer changing with time. 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.

Setting Up Solvers for Physical Models

In this section...

“About Simulink and Simscape Solvers” on page 4-11

“Choosing Simulink and Simscape Solvers” on page 4-11

“Harmonizing Simulink and Simscape Solvers” on page 4-13

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 4-21.

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 4-13.

- “Working with Global Simulink Solvers” on page 4-11
- “Working with Local Simscape Solvers” on page 4-12

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 “Solvers” in the Simulink documentation.

When you first create a model, the default Simulink solver is ode45. To select a different solver, follow a procedure similar to the procedure in “Modifying Initial Settings” on page 1-26.

- You can choose one from a suite of both variable-step and fixed-step solvers. A variable-step solver is the default.
- You can also select from among explicit and implicit solvers. An explicit solver is the default. But for physical models, MathWorks recommends implicit solvers, such as

ode14x, 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 4-14.

- 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 “Enabling or Disabling Simulink Zero-Crossing Detection” on page 4-16.

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 or Trapezoidal Rule) 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. Fixed-cost simulation requires a global fixed-step solver.

Choosing Multirate Simulation

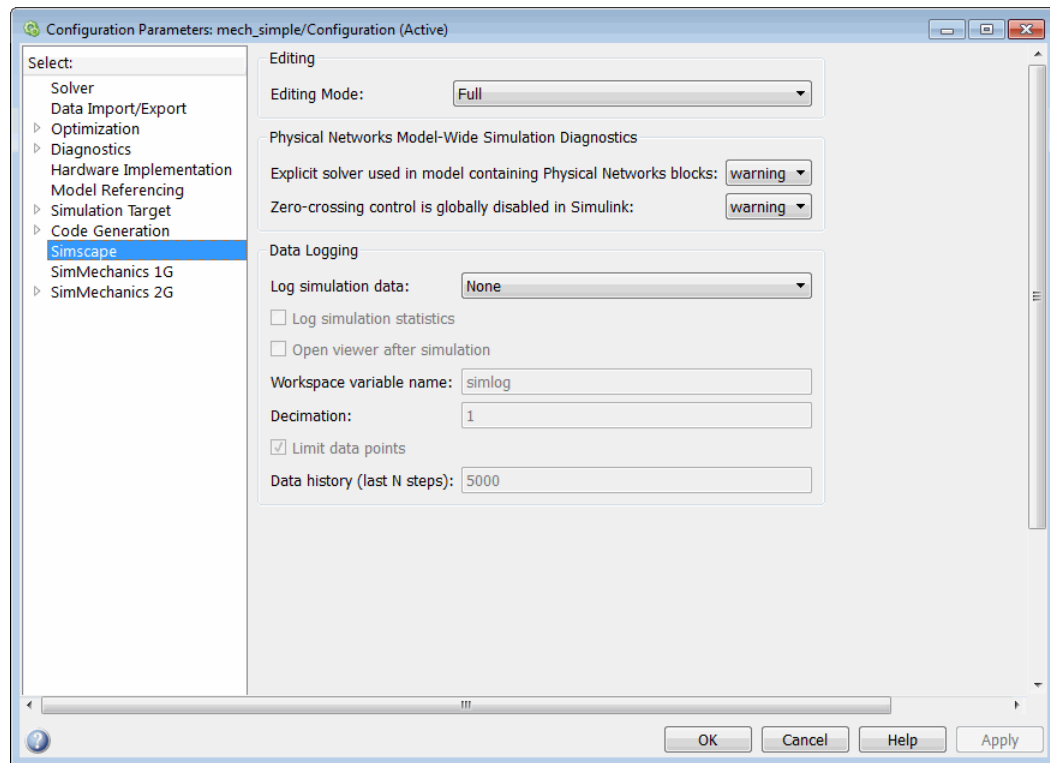
With the local solver option, you can perform multirate simulations, with:

- 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. From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. Review and adjust the following settings.

- “Switching from the Default Explicit Solver to Other Simulink Solvers” on page 4-14
- “Filtering Input Signals and Providing Time Derivatives” on page 4-14
- “Enabling or Disabling Simulink Zero-Crossing Detection” on page 4-16
- “Making Multirate Simulation Consistent” on page 4-16



Simscape Pane of the Configuration Parameters Dialog Box

Switching from the Default Explicit Solver to Other Simulink Solvers

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 4-17.

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**.

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.

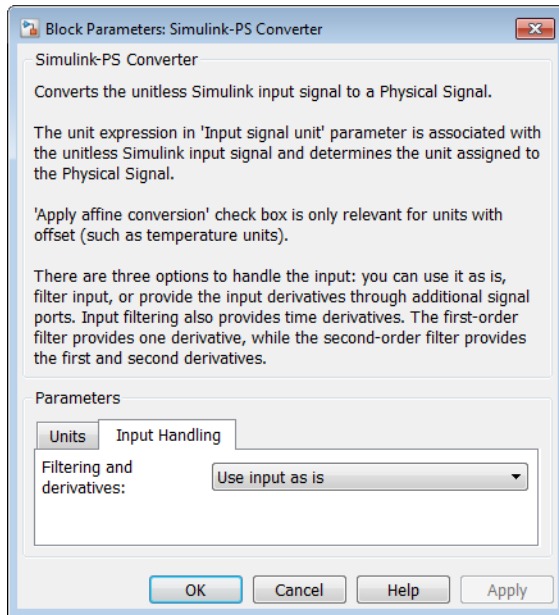
The filter time constant controls the filtering of the input signal. The filtered input follows the true input but is smoothed, with a lag on the order of the time constant that

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 physical signals.

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 Click the **Input handling** tab.



- 3 To turn on input filtering, set the **Filtering and derivatives** parameter to **Filter input**. Select the first-order or second-order filter, by using the **Input filtering order** parameter, and set the appropriate **Input filtering time constant** parameter value for your model.
- 4 To avoid filtering the input signal, set the **Filtering and derivatives** parameter to **Provide input derivative(s)**. Then set the **Input derivatives** parameter value:

- **Provide 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 derivatives.
- **Provide first and second 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.

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 “Working with Simscape Representation” on page 4-3.

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

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 4-21. For background information, consult “How Simscape Models Represent Physical Systems” on page 4-2 and “How Simscape Simulation Works” on page 4-5.

In this section...

“Variable-Step and Fixed-Step Solvers” on page 4-17

“Explicit and Implicit Solvers” on page 4-18

“Full and Sparse Linear Algebra” on page 4-18

“Event Detection and Location” on page 4-18

“Unbounded, Bounded, and Fixed-Cost Simulation” on page 4-19

“Global and Local Solvers” on page 4-19

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 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

For the key simulation concepts to consider before making these choices, see “Important Concepts and Choices in Physical Simulation” on page 4-17.

In this section...

“Simulating with Variable Time Step” on page 4-21

“Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers” on page 4-21

“Simulating with Fixed Cost” on page 4-22

“Troubleshooting and Improving Solver Performance” on page 4-23

“Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System” on page 4-24

Simulating with Variable Time Step

For a typical Simscape model, MathWorks recommends the Simulink variable-step solvers `ode15s` and `ode23t`. Of these two global 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 accurate and efficient.

Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers

In a Simscape model, MathWorks recommends 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 Backward Euler and Trapezoidal Rule. Of these two local solvers:

- 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.

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 `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 any non-Simscape parts of your model have only discrete states, then the global solver effectively sees only discrete states. In that case, MathWorks recommends a discrete, fixed-step global solver. If you are attempting a fixed-cost simulation with discrete states, you must use a discrete, fixed-step global solver.

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 4-19 and “Fixed-Cost Simulation for Real-Time Viability” on page 7-55.

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.

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 4-2.) 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.

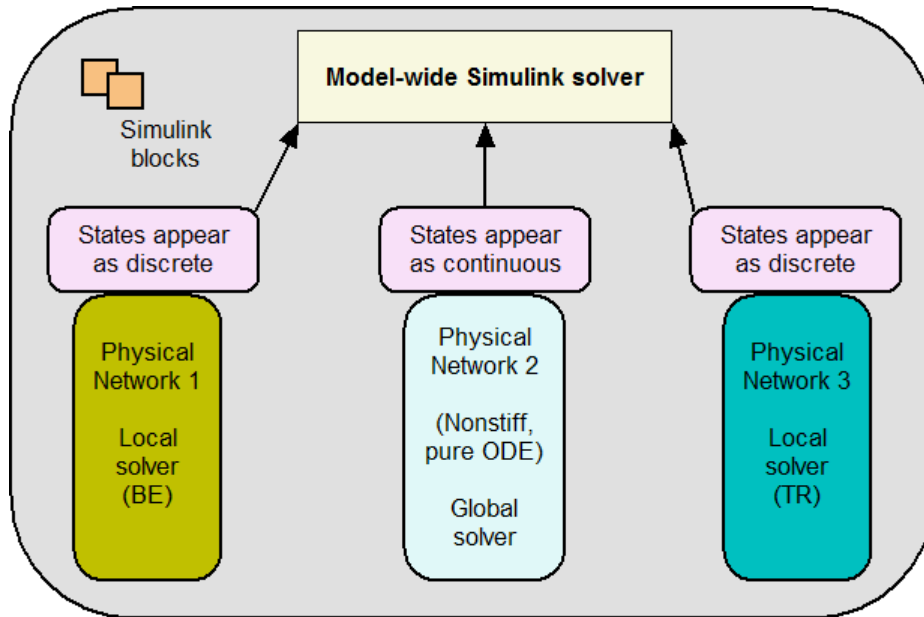
Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System

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.



Troubleshooting Simulation Errors

In this section...
“Troubleshooting Tips and Techniques” on page 4-26
“System Configuration Errors” on page 4-27
“Numerical Simulation Issues” on page 4-29
“Initial Conditions Solve Failure” on page 4-30
“Transient Simulation Issues” on page 4-30

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 4-5, 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.

MathWorks recommends 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 4-27
- “Extra Fluid Block or Gas Properties Block” on page 4-27
- “Missing Reference Block” on page 4-28
- “Basic Errors in Physical System Representation” on page 4-28

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 Block 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 SimHydraulics 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 Gas Properties block in a pneumatic 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, or Gas Properties 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 1-36.

Basic Errors in Physical System Representation

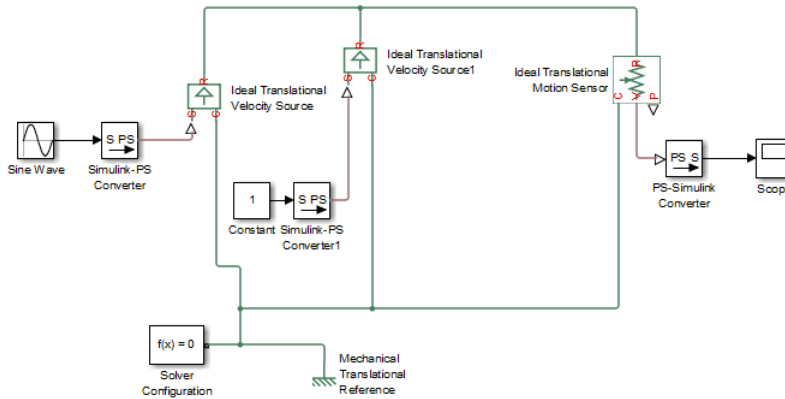
Physical systems are represented in the Simscape modeling environment as Physical Networks according to the Kirchhoff’s generalized circuit laws. Certain model configurations violate these laws and are therefore illegal. There are two broad violations:

- 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 4-29
- “Parameter Discontinuities” on page 4-30

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 4-30).

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 4-27 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 4-29 for more information.
- The constraint 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 **Constraint Residual 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 4-26.

Transient Simulation Issues

- “Transient Initialization Not Converging” on page 4-31
- “Step-Size-Related Errors — Dependent States — High Stiffness” on page 4-31

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 4-26.

You can also try to decrease the **Constraint Residual 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 **Constraint Residual 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 4-29.

Limitations

In this section...
“Sample Time and Solver Restrictions” on page 4-32
“Algebraic Loops” on page 4-32
“Restricted Simulink Tools” on page 4-33
“Unsupported Simulink Tools and Features” on page 4-35
“Simulink Tools Not Compatible with Simscape Blocks” on page 4-35
“Code Generation” on page 4-35

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.

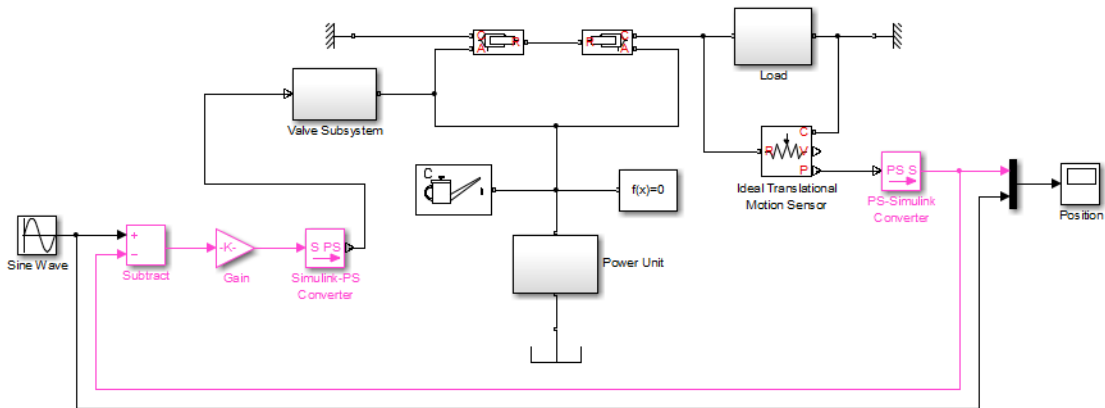
If you switch to a local solver in the **Solver Configuration** block, the states of the associated physical network become discrete. If there are no continuous Simulink or Simscape states anywhere in a model, you are free to use a discrete solver to simulate the model.

You cannot override the sample time of a nonvirtual subsystem containing Simscape blocks.

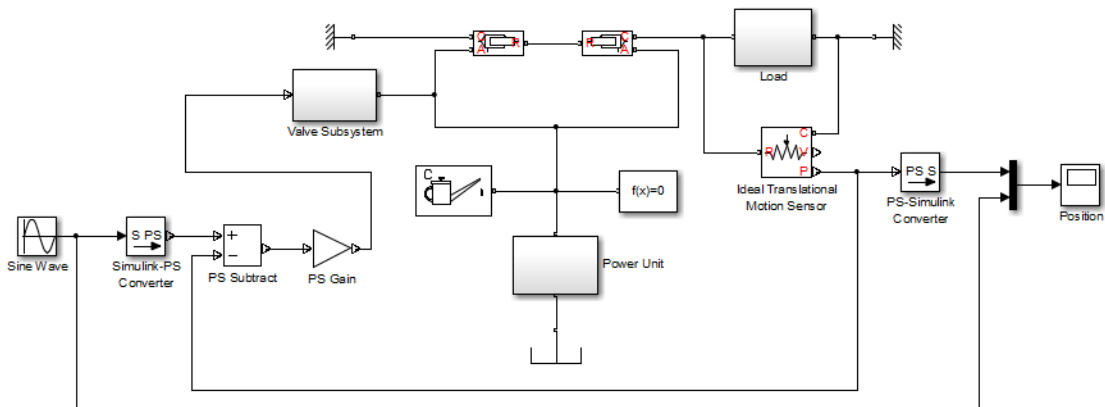
Algebraic Loops

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.



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. MathWorks® does not recommend 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 choices have consistent port signatures.
- When using `SimState` to save and restore simulations of models, you cannot make any changes to the Simscape blocks in the model between the time at which you save the `SimState` and the time at which you restore the simulation using the `SimState`.

This is an extension of the Simulink limitation prohibiting structural changes to the model between these two points in time (see “Limitations of `SimState`”). Changes to Simscape block parameters can cause equation changes and result in changes to the state representation. Therefore, modifying parameters of Simscape blocks between saving and restoring the `SimState` is not allowed.

- 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.
- 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.

Unsupported Simulink Tools and Features

Certain Simulink tools and features do not work with Simscape software:

- The Simulink Profiler tool does not work with Simscape models.
- Exporting a model to a format used by an earlier version (**File > Export Model to > Previous Version**) is not supported for models containing Simscape blocks.
- Physical signals and physical connection lines between conserving ports are different from Simulink signals. Therefore, the Signal and Scope Manager tool and the signal label functionality are not supported.
- Simscape block names cannot contain cross-locale non-ASCII characters.

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.

- Code reuse is not supported.
- Encapsulated C++ code generation is not supported.
- Tunable parameters are not supported.
- Run-time parameter inlining ignores global exceptions.
- 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 4-33 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 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.

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

Variable Initialization and State Viewer

- “About Variable Initialization” on page 5-2
- “Set Priority and Initial Target for Block Variables” on page 5-5
- “Initialize Variables for a Mass-Spring-Damper System” on page 5-7
- “Variable Viewer” on page 5-23

About Variable Initialization

In this section...

“Initializing Block Variables for Model Simulation” on page 5-2

“Variable Initialization Priority” on page 5-3

“Suggested Workflow” on page 5-4

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 4-8. 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 on the **Variables** tab 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 5-23.

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 **Variables** tab of the block dialog box, selecting the **Override** 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 5-7.

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 5-5.

Suggested Workflow

- 1 Using the **Variables** tab of the respective block dialog boxes, specify the variable targets for initialization, by setting the priority, target values, and units for block variables as required by your model.
- 2 Open 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.

Related Examples

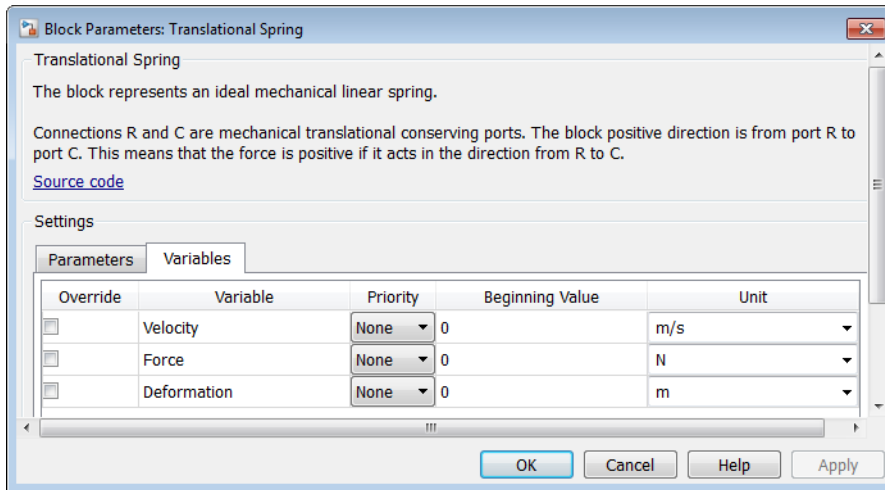
- “Set Priority and Initial Target for Block Variables” on page 5-5
- “Initialize Variables for a Mass-Spring-Damper System” on page 5-7

More About

- “Variable Viewer” on page 5-23

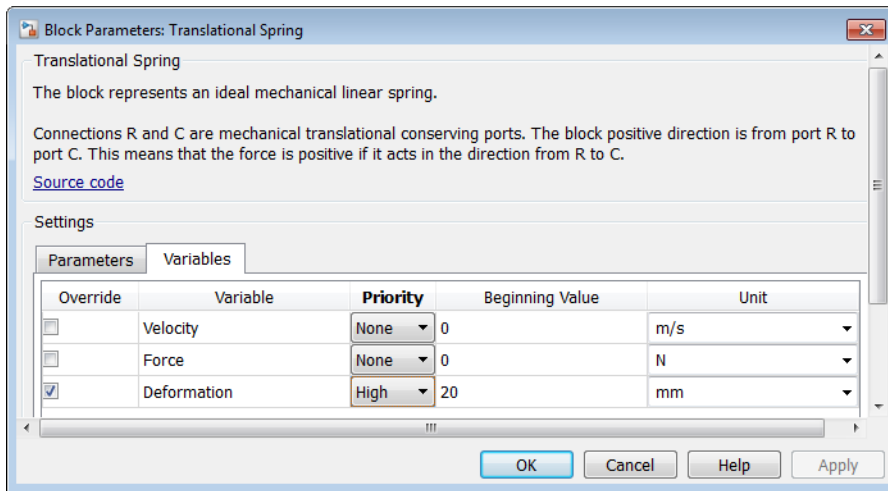
Set Priority and Initial Target for Block Variables

When you open the **Variables** tab of a 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 click the **Variables** tab, it looks like this:



For details on these variables and their usage in the block equations, click the **Source code** link in the block dialog box to view the underlying Simscape source file. The **Source code** link is available for all the Foundation library blocks that have a **Variables** tab.

To specify the initial deformation of the spring, select the **Override** 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 **Beginning 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 **Override** check box next to a variable name, its **Priority**, **Beginning Value**, and **Unit** 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.

Related Examples

- “Initialize Variables for a Mass-Spring-Damper System” on page 5-7

More About

- “About Variable Initialization” on page 5-2

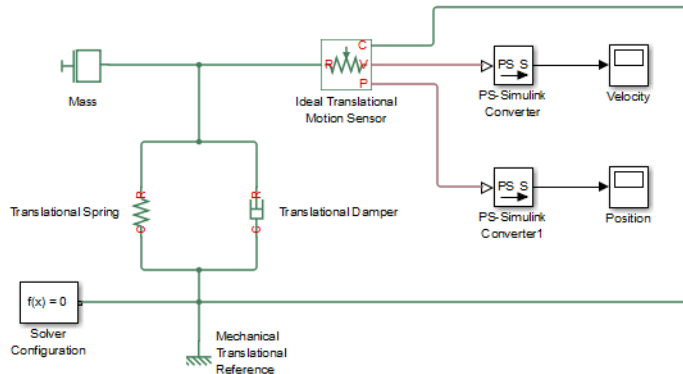
Initialize Variables for a Mass-Spring-Damper System

This example shows how to use block variable initialization, and how it affects the simulation results of a simple mechanical system.

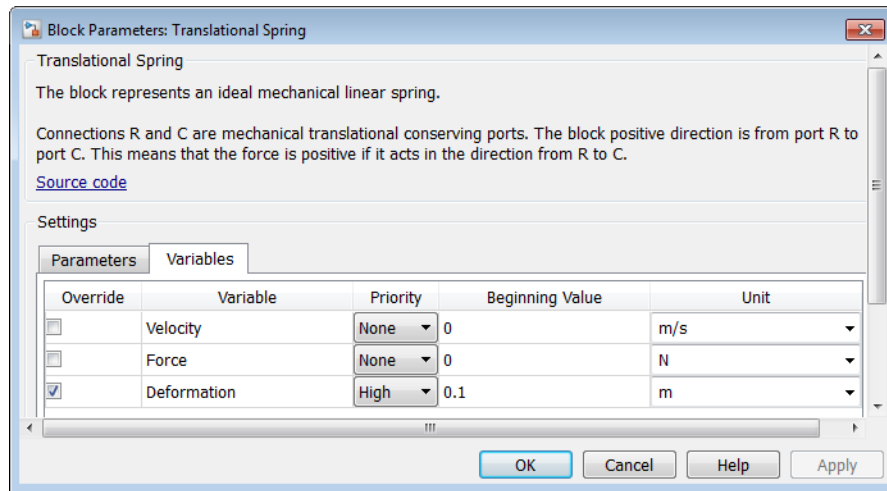
The model is a classical unforced mass-spring-damper system, with the oscillations of the mass caused by the initial deformation of the spring.

Create and Set Up the Model

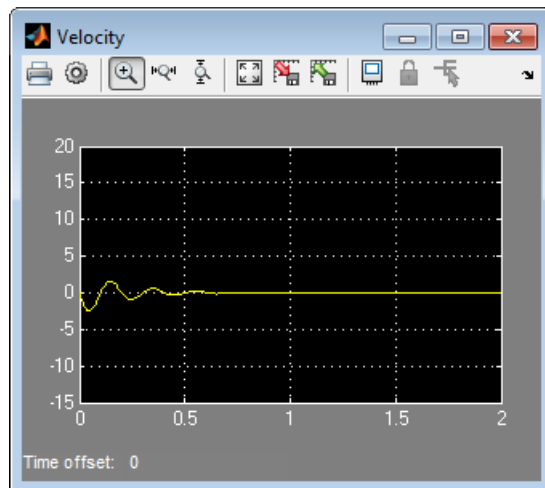
- 1 Create a simple mass-spring-damper system. Use the Mass, Translational Spring, Translational Damper, Mechanical Translational Reference, Ideal Translational Motion Sensor, PS-Simulink Converter, Solver Configuration, and Scope blocks, and connect them as shown in the following illustration.

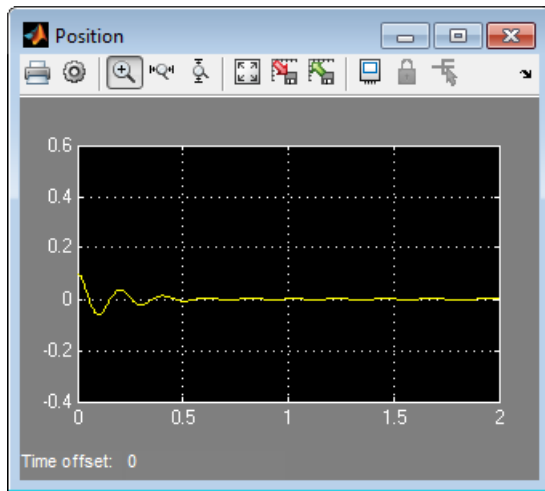


- 2 Prepare the model for simulation. On the top menu bar of the model window, select **Simulation > Model Configuration Parameters**. Under **Solver options**, set **Solver** to `ode23t (mod.stiff/Trapezoidal)` and **Max step size** to `0.2`. Also adjust the **Simulation time** to be between 0 and 2 seconds, by setting **Stop time** to `2.0`.
- 3 Specify the initial deformation of the spring. Double-click the Translational Spring block. In the block dialog box, click the **Variables** tab, and then select the check box next to the **Deformation** variable. Change its **Priority** to **High**. Change the **Beginning Value** to `0.1`. Leave the **Unit** unchanged as `m`.



- 4 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.
- 5 Simulate the model.





- 6 Open the Variable Viewer. In the top menu bar of the model window, select **Analysis > Simscape > Variable Viewer**.

Name	Status	Priority	Target	Start	Unit
Ideal_Translational_Motion_Sensor	●				
C	●				
v	●			0.0	m/s
p	●			0.1	m
R	●				
v	●			0.0	m/s
V	●			0.0	m/s
f	●			0.0	N
v	●			0.0	m/s
x	●	High	0.1	0.1	m
Mass	●				
M	●				
v	●			0.0	m/s
f	●			-100.0	N
v	●			0.0	m/s
Mechanical_Translational_Reference	●				
V	●				
v	●			0.0	m/s
f	●			100.0	N
Translational_Damper	●				
C	●				
v	●			0.0	m/s
R	●				
v	●			0.0	m/s
f	●			0.0	N
v	●			0.0	m/s
Translational_Spring	●				
C	●				
v	●			0.0	m/s
R	●				
v	●			0.0	m/s
f	●			100.0	N
v	●			0.0	m/s
x	●	High	0.1	0.1	m

● All targets satisfied Variables at start ▼

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.

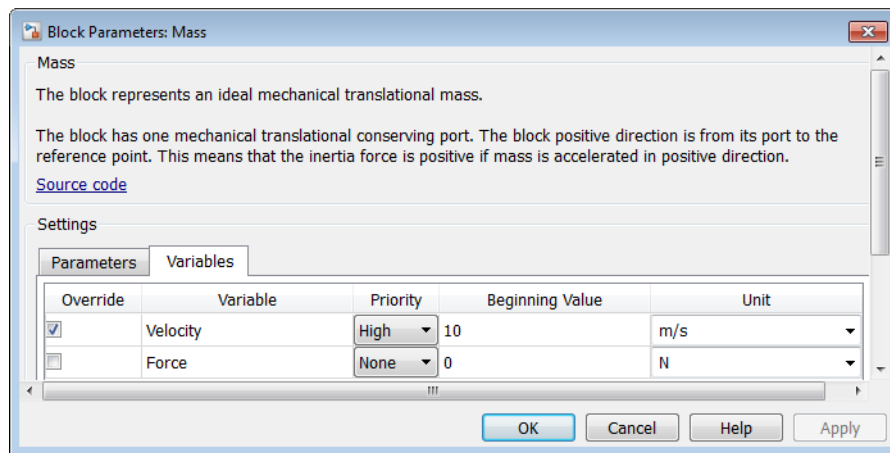
The other high-priority variable in this model is the position, x , of the Ideal Translational Motion Sensor block, which is set inside the component file because it is necessary for the correct operation of the sensor. Its actual start value also matches its target value, and its **Status** column also 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 variable targets are satisfied.

Change Initialization Targets

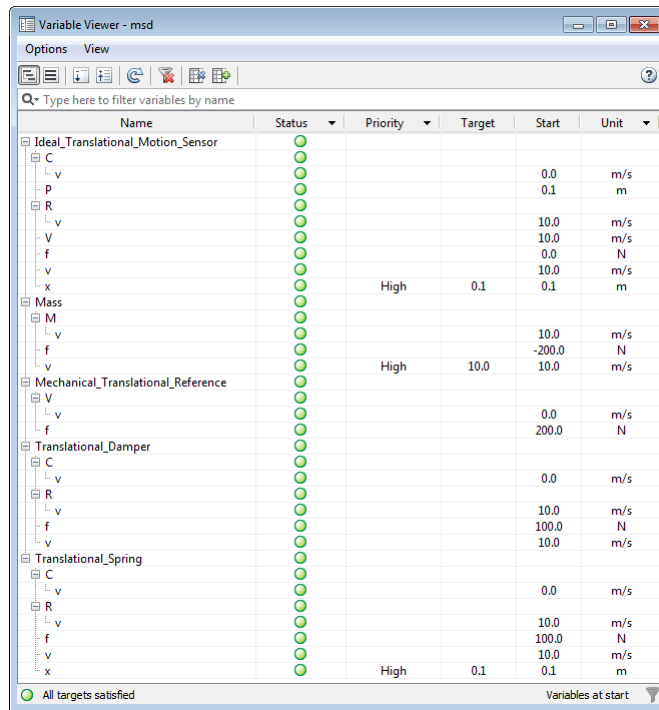
You can now see how specifying different variable targets affects system initialization and simulation results.

- 1 Specify the initial velocity of the mass. Double-click the Mass block, go to the **Variables** tab, select the check box next to the **Velocity** variable, change its **Priority** to **High**, and enter a beginning value of **10**. Keep the unit **m/s**.



- 2 Refresh the Variable Viewer by clicking .

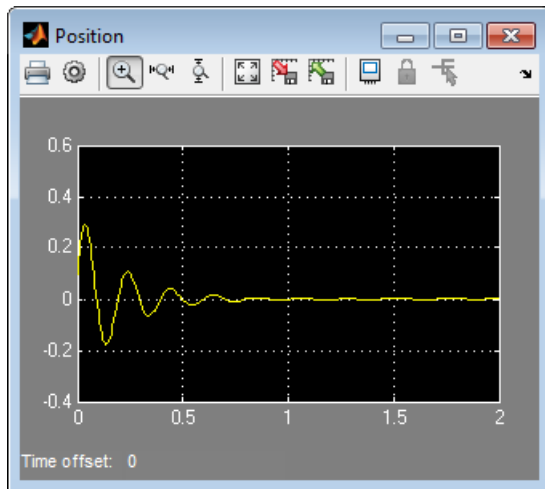
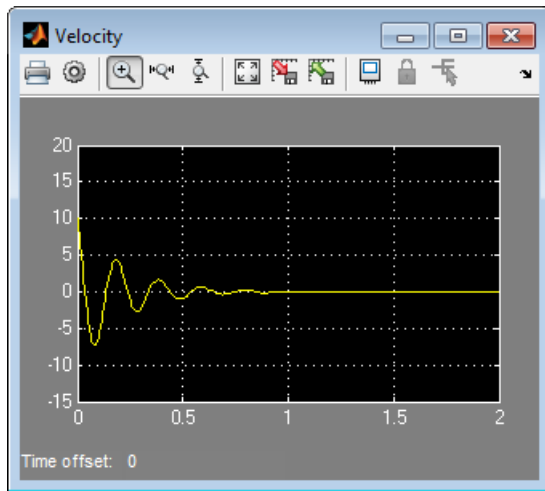
5 Variable Initialization and State Viewer



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 variable 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.

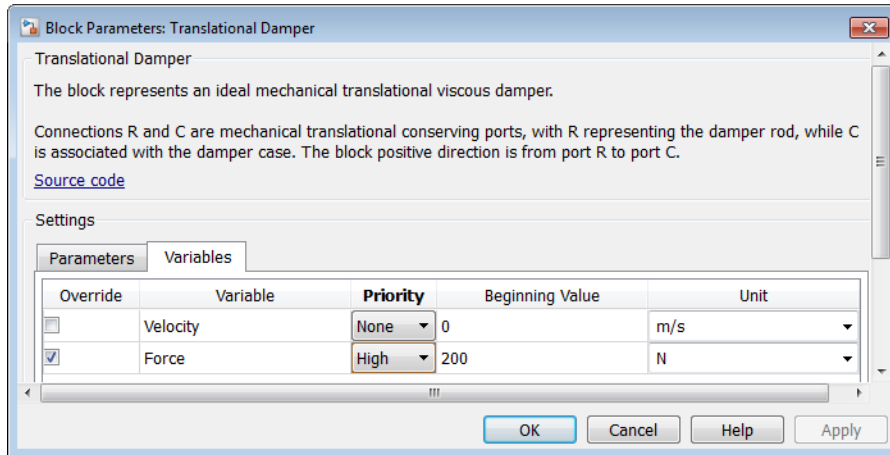
Rerun the simulation and examine the Velocity and Position scope windows, to see the effect of the new initial value for mass velocity on the simulation results.



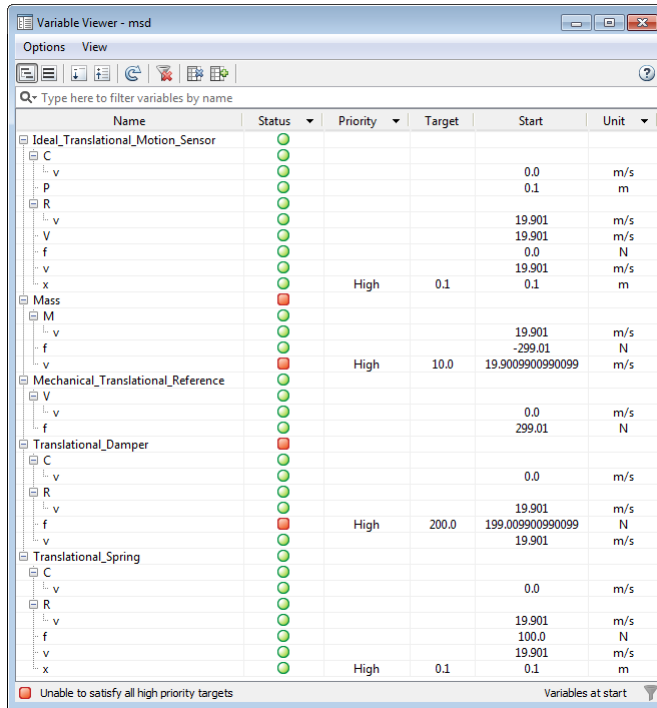
Deal with Over-Specification

As you specify additional variable targets, sometimes it is possible to over-specify the constraints.

- 1 Double-click the Translational Damper block, go to the **Variables** tab, select the check box next to the **Force** variable, change its **Priority** to **High**, and enter a beginning value of 200. Keep the unit N.

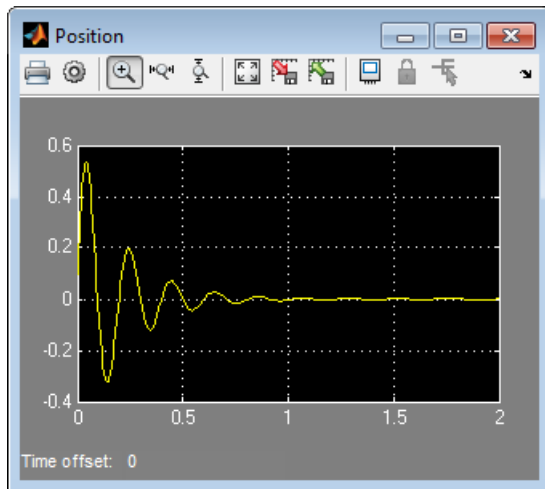
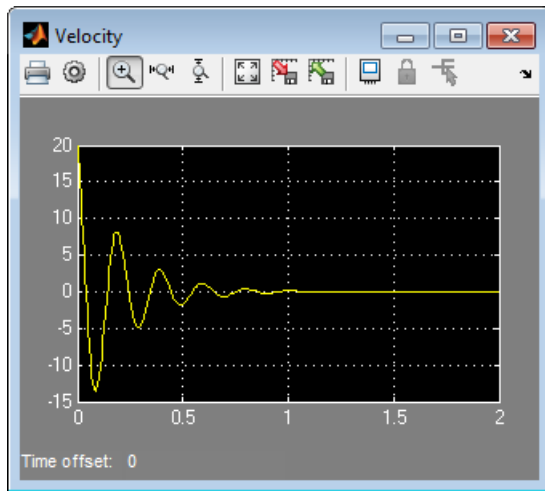


- 2 Refresh the Variable Viewer.



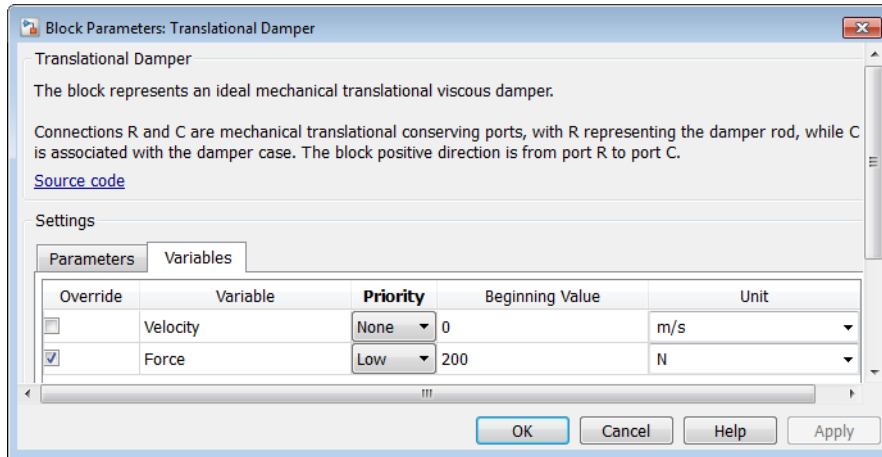
The overall status at the bottom of the Variable Viewer window now displays a red square and says that the solver is unable to satisfy all the high-priority variable targets. There are red squares in the **Status** column for the two high-priority variables with targets not satisfied, as well as for their parent blocks.

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, go to the **Variables** tab, and change the priority of the **Force** variable to **LOW**.



- 4 Refresh the Variable Viewer.

5 Variable Initialization and State Viewer

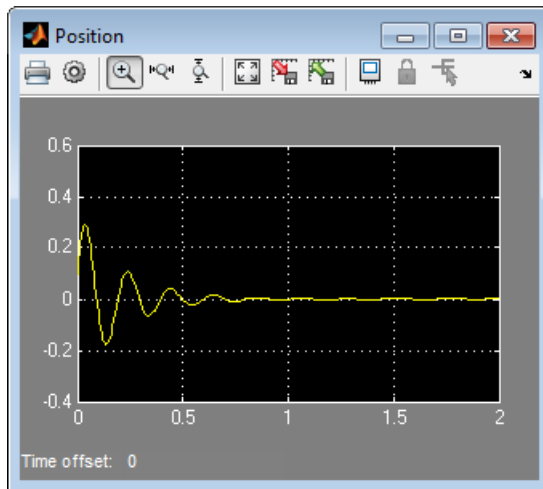
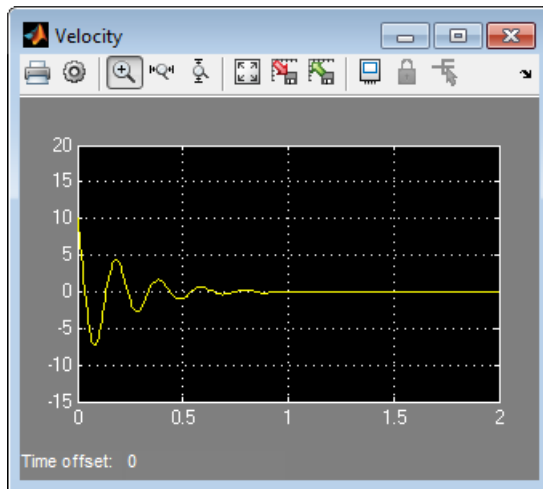
Name	Status	Priority	Target	Start	Unit
Ideal_Translational_Motion_Sensor	●				
C	●				
v	●			0.0	m/s
P	●			0.1	m
R	●				
v	●			10.0	m/s
V	●			10.0	m/s
f	●			0.0	N
v	●			10.0	m/s
x	●	High	0.1	0.1	m
Mass	●				
M	●				
v	●			10.0	m/s
f	●	High	10.0	-200.0	N
v	●			10.0	m/s
Mechanical_Translational_Reference	●				
V	●				
v	●			0.0	m/s
f	●			200.0	N
Translational_Damper	▲				
C	●				
v	●			0.0	m/s
R	●				
v	●			10.0	m/s
f	▲	Low	200.0	100.0	N
v	●			10.0	m/s
Translational_Spring	●				
C	●				
v	●			0.0	m/s
R	●				
v	●			10.0	m/s
f	●			100.0	N
v	●			10.0	m/s
x	●	High	0.1	0.1	m

▲ All high priority targets satisfied but some low priority targets not satisfied

Variables at start ▼

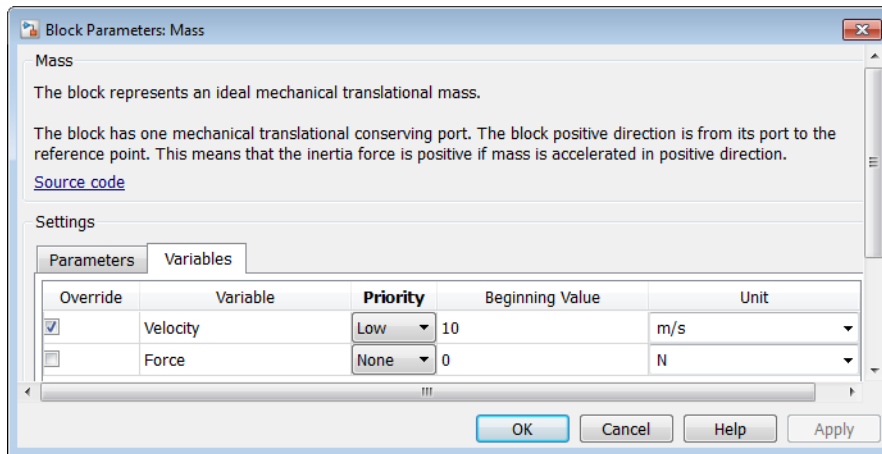
The overall status at the bottom of the Variable Viewer window now displays a yellow triangle and says that all the high-priority targets are satisfied, but some of the low-priority targets are not satisfied. There are now two yellow triangles in the status column: one for the low-priority force variable f and one for its parent block, Translational Damper.

Essentially, the solution found in this case is the same as when you previously specified high-priority target for the mass velocity, and the simulation results are the same.

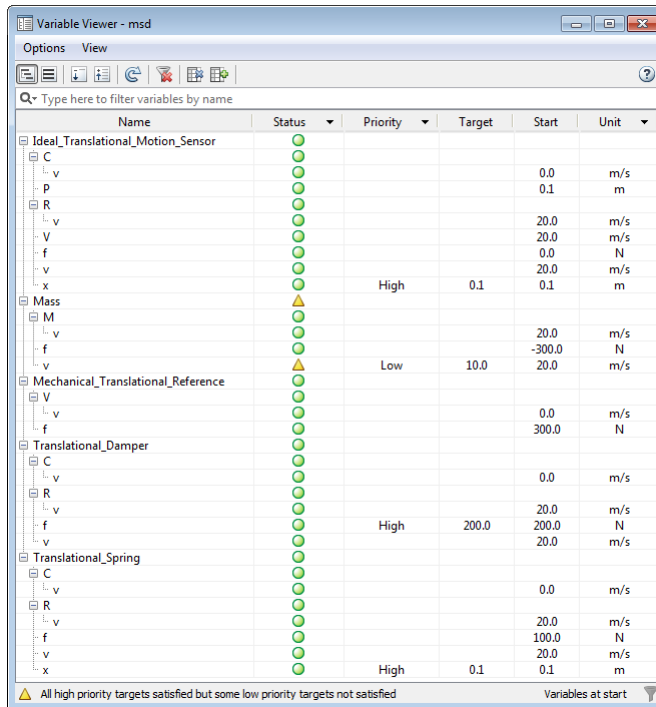


- 5 Another way to deal with over-specification is to keep the high priority on the damper force and relax the priority on mass initial velocity. Double-click the Translational Damper block again, go to the **Variables** tab, and change the priority of the **Force** variable back to **High**. Then double-click the Mass block, go to the **Variables** tab, and change the priority of the **Velocity** variable to **Low**.

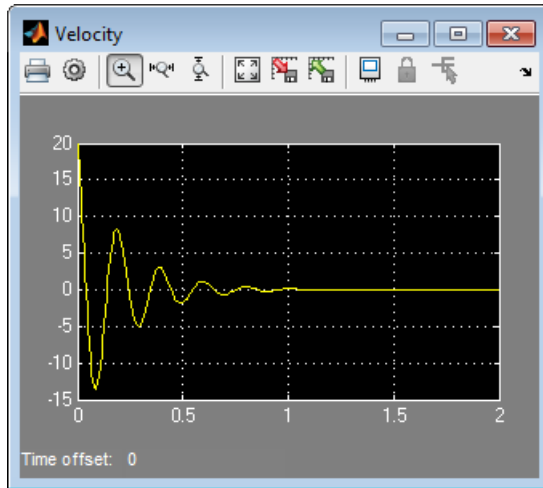
5 Variable Initialization and State Viewer

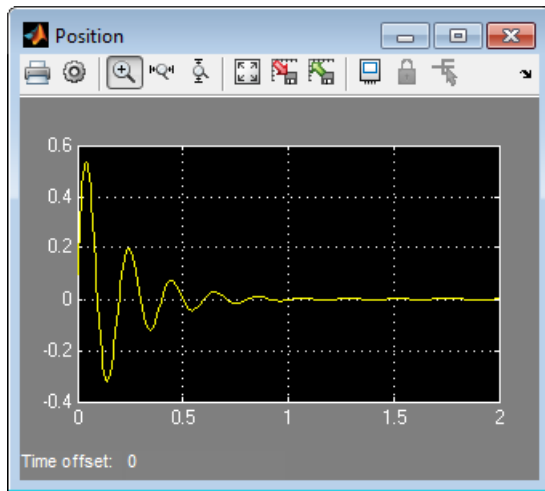


6 Refresh the Variable Viewer.



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.





Related Examples

- “Set Priority and Initial Target for Block Variables” on page 5-5

More About

- “About Variable Initialization” on page 5-2
- “Variable Viewer” on page 5-23

Variable Viewer

In this section...

“About Variable Viewer” on page 5-23

“Advanced Configuration” on page 5-25

“Switching Between Tree View and Flat View” on page 5-27

“Useful Filtering Techniques” on page 5-29

“Link to Block Diagram” on page 5-30

“Interaction with Model Updates and Simulation” on page 5-31

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 top menu bar of the model window, select **Analysis > Simscape > Variable Viewer**.

Name	Status	Priority	Target	Start	Unit
DC_Motor	●				
Friction	●				
C	●				
w	●			0.0	rad/s
R	●				
w	●			0.0	rad/s
t	●			0.0	N*m
w	●			0.0	rad/s
Inertia	●				
I	●				
w	●			0.0	rad/s
t	●			1.03132E-12	N*m
w	●	High	0.0	0.0	rad/s
Rotational_Electromechanical_Converter	●				
C	●				

● All targets satisfied









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, which has the following columns:

Name	Description
Status	Initialization status of each variable, can be one of: <ul style="list-style-type: none"> • Green circle — Displayed for variables with initialization targets satisfied, and also for all variables with no initialization priority. • Yellow triangle — Displayed for low-priority variables if the target is not satisfied. • Red square — Displayed for high-priority variables if the target is not satisfied. • Red cross — If initial condition solve fails, displayed for variables that could not be initialized. • 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 5-31.
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 5-5 and “Variable Priority for Model Initialization”. If the variable has no initialization priority (<code>None</code> or <code>priority.none</code>), then this field is empty.
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.


A downward-pointing arrow next to a column name indicates that you can filter the table rows based on their value in this column. For more information on the filtering options, see “Useful Filtering Techniques” on page 5-29.

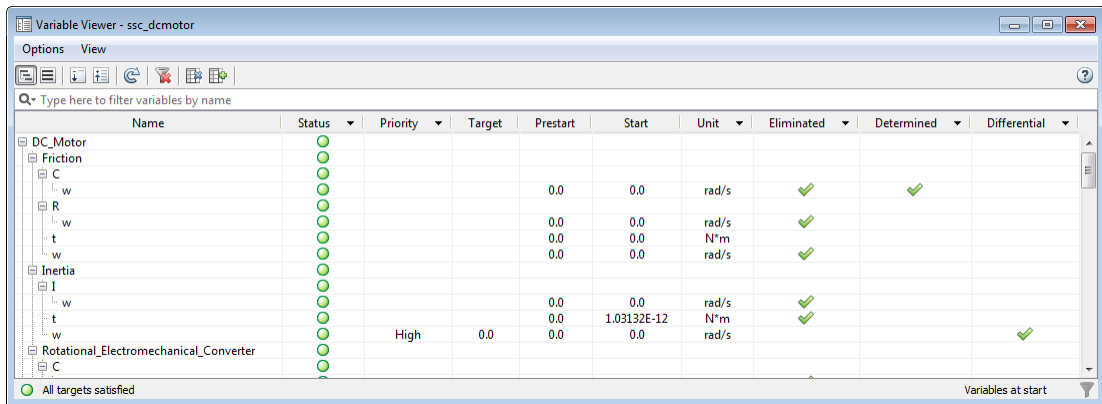
The Variable Viewer toolbar buttons perform the following actions:

-  Displays the data in the Variable Viewer in tree view, with variable nodes grouped under the parent port, block, and subsystem nodes. This is the default view.
-  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.
-  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. For more information, see “Interaction with Model Updates and Simulation” on page 5-31.
-  Clears all the column filtering options and displays all the rows in the table. For more information, see “Useful Filtering Techniques” on page 5-29.
-  Shows the Variable Viewer in its default, basic, configuration, with only the following columns displayed: **Status**, **Priority**, **Target**, **Start**, and **Unit**.
-  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 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  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

Name	Description
	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.

You can change the default order of columns by clicking a column heading and dragging it, while holding down the mouse button, to the desired location. You can also hide columns by right-clicking their headers and selecting **Hide This Column** from the

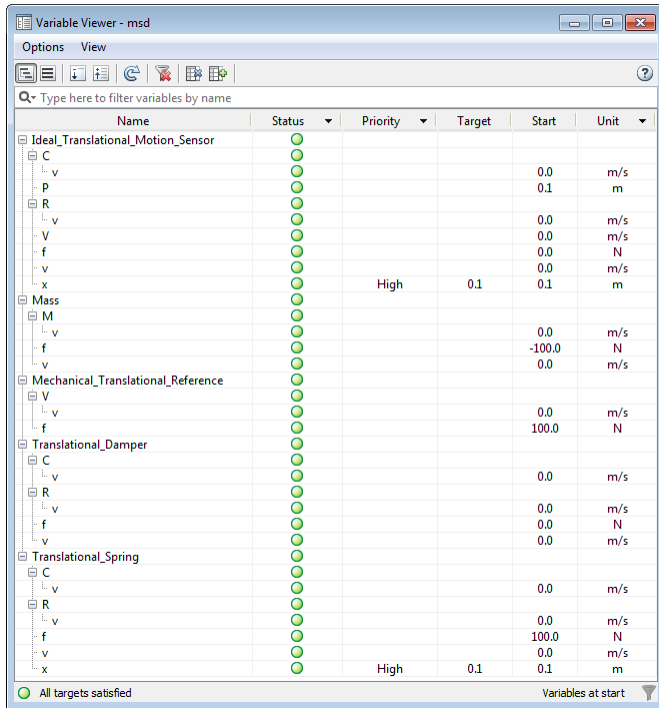
context menu, or clearing the check mark next to a column name. Clicking  or  in the Variable Viewer toolbar restores the default basic or advanced layout, respectively.

Switching Between Tree View and Flat View

You can control the number of rows in the Variable Viewer by switching between the tree view (the default) and the flat view. By default, the Variable Viewer opens in tree view, with variable nodes 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).

5 Variable Initialization and State Viewer



The screenshot shows the 'Variable Viewer - msd' window. It features a toolbar with icons for options and view, and a search bar. The main area displays a table of variables with columns for Name, Status, Priority, Target, Start, and Unit. The variables are organized into a tree structure under various component names. The Status column shows green circles, indicating that all targets are satisfied. The Priority column shows 'High' for some variables, and the Target column shows values like 0.1. The Start column shows values like 0.0 and 100.0. The Unit column shows units like m/s and m.

Name	Status	Priority	Target	Start	Unit
Ideal_Translational_Motion_Sensor	●				
C	●				
v	●			0.0	m/s
P	●			0.1	m
R	●				
v	●			0.0	m/s
V	●			0.0	m/s
f	●			0.0	N
v	●			0.0	m/s
x	●	High	0.1	0.1	m
M	●				
v	●			0.0	m/s
f	●			-100.0	N
v	●			0.0	m/s
Mechanical_Translational_Reference	●				
V	●				
v	●			0.0	m/s
f	●			100.0	N
Translational_Damper	●				
C	●				
v	●			0.0	m/s
R	●				
v	●			0.0	m/s
f	●			0.0	N
v	●			0.0	m/s
Translational_Spring	●				
C	●				
v	●			0.0	m/s
R	●				
v	●			0.0	m/s
f	●			100.0	N
v	●			0.0	m/s
x	●	High	0.1	0.1	m

● All targets satisfied Variables at start

To switch to the flat view, click  in the Variable Viewer toolbar.

Name	Status	Priority	Target	Start	Unit
Ideal_Translational_Motion_Sensor-->C-->v	●			0.0	m/s
Ideal_Translational_Motion_Sensor-->P	●			0.1	m
Ideal_Translational_Motion_Sensor-->R-->v	●			0.0	m/s
Ideal_Translational_Motion_Sensor-->V	●			0.0	m/s
Ideal_Translational_Motion_Sensor-->f	●			0.0	N
Ideal_Translational_Motion_Sensor-->v	●			0.0	m/s
Ideal_Translational_Motion_Sensor-->x	●	High	0.1	0.1	m
Mass-->M-->v	●			0.0	m/s
Mass-->f	●			-100.0	N
Mass-->v	●			0.0	m/s
Mechanical_Translational_Reference-->V-->v	●			0.0	m/s
Mechanical_Translational_Reference-->f	●			100.0	N
Translational_Damper-->C-->v	●			0.0	m/s
Translational_Damper-->R-->v	●			0.0	m/s
Translational_Damper-->f	●			0.0	N
Translational_Damper-->v	●			0.0	m/s
Translational_Spring-->C-->v	●			0.0	m/s
Translational_Spring-->R-->v	●			0.0	m/s
Translational_Spring-->f	●			100.0	N
Translational_Spring-->v	●			0.0	m/s
Translational_Spring-->x	●	High	0.1	0.1	m

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.


If the Variable Viewer is in flat view, the buttons that expand and collapse nodes are disabled.

To switch back to the tree view, click  in the Variable Viewer toolbar.

Useful Filtering Techniques

A downward-pointing arrow next to a column name indicates that you can filter the table rows based on their value in this column.

To filter the rows, click the arrow, and then select or clear the check boxes in the drop-down list to indicate which rows you want to be displayed, based on their value. Selecting

All clears all the filters for that column. To clear all filters for all columns, click  in the Variable Viewer toolbar.

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.

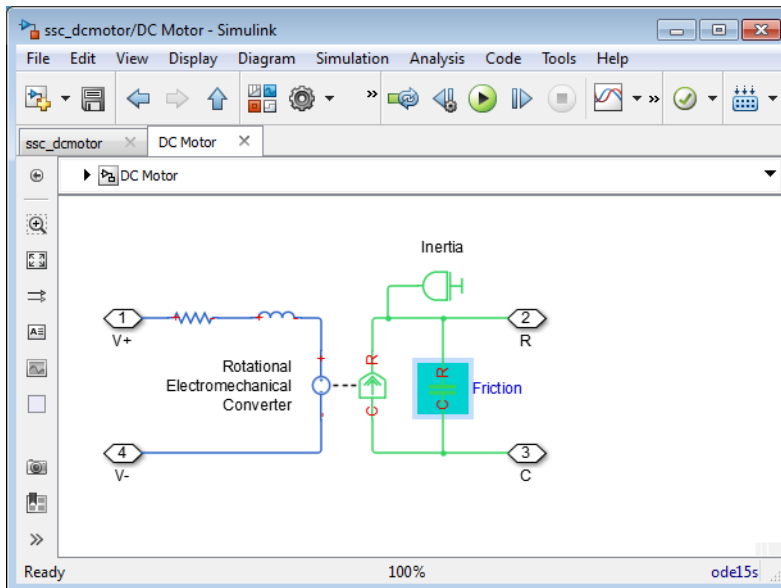
Link to Block Diagram

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, click the **Variables** tab to view or modify the variable priorities and targets. If the selected row represents a subsystem, this option is not available.

Name	Status	Priority	Target	Start	Unit
DC_Motor	●				
Friction	●				
C	●			0.0	rad/s
w	●			0.0	rad/s
R	●			0.0	N*m
t	●			0.0	rad/s
w	●			0.0	rad/s
Inertia	●				
I	●			0.0	rad/s
w	●			1.03132E-12	N*m
t	●			0.0	rad/s
w	●			0.0	rad/s
Rotational_Electromechanical_Converter	●	High	0.0		
C	●				




Interaction with Model Updates and Simulation

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 open or refresh the viewer, otherwise you get an error message.

- If the scopes are open, they turn blank every time you open or 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 while the Variable Viewer is open, the results in the viewer are not updated automatically.

Refresh the viewer (by clicking  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 **Simulation > Update Diagram** in the top menu bar of 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.

Related Examples

- “Initialize Variables for a Mass-Spring-Damper System” on page 5-7

More About

- “About Variable Initialization” on page 5-2

Linearization and Trimming

- “Finding an Operating Point” on page 6-2
- “Linearizing at an Operating Point” on page 6-7
- “Linearize an Electronic Circuit” on page 6-13
- “Linearize a Plant Model for Use in Feedback Control Design” on page 6-23

Finding an Operating Point

In this section...

“What Is an Operating Point?” on page 6-2

“Finding Operating Points in Physical Models” on page 6-3

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 \mathbf{x} and inputs \mathbf{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, \mathbf{y} , in addition to inputs.

See “Linearizing at an Operating Point” on page 6-7.

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, MathWorks recommends Simulink Control Design software.

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 \mathbf{x}_0 and input vector \mathbf{u}_0 .

- “Simulating in Time to Search for an Operating Point” on page 6-3
- “Using the Simscape Initial Condition Solver” on page 6-4
- “Using Simulink Control Design Techniques to Find Operating Points” on page 6-4
- “Using Sources to Find Operating Points Not Recommended” on page 6-6
- “Simulink `trim` Function Not Supported with Simscape Models” on page 6-6

Simulating in Time to Search for an Operating Point

One way to identify operating points is to simulate your model and inspect its state \mathbf{x} and output \mathbf{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 block dialog box, 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 \mathbf{x} values in your workspace.

If you also have input signals \mathbf{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 simulation.

The first vector of values $\mathbf{x}(t=0)$ that you capture during simulation reflects the steady state \mathbf{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 4-8.

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 $\mathbf{x}(t=0)$.

Using Simulink Control Design Techniques to Find Operating Points

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. Simulink Control Design offers both command line and graphical interfaces for finding and analyzing operating points.

Simulink Control Design methods are state-based, giving you full access to state names and values, and allow you to impose operating specifications or use simulation snapshots. MathWorks does not recommend imposing operating specifications state-by-state using the Simulink Control Design dialogs (or `findop` function), but simulation snapshots work well.

To find operating points, it is simplest to use the `operspec` and `findop` functions, customizing where necessary. Create an operating specification object with `operspec`, then compute an operating point object with `findop`. The `findop` function attempts to find an operating point that satisfies the operating specifications and reports on its success or failure. If the search is successful, `find_op` returns state values satisfying the operating specifications.

You have several choices for operating specifications for the components of the state vector.

Assumed Operating Condition	Operating Specification
Default	Request that all state component derivatives be zero. This is a steady-state for the whole model, not just a Simscape network within the model.
Nondefault	Request any value you want independently for each state component.
Nondefault	Request that a particular state component derivative be zero. This is a steady-state condition for that state component.

Additional Simulink Control Design Methods

You can also use the graphical user interface, through the model menu bar: **Analysis > Control Design > Linear Analysis**. This interface gives you access to state, input, and output names, structure, and initial values.

For more details on the use of operating point specification objects, related functions, and the graphical interface, see the Simulink Control Design documentation.

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 6-7

“Linearizing a Physical Model” on page 6-9

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 6-2.

- “What Is a Linearized Model?” on page 6-7
- “Example” on page 6-8
- “Choosing a Good Operating Point for Linearization” on page 6-8

What Is a Linearized Model?

Near an operating point, you can express the system state \mathbf{x} , inputs \mathbf{u} , and outputs \mathbf{y} relative to that operating point in terms of $\mathbf{x} - \mathbf{x}_0$, $\mathbf{u} - \mathbf{u}_0$, and $\mathbf{y} - \mathbf{y}_0$. For convenience, shift the vectors by subtracting the operating point: $\mathbf{x} - \mathbf{x}_0 \rightarrow \mathbf{x}$, 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:

$$\begin{aligned} \frac{d\mathbf{x}}{dt} &= \mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{u} \\ \mathbf{y} &= \mathbf{C} \mathbf{x} + \mathbf{D} \mathbf{u}. \end{aligned}$$

The matrices \mathbf{A} , \mathbf{B} , \mathbf{C} , \mathbf{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 \mathbf{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 \mathbf{A} , \mathbf{B} , \mathbf{C} , \mathbf{D} are not constant and depend on the simulation time t_0 , as well as the operating point \mathbf{x}_0 and \mathbf{u}_0 [3].

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 \mathbf{u} and one output component in \mathbf{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 \mathbf{x}_0 and inputs \mathbf{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 MathWorks recommends the Simulink Control Design product for linearization analysis.

- “Independent Versus Dependent States” on page 6-9
- “Linearizing with Simulink Control Design Software” on page 6-10
- “Linearizing with the Simulink `linmod` and `dlinmod` Functions” on page 6-10
- “Linearizing with Simulink Linearization Blocks” on page 6-12

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 4-5.

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 .

Additional Simulink Control Design Methods

You can also use the graphical user interface, through the model menu bar: **Analysis > Control Design > Linear Analysis**. For more details on linearization, operating points and state-space objects, related functions, and the graphical interface, see the Simulink Control Design documentation.

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” in the Simulink documentation.

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, MathWorks recommends 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.

In this section...

“Explore the Model” on page 6-13

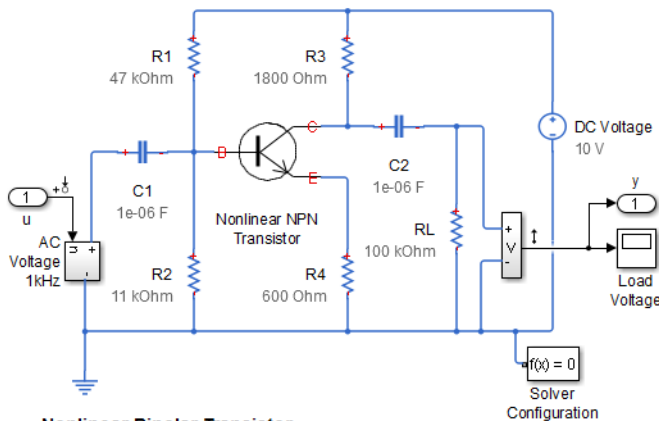
“Linearize with Steady-State Solver and linmod Function” on page 6-17

“Linearize with Simulink Control Design Software” on page 6-19

“Use Control System Toolbox Software for Bode Analysis” on page 6-20

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 an 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.

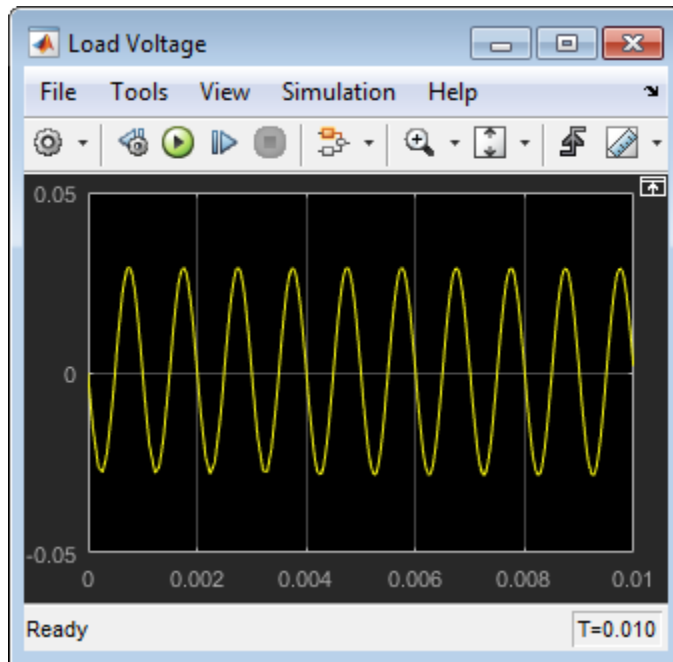
R1 and R2 set the nominal operating point, and the small signal gain is approximately set by the ratio R3/R4. The decoupling capacitors C1 and C2 have a capacitance of 1 μ F, to present negligible impedance at 1 kHz.

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.

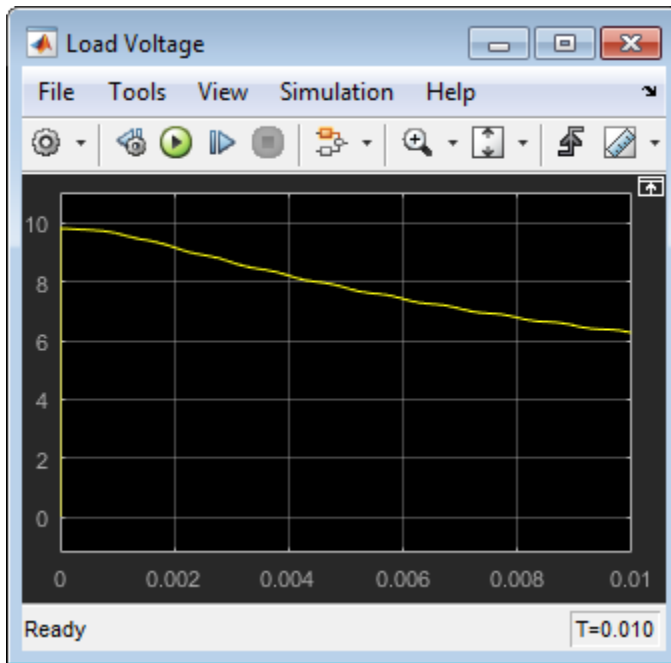
In general, to obtain a nontrivial linearized input-output model and generate a frequency response, you must specify model-level inputs and outputs. The Nonlinear Bipolar Transistor model meets this requirement in two ways, depending on how you linearize:

- 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.

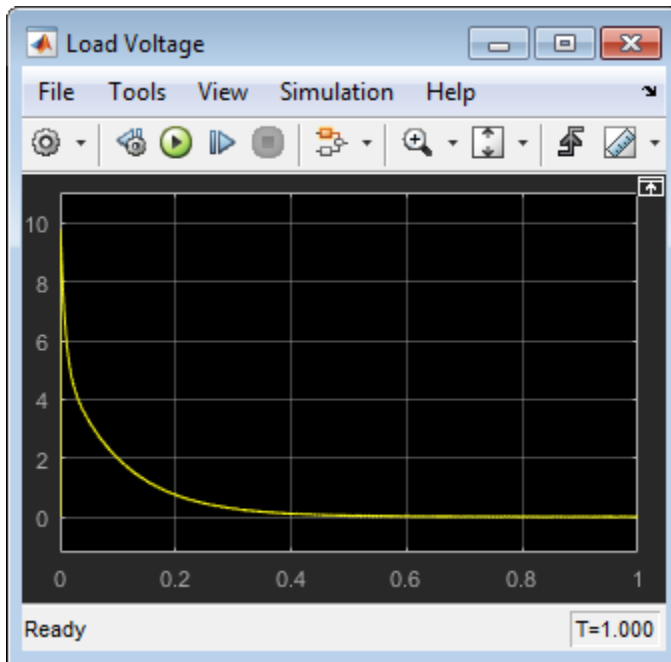
Open the Solver Configuration block and see that the **Start simulation from steady state** check box is selected. Then open the Load Voltage scope and run the simulation to see the basic circuit behavior. The transistor junction capacitance initial voltages are set to be consistent with the bias conditions 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');
```

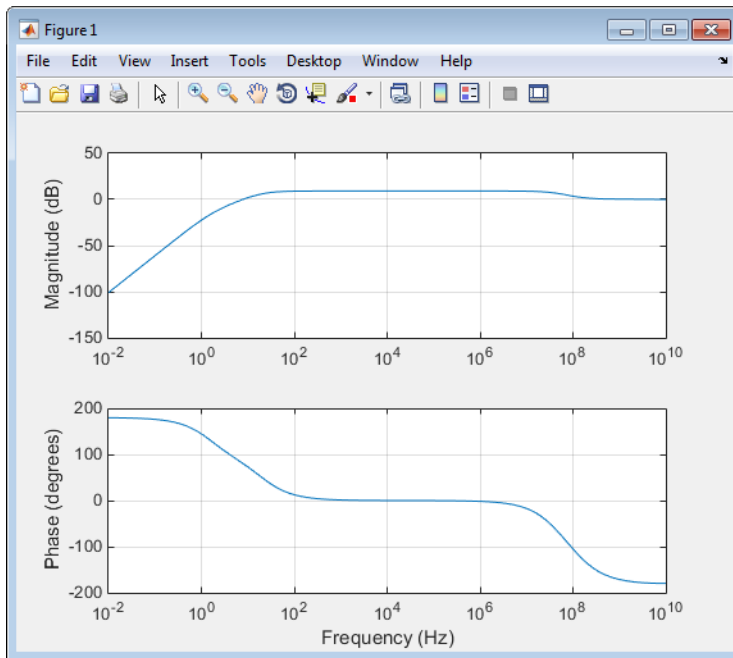
You can alternatively call the `linmod` function with a single output argument, in which case it generates a structure with states, inputs, and outputs, as well as the linear time-invariant (LTI) model.

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 with negative feedback convention, type the following in the MATLAB Command Window:

```
c = -c; d = -d;
npts = 100; f = logspace(-2,10,npts); G = zeros(1,npts);
for i=1:npts
    G(i) = c*(2*pi*i*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
```

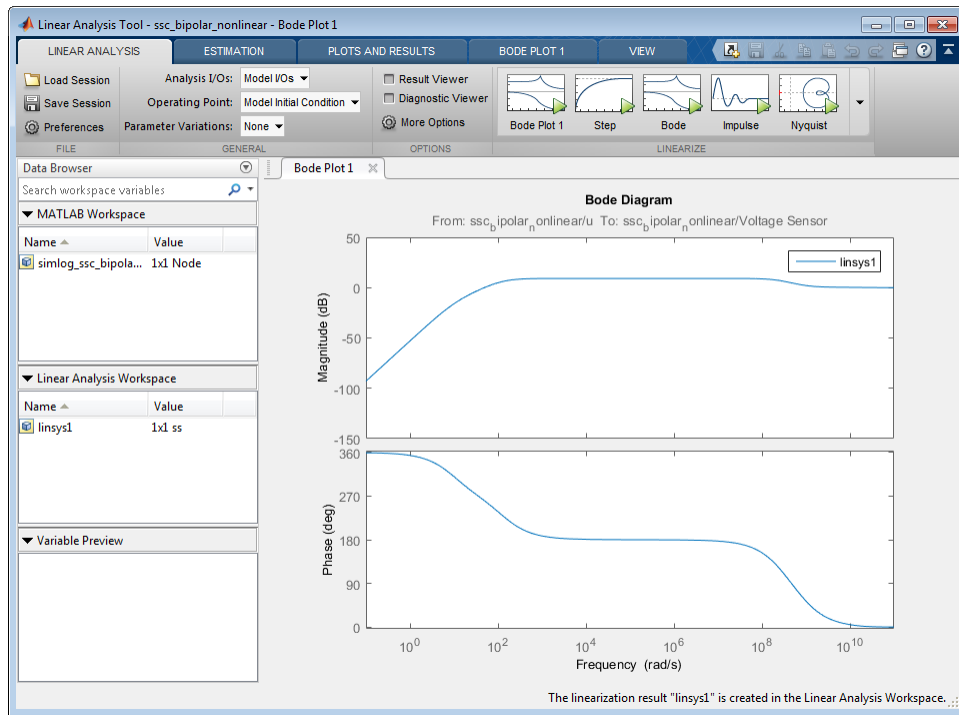


Linearize with Simulink Control Design Software

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 top menu bar of the Nonlinear Bipolar Transistor model, select **Analysis > Control Design > Linear Analysis** .
- 2 In the Linear Analysis Toolstrip, 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.

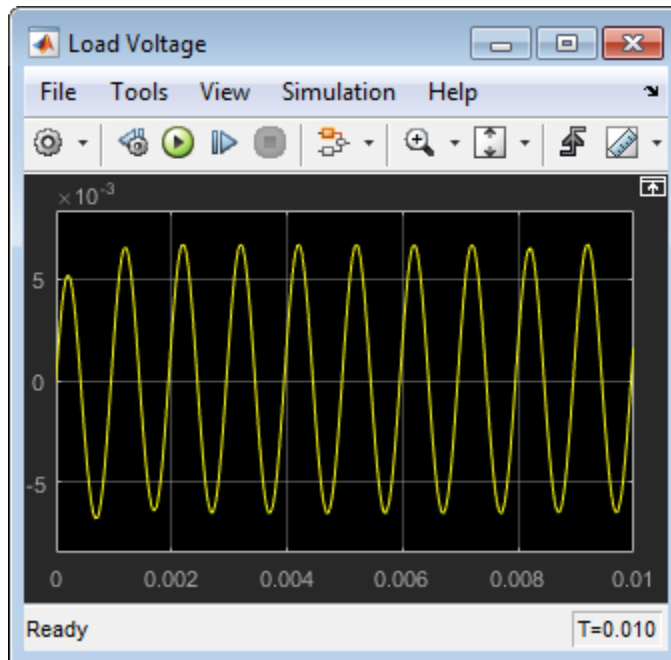
```
[a0,b0,c0,d0] = minreal(a,b,c,d);
```

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.

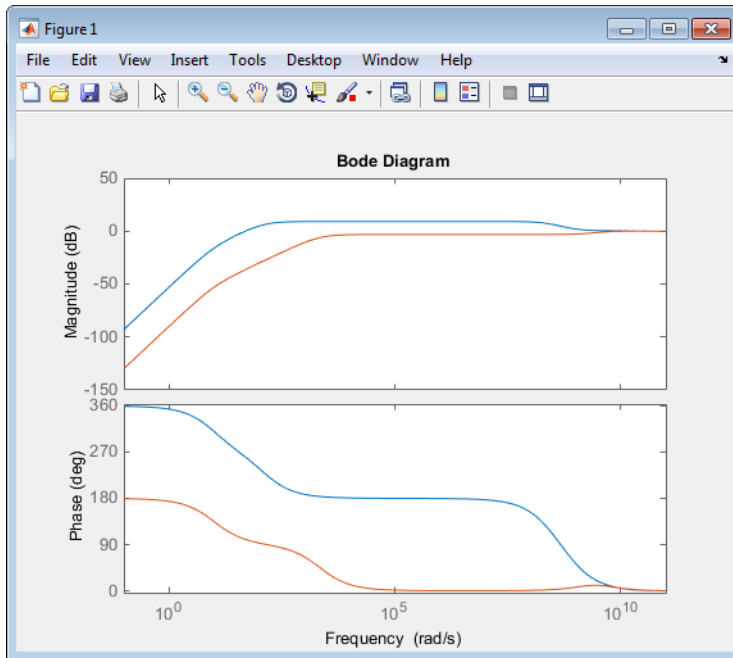
```
bode(a0,b0,c0,d0) % Creates first Bode plot
```

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.

Related Examples

- “Linearize a Plant Model for Use in Feedback Control Design” on page 6-23

More About

- “Finding Operating Points in Physical Models” on page 6-3
- “Linearizing a Physical Model” on page 6-9

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.

In this section...

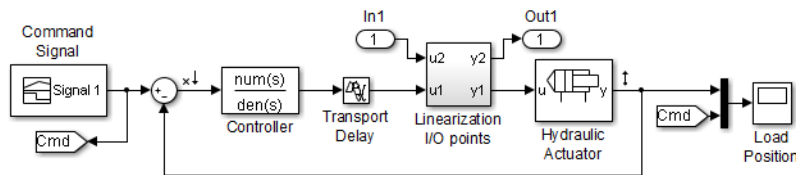
“Explore the Model” on page 6-23

“Trim Using the Controller and Linearize with Simulink linmod Function” on page 6-26

“Linearize with Simulink Control Design Software” on page 6-27

Explore the Model

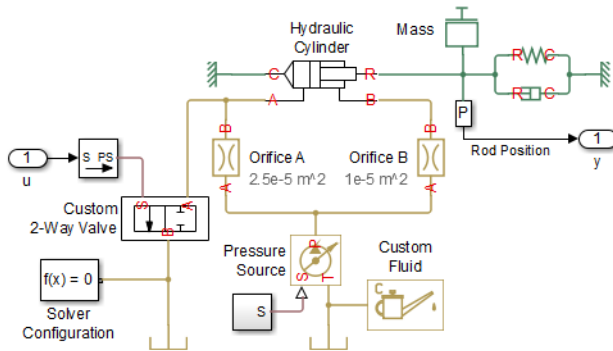
To open the Hydraulic Actuator with Digital Position Controller example model, type `ssc_hydraulic_actuator_digital_control` in the MATLAB Command Window.



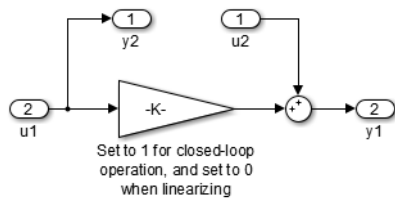
Hydraulic Actuator with Digital Position Controller

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.



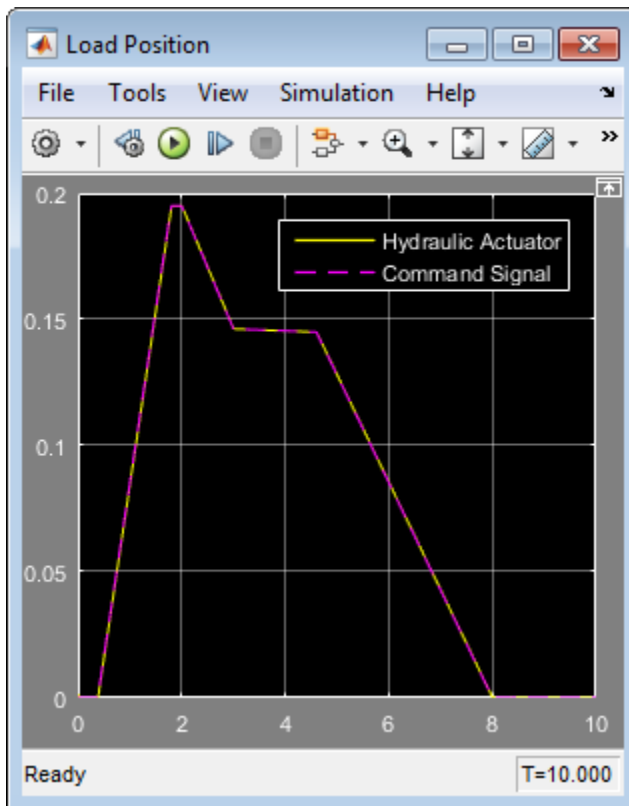
The model is configured for linearization. You can quickly generate and view the small-signal frequency response by clicking the [Linearize](#) hyperlink in model annotation. To view the MATLAB script that generates the frequency response, click the next hyperlink in that annotation, [see code](#). This 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);
```

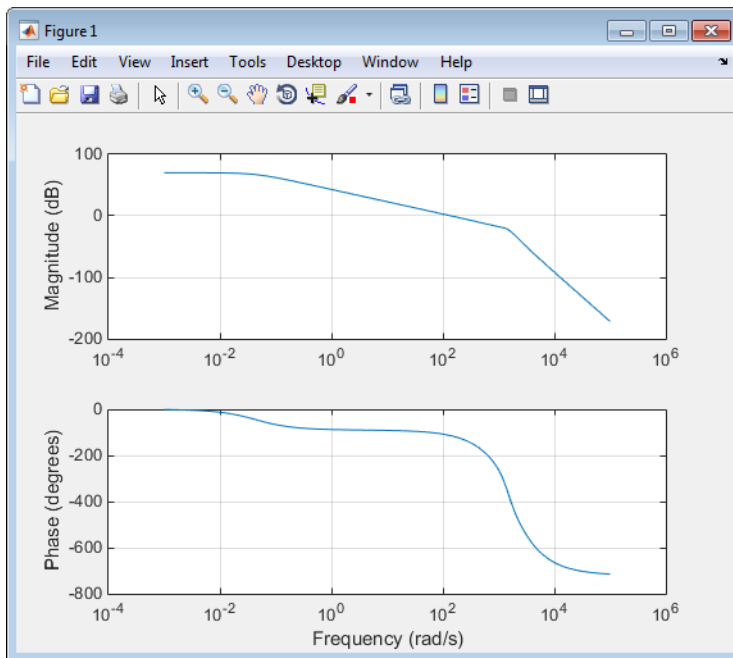
- 4 To generate a Bode plot with negative feedback convention, type the following in the MATLAB Command Window:

```
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

```

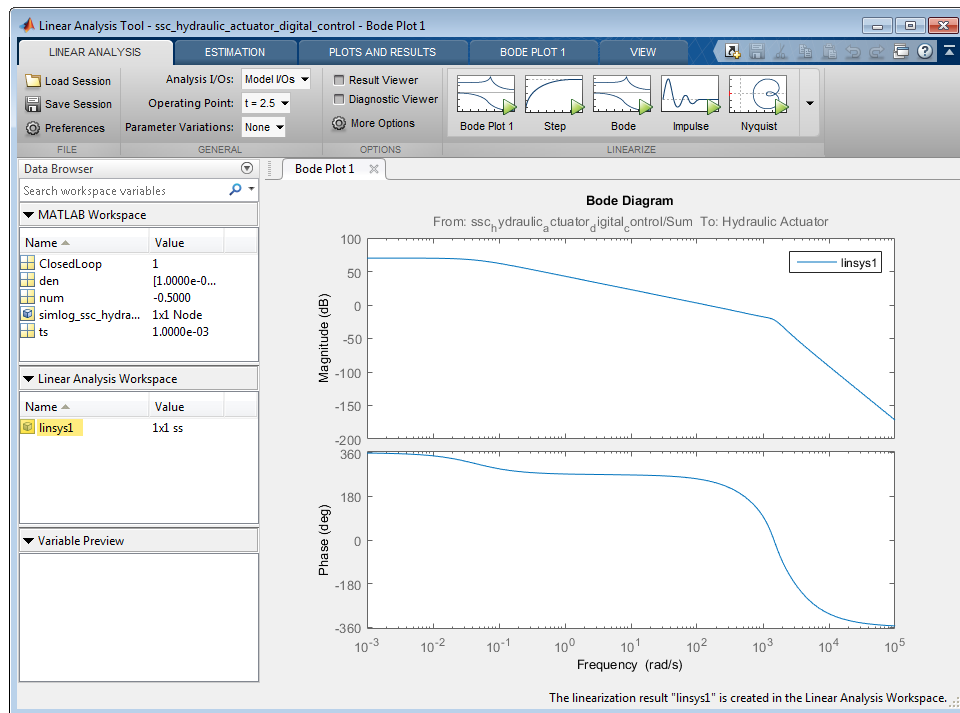


Linearize with Simulink Control Design Software

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 top menu bar of the Hydraulic Actuator with Digital Position Controller model, select **Analysis > Control Design > Linear Analysis**.
- 2 In the Linear Analysis Tool, 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.

Related Examples

- “Linearize an Electronic Circuit” on page 6-13

More About

- “Finding Operating Points in Physical Models” on page 6-3
- “Linearizing a Physical Model” on page 6-9

Real-Time Simulation

- “Model Preparation Objectives” on page 7-2
- “Real-Time Model Preparation Workflow” on page 7-5
- “Improving Speed and Accuracy” on page 7-10
- “Determine Step Size” on page 7-15
- “Reduce Computation Costs” on page 7-25
- “Reduce Numerical Stiffness” on page 7-31
- “Reduce Zero Crossings” on page 7-41
- “Fixed-Cost Simulation for Real-Time Viability” on page 7-55
- “Real-Time Simulation Workflow” on page 7-57
- “Solvers for Real-Time Simulation” on page 7-63
- “Determine System Stiffness” on page 7-68
- “Estimate Computation Costs” on page 7-76
- “Choose Step Size and Number of Iterations” on page 7-79
- “What Is Hardware-in-the-Loop Simulation?” on page 7-96
- “Hardware-in-the-Loop Simulation Workflow” on page 7-100
- “Code Generation Requirements” on page 7-106
- “Generate, Download, and Execute Code” on page 7-108
- “Requirements for Using Alternative Platforms” on page 7-111

Model Preparation Objectives

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. 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.

Related Examples

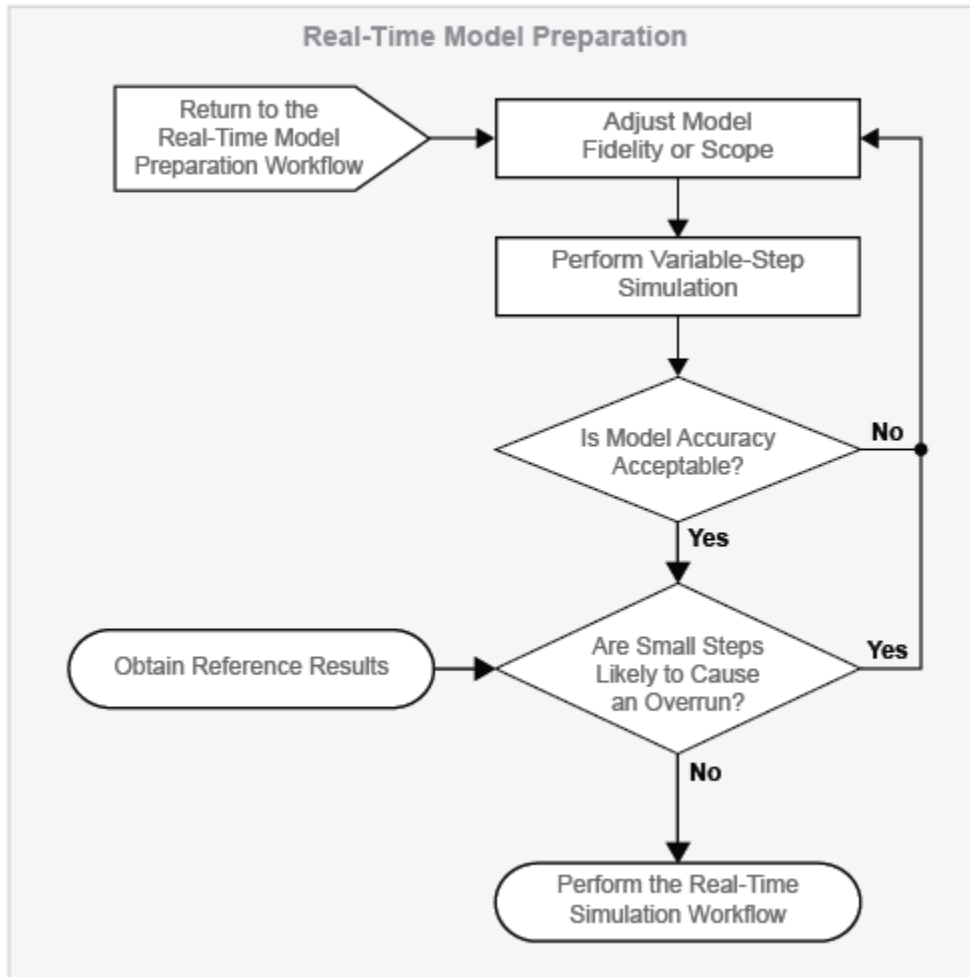
- “Determine Step Size” on page 7-15
- “Reduce Computation Costs” on page 7-25
- “Reduce Numerical Stiffness” on page 7-31
- “Reduce Zero Crossings” on page 7-41

More About

- “Fixed-Cost Simulation for Real-Time Viability” on page 7-55
- “Improving Speed and Accuracy” on page 7-10
- “Real-Time Model Preparation Workflow” on page 7-5
- “Real-Time Simulation Workflow” on page 7-57

Real-Time Model Preparation Workflow

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, 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. 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 numerical stiffness.
- Reduce zero crossings.
- Reduce computation costs.
- 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 7-57 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 lacks the computational capability for running your model in real time. If, after multiple iterations of the workflow, it appears that there is no level of model complexity that can make your model real-time capable, consider these options for increasing processing power:

- “Upgrading Target Hardware” on page 7-13
- “Simulating Parts of the System in Parallel” on page 7-13

Related Examples

- “Determine Step Size” on page 7-15

- “Estimate Computation Costs” on page 7-76
- “Reduce Computation Costs” on page 7-25
- “Reduce Numerical Stiffness” on page 7-31
- “Reduce Zero Crossings” on page 7-41

More About

- “Essential Physical Modeling Techniques” on page 1-15
- “Hardware-in-the-Loop Simulation Workflow” on page 7-100
- “Improving Speed and Accuracy” on page 7-10
- “Model Preparation Objectives” on page 7-2
- “Modeling Dynamic Systems”
- “Real-Time Simulation Workflow” on page 7-57

Improving Speed and Accuracy

In this section...
“Why Speed and Accuracy Matter for Real-Time Simulation” on page 7-10
“Balancing Speed and Accuracy” on page 7-11
“Eliminating Effects That Require Intensive Computation” on page 7-12
“Optimizing Local and Global Solver Configurations” on page 7-13
“Upgrading Target Hardware” on page 7-13
“Simulating Parts of the System in Parallel” on page 7-13

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 Tether or not 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 7-15 and “Estimate Computation Costs” on page 7-76.

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. 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 7-12. For more information, on how solver configurations affect speed and accuracy, see “Optimizing Local and Global Solver Configurations” on page 7-13.

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, or if the accuracy is unacceptable, consider these options for increasing speed and accuracy:

- “Upgrading Target Hardware” on page 7-13
- “Simulating Parts of the System in Parallel” on page 7-13

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 delivers does not necessarily correlate to a specific step size across all networks in a single model. A real-time target 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 7-63.

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 to use this approach.

Related Examples

- “Determine Step Size” on page 7-15
- “Estimate Computation Costs” on page 7-76
- “Reduce Computation Costs” on page 7-25
- “Reduce Numerical Stiffness” on page 7-31
- “Reduce Zero Crossings” on page 7-41

More About

- “Model Preparation Objectives” on page 7-2
- “Real-Time Model Preparation Workflow” on page 7-5
- “Solvers for Real-Time Simulation” on page 7-63

Determine Step Size

For the first step in “Real-Time Model Preparation Workflow” on page 7-5, 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:

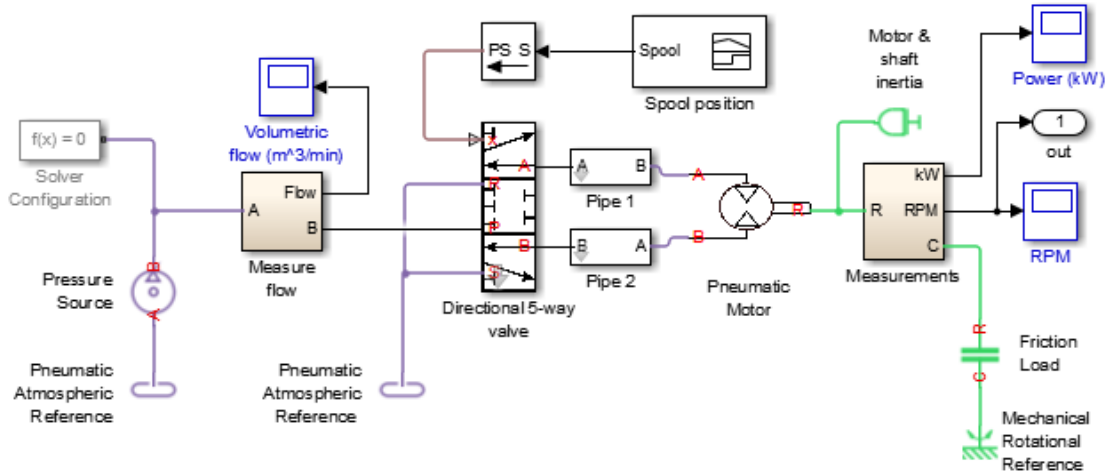
- Estimate the maximum step size that you can use for a fixed-step simulation
- Identify events that have the potential to limit the maximum step size

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, $T_{s_{max}}$, is too small to 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 possible 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 7-2.

1 To open the reference model, at the MATLAB command prompt, enter:

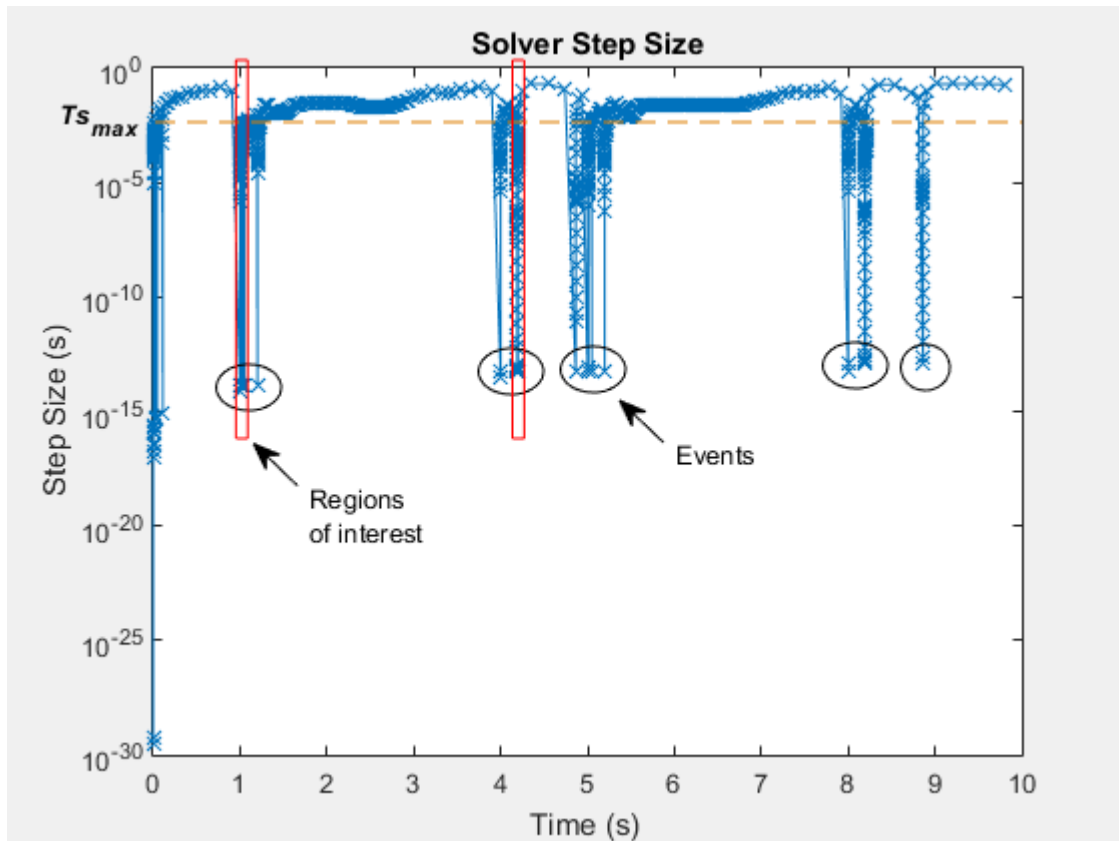
```
ssc_pneumatic_rts_reference
```



- 2 Simulate the 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)');
    
```



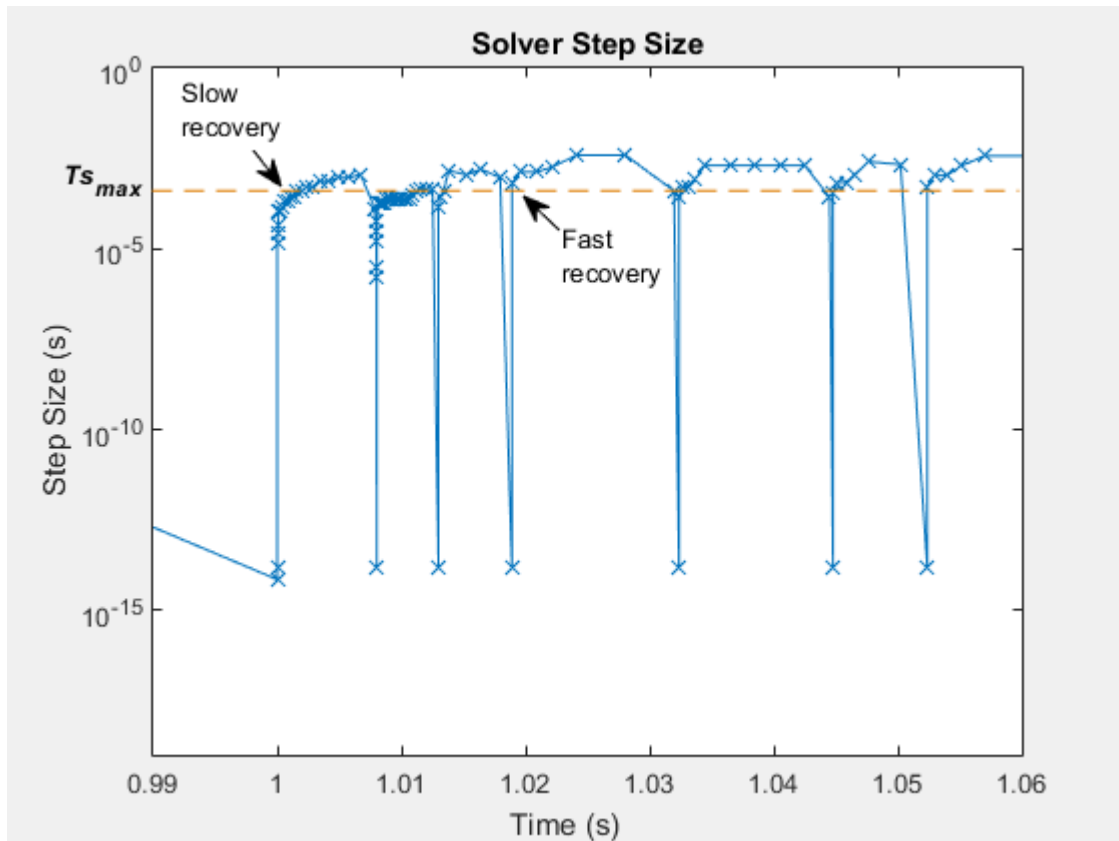
For much of the simulation, the step size is greater than the value of the Ts_{max} in the plot. The corresponding value, ~ 0.001 seconds, is an estimated maximum step size for achieving accurate results during fixed-step simulation with the model. To see how to configure the step size for fixed-step solvers for real-time simulation, see “Choose Step Size and Number of Iterations” on page 7-79.

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 $(t) = \sim 1$ second.

```
h1;  
xStart = 0; xEnd = 10; yStart = 0; yEnd = 10e0;  
xZoomStart1 = 0.99; xZoomEnd1 = 1.06;  
yZoomStart1 = 10e-20; yZoomEnd1 = 10e-1;  
axis([xZoomStart1 xZoomEnd1 yZoomStart1 yZoomEnd1]);
```



Between $t = 0$ and 1 second, the solver takes a step every 0.01 seconds. The step size decreases to less than $10e-10$ seconds to capture an event that occurs one second into the simulation. The step size increases quickly to $10e-5$ seconds, and then increases slowly to the step size from before the event. The slow rate of recovery indicates 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:

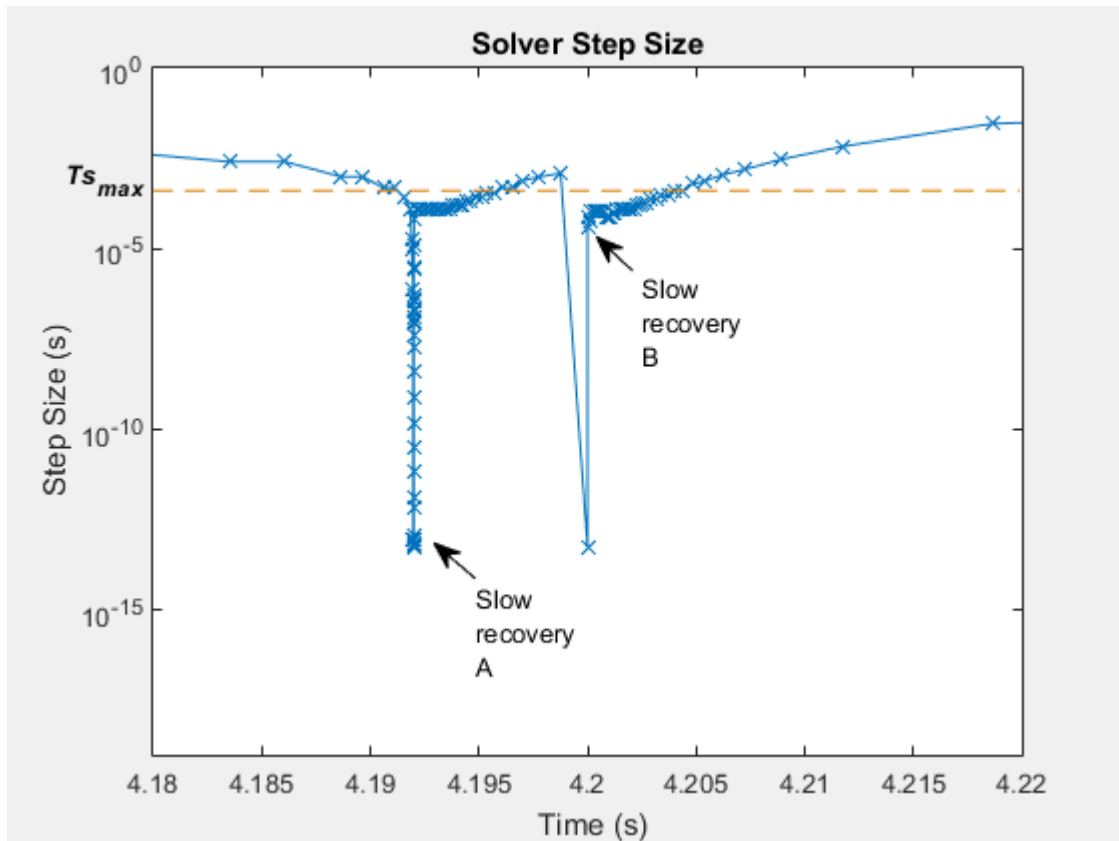
- Elements that cause discontinuities, such as hard-stops and stick-slip friction

- 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 events that occur between the simulation times where $t = 1.02$ and 1.06 seconds. The events that occur during this time are 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 $t = \sim 4.2$ seconds.

```
h1;  
xZoomStart2 = 4.16; xZoomEnd2 = 4.24;  
yZoomStart2 = 10e-20; yZoomEnd2 = 10e-1;  
axis([xZoomStart2 xZoomEnd2 yZoomStart2 yZoomEnd2]);
```




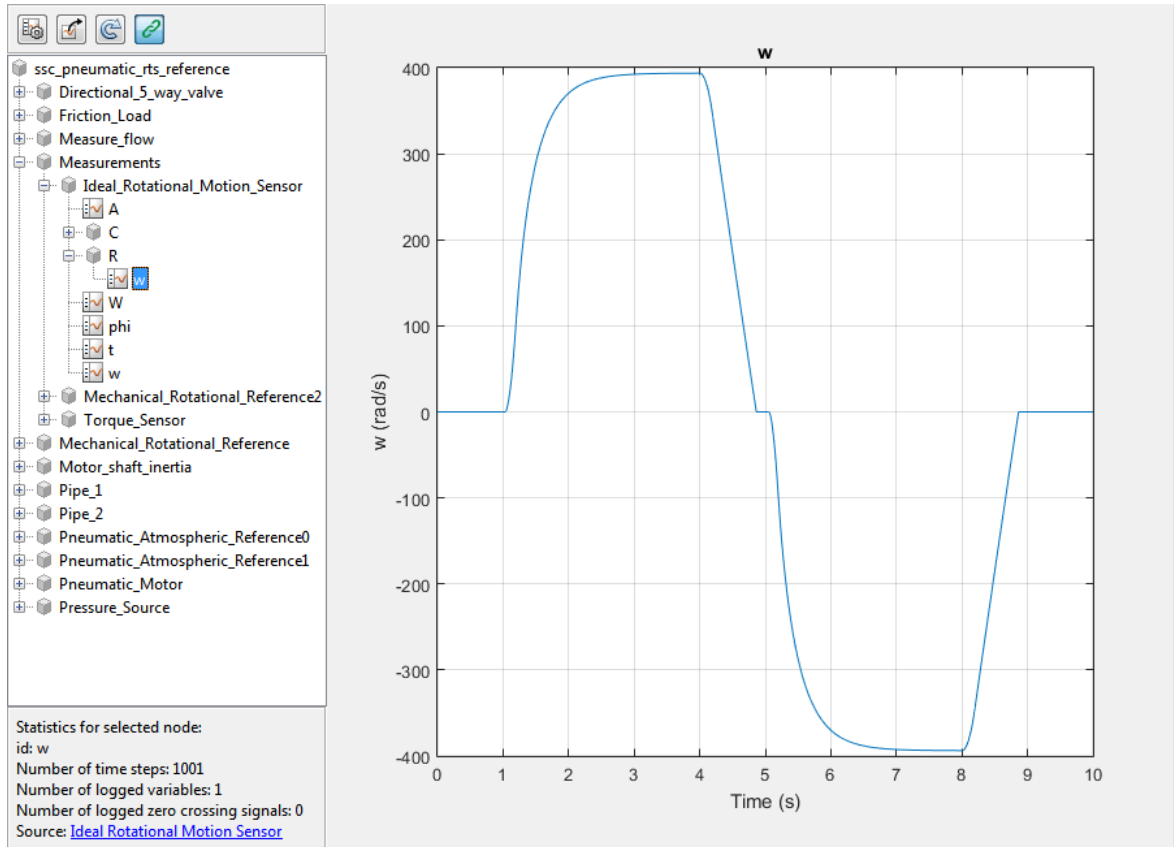
Just as there are different types of events that cause solvers to slow down, there are different types of slow solver recovery. The events that occur just before $t = 4.19$ and 4.2 seconds both involve zero-crossings. The solver takes a series of progressively larger steps as it reaches the step size 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.

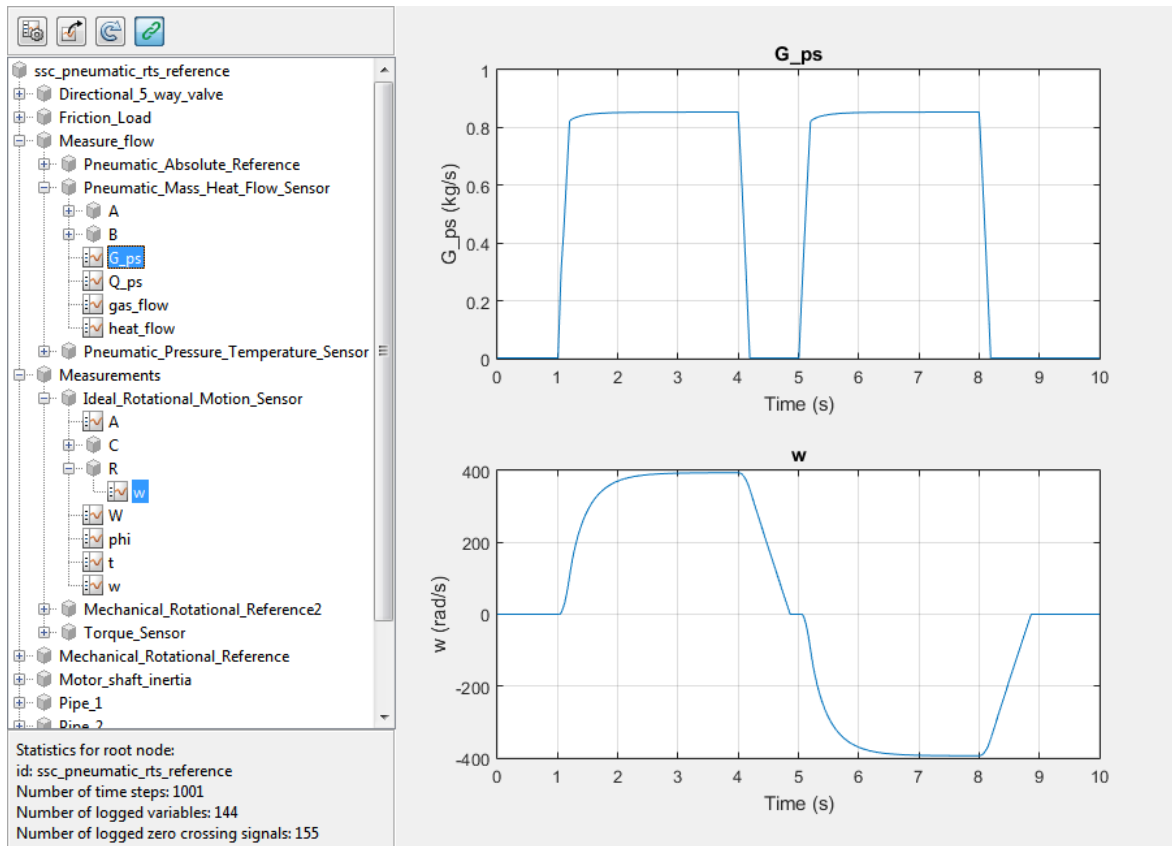
- 6 To see the results, open the Simscape Results Explorer.

sscexplore(simlog)

- 7 In the Simscape Results Explorer window, in the simulation log tree hierarchy, expand the Measurements node and the Ideal_Rotational_Motion_Sensor node. Select the w Plot  to see the plot for angular speed.



- 8 To add the gas flow results to the Simscape Results Explorer window, expand the Measure_Flow node and the Pneumatic_Mass_Heat_Flow_Sensor node. Use Ctrl +click to select both the G_ps Plot  and the w Plot .



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 $t = \sim 1, 5,$ 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)

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 7-31 and “Reduce Zero Crossings” on page 7-41.

Related Examples

- “Choose Step Size and Number of Iterations” on page 7-79
- “Reduce Numerical Stiffness” on page 7-31
- “Reduce Zero Crossings” on page 7-41

More About

- “About Simulation Data Logging” on page 9-2
- “Events and Zero Crossings” on page 4-3
- “Improving Speed and Accuracy” on page 7-10
- “Log and View Simulation Data for Selected Blocks” on page 9-17
- “Model Preparation Objectives” on page 7-2
- “Real-Time Model Preparation Workflow” on page 7-5
- “Real-Time Simulation Workflow” on page 7-57
- “Solvers for Real-Time Simulation” on page 7-63
- “Stiffness” on page 4-3
- “Zero-Crossing Detection”

Reduce Computation Costs

In this section...

“Data Logging and Monitoring Guidelines” on page 7-25

“Improve Data Logging and Monitoring Efficiency” on page 7-25

“Additional Methods for Reducing Computational Cost” on page 7-29

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.

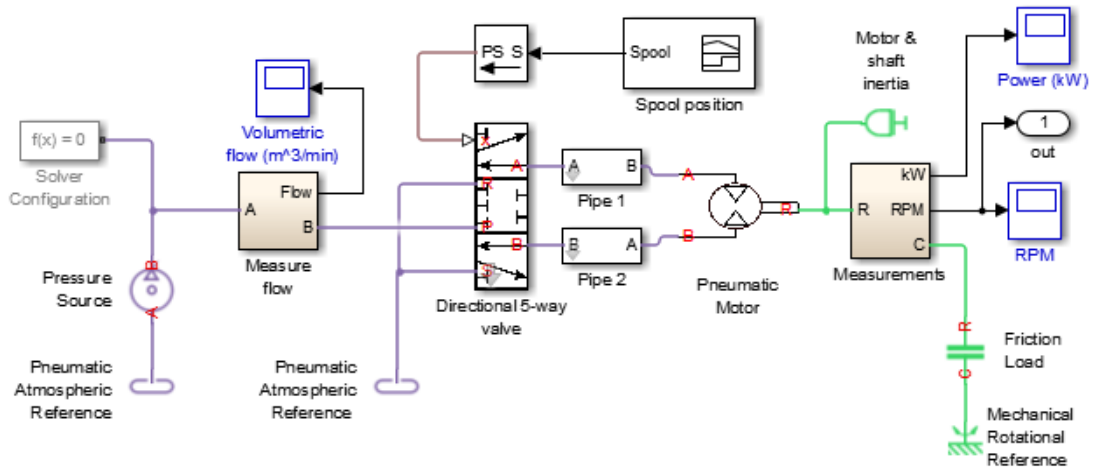
Note: Simscape simulation data logging is not supported for generated code.

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.

- To open the model, at the MATLAB command prompt, enter:

```
ssc_pneumatic_rts_zc_redux
```



The model contains three scope blocks and one output block. The Power (kW) scope, RPM scope and output block receive data from the Measurements subsystem.

- Simulate the model.

Name	
	tout
	yout
	simlog
	Pneu_rts_RPM_DATA
	Pneu_rts_Vol_Flow_DATA

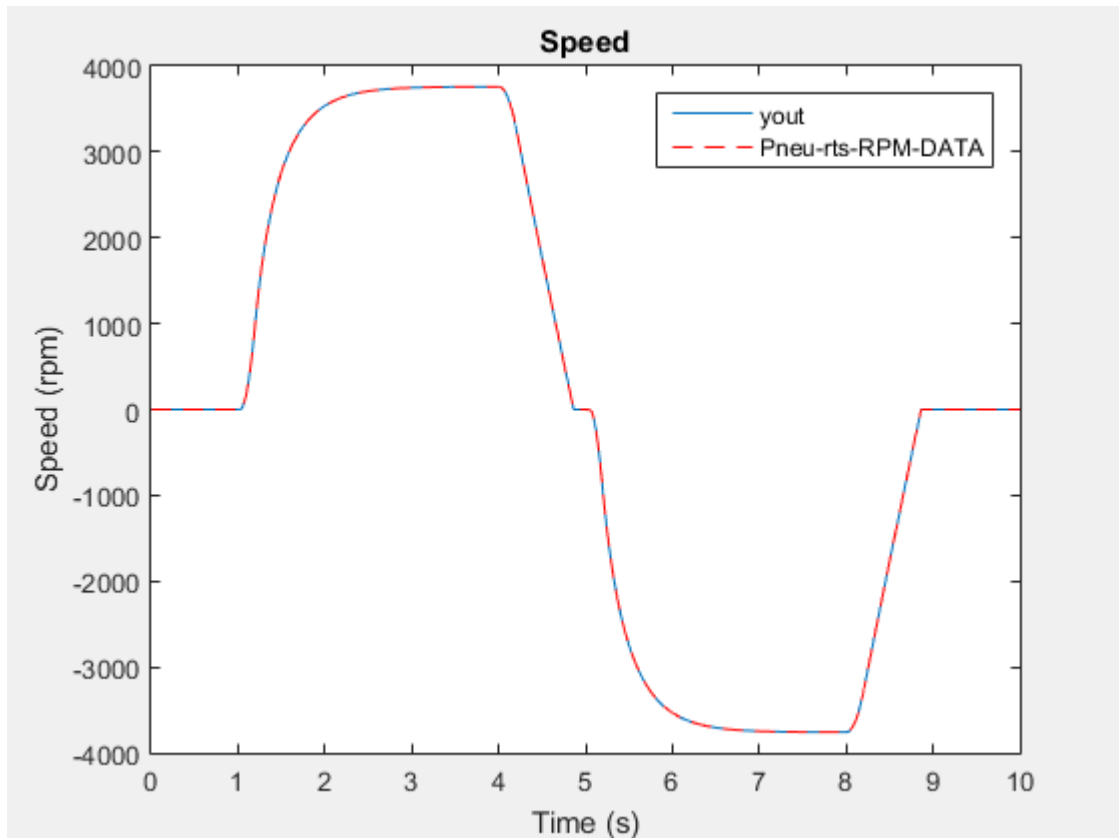
The model logs five variables to the workspace, including a Simscape simulation data logging node.

- 3 To determine the source for the `Pneu_rts_RPM_DATA`, In the MATLAB workspace, open the structure.

The `blockName` variable shows that the RPM scope logs the data. In the model, the output 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 output block.
- If you only need to monitor the speed data, delete the output 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 need to log data while you prepare your model for real-time simulation, configure the model to log only the data that you need with the `simlog` node in the MATLAB workspace. For information, see “Log Data for Selected Blocks Only” on page 9-5.

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).

Related Examples

- “Determine Step Size” on page 7-15
- “Estimate Computation Costs” on page 7-76
- “Reduce Numerical Stiffness” on page 7-31
- “Reduce Zero Crossings” on page 7-41

More About

- “About Simulation Data Logging” on page 9-2
- “Limitations” on page 9-3
- “Improving Speed and Accuracy” on page 7-10
- “Log, Navigate, and Plot Simulation Data” on page 9-21

- “Model Preparation Objectives” on page 7-2
- “Real-Time Model Preparation Workflow” on page 7-5
- “Real-Time Simulation Workflow” on page 7-57

Reduce Numerical Stiffness

This example helps you to complete the steps outlined in “Real-Time Model Preparation Workflow” on page 7-5 and to meet the goals described in “Model Preparation Objectives” on page 7-2.

In “Determine Step Size” on page 7-15, 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 7-15 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?

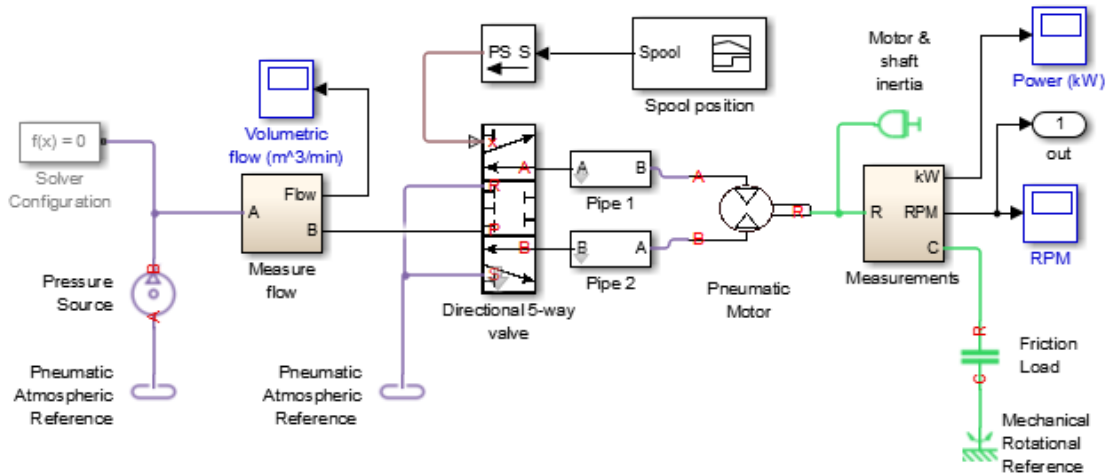
Numerical stiffness can prevent your model from being real-time capable. A real-time-capable model is one that produces acceptable results without incurring overruns when you simulate it on your target processor. Stiff systems contain dynamics that vary both quickly and slowly. When solvers take large 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:

```
ssc_pneumatic_rts_reference
```

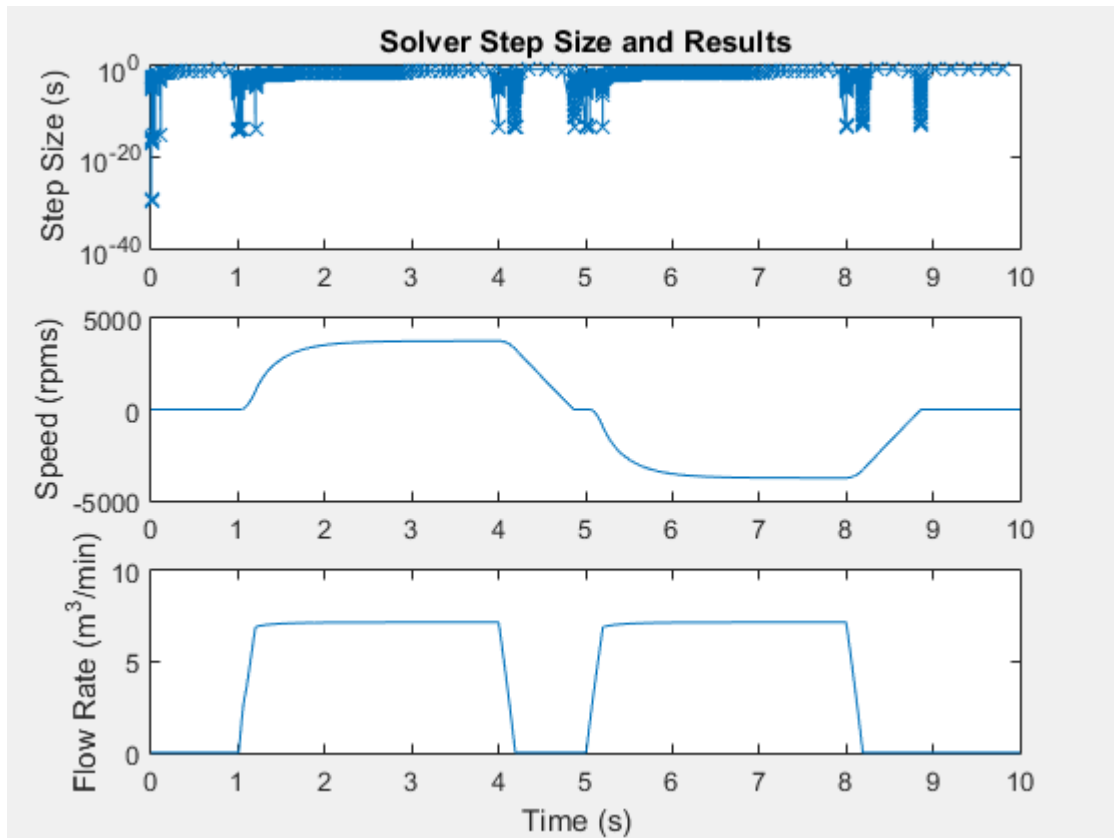


- 2 Simulate the model.
- 3 Create a figure that contains a semilogarithmic plot of the solver step size, a plot of the motor speed results, and a plot of the gas flow results.

```

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.

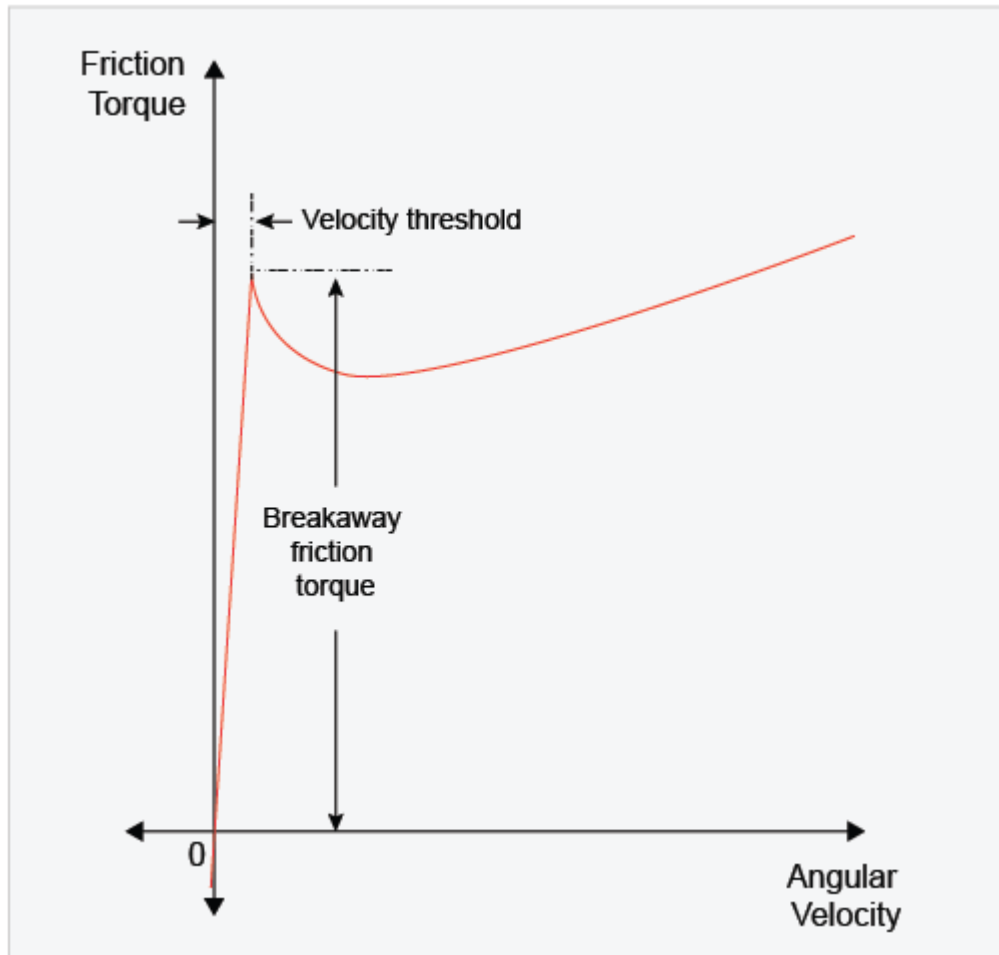
- 4 Assign the simulation results to new variables in the MATLAB workspace so that you can compare the data to results from a model that you modify.

```
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.

- 1 Save the model as `rts_stiffness_model` in a writable folder on the MATLAB path.
- 2 Open the Friction Load block dialog box, 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 value for the **Linear region velocity threshold** that is not close to zero.

- 3 On the **Parameters** tab of the dialog box, change the value of the **Linear region velocity threshold** from .005 to .5 rad/s.

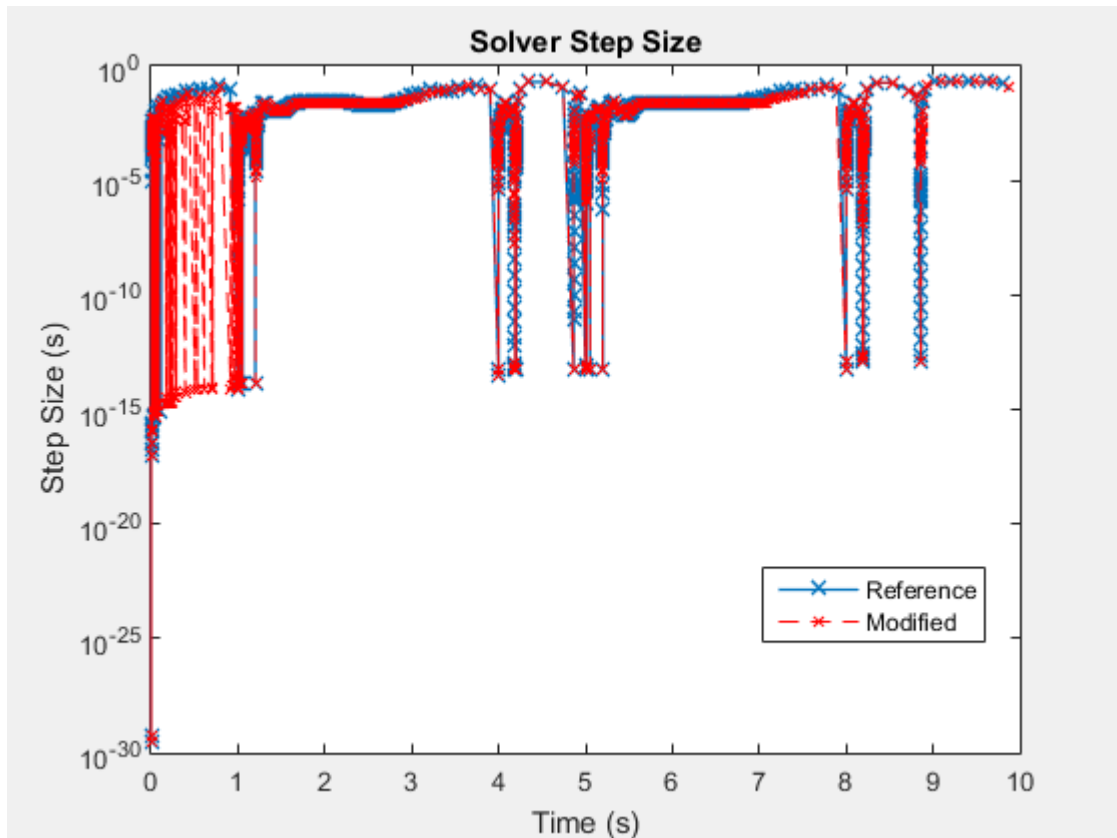
- 4 Simulate the modified model.

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 those 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');
```



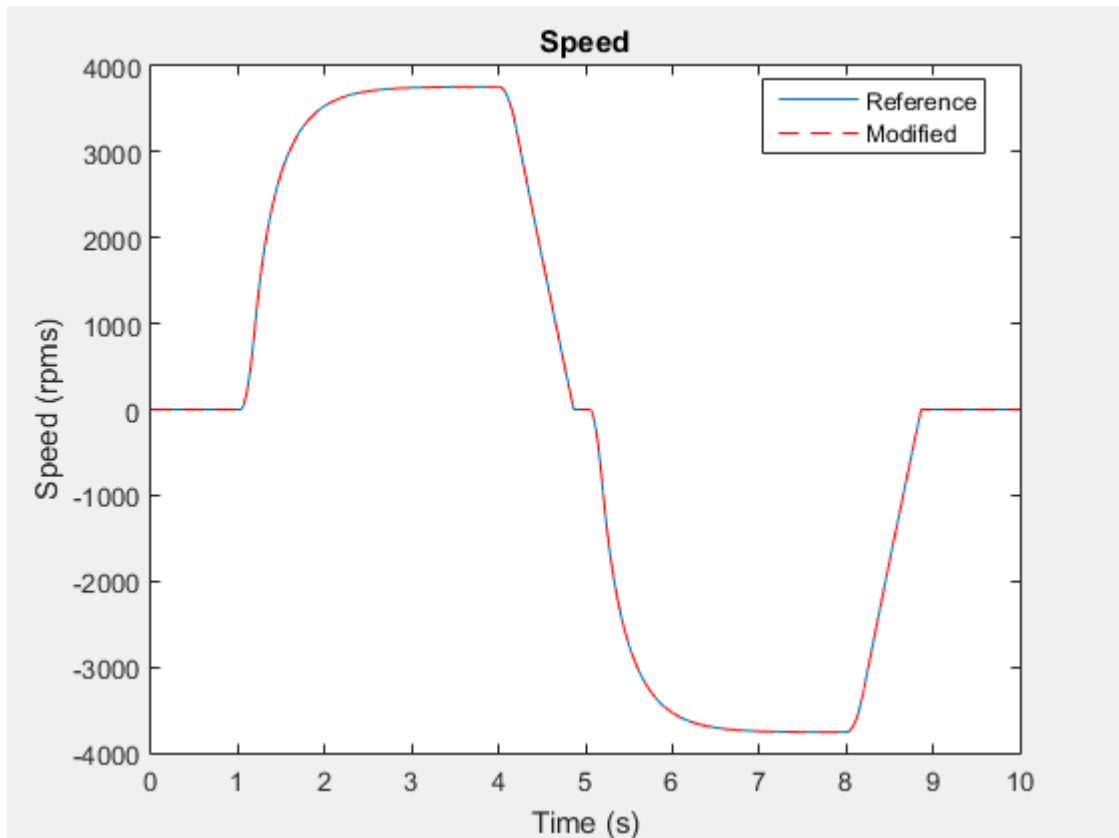
The step size recovers more quickly from events that occur at simulation time $t \approx 4$, 5, 8, and 9 seconds. The simulation is less stiff at these times.

- 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;
```

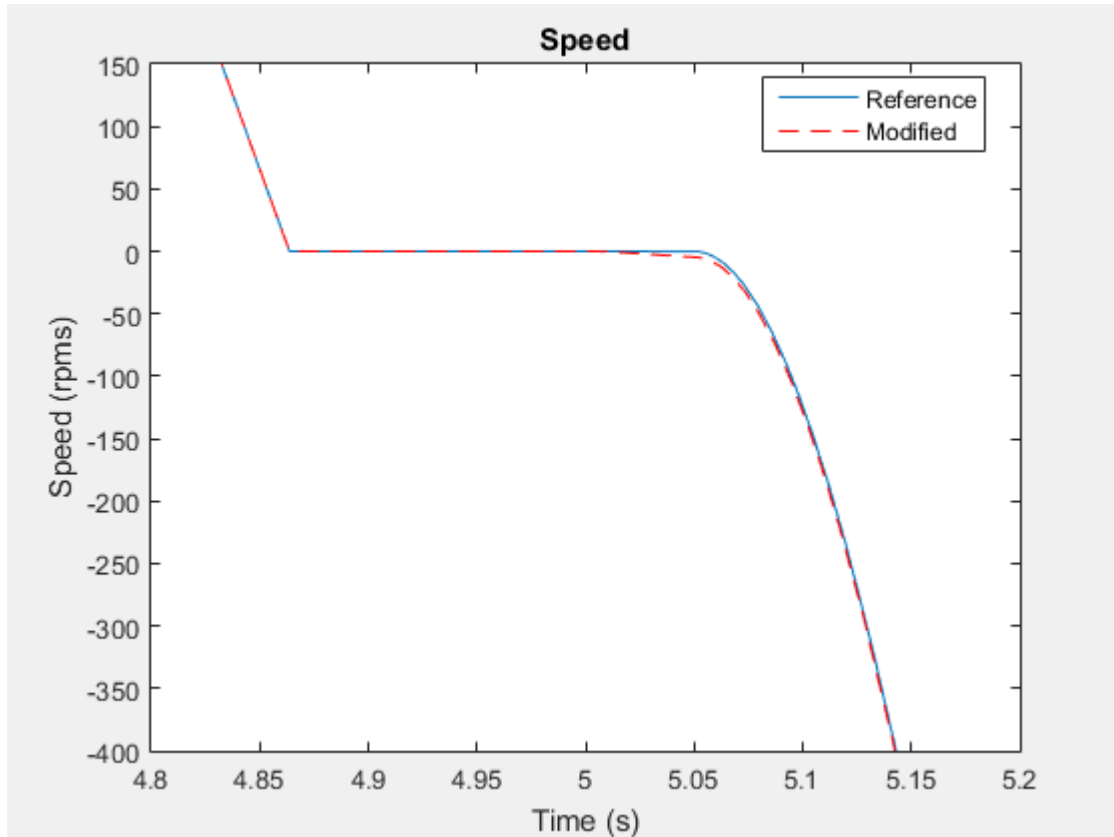
- Plot and compare the results for the speed data for both simulations to make sure that the modified model is accurate.

```
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');
```



- 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 a small difference in the results, but 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 7-25 and “Reduce Zero Crossings” on page 7-41. 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 7-79.

See Also

Rotational Friction

Related Examples

- “Determine Step Size” on page 7-15
- “Estimate Computation Costs” on page 7-76
- “Reduce Computation Costs” on page 7-25
- “Reduce Zero Crossings” on page 7-41

More About

- “About Simulation Data Logging” on page 9-2
- “Events and Zero Crossings” on page 4-3
- “Log and Plot Simulation Data” on page 9-8
- “Model Preparation Objectives” on page 7-2
- “Real-Time Model Preparation Workflow” on page 7-5
- “Stiffness” on page 4-3

Reduce Zero Crossings

In this section...

“Why Reduce Zero Crossings?” on page 7-41

“Obtain Reference Results and Plot Simulation Step Size” on page 7-41

“Identify and Modify Elements That Cause Zero Crossings” on page 7-45

“Analyze the Results of the Modified Model” on page 7-50

Why Reduce Zero Crossings?

Real-time deployment requires using a fixed-step solver. You might want to run the same model using 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:

```
ssc_pneumatic_rts_stiffness_redux
```

2 Simulate the model.

3 Save the data to the workspace.

```
simlogRef = simlog;  
timeRef = tout;
```

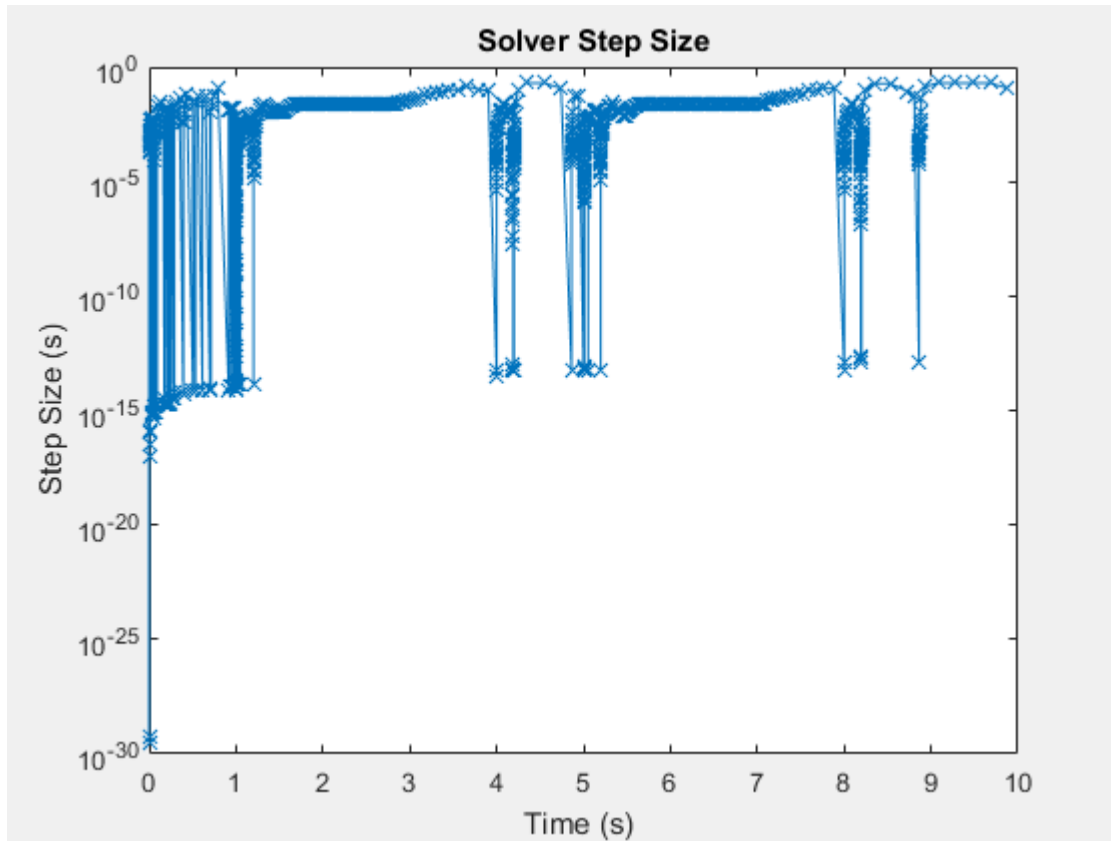
4 Plot the step size against the simulation time.

```
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 $t \sim 4, 5, 8,$ and 9 seconds.

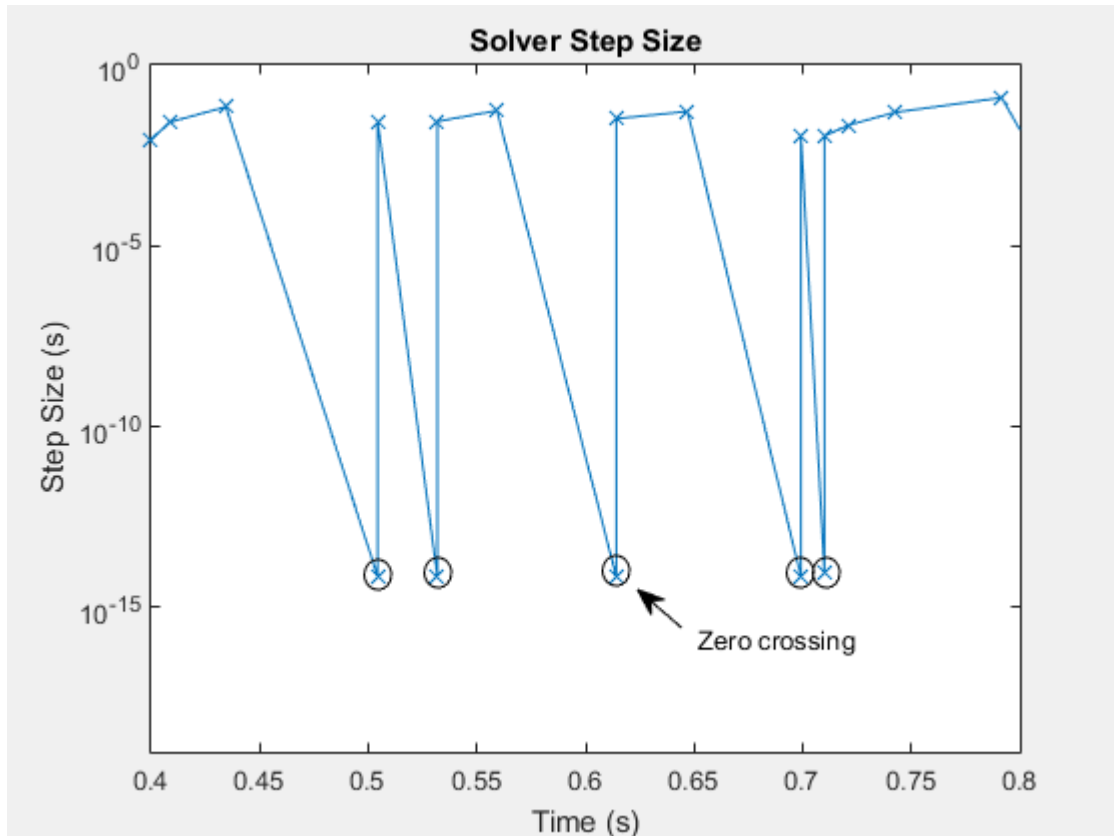
- 5 Zoom to examine the data between time $t = 0.4$ and 0.8 seconds.

```

h1;
xStart = 0; xEnd = 10; yStart = 0; yEnd = 10e0;
xZoomStart1 = 0.4; xZoomEnd1 = 0.8;
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.sli.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.sli.findNode(simlogRef, gcbh)
```

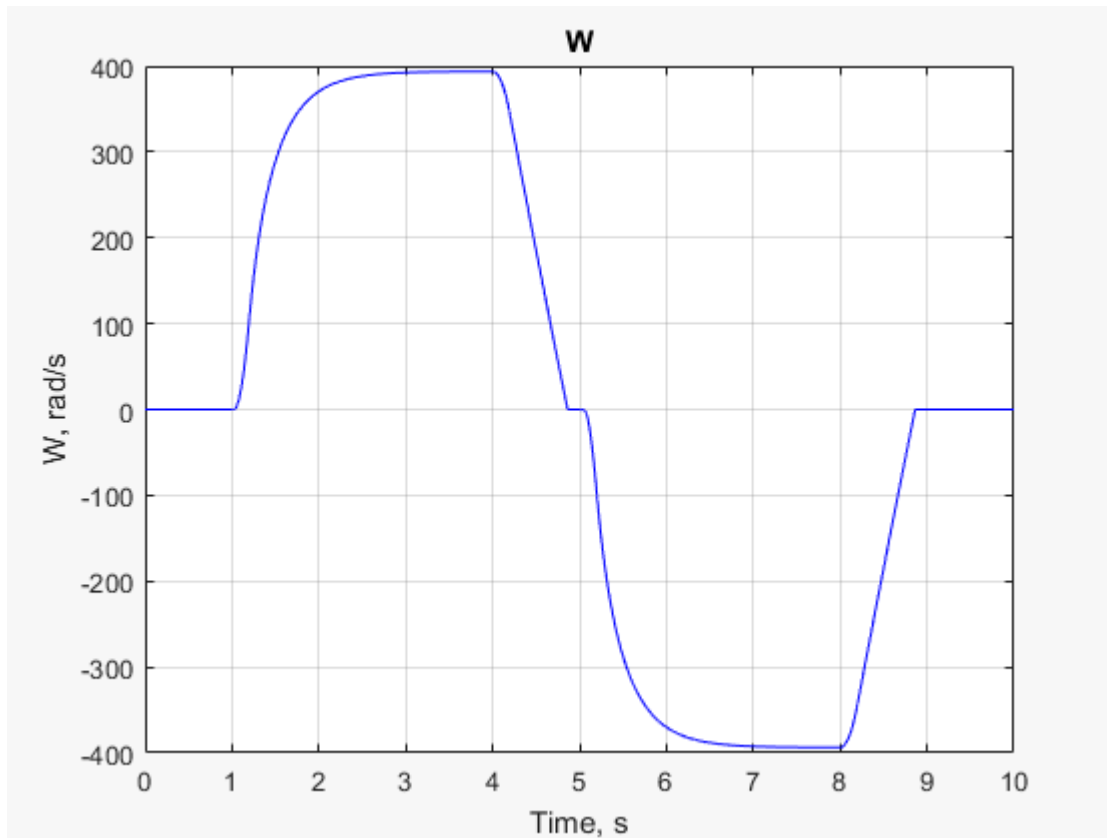
```
nRef =
```

```
Node with properties:
```

```
    id: 'Ideal_Rotational_Motion_Sensor'  
    w: [1x1 simscape.logging.Node]  
phi: [1x1 simscape.logging.Node]  
    R: [1x1 simscape.logging.Node]  
    W: [1x1 simscape.logging.Node]  
    t: [1x1 simscape.logging.Node]  
    A: [1x1 simscape.logging.Node]  
    C: [1x1 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 (155 signals, 101 crossings)
+-Directional_5_way_valve (144 signals, 36 crossings)
```

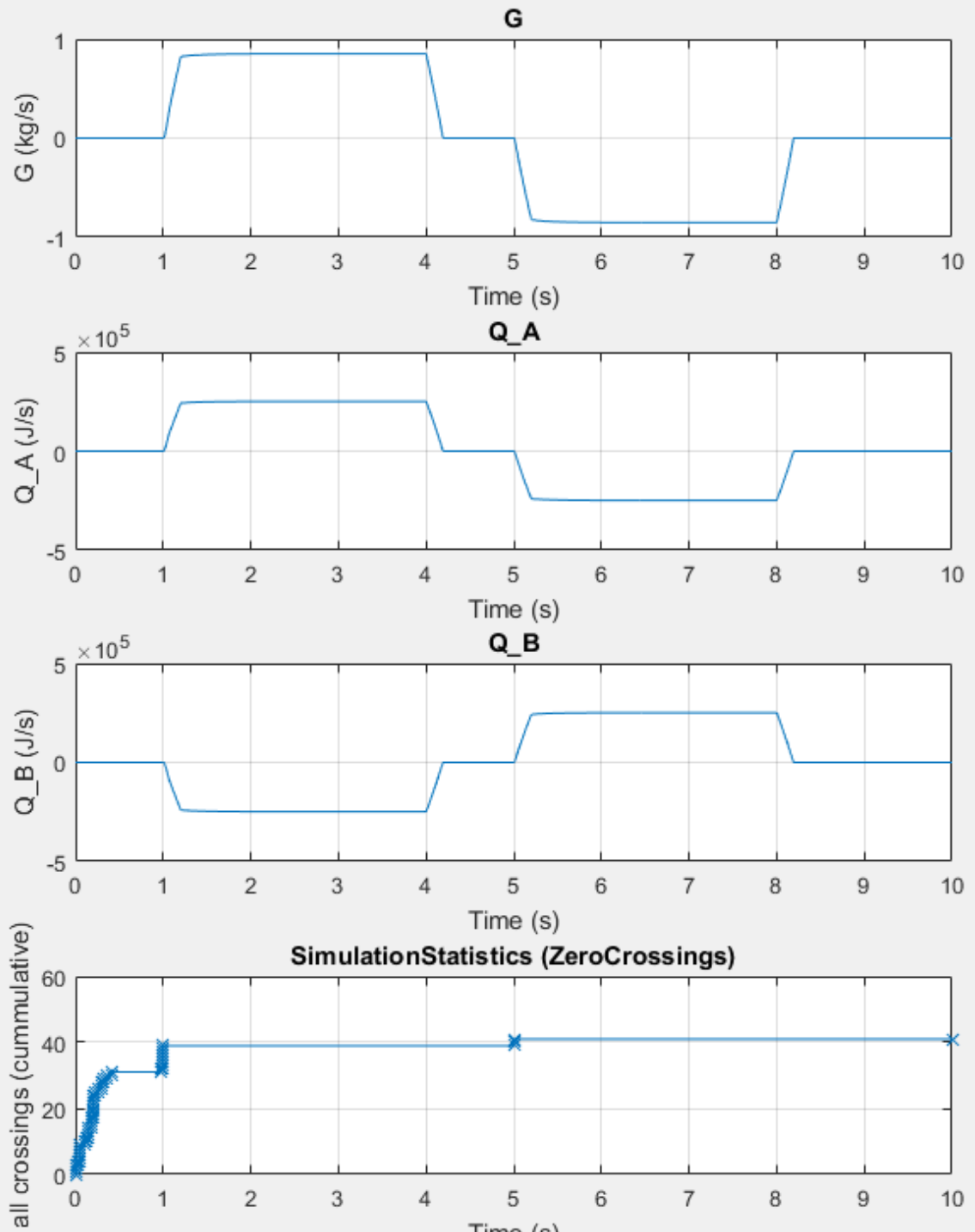
```
| +-Area_A_R (12 signals, 2 crossings)
| +-Area_B_S (12 signals, 2 crossings)
| +-Area_P_A (12 signals, 2 crossings)
| +-Area_P_B (12 signals, 2 crossings)
| +-Variable_Area_Orifice_1 (24 signals, 6 crossings)
| +-Variable_Area_Orifice_2 (24 signals, 8 crossings)
| +-Variable_Area_Orifice_3 (24 signals, 8 crossings)
| +-Variable_Area_Orifice_4 (24 signals, 6 crossings)
+-Friction_Load (2 signals, 4 crossings)
+-Pipe_1 (2 signals, 0 crossings)
| +-Constant_Chamber (2 signals, 0 crossings)
+-Pipe_2 (2 signals, 0 crossings)
| +-Constant_Chamber (2 signals, 0 crossings)
+-Pneumatic_Motor (5 signals, 61 crossings)
```

The results show that many of the 101 detected zero crossings occur in the Directional 5-way valve block (36 crossings) and the Pneumatic Motor block (61 crossings).


- 2 Use the `sscexplore` function to open the Simscape Results Explorer to interact with logged simulation data.

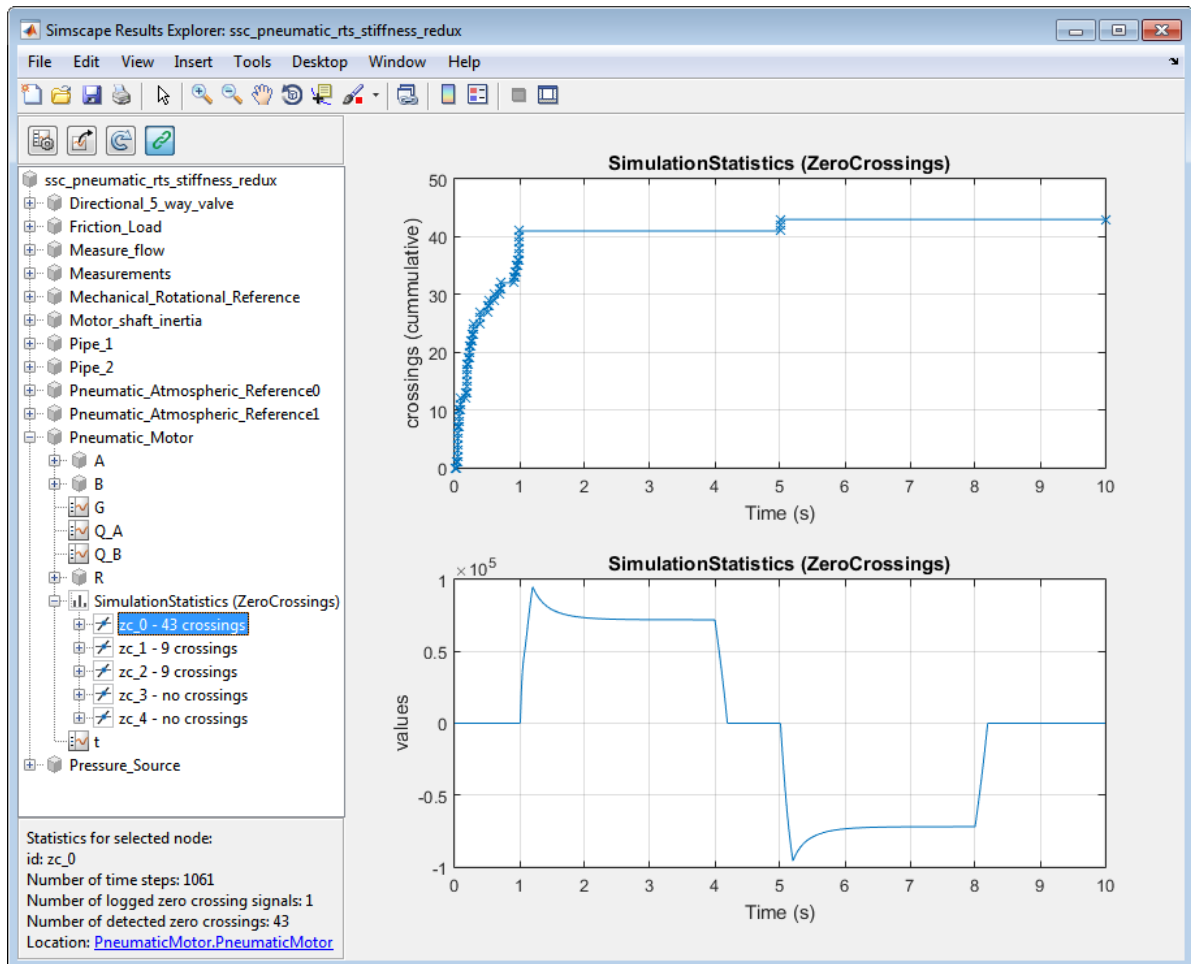
```
sscexplore(simlogRef)
```

- 3 In the results tree, select the `Pneumatic_Motor` node 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 $t = 5$ and 10 seconds.

- 4 To identify the source code that triggers most of the zero crossings, expand the `Pneumatic_Motor` node and the `SimulationsStatistics (ZeroCrossings)` node. Click the `zc_0 - 43 crossings` **Zero Crossing** button  and then the **PneumaticMotor.PneumaticMotor** 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:

```
% Flow direction - "dir" is positive if flow is from port A to port B, otherwise negative
dir = if A.p>B.p, 1 else -1 end;
```

The conditional statement that is responsible for the zero crossings is related to the flow direction in the pipe blocks. The Pipe 1 and Pipe 2 blocks simply model

pneumatic pipes with constant parameters. The Directional 5-way valve block feeds the pipe blocks.

- 5 Look under the mask of the valve block. The spool position dictates the orifice that receives the gas flow, and therefore, the direction of flow in the pipes that lead to the motor.
- 6 Open the Signal Builder block that determines the spool position. Between the simulation time, t , of 0 and 1 second, the signal puts the spool at position 0. The zero crossings reflect the chatter that occurs as the system fills.
- 7 Decrease the chatter, and therefore the number of zero crossings, by increasing the leakage area of the Directional 5-way valve. Open the Directional 5-way valve block dialog box and specify $1e-2$ for the **Leakage area at $x=0$ (mm²)**.

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.
- 2 Print the zero crossing data.

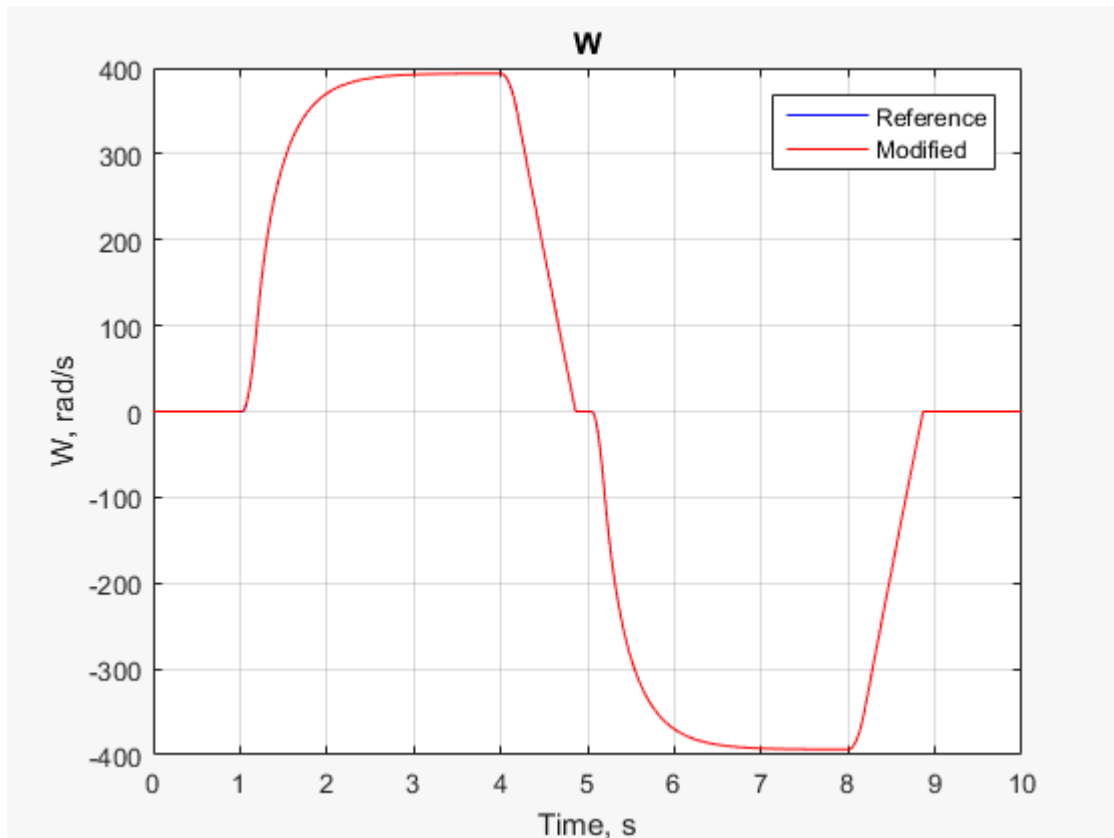
```
sscprintzcs(simlog)

ssc_pneumatic_rts_stiffness_redux (155 signals, 65 crossings)
+-Directional_5_way_valve (144 signals, 28 crossings)
| +-Area_A_R (12 signals, 2 crossings)
| +-Area_B_S (12 signals, 2 crossings)
| +-Area_P_A (12 signals, 2 crossings)
| +-Area_P_B (12 signals, 2 crossings)
| +-Variable_Area_Orifice_1 (24 signals, 4 crossings)
| +-Variable_Area_Orifice_2 (24 signals, 6 crossings)
| +-Variable_Area_Orifice_3 (24 signals, 6 crossings)
| +-Variable_Area_Orifice_4 (24 signals, 4 crossings)
+-Friction_Load (2 signals, 4 crossings)
+-Pipe_1 (2 signals, 0 crossings)
| +-Constant_Chamber (2 signals, 0 crossings)
+-Pipe_2 (2 signals, 0 crossings)
| +-Constant_Chamber (2 signals, 0 crossings)
+-Pneumatic_Motor (5 signals, 33 crossings)
```

The overall number of zero crossings has decreased from 101 to 65. The number of zero crossings in the motor has decreased from 50 to 33.

- 3 To compare the results, use 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.

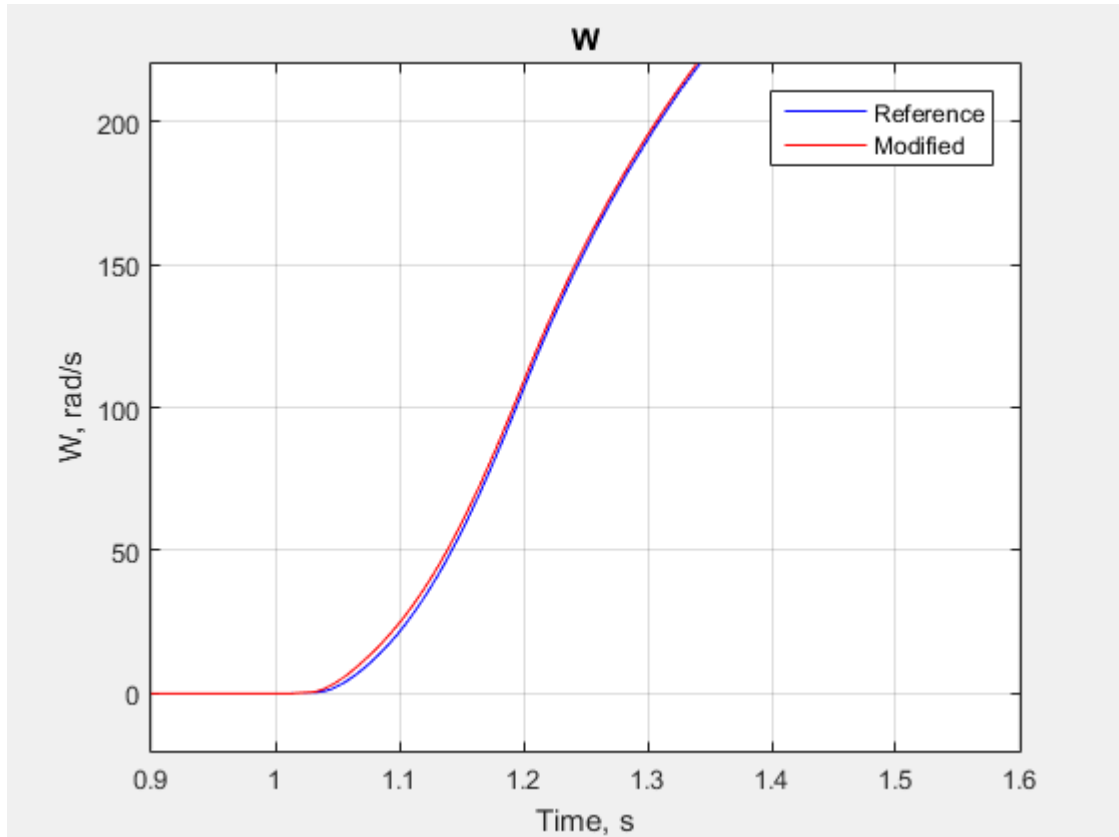
- 4 Zoom control for a closer look at the inflection point at $t \approx 5$ seconds.

```
xStart = 0; xEnd = 10; yStart = -400; yEnd = 400;
xZoomStart1 = .9; xZoomEnd1 = 1.6;
```

```

yZoomStart1 = -20; yZoomEnd1 = 220;
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.

- 5 Examine the step size for the simulation with the modified model.

```

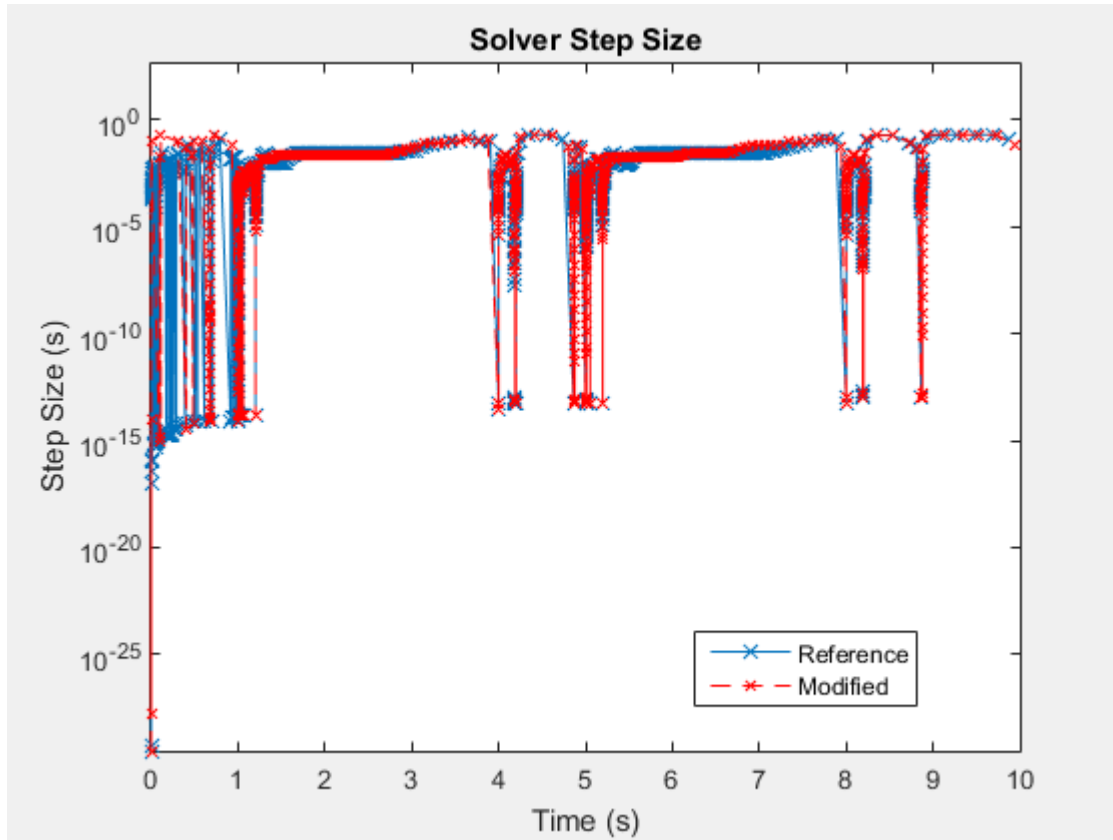
h1;
axis([xStart xEnd yStart yEnd]);
h1;
hold on;
semilogy(tout(1:end-1),diff(tout),'--x', 'Color','r',...

```

```

'LineWidth',.1,'MarkerSize',5);
title('Solver Step Size');
xlabel('Time (s)');
ylabel('Step Size (s)');
h1Leg2 = legend({'Reference','Modified'},[300,65,0.3,0.7]);

```



You have reduced the number of zero crossings by eliminating most of the chatter at the beginning of the simulation. Events still slow the simulation at $t \approx 1, 4, 5, 8,$ and 9 seconds. 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.sli.findNode` | `sscexplore` | `sscprintzcs`

Related Examples

- “Determine Step Size” on page 7-15
- “Estimate Computation Costs” on page 7-76
- “Improving Speed and Accuracy” on page 7-10
- “Log, Navigate, and Plot Simulation Data” on page 9-21
- “Reduce Computation Costs” on page 7-25
- “Reduce Numerical Stiffness” on page 7-31

More About

- “About Simulation Data Logging” on page 9-2
- “About the Simscape Results Explorer” on page 9-26
- “Events and Zero Crossings” on page 4-3
- “Model Preparation Objectives” on page 7-2
- “Improving Speed and Accuracy” on page 7-10
- “Real-Time Model Preparation Workflow” on page 7-5
- “Using Conditional Expressions in Equations”
- “Zero-Crossing Detection”

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 solver ode14x, specify the number of Newton's iterations.

To obtain accurate results, for both local and global solvers start with two or three iterations and increase as required.

Related Examples

- “Choose Step Size and Number of Iterations” on page 7-79

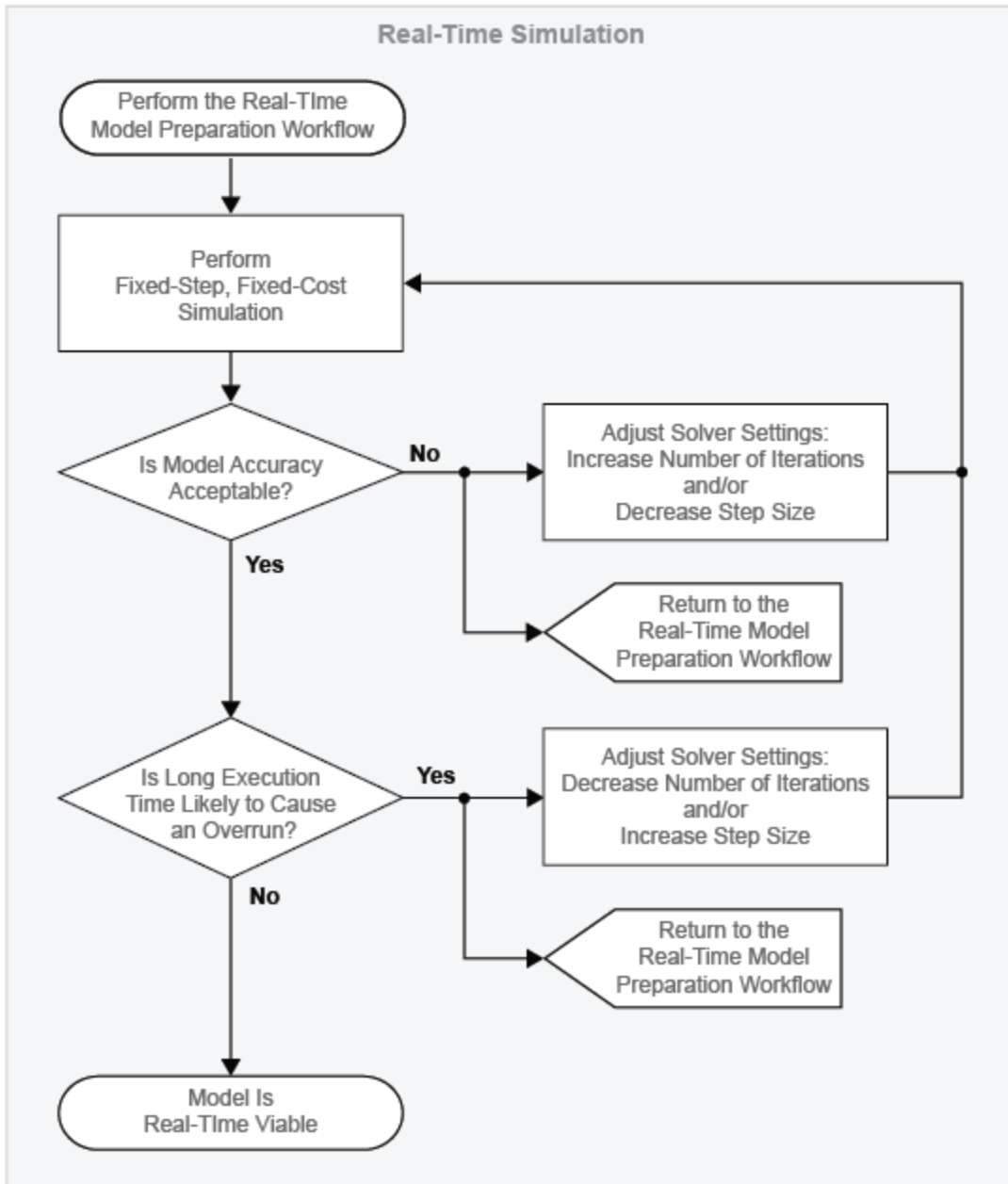
More About

- “Simulating with Fixed Cost” on page 4-22
- “Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers” on page 4-21

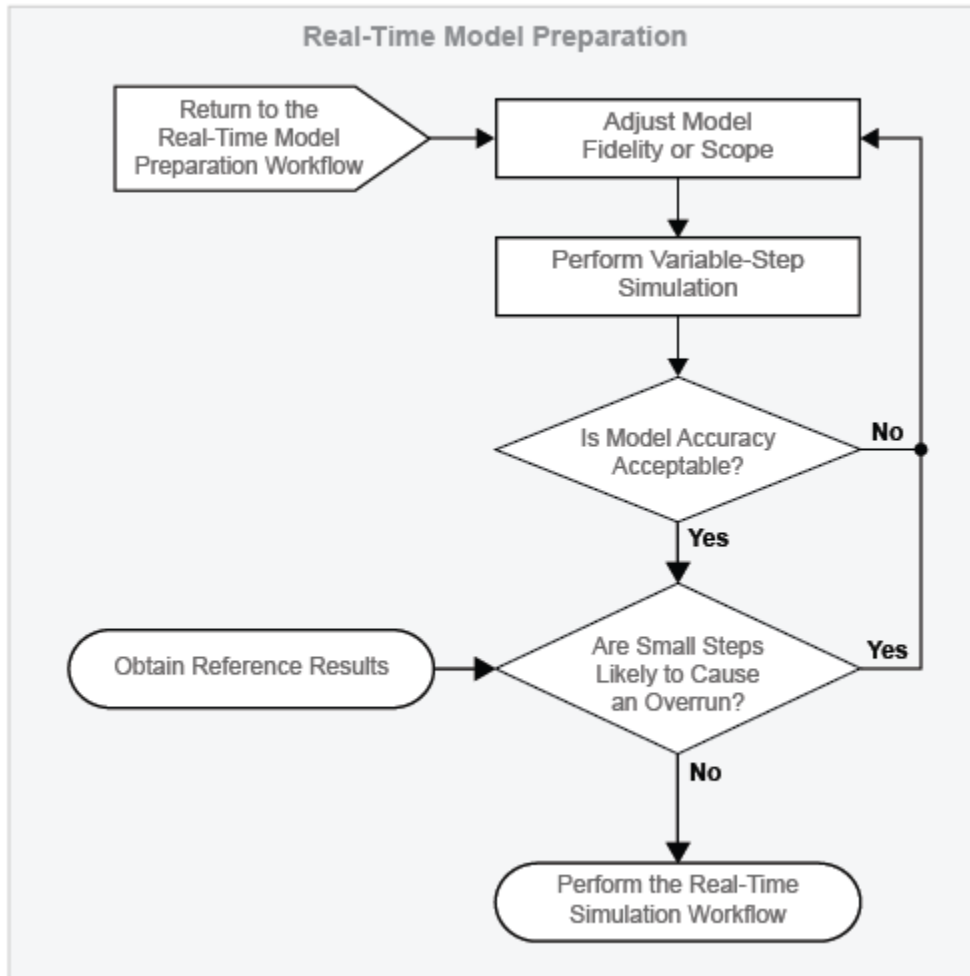
- “Solvers for Real-Time Simulation” on page 7-63

Real-Time Simulation Workflow

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 7-5. 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 7-79 and “Fixed-Cost Simulation for Real-Time Viability” on page 7-55.

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), 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 7-5.

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 7-76.

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, 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), 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 “What Is Hardware-in-the-Loop Simulation?” on page 7-96.

Insufficient Computational Capability for Real-Time Viability

It is possible that your real-time target lacks the computational capability for running your model in real time. If, after multiple iterations of the workflow, it appears that there is no combination of model complexity and solver settings that can make your model real-time viable, consider these options for increasing processing power:

- “Upgrading Target Hardware” on page 7-13
- “Simulating Parts of the System in Parallel” on page 7-13

Related Examples

- “Choose Step Size and Number of Iterations” on page 7-79
- “Determine System Stiffness” on page 7-68
- “Estimate Computation Costs” on page 7-76

More About

- “Fixed-Cost Simulation for Real-Time Viability” on page 7-55
- “Hardware-in-the-Loop Simulation Workflow” on page 7-100
- “Improving Speed and Accuracy” on page 7-10
- “Real-Time Model Preparation Workflow” on page 7-5
- “Solvers for Real-Time Simulation” on page 7-63
- “What Is Hardware-in-the-Loop Simulation?” on page 7-96

Solvers for Real-Time Simulation

In this section...

“Choosing Between Discrete and Continuous Solvers” on page 7-64

“Computational Cost for Continuous Solvers” on page 7-64

“How Numerical Stiffness Affects Solver Choice” on page 7-65

“Using Simscape Local Fixed-Step Solvers” on page 7-66

To run your model on a real-time target, 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 will use 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)

Realm	Type	Numerical Method	Solver
		Implicit	ode14x (extrapolation)
	Discrete	Not applicable	Discrete (no continuous states)
Simscape local network	Continuous	Implicit	Backward Euler
			Trapezoidal Rule

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 a controller, see Hydraulic Actuator Configured for HIL Testing.

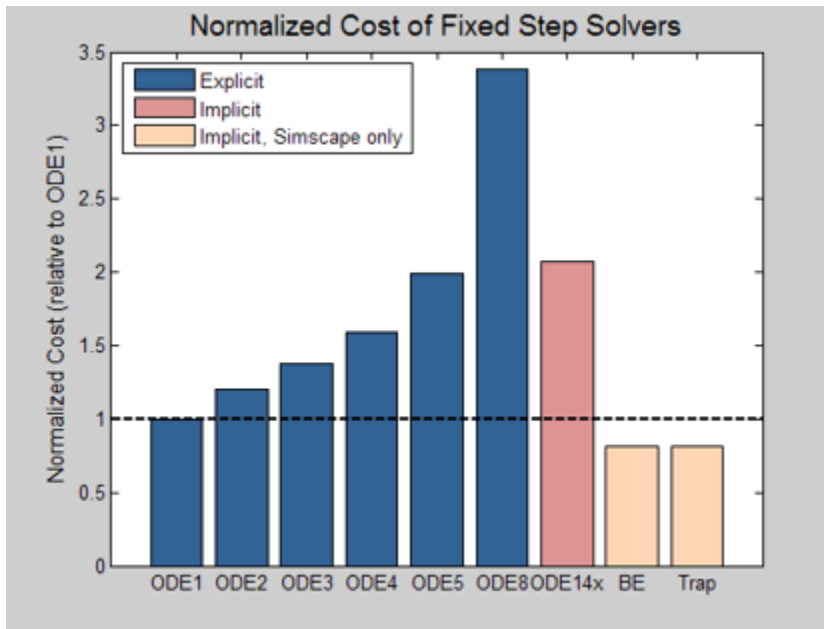
Note: A physical network using a local solver appears to the global Simulink solver as if it has discrete states.

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 all 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 7-68.

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.
- 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 7-79.

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 two Simscape fixed-step solvers for real-time simulation. Both are implicit.

- Backward Euler

- Trapezoidal Rule

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 those found in electrical systems with AC waveforms. 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 7-68
- “Reduce Numerical Stiffness” on page 7-31
- “Choose Step Size and Number of Iterations” on page 7-79

More About

- “Choose a Fixed-Step Solver”
- “Fixed-Cost Simulation for Real-Time Viability” on page 7-55
- “Making Optimal Solver Choices for Physical Simulation” on page 4-21
- “Solver Classification Criteria”
- “Solvers”

Determine System Stiffness

In this section...

“Obtain Reference Results” on page 7-68

“Simulate with an Implicit Fixed-Step Solver” on page 7-69

“Simulate with an Explicit Fixed-Step Solver” on page 7-71

“Analyze the Results” on page 7-73

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 explicit 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:

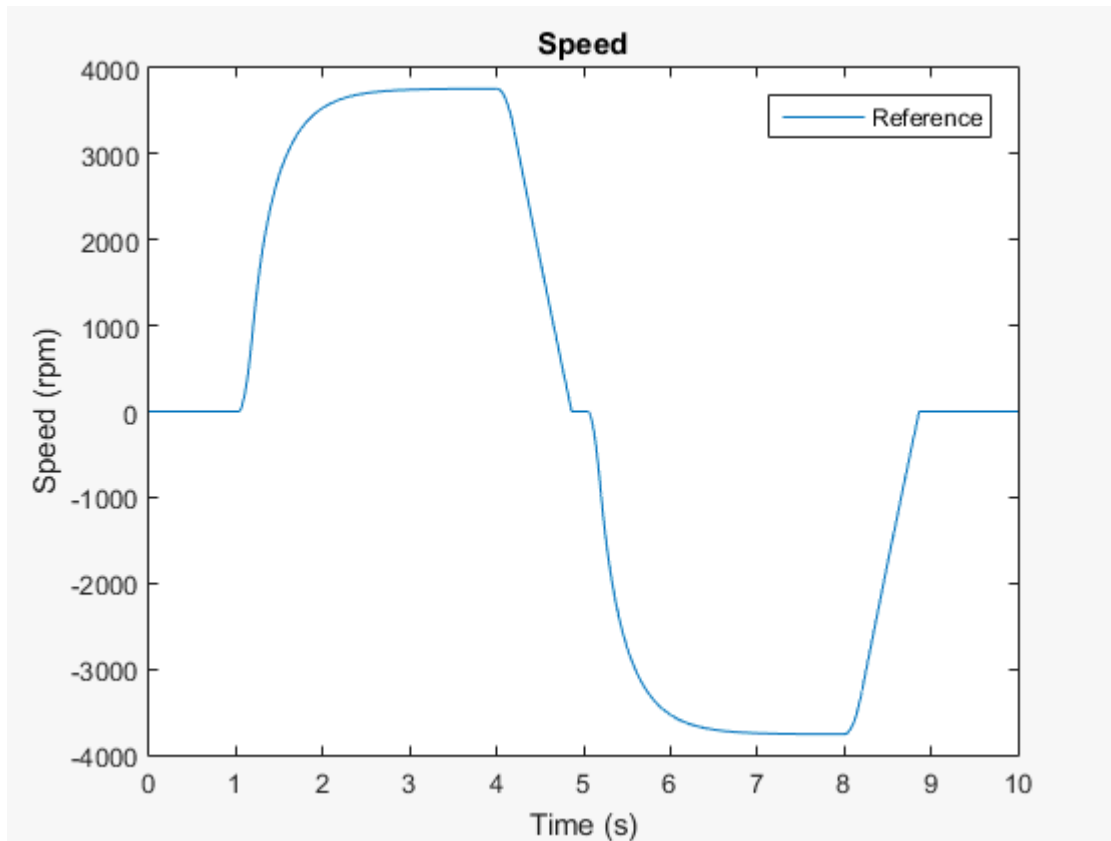
```
ssc_pneumatic_rts_reference
```

- 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 **OK**.

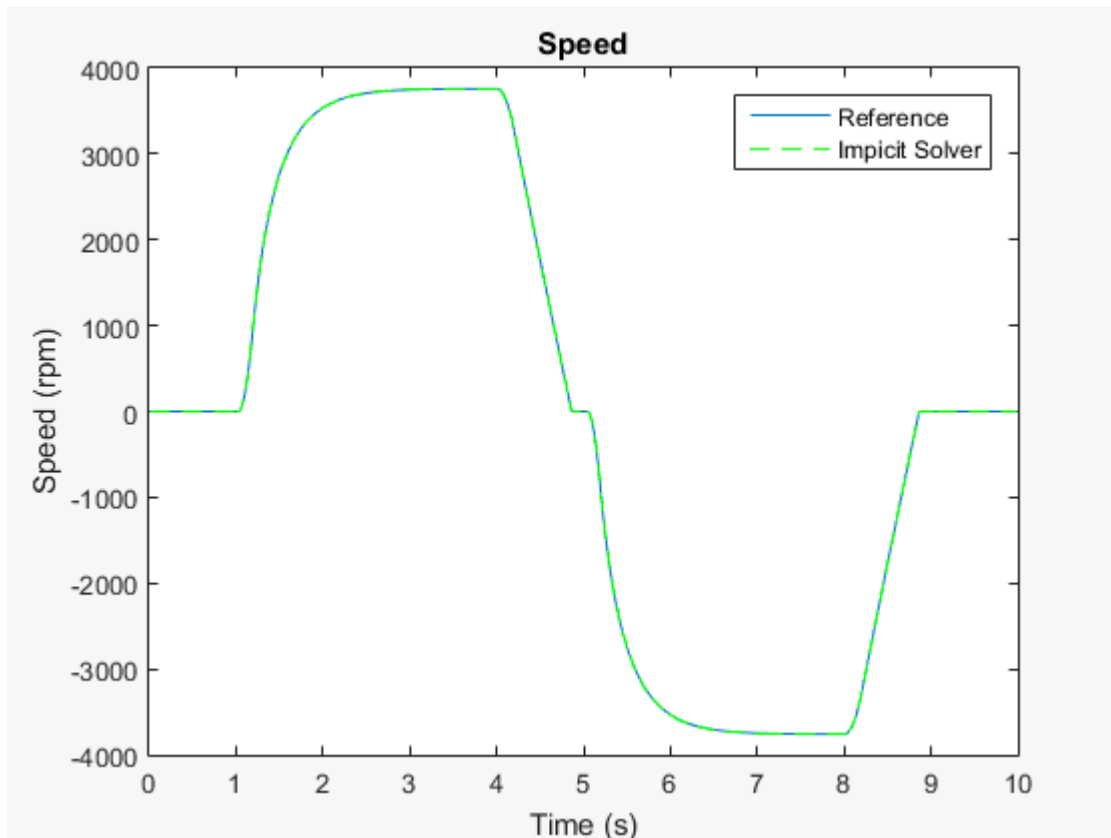
- 2 Simulate the model.
- 3 Assign the simulation results to new variables.

```
y0de14x = yout;  
t0de14x = 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(t0de14x,y0de14x,'g--');  
h1Leg = legend({'Reference','Implicit Solver'});
```

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)

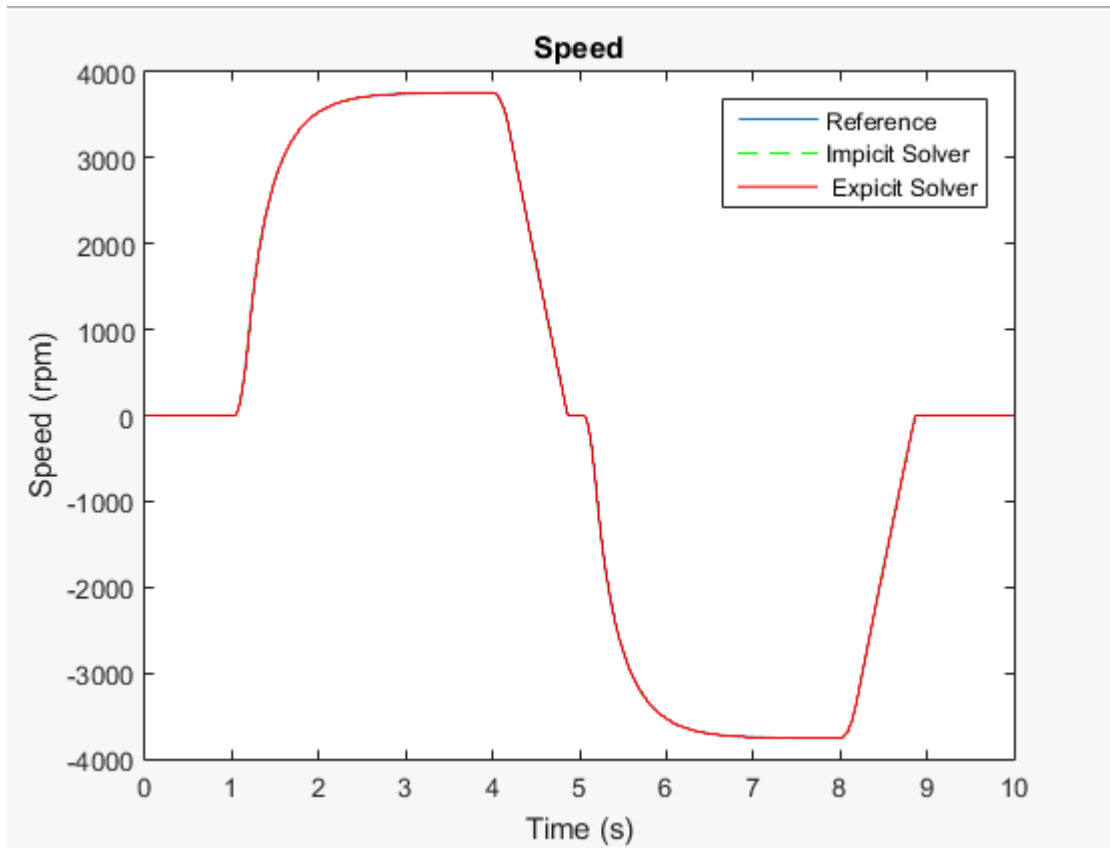
Click **OK**.

- 2 Filter the input signal to provide the required input derivative for the explicit solver. In the PS-S Simulink Converter block dialog box, 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.

```
y0de5 = yout;  
t0de5 = tout;
```

- 5 Use the `stairs` function to plot the results of the explicit fixed-step simulation.

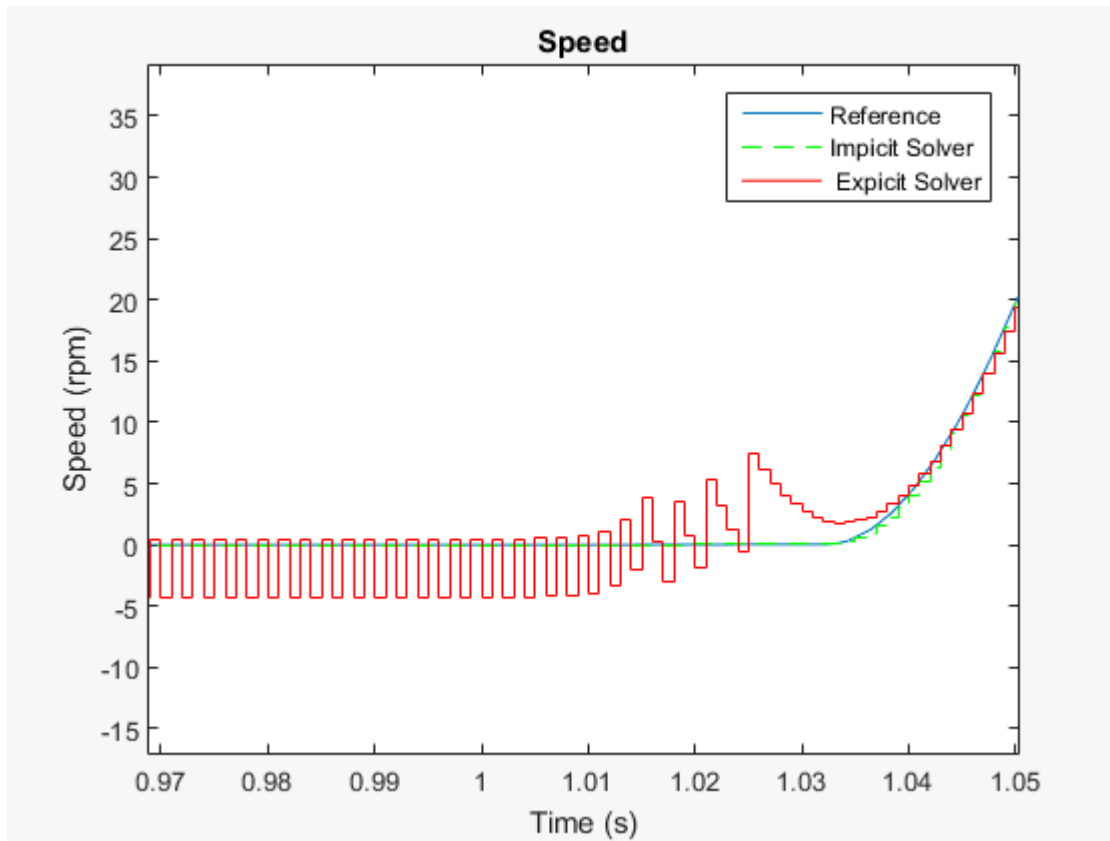
```
h1;  
hold on;  
stairs(t0de5,y0de5,'r-');  
h1Leg = legend({'Reference', 'Implicit Solver', 'Explicit Solver'});
```



The results appear the same.

Analyze the Results

- 1 To see the results more closely, zoom to the inflection point at 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 7-31
- “Determine Step Size” on page 7-15

More About

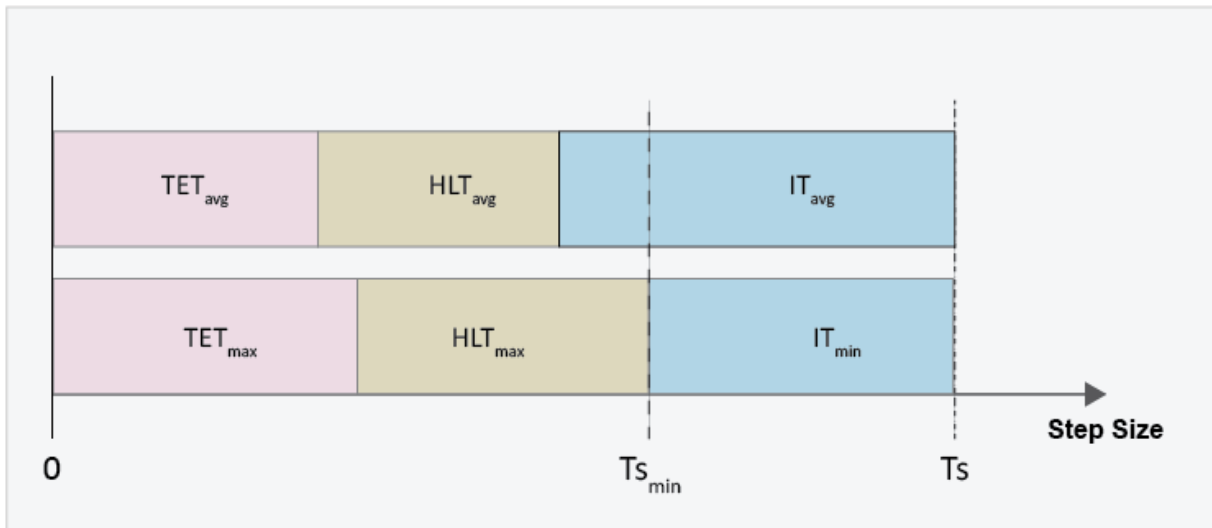
- “Filtering Input Signals and Providing Time Derivatives” on page 4-14
- “Real-Time Model Preparation Workflow” on page 7-5
- “Solvers for Real-Time Simulation” on page 7-63
- “Stiffness” on page 4-3

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 a real-time hardware target, estimate the simulation execution-time budget for your real-time target.

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 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 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 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

$$T_{s_{\min}} = TET_{\max} + HLT_{\max},$$

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.
- *THL* is the hardware latency time. Hardware latency time includes scheduling, interrupt and input/output (I/O) latency.
- $T_{s_{\min}}$ is the minimum step size.

If the time that it takes for the target 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,

$$T_s = TET_{\max} + HLT_{\max} + IT,$$

where

- *T_s* is the step size that you specify for the fixed-step solver.
- *IT* is the idle time.

This equation can be rearranged as:

$$TET_{\max} = T_s - HLT_{\max} - IT,$$

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:

$$TET_{\max} = T_s - HLT_{\max} - [(SMT) * (T_s)],$$

where

- *SMT* is the desired safety margin, specified as a percent.

Related Examples

- “Reduce Computation Costs” on page 7-25

More About

- “Fixed-Cost Simulation for Real-Time Viability” on page 7-55
- “Simulation Phases in Dynamic Systems”

Choose Step Size and Number of Iterations

In this section...

“Obtain Reference Results” on page 7-80

“Determine Maximum Step Size for Accurate Results” on page 7-82

“Parameterize Global and Local Solver Settings” on page 7-84

“Perform Fixed-Step, Fixed-Cost Simulation” on page 7-85

“Adjust Solver Settings to Improve Accuracy” on page 7-89

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, specify a combination of step size (T_s) 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 T_s 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:

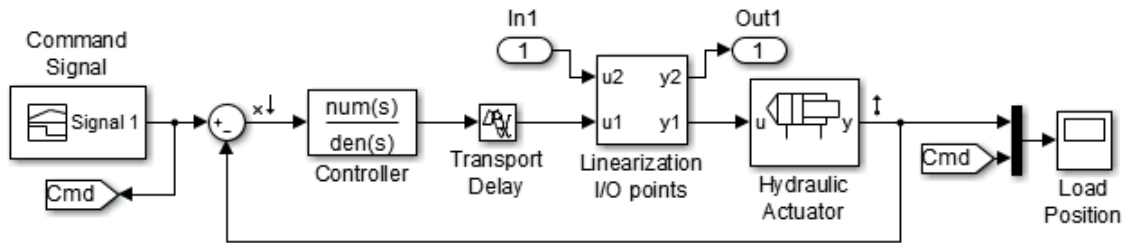
- Obtain reference results by performing variable-step simulation on a model of a hydraulic actuator.
- Use a modified version of the model to determine the maximum step size to use to achieve accurate enough results from a fixed-step, fixed-cost simulation. Fixed-step, fixed-cost simulation is required 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:

```
ssc_hydraulic_actuator_digital_control
```

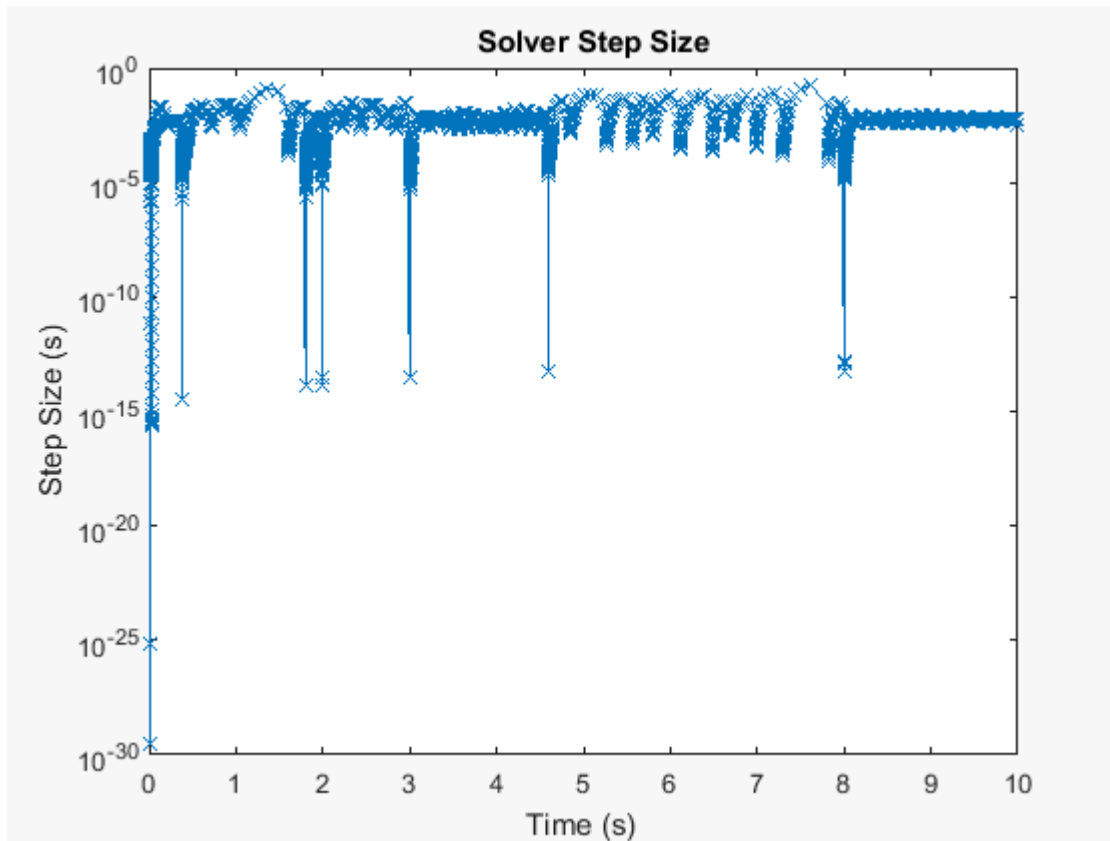


- 2 Simulate the model.
- 3 Extract the data for pressure and simulation-step time from the logged Simscape node.

```
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;
```

- 4 Plot the step size.

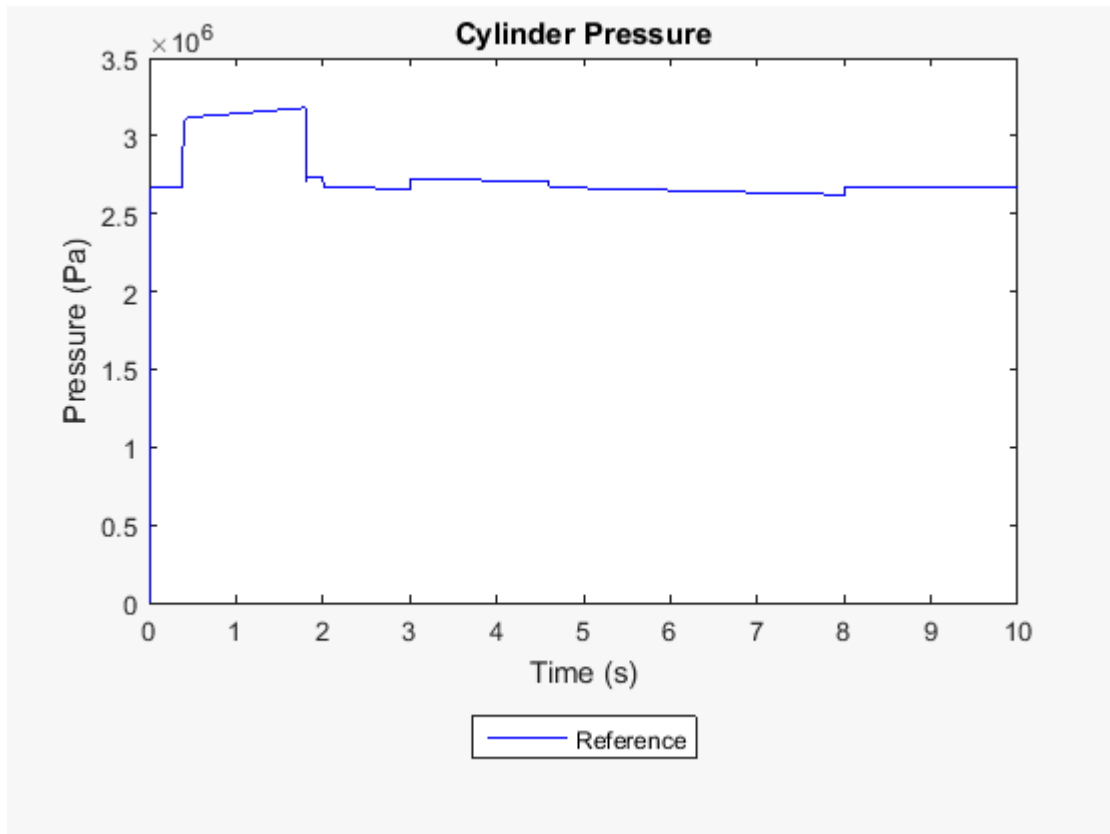
```
h1 = figure;
semilogy(tRef(1:end-1),diff(tRef),'-x')
title('Solver Step Size');
xlabel('Time (s)');
ylabel('Step Size (s)');
```



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 7-15.

5 Plot the simulation results.

```
h2 = figure;
plot(tRef,pRef, 'b-');
h2Legend1 = legend({'Reference'}, 'Location', 'southoutside');
title('Cylinder Pressure');
xlabel('Time (s)');
ylabel('Pressure (Pa)');
```

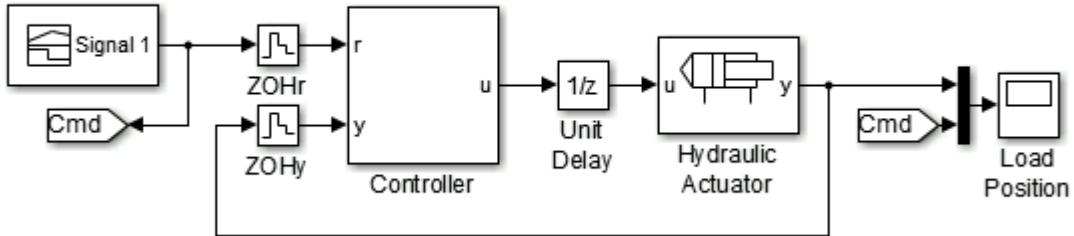


Determine Maximum Step Size for Accurate Results

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](#).

- 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 dialog box, 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;
```

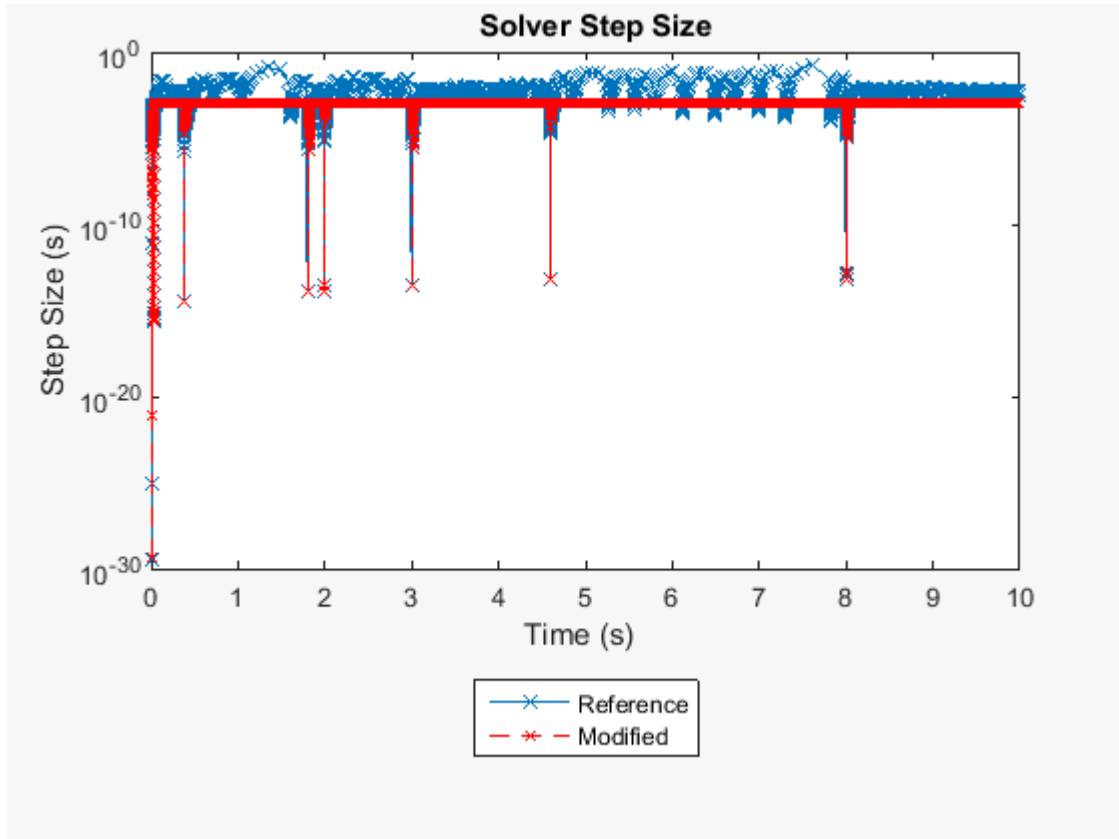
- 5 Plot the step size to the figure that contains the step-size data for the original model.

```
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');

```



The discretized controller makes the modified model less stiff than the original model, but it also reduces the maximum step size for obtaining accurate real-time results. For the discretized model, $T_{s_{max}}$ 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 dialog box 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 dialog box, 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 7-76.

- 1 For the first simulation, specify both the global and local step size as the largest possible value of $T_{s_{max}}$ 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.

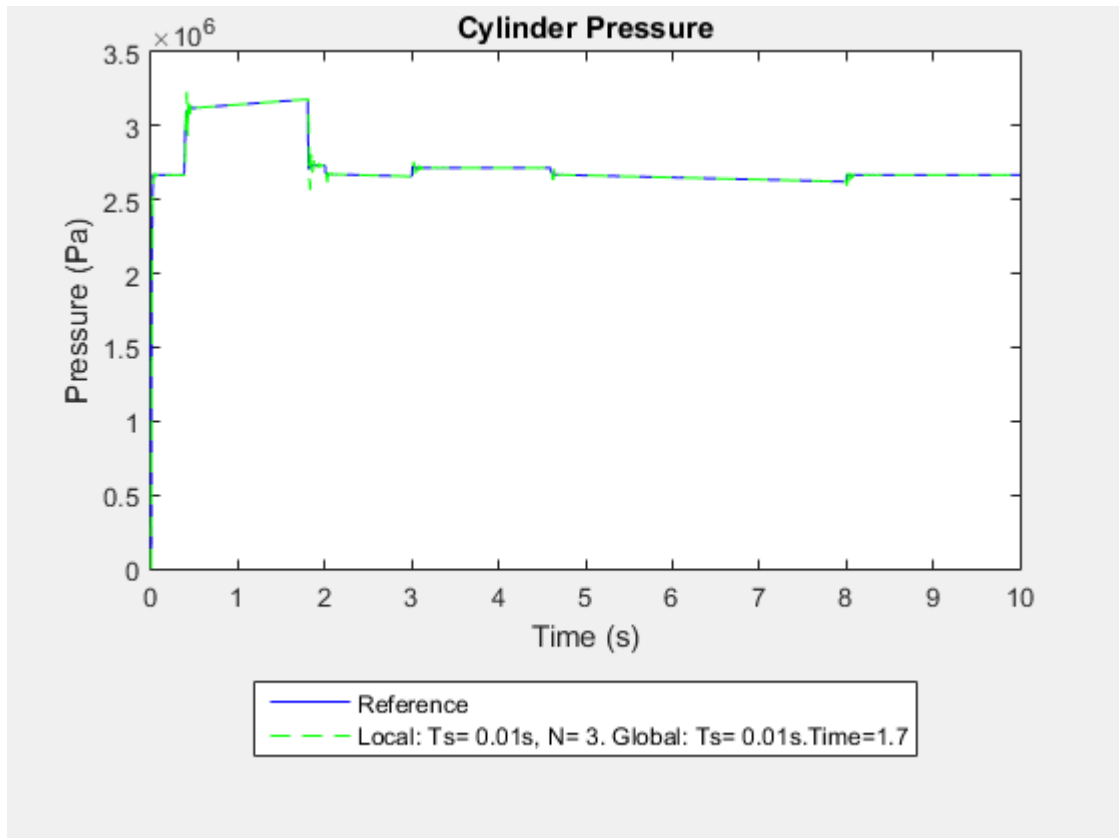
```
tic; sim('ssc_hydraulic_actuator_HIL'); tSim1 = toc;  
time1 = max(tSim1);
```

- 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');
```

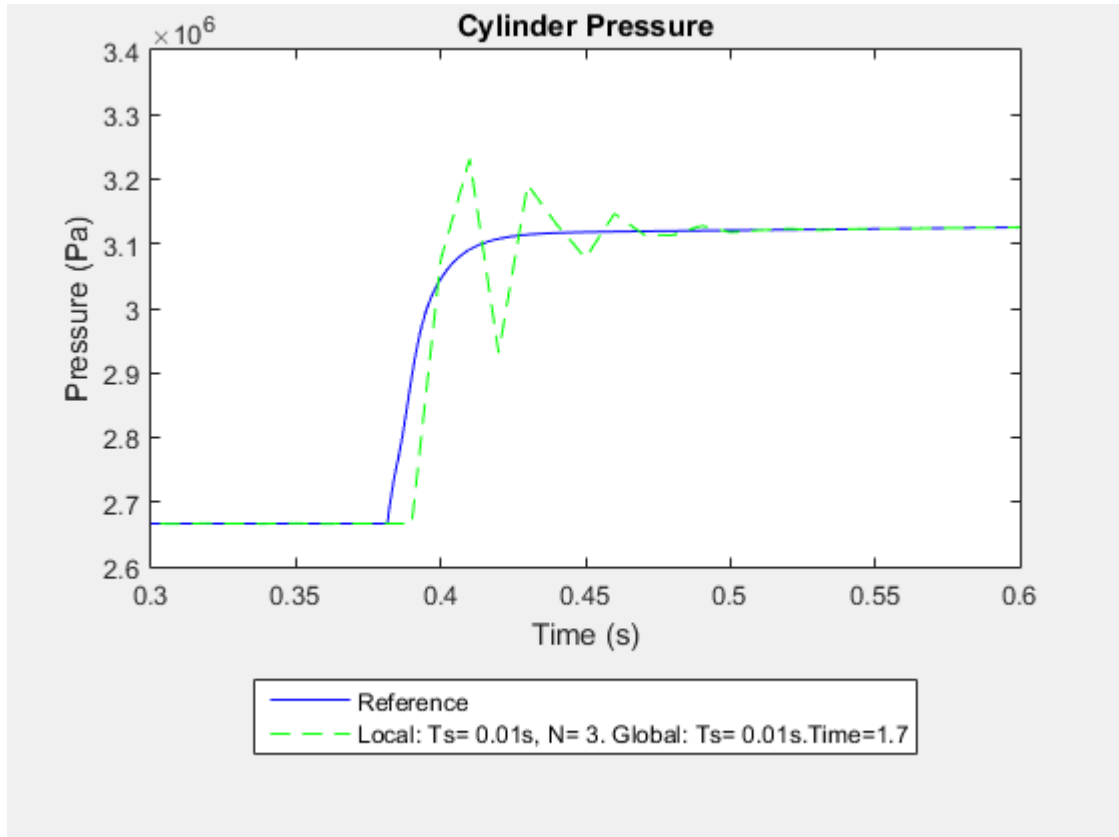



The elapsed time varies because it depends on the immediate computational capacity of the computer that runs the simulation. The elapsed times in the legend are from simulation on a 3.6 GHz Intel[®] CPU with a 16 GB memory. Your legend contains the elapsed time for the simulation on your computer.

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 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 = 3.4e6;  
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 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);
```

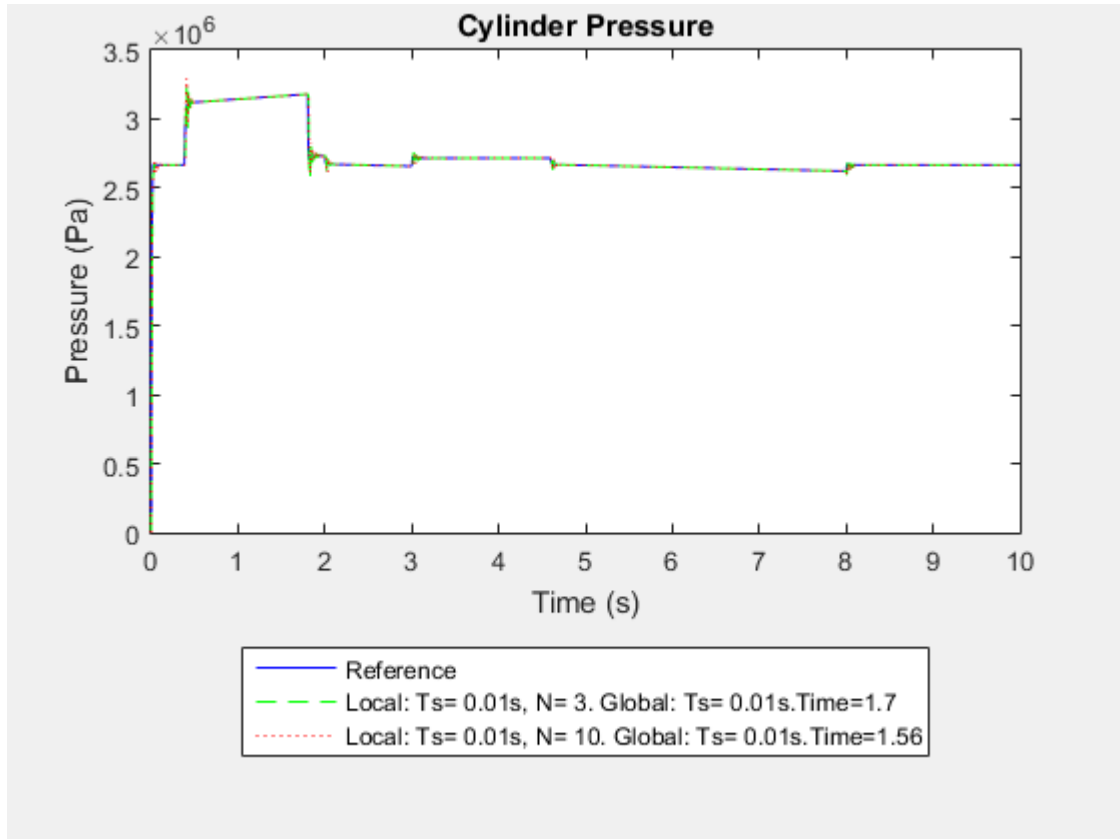
- 3 Extract the pressure and simulation time data.

```
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];
```

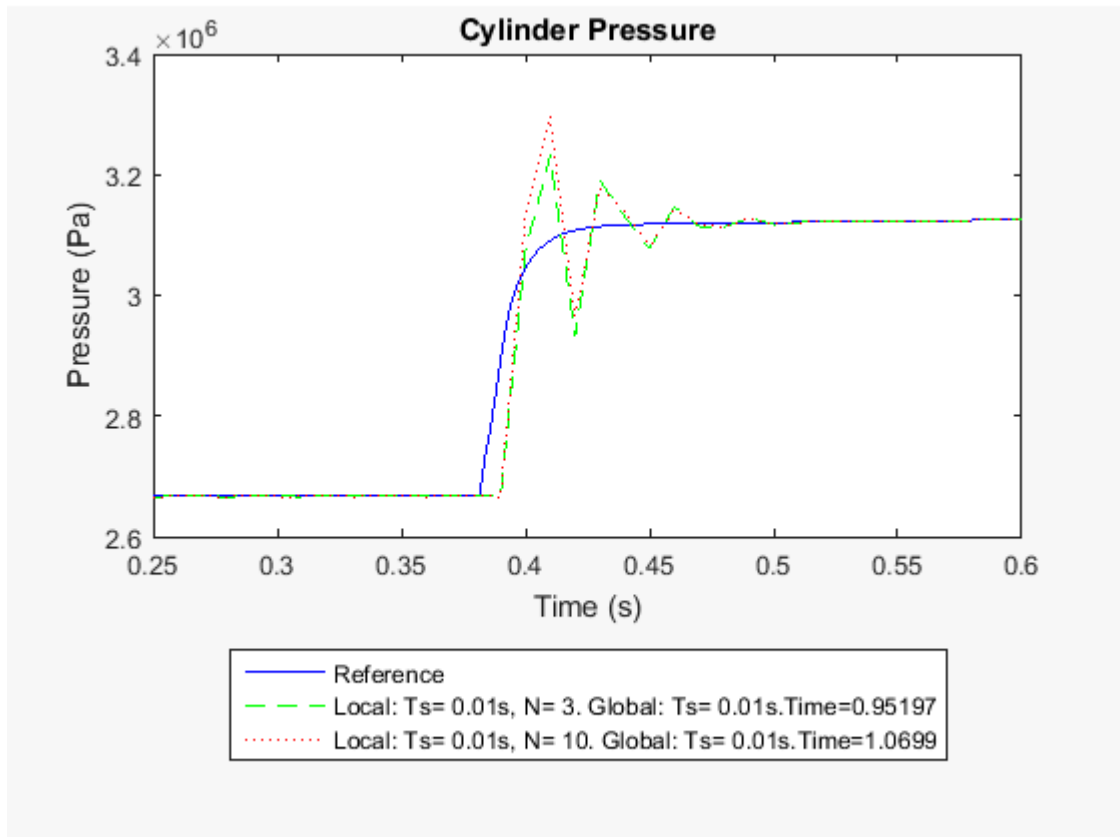
```
h2Legend3 = legend({'Reference', num2str(cfgSim1), num2str(cfgSim2)}, ...
    'Location', 'southoutside');
```



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.

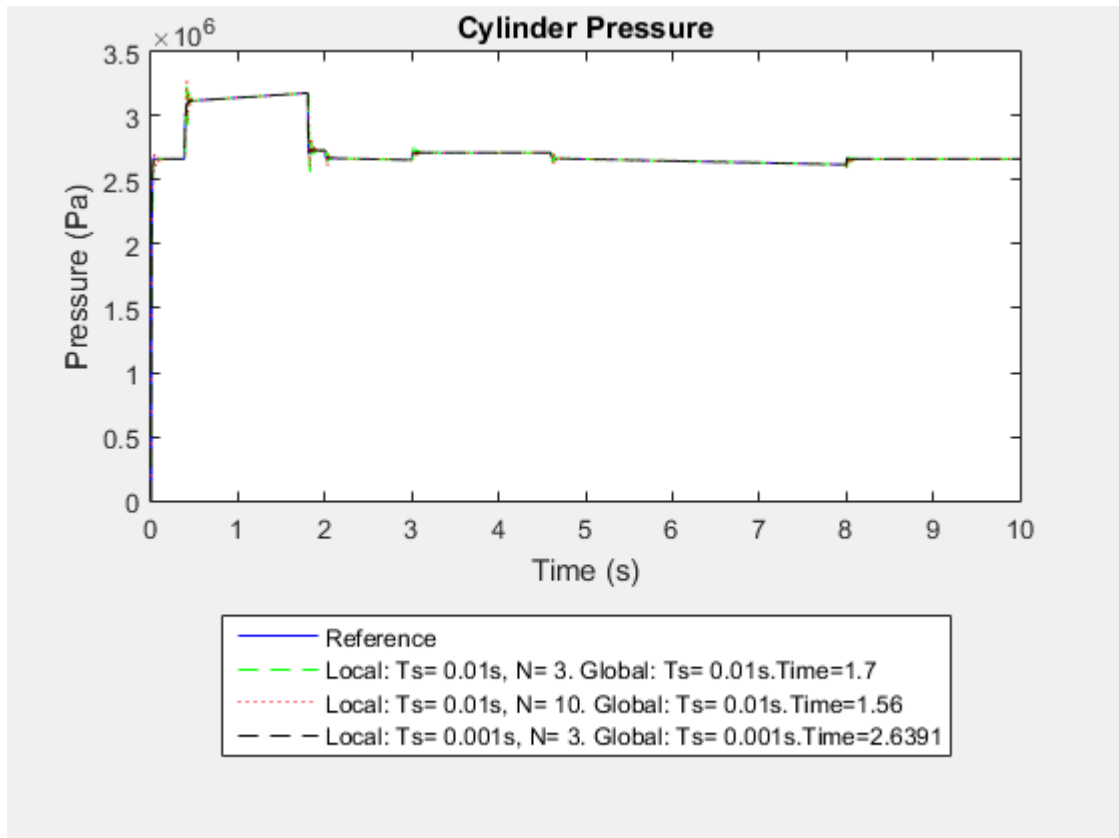
```
tic; sim('ssc_hydraulic_actuator_HIL'); tSim3 = toc;
time3 = max(tSim3);
```

- 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.

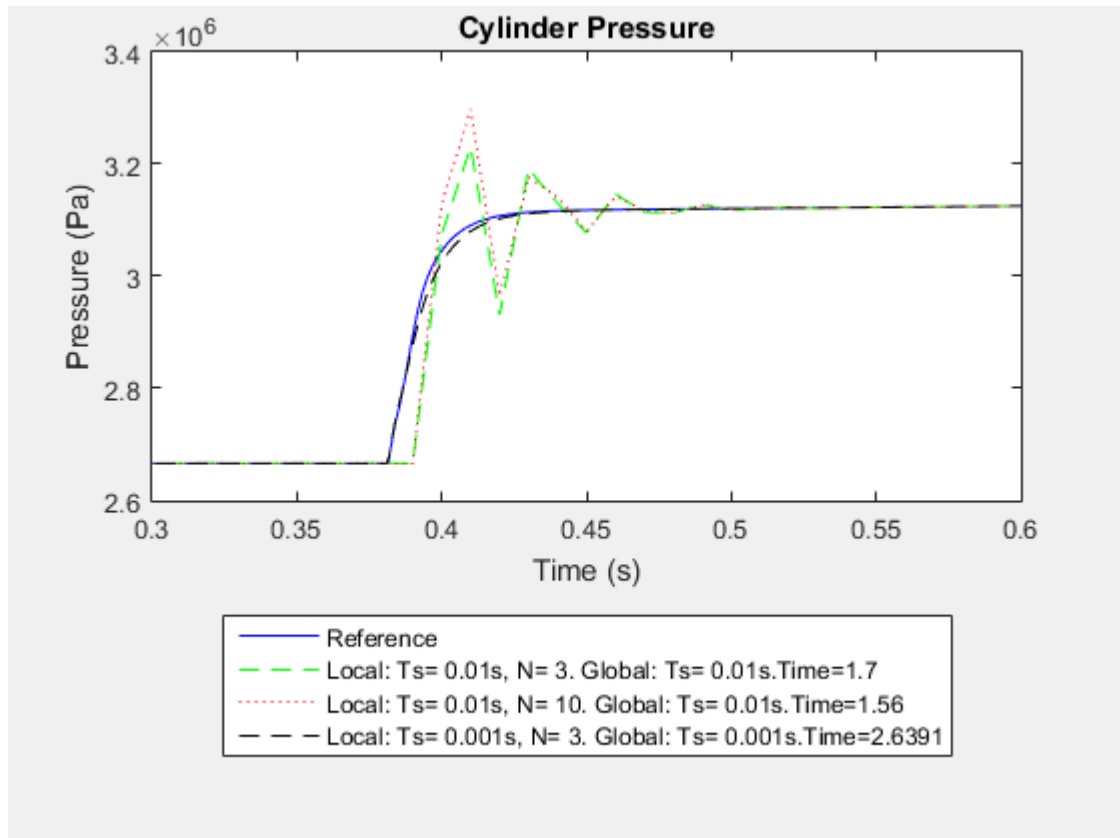
```
figure(h2)
hold on;
plot(tSim3, pSim3, 'k--');
delete(h2Legend3)
axis([xStart xEnd yStart yEnd]);
configSim3L = ['Local: Ts= ', num2str(ts), 's, N= ', num2str(N), '.'];
configSim3G = [' Global: Ts= ', num2str(tsG), 's.'];
timeSim3T = ['Time= ', num2str(time3)];
cfgSim3 = [configSim3L, configSim3G, timeSim3T];
h2Legend4 = legend...
    ({'Reference', num2str(cfgSim1), num2str(cfgSim2), num2str(cfgSim3)}, ...
    'Location', 'southoutside');
```



The simulation takes longer (1.68 s) but is fast enough given the four-second simulation execution-time budget.

10 Zoom to better evaluate accuracy.

```
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. To run your model on a real-time target, 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 7-5.

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 7-13 and “Simulating Parts of the System in Parallel” on page 7-13.

Related Examples

- “Determine Step Size” on page 7-15
- “Determine System Stiffness” on page 7-68
- “Estimate Computation Costs” on page 7-76

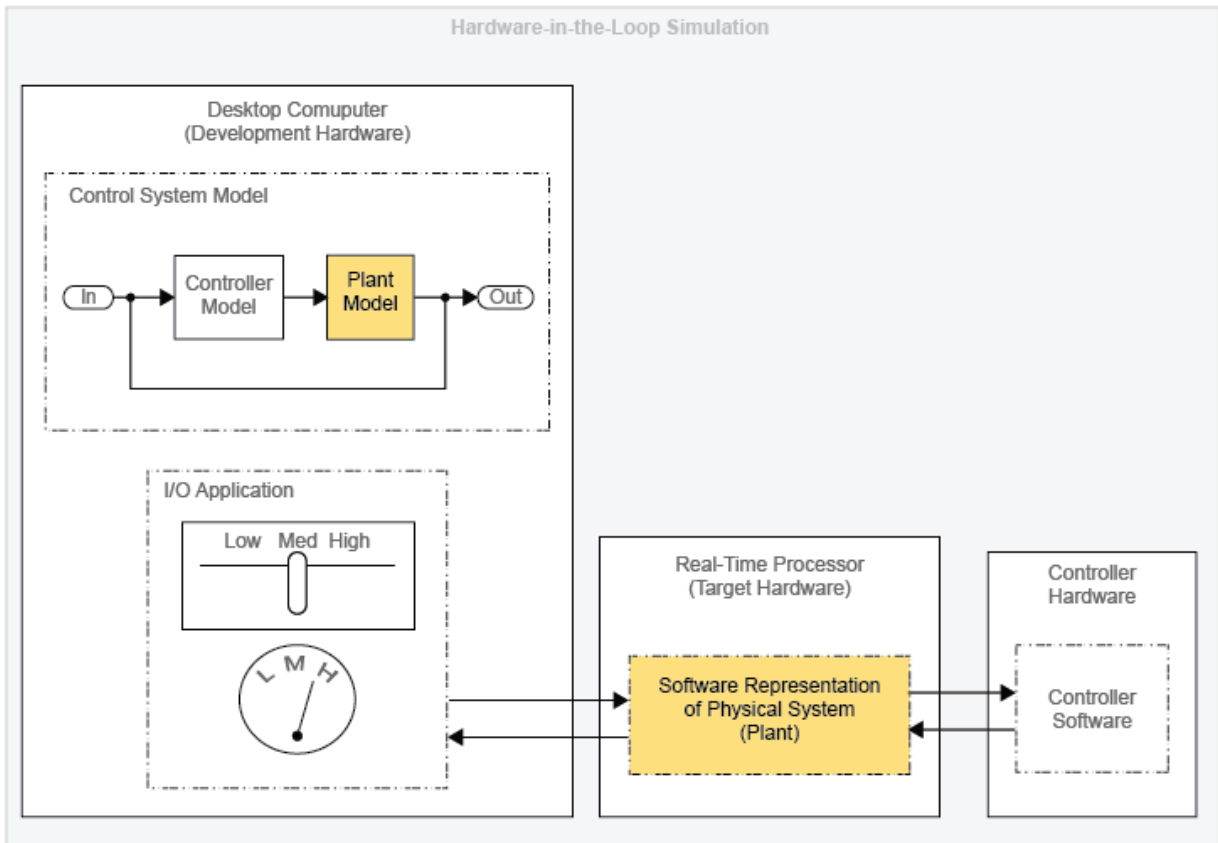
More About

- “Filtering Input Signals and Providing Time Derivatives” on page 4-14
- “Fixed-Cost Simulation for Real-Time Viability” on page 7-55
- “Hardware-in-the-Loop Simulation Workflow” on page 7-100
- “Improving Speed and Accuracy” on page 7-10
- “Log and Plot Simulation Data” on page 9-8
- “Real-Time Model Preparation Workflow” on page 7-5
- “Solvers for Real-Time Simulation” on page 7-63
- “What Is Hardware-in-the-Loop Simulation?” on page 7-96

What Is Hardware-in-the-Loop Simulation?

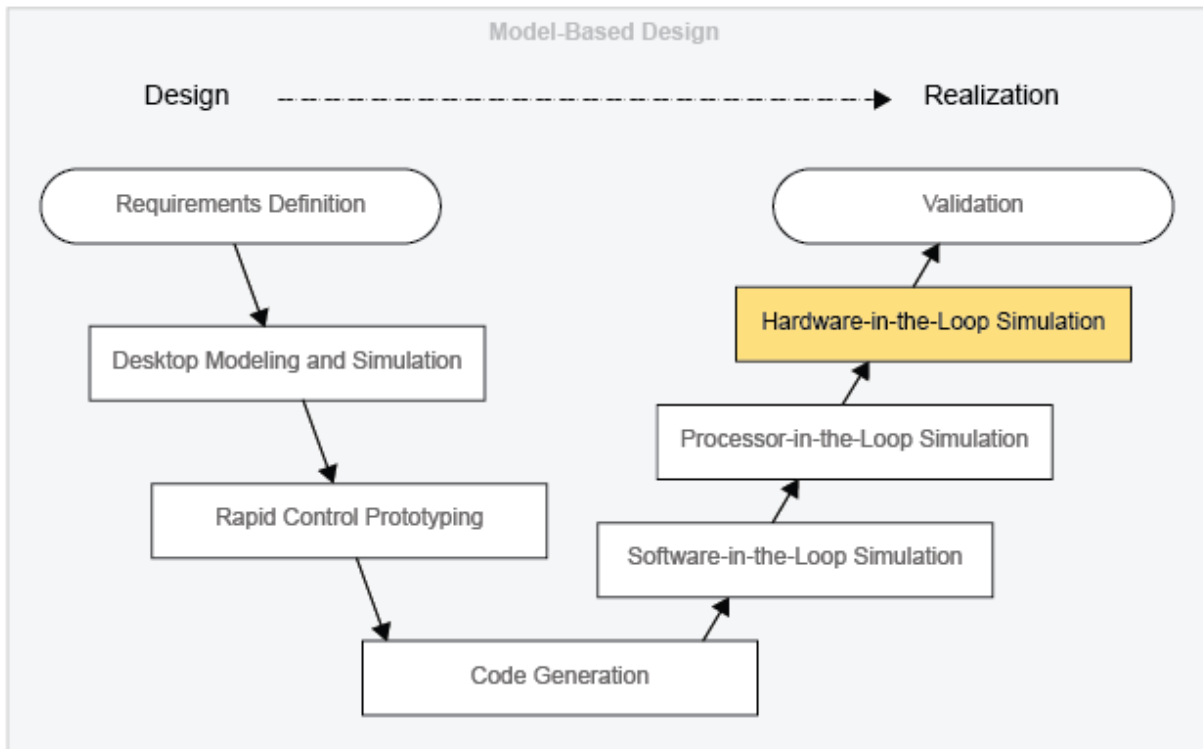
Hardware-in-the-loop (HIL) simulation is a type of real-time simulation. You use HIL simulation to test your controller design. HIL simulation shows how your controller responds, in real time, to realistic virtual stimuli. You can also use HIL to determine if your physical system (plant) model is valid.

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.



Why Perform 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 when it comes to 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.

Related Examples

- “Generate, Download, and Execute Code” on page 7-108

More About

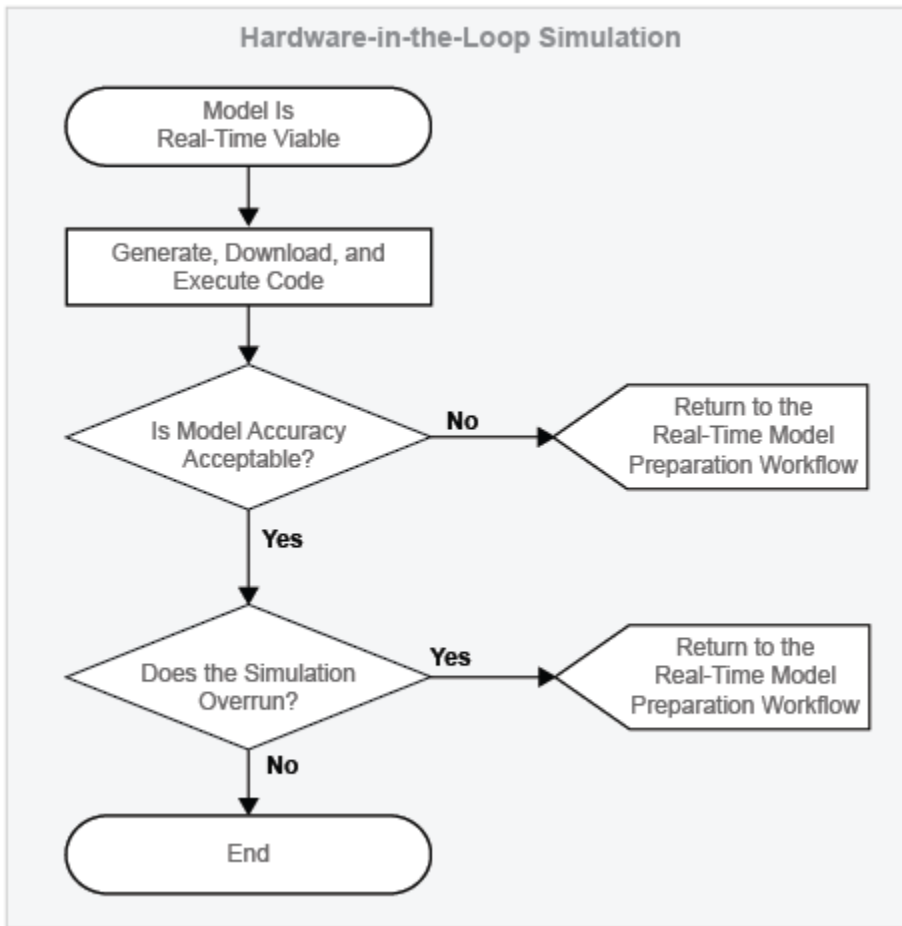
- “Hardware-in-the-Loop Simulation Workflow” on page 7-100

Hardware-in-the-Loop Simulation Workflow

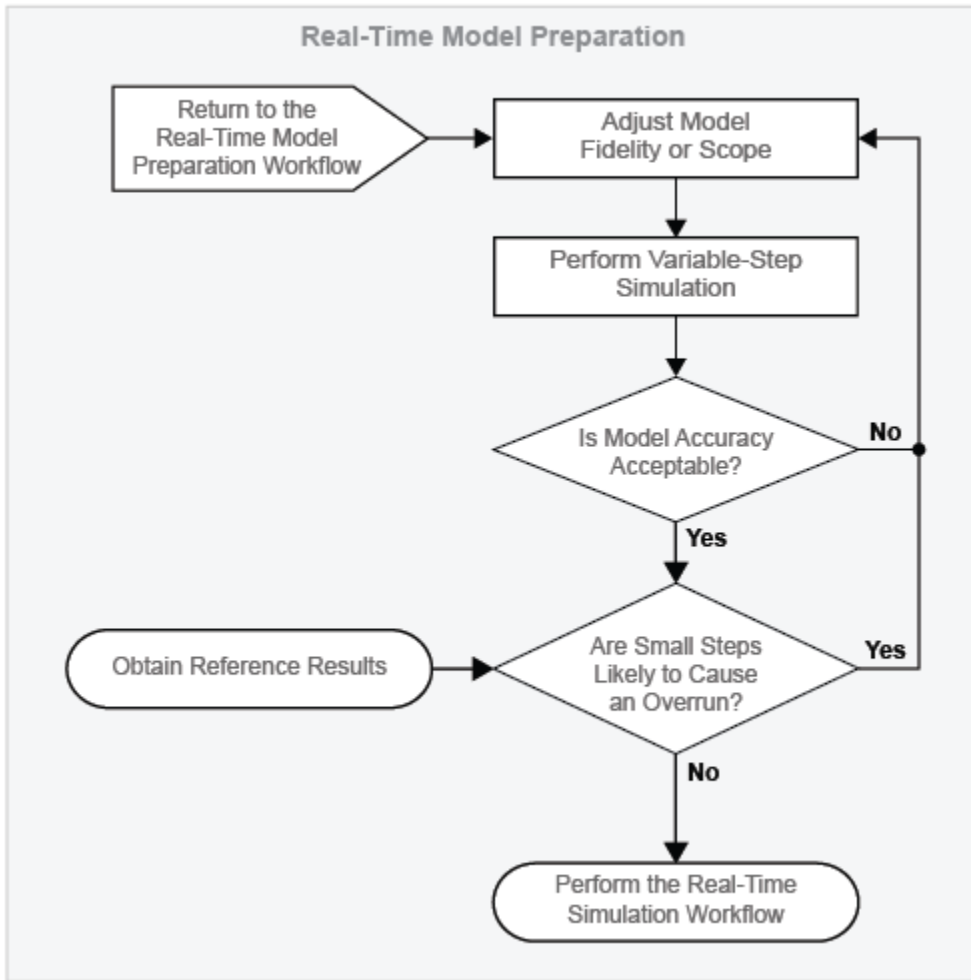
In this section...
“Perform Hardware-in-the-Loop Simulation” on page 7-104
“Insufficient Computational Capability for Hardware-in-the-Loop Simulation” on page 7-105

You must have a Simulink Real-Time license to perform this workflow.

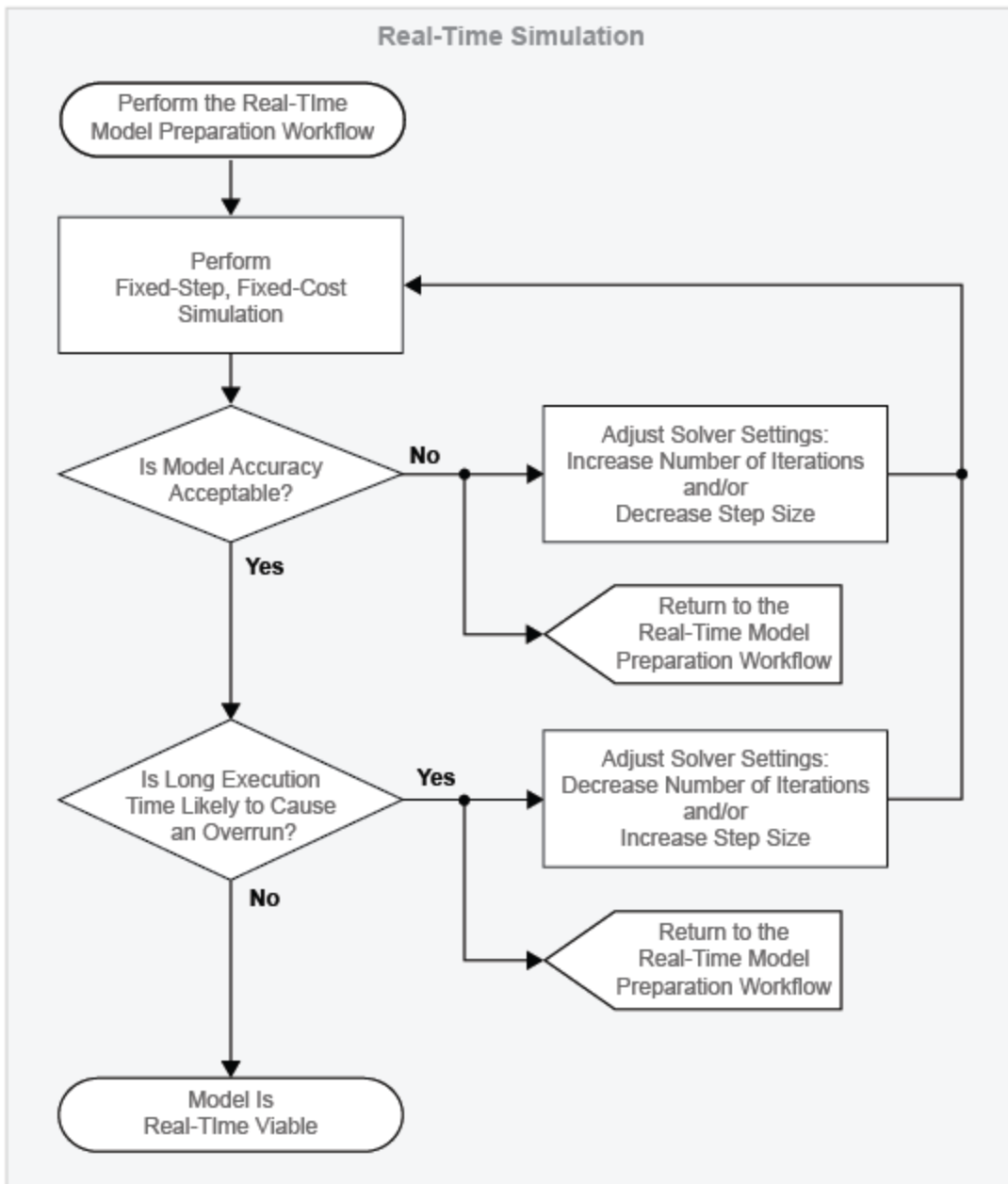
This figure shows the hardware-in-the loop simulation workflow. The connectors are exit points for returning to the real-time model preparation workflow.



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 7-5 and “Real-Time Simulation Workflow” on page 7-57.
- 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 Simulink Real-Time Setup and Configuration.
- 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 7-108.

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 generates an overrun or produces results that do not match your reference results closely enough. To make your model real-time capable, return 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 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 7-5 and “Real-Time Simulation Workflow” on page 7-57.

Insufficient Computational Capability for Hardware-in-the-Loop Simulation

Your real-time target 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 after multiple iterations of the real-time workflows, consider these options for increasing processing power:

- “Upgrading Target Hardware” on page 7-13
- “Simulating Parts of the System in Parallel” on page 7-13

Related Examples

- “Generate, Download, and Execute Code” on page 7-108

More About

- “Real-Time Model Preparation Workflow” on page 7-5
- “What Is Hardware-in-the-Loop Simulation?” on page 7-96
- “Real-Time Simulation Workflow” on page 7-57
- “Code Generation Requirements” on page 7-106

Code Generation Requirements

In this section...
“Hardware Requirements” on page 7-106
“Software Requirements” on page 7-107

Performing 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*.
- Real-time target computer system, containing this hardware:
 - CPU. For information on real-time target computer specifications, see “*Target Computer Requirements*”
 - I/O board. For information on supported I/O boards and for writing device drivers for unsupported boards, see *Supported Hardware: Hardware Drivers*, “*I/O Boards*”, and “*I/O Driver Support*”.
 - Protocol interface
- Controller, preconfigured with code from your controller model
- Connection cable for linking the development computer to the real-time target. For information, see “*Links Between Development and Target Computers*”.
- 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

For information, see “*Peripherals*”.

- 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. For information, see Simulink Real-Time C Compiler Requirements and “Command-Line C Compiler Configuration”.

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”.

Related Examples

- “Set Up and Configure Simulink Real-Time”
- “Create and Run a Real-Time Application”

More About

- “About Code Generation from Simscape Models” on page 8-2
- “How Simscape Code Generation Differs from Simulink” on page 8-5
- “Limitations” on page 9-3
- “Real-Time Test Environment”
- Simulink Real-Time Setup and Configuration
- “Simulink Real-Time Explorer”
- “What Is Hardware-in-the-Loop Simulation?” on page 7-96
- “Hardware-in-the-Loop Simulation Workflow” on page 7-100

Generate, Download, and Execute Code

In this section...
“Requirements for Building and Executing Simulink Real-Time Applications” on page 7-108
“Create, Build, and Download a Real-Time Application” on page 7-108
“Execute Real-Time Application” on page 7-109

You must have a Simulink Real-Time license to perform this workflow.

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 7-5 and “Real-Time Simulation Workflow” on page 7-57.
- 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 Simulink Real-Time Setup and Configuration.

Create, Build, and Download a Real-Time Application

To generate code for the model on your development computer and transfer it to your real-time computer:

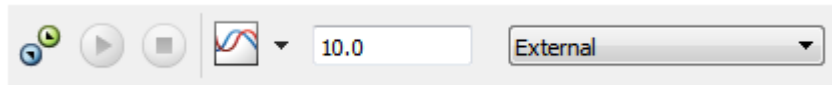
- 1 Set the Simulink Real-Time code generation configuration parameters. For information, see “Set Configuration Parameters”.
- 2 Start the target computer. For information, see “Start Target Computer”.

- 3 To compile your code, link your hardware, and download the real-time application to your target in one step, in the Simulink editor, click the **Build Model** button . For information, see “Build and Download Real-Time Application”.



Execute Real-Time Application

After you build and download a real-time application to the target computer, you can run the real-time application.

- 1 In the Simulink window, on the toolbar, set the simulation mode to **External**.



Selecting external mode allows you to connect your development computer to your real-time target.

- 2 To connect your development and target computers and to transfer your model parameters to the target, click the **Connect to Target** button .
- 3 To execute your real-time application on the target computer, click the **Run** button .

See Also

slrtexplr

Related Examples

- “Set Up and Configure Simulink Real-Time”
- “Create and Run a Real-Time Application”

More About

- “Hardware-in-the-Loop Simulation Workflow” on page 7-100
- “What Is Hardware-in-the-Loop Simulation?” on page 7-96
- “Hardware-in-the-Loop Simulation Process”

- “About Code Generation from Simscape Models” on page 8-2
- “How Simscape Code Generation Differs from Simulink” on page 8-5
- “Code Generation Requirements” on page 7-106
- “Configuration Parameters”
- “Real-Time Application Execution”
- “Simulink Real-Time Options Configuration Parameters”

Requirements for Using Alternative Platforms

In this section...

“Hardware Requirements” on page 7-111

“Software Requirements” on page 7-111

Performing hardware-in-the-loop (HIL) simulation with a custom standalone application 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, as appropriate for communicating with the real-time processor
- Real-time target, that is, a real-time capable CPU or computer
- I/O board that the real-time target supports
- Controller, preconfigured with code from your controller model
- Peripheral for transferring code to the real-time target
- Wiring harness to connect the real-time target to the controller

Note: You might also need a real-time operating system, depending on the requirements of your real-time target.

Software Requirements

The minimum software requirements for HIL simulation with a custom application are:

- Embedded Coder and the Embedded Coder Software Requirements, including:
 - Simulink Coder and the Simulink Coder Software Requirements.
 - C compiler. For more about the Embedded Coder C compiler requirements, see Embedded Coder C Compiler Requirements.
 - Cross compiler.

- Template example main function that you can manually or automatically combine with generated code
- I/O driver. Options are:
 - C code I/O driver for the code generation build
 - Precompiled static or dynamic library with the necessary documentation

More About

- “About Code Generation from Simscape Models” on page 8-2
- “How Simscape Code Generation Differs from Simulink” on page 8-5
- “Hardware-in-the-Loop Simulation Workflow” on page 7-100
- “Limitations” on page 9-3
- “What Is Hardware-in-the-Loop Simulation?” on page 7-96

Code Generation

- “About Code Generation from Simscape Models” on page 8-2
- “Reasons for Generating Code” on page 8-3
- “Using Code-Related Products and Features” on page 8-4
- “How Simscape Code Generation Differs from Simulink” on page 8-5

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 8-5, as well as “Limitations” on page 4-32.

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 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 8-5

“Compiler and Processor Architecture Requirements” on page 8-5

“Precompiled Libraries Provided for Selected Compilers” on page 8-5

“Simscape Code Reuse Not Supported” on page 8-6

“Tunable Parameters Not Supported” on page 8-6

“Simscape Run-Time Parameter Inlining Override of Global Exceptions” on page 8-6

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
- 32-bit integer size

For details on supported compiler versions, see

http://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. These are the standard UNIX

compilers for UNIX operating systems, lcc and Microsoft® Visual Studio® for 32-bit Windows®, and Microsoft Visual Studio for 64-bit Windows.

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.

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.

Data Logging

- “About Simulation Data Logging” on page 9-2
- “Enable Data Logging for the Whole Model” on page 9-4
- “Log Data for Selected Blocks Only” on page 9-5
- “Data Logging Options” on page 9-6
- “Log and Plot Simulation Data” on page 9-8
- “Log Simulation Statistics” on page 9-13
- “Log and View Simulation Data for Selected Blocks” on page 9-17
- “Log, Navigate, and Plot Simulation Data” on page 9-21
- “About the Simscape Results Explorer” on page 9-26
- “Use Custom Units to Plot Simulation Data” on page 9-27
- “View Sparkline Plots of Simulation Data” on page 9-31

About Simulation Data Logging

In this section...
“Suggested Workflows” on page 9-2
“Limitations” on page 9-3

Suggested Workflows

You can log simulation data to the workspace 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 to the workspace, you can later query, plot, and analyze it without rerunning the simulation.

Simulation data logging can also 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 9-8 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 9-5.

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 stored in the simulation log variable, see the reference pages for the classes `simscape.logging.Node`, `simscape.logging.Series`, and their associated methods.

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. Set up your model to log simulation data, either for the whole model or on a block-by-block basis, and enable data recording. Simulate the model, and then open the Simulation Data Inspector and view the results. For detailed information on how to enable data recording and how to configure and use the Simulation Data Inspector, see “Inspect Signal Data with Simulation Data Inspector”.

To make your model simulation and data logging compatible with the `parfor` command, select the **Save simulation output as single object** 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 stored as a separate workspace variable. For more information, see “Save simulation output as single object”.

Limitations

Simulation data logging is not supported for:

- Model reference
- Generated code
- Accelerator mode
- Rapid Accelerator mode

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 9-21
- “Log and View Simulation Data for Selected Blocks” on page 9-17
- “Log and Plot Simulation Data” on page 9-8
- “Log Simulation Statistics” on page 9-13

More About

- “Data Logging Options” on page 9-6

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, from the top menu bar, select **Simulation > Model Configuration Parameters**. The Configuration Parameters dialog box opens.
- 2 In the Configuration Parameters dialog box, in the left pane, select **Simscape**. The right pane displays the **Log simulation data** option, which is set to **None**, by default.
- 3 From the drop-down list, select **All**, then click **OK**.
- 4 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 9-6.

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 9-21
- “Log and Plot Simulation Data” on page 9-8
- “Log Simulation Statistics” on page 9-13

More About

- “Data Logging Options” on page 9-6

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 Set the logging configuration parameter to enable simulation data logging on a block-by-block basis.

In the model window, from the top menu bar, select **Simulation > Model Configuration Parameters**. 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**.

- 2 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 on the block. From the context menu, select **Simscape > Log simulation data**. A check mark appears in front of the **Log simulation data** option.

- 3 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 on 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.

Related Examples

- “Log and View Simulation Data for Selected Blocks” on page 9-17

More About

- “Data Logging Options” on page 9-6

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 9-13.
- **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 9-26.
- **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.

Note The **Output options** parameter, on the **Data Import/Export** pane of the Configuration Parameters dialog box, also affects which data points are logged. For more information, see “Data Import/Export Pane” in the Simulink documentation.

After changing your data logging preferences, rerun the simulation to generate a new data log.

Related Examples

- “Enable Data Logging for the Whole Model” on page 9-4
- “Log Data for Selected Blocks Only” on page 9-5

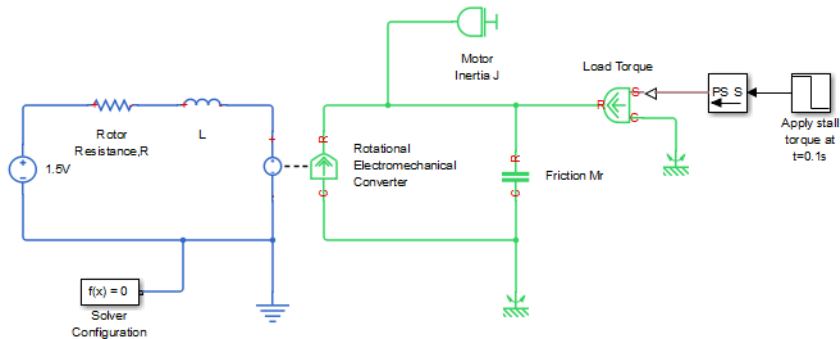
More About

- “About Simulation Data Logging” on page 9-2

Log and Plot Simulation Data

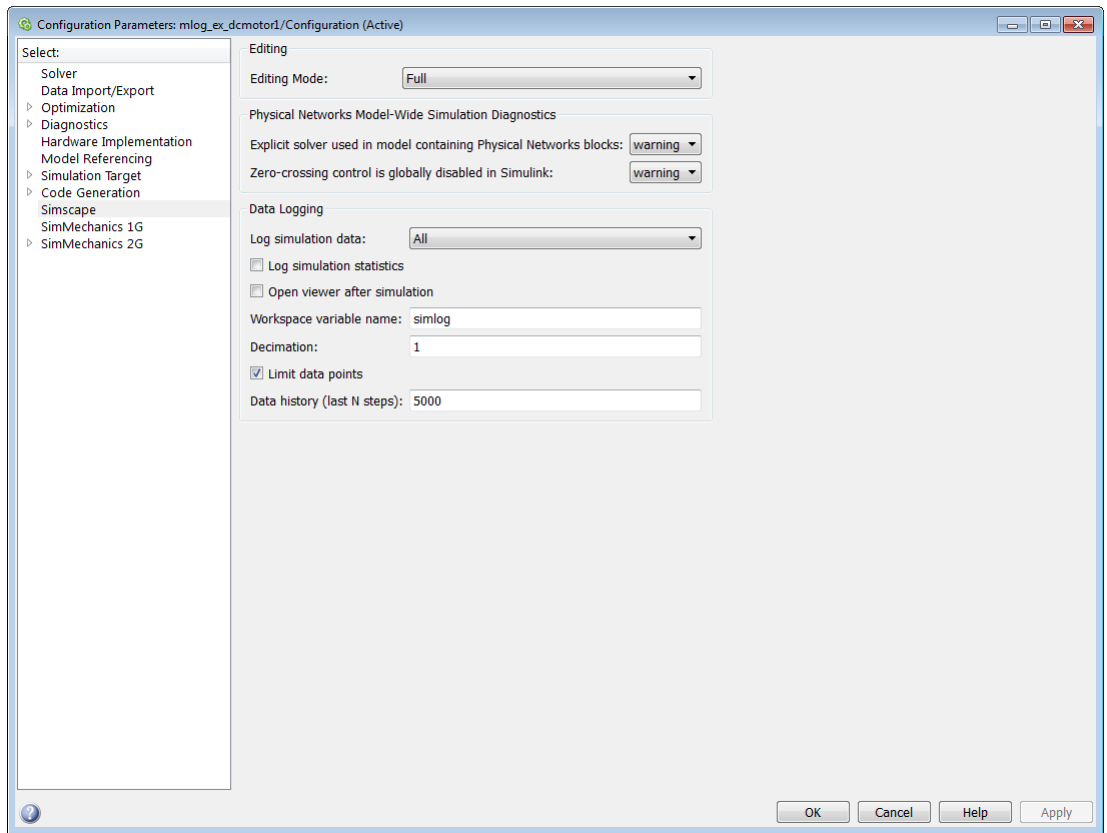
This example shows how you can log and plot simulation data instead of adding sensors to your model.

The model shown represents a permanent magnet DC motor.



This model is very similar to the Permanent Magnet DC Motor 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 To enable data logging, open the Configuration Parameters dialog box, in the left pane, select **Simscape**, then set the **Log simulation data** parameter to **All** and click **OK**.



- 3 Simulate the model. This creates a workspace variable named `simlog` (as specified by the **Workspace variable name** parameter), which contains the simulation data.
- 4 The `simlog` variable has the same hierarchy as the model. To see the whole variable structure, at the command prompt, type:

```
simlog.print
```

This command prints the whole data tree.

```
mlog_ex_dcmotor1
+-Electrical_Reference2
| +-V
| | +-v
| +-i
+-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
| +-t
+-Mechanical_Rotational_Reference1
| +-W
| | +-w
| +-t
+-Motor_Inertia_J
| +-I
| | +-w
| +-t
+-Rotational_Electromechanical_Converter
| +-C
| | +-w
| +-R
| | +-w
| +-i
| +-n
| | +-v
| +-p
| | +-v
| +-t
| +-v
| +-w
+-Rotor_Resistance_R
| +-i
| +-n
| | +-v
| +-p
| | +-v
| +-v
+-Simulink_PS_Converter

```

```

+-x1_5V
+-i
+-n
| +-v
+-p
| +-v
+-v

```

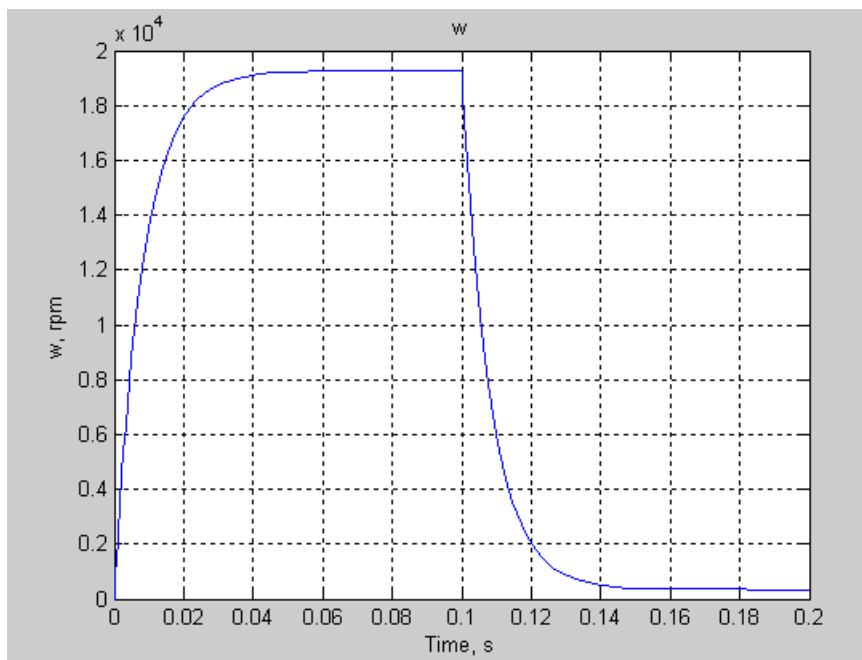
- 5 Every node that represents an Across, Through, or internal block variable contains series data. To get to the series, you have to specify the complete path to it through the tree, starting with the top-level variable name. For example, to get a handle on the series representing the angular velocity 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.

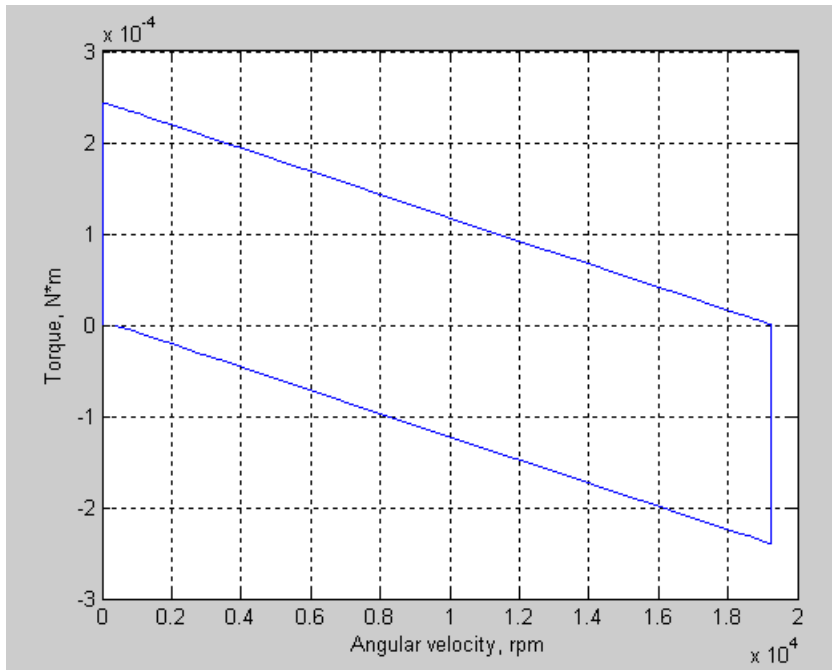
- 6 You do not have to isolate series data to plot its values against time, or against another series. For example, to see how the motor speed (in revolutions per minute) changes with time, type:

```
plot(simlog.Rotational_Electromechanical_Converter.R.w, 'units', 'rpm')
```



- 7 Compare this figure to the RPM scope display in the Permanent Magnet DC Motor example. The results are exactly the same.
- 8 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')
```

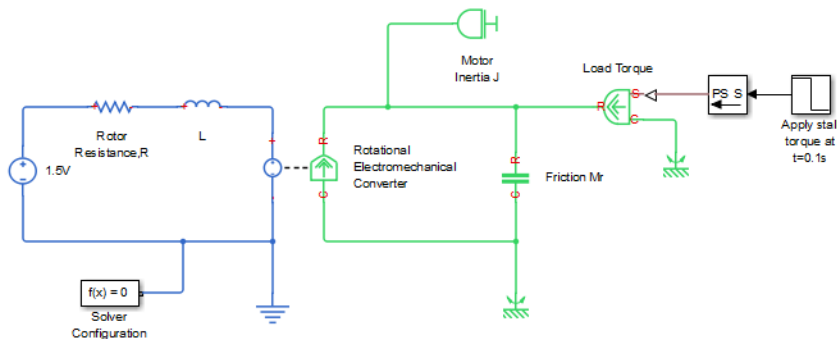


For more information on plotting logged simulation data, see the `simscape.logging.plot` and `simscape.logging.plotxy` reference pages.

Log Simulation Statistics

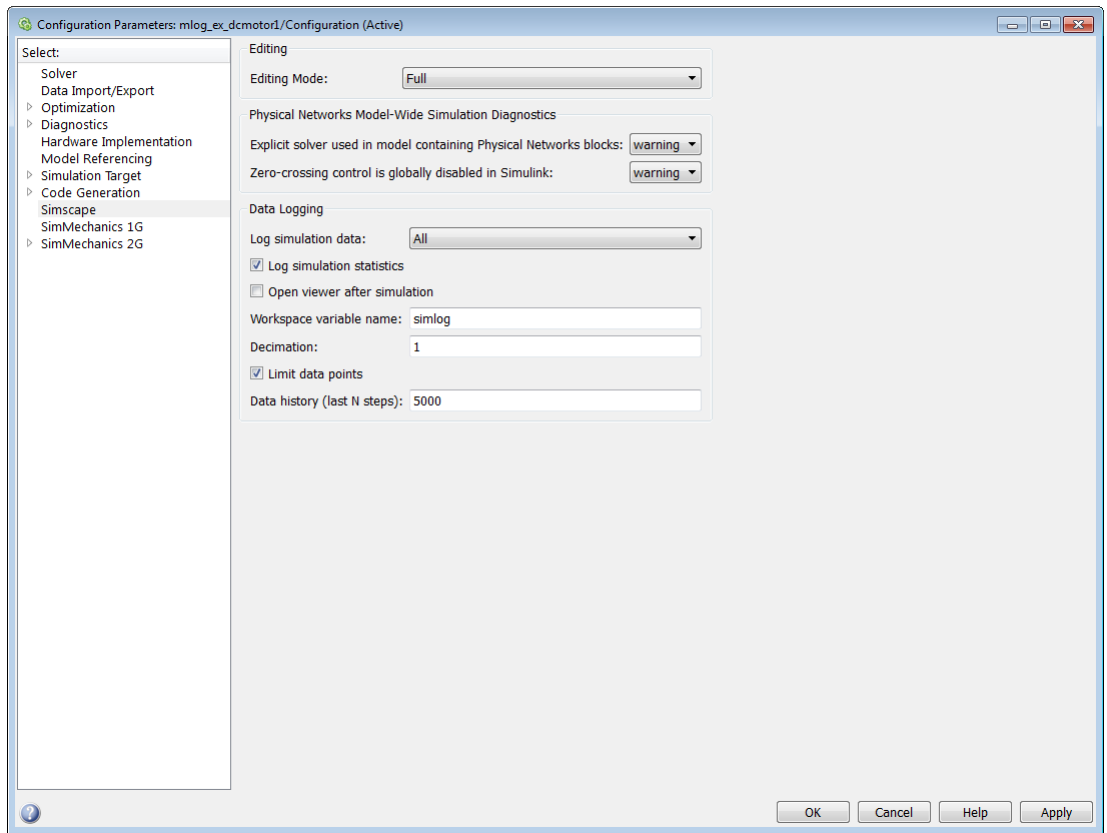
This example shows how you can access and analyze information on zero crossings during simulation. By default, the zero-crossing data is not logged. If you select the **Log simulation statistics** check box, the simulation log variable contains an additional `SimulationStatistics` node for each block that can produce zero crossings, at the price of slower simulation speed and heavier memory consumption.

The model shown represents a permanent magnet DC motor.



This model is the same as the one used in the “Log and Plot Simulation Data” on page 9-8 example.

- 1 Build the model, as shown in the preceding illustration.
- 2 To enable data logging, open the Configuration Parameters dialog box, in the left pane, select **Simscape**, then set the **Log simulation data** parameter to **All**, select the **Log simulation statistics** check box, and click **OK**.



- 3 Simulate the model. This creates a workspace variable named `simlog` (as specified by the **Workspace variable name** parameter), which contains the simulation data. Because you selected the **Log simulation statistics** checkbox, 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.print
```

This command prints the whole data tree.

```
mlog_ex_dcmotor1
+-Electrical_Reference2
| +-V
```

```

| | +-v
| +-i
+-Friction_Mr
| +-C
| | +-w
| +-R
| | +-w
| +-SimulationStatistics
| | +-zc_0
| | | +-crossings
| | | +-values
| | +-zc_1
| | | +-crossings
| | | +-values
| | +-zc_2
| | | +-crossings
| | | +-values
| +-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
| +-t
+-Mechanical_Rotational_Reference1
| +-W
| | +-w
| +-t
+-Motor_Inertia_J
| +-I
| | +-w
| +-t
+-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

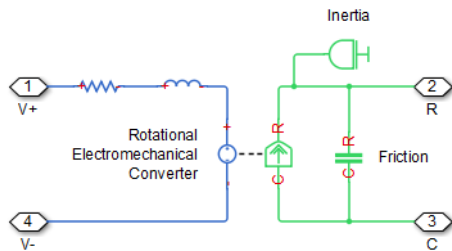
```

- 5 If you compare this tree to the one used in the “Log and Plot Simulation Data” on page 9-8 example, you can see that under the `Friction_Mr` node there is now an additional node called `SimulationStatistics`. The rest of the tree is unchanged. This means that `Friction Mr` 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` class and `simscape.logging.Series` class reference pages.

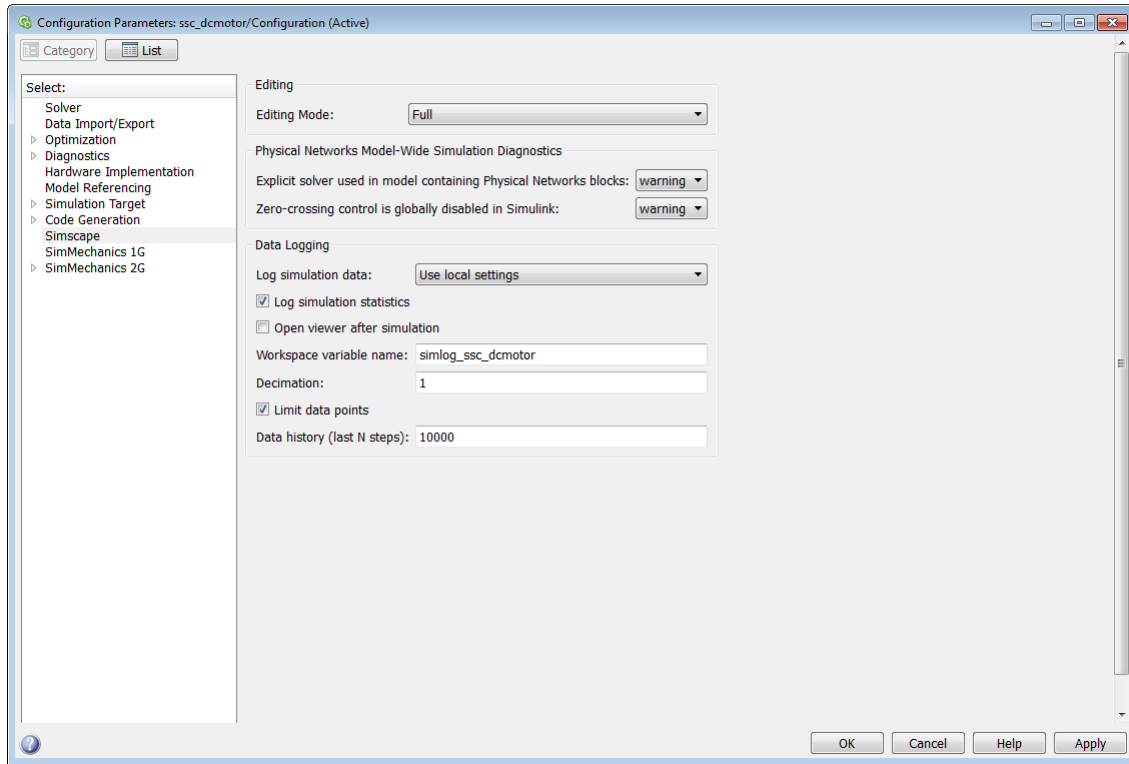
Log and View Simulation Data for Selected Blocks

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.

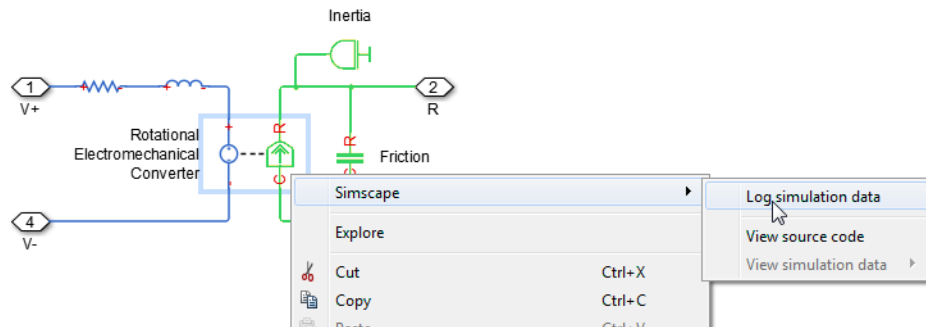
- 1 Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window. Double-click the DC Motor subsystem to open it.



- 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**.

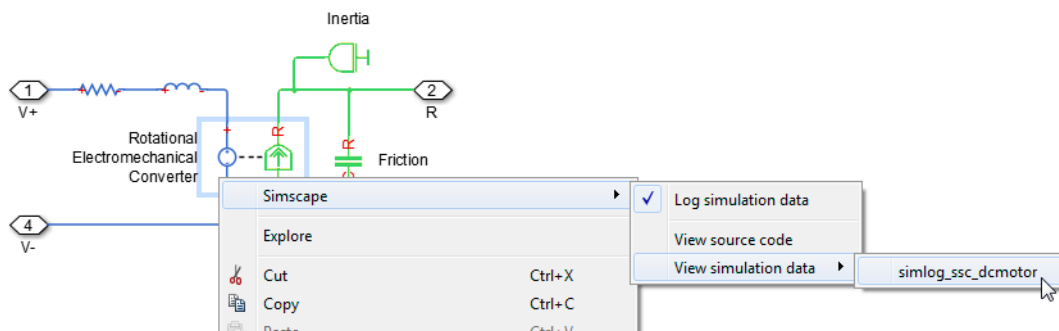


- 3 Select the blocks for data logging. Right-click the Rotational Electromechanical Converter block. From the context menu, select **Simscape** > **Log simulation data**.



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.

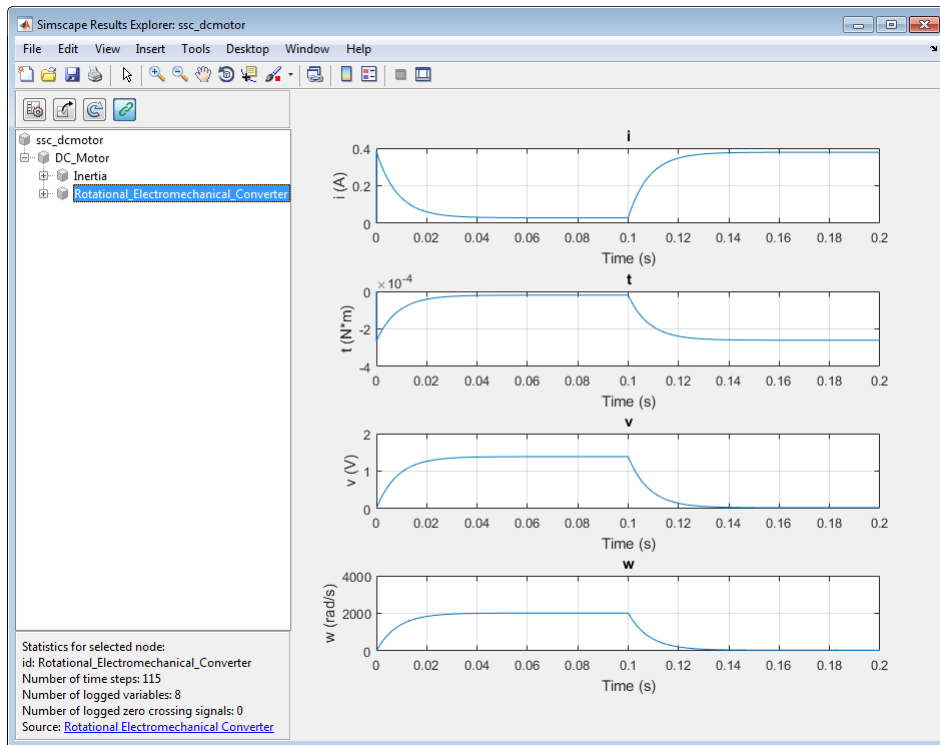
- 4 Right-click the Motor Inertia block and select it for data logging, as described in the previous step.
- 5 Simulate the model. This creates a workspace variable named `simlog_ssc_dcmotor` (as specified by the **Workspace variable name** parameter), which contains the simulation data for selected 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 9-26.

The Simscape Results Explorer window opens, with the `Rotational_Electromechanical_Converter` node already selected 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 `Motor Inertia`.



Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 9-21

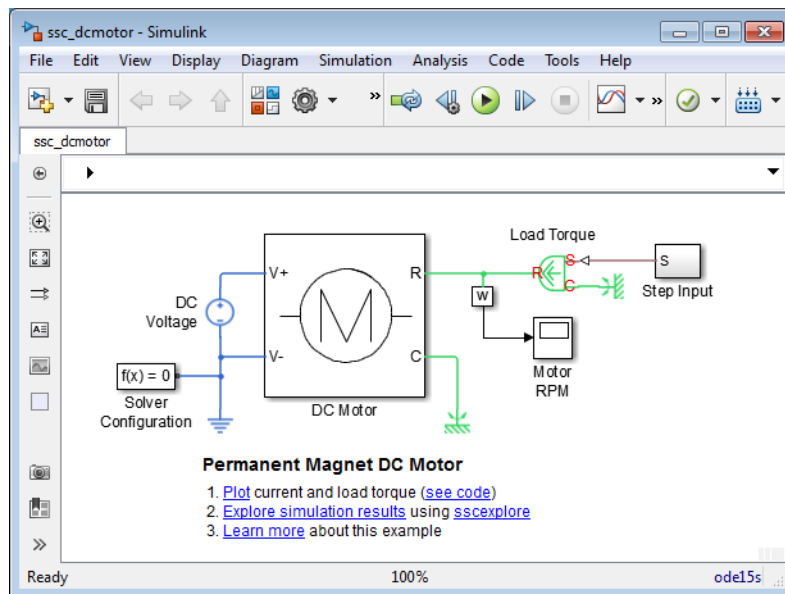
More About

- “About Simulation Data Logging” on page 9-2
- “About the Simscape Results Explorer” on page 9-26

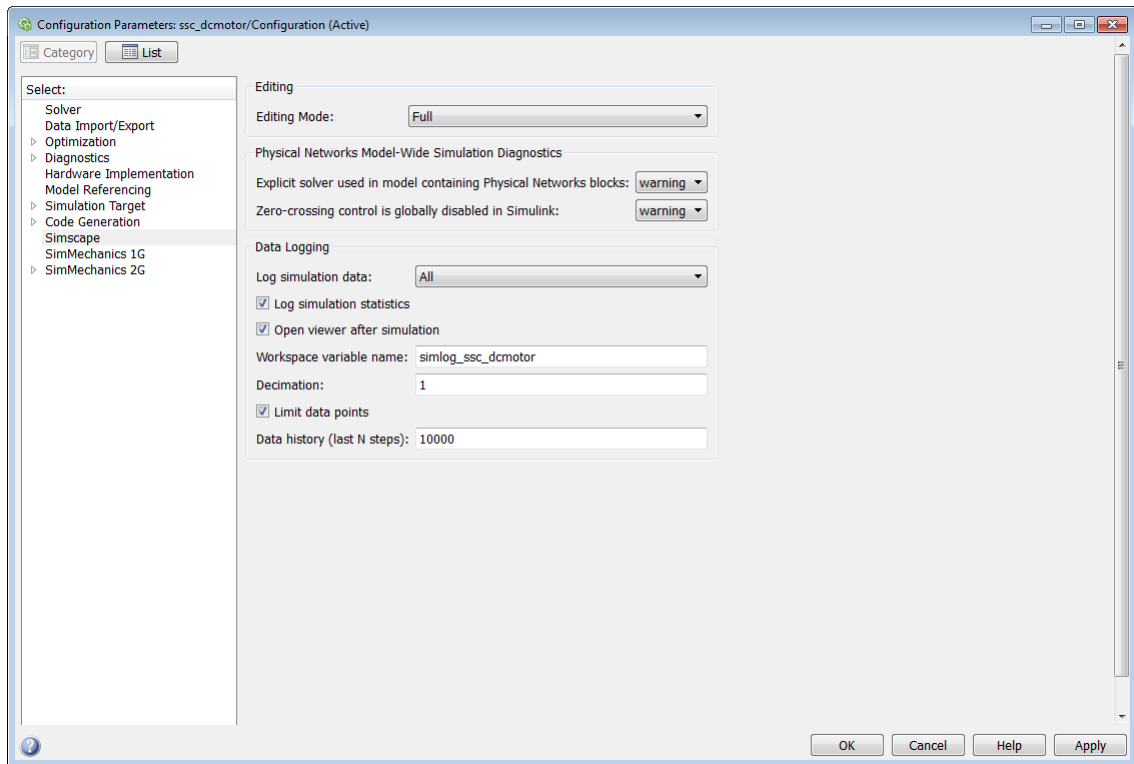
Log, Navigate, and Plot Simulation Data

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.

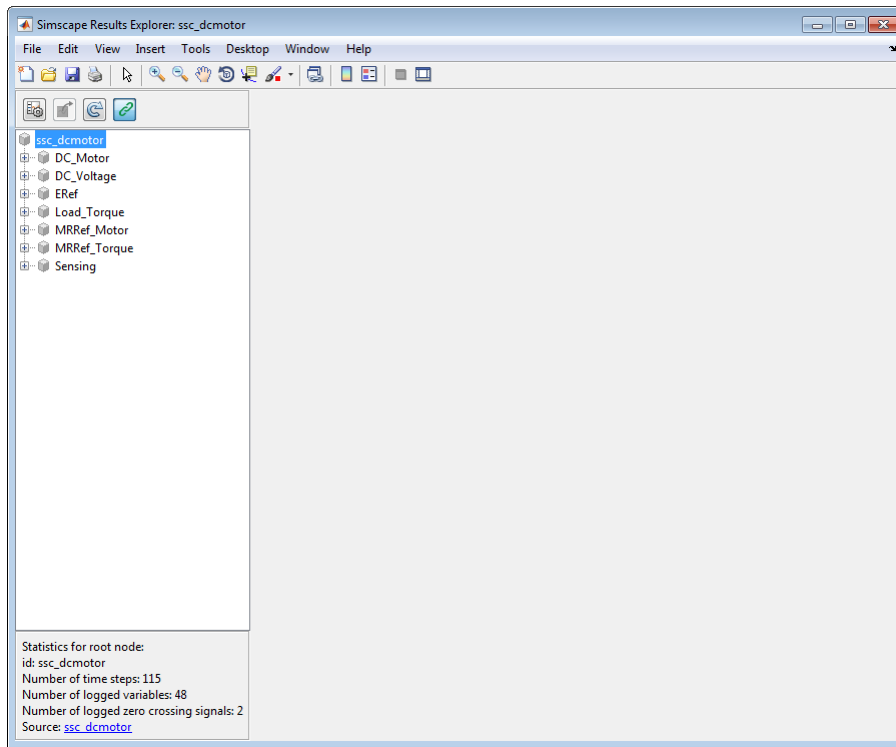
- 1 Open the Permanent Magnet DC Motor example model by typing `ssc_dcmotor` in the MATLAB Command Window.



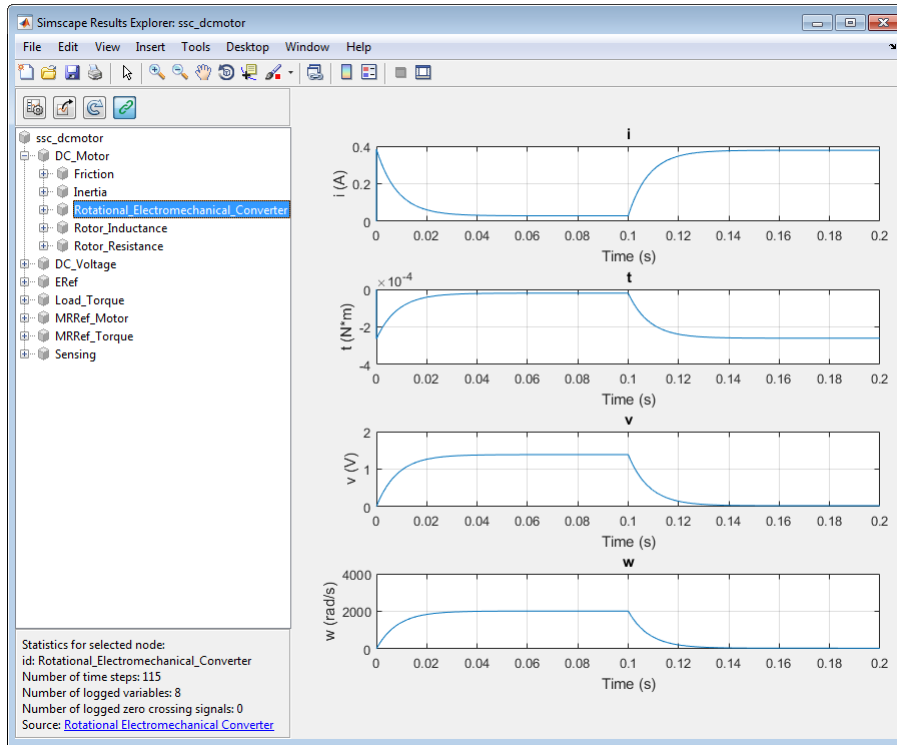
- 2 Open the Configuration Parameters dialog box and then, in the left pane, select **Simscape**. You can see that this example model already has data logging for the whole model enabled, as well as simulation statistics, and that the workspace variable name is `simlog_ssc_dcmotor`. Select the **Open viewer after simulation** check box and click **OK**.



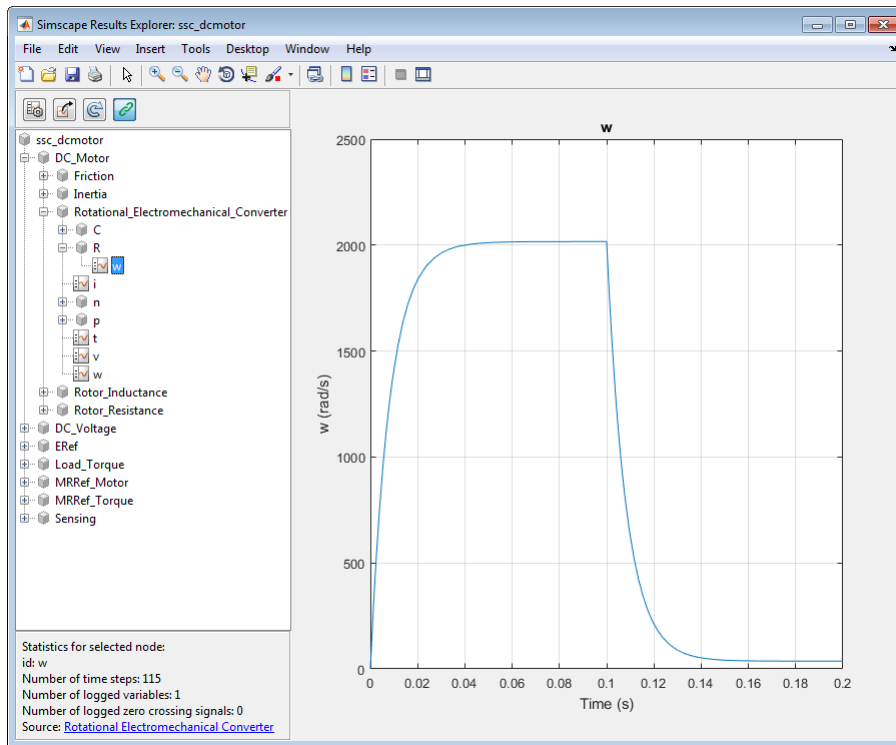
- 3 Simulate the model. When the simulation is done, the Simscape Results Explorer window opens. In the left pane, it contains the simulation log tree hierarchy, which corresponds to the model hierarchy.



- 4 When you click on 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.



Related Examples

- “Log and View Simulation Data for Selected Blocks” on page 9-17

More About

- “About Simulation Data Logging” on page 9-2
- “About the Simscape Results Explorer” on page 9-26

About the Simscape Results Explorer




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 9-21.

Another way to open the Simscape Results Explorer window is to right-click on 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 9-17.

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  button located in the toolbar above the left pane. The button appearance changes to  and, when the new window opens after simulation, that window will be 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 on its  button) unlinks the previous one.

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 9-21
- “Log and View Simulation Data for Selected Blocks” on page 9-17
- “Use Custom Units to Plot Simulation Data” on page 9-27

Use Custom Units to Plot Simulation Data

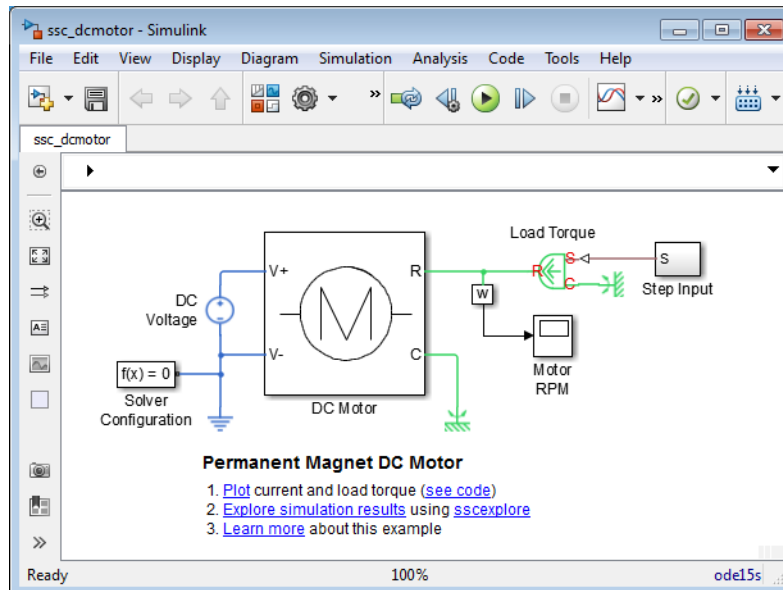
Simscape Results Explorer has a set of default units for plotting the logged data. This example shows how you can change to a custom unit; for example, to plot rotations in degrees rather than radians.

- 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
```

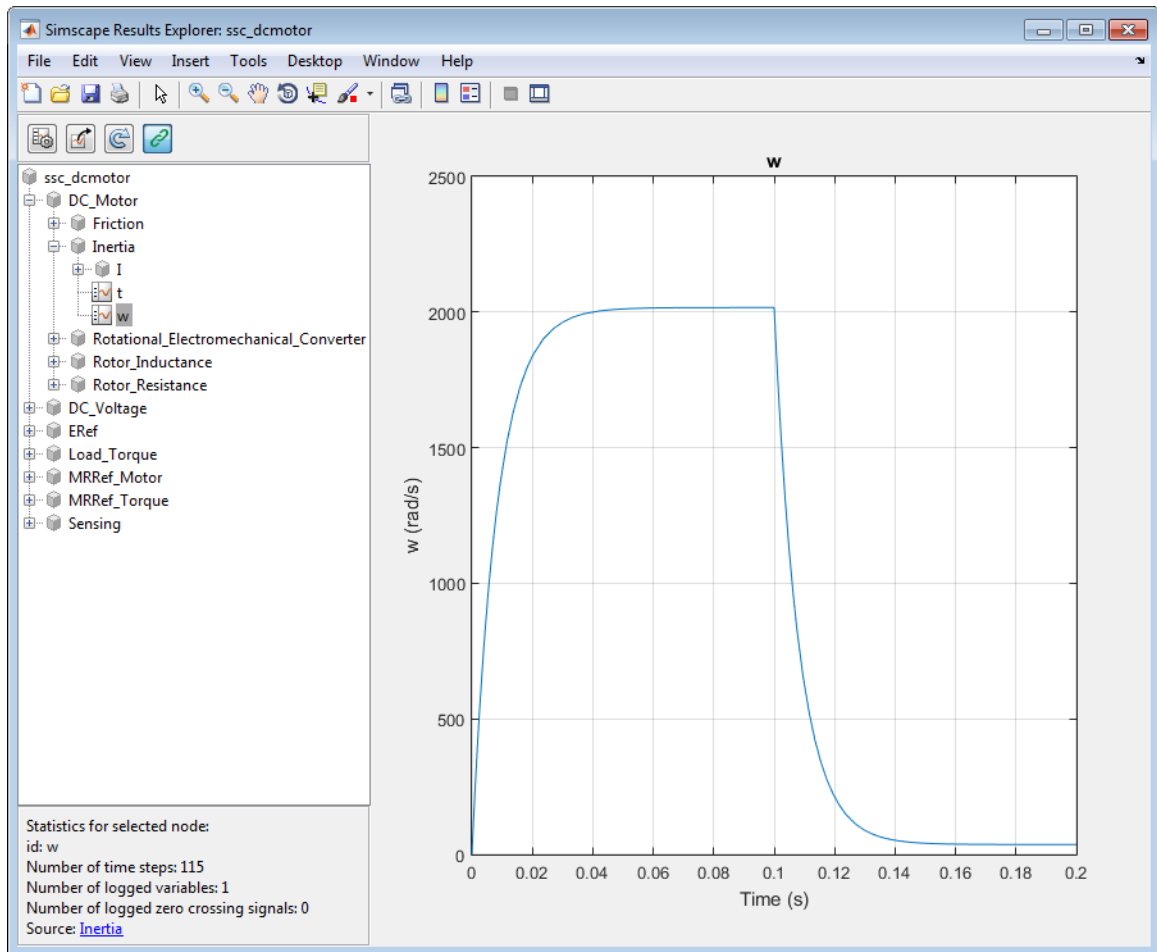
Include only the units you want to customize. For everything else, Simscape Results Explorer will use the default units.

- 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:

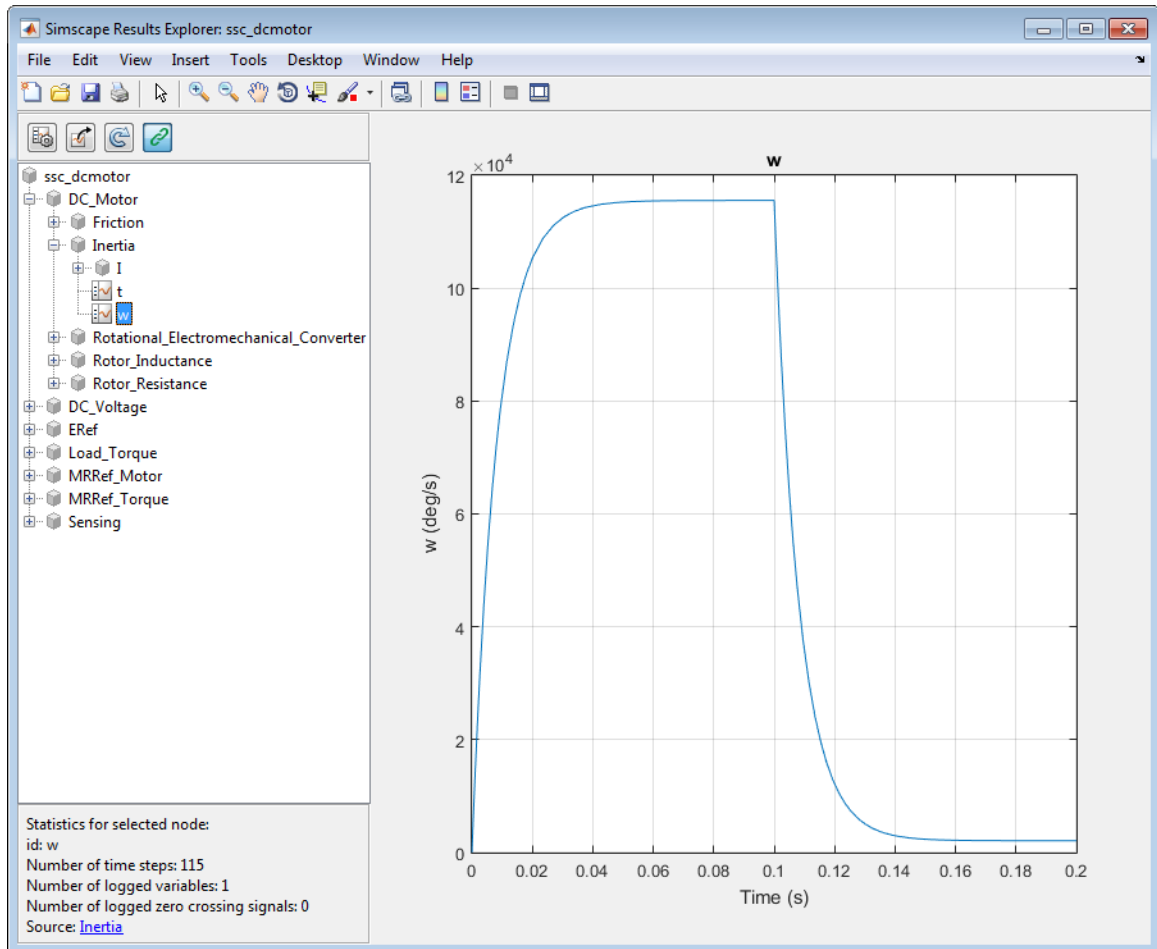
```
sscexplore(simlog_ssc_dcmotor, 'DC_Motor.Inertia.w')
```



By default, Simscape Results Explorer plots rotational velocity in rad/s.

5

To switch to custom units, click the **Plot options** icon  and then, in the Options dialog box, change **Units** from **Default** to **Custom** and click **OK**. The rotational velocity plot is redrawn in deg/s.



Tip Use the function `pm_getunits` to get the full list of available units.

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 9-21
- “Log and View Simulation Data for Selected Blocks” on page 9-17

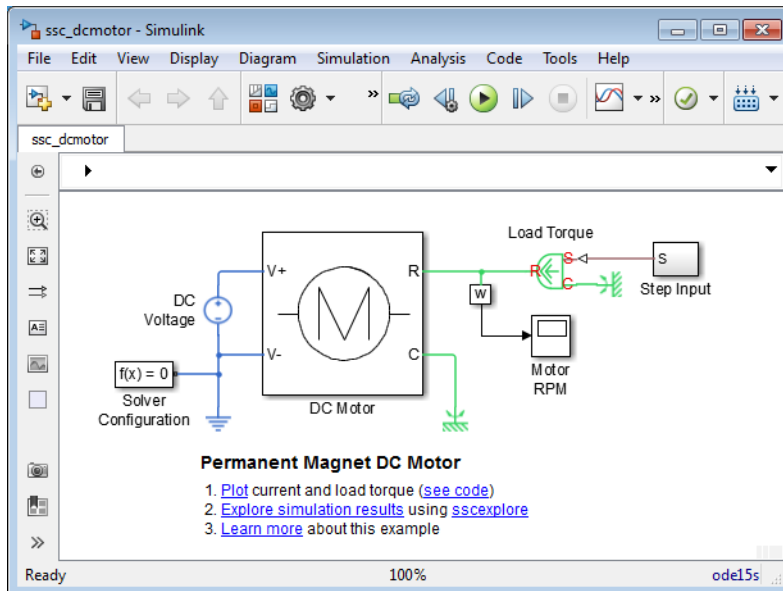
More About

- “About Simulation Data Logging” on page 9-2
- “About the Simscape Results Explorer” on page 9-26

View Sparkline Plots of Simulation Data

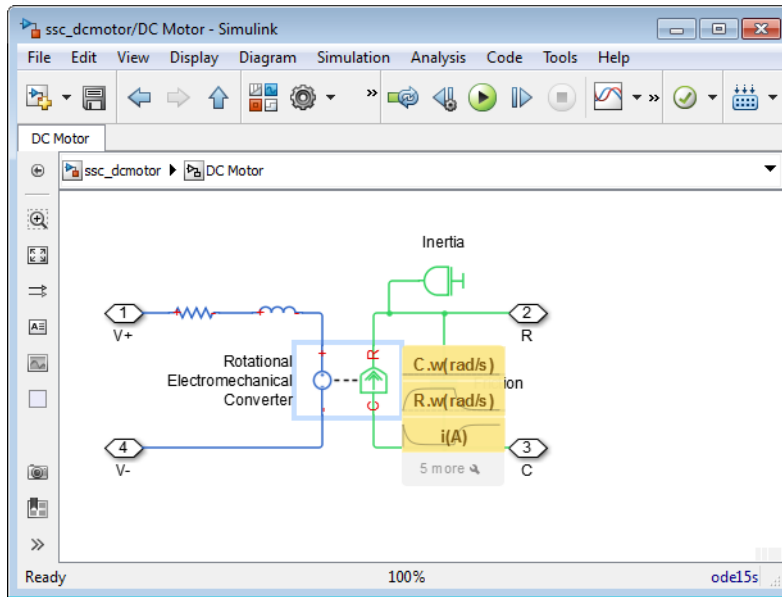
This example shows the basic workflow for viewing sparkline plots of logged simulation data for selected blocks and variables directly on the model canvas. Before viewing sparkline plots, you must enable data logging for the whole model, or at least for those blocks where you want to display the data, and run the simulation.

- 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.

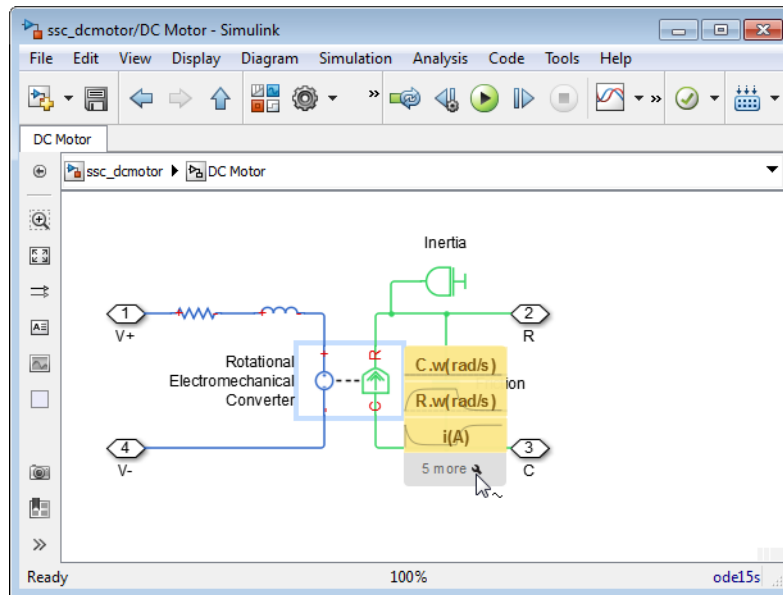


- 2 Double-click the DC Motor subsystem to open it.
- 3 Simulate the model.
- 4 To enable display of the sparkline plots on the model canvas, in the model window, from the top menu bar, select **Display > Simscape > Toggle Sparklines When Clicked**. This action adds the check mark next to the **Toggle Sparklines When Clicked** menu option, and you can start selecting blocks to display sparkline plots of logged data for their variables. Repeatedly selecting a block toggles the display of its sparkline plots on and off.

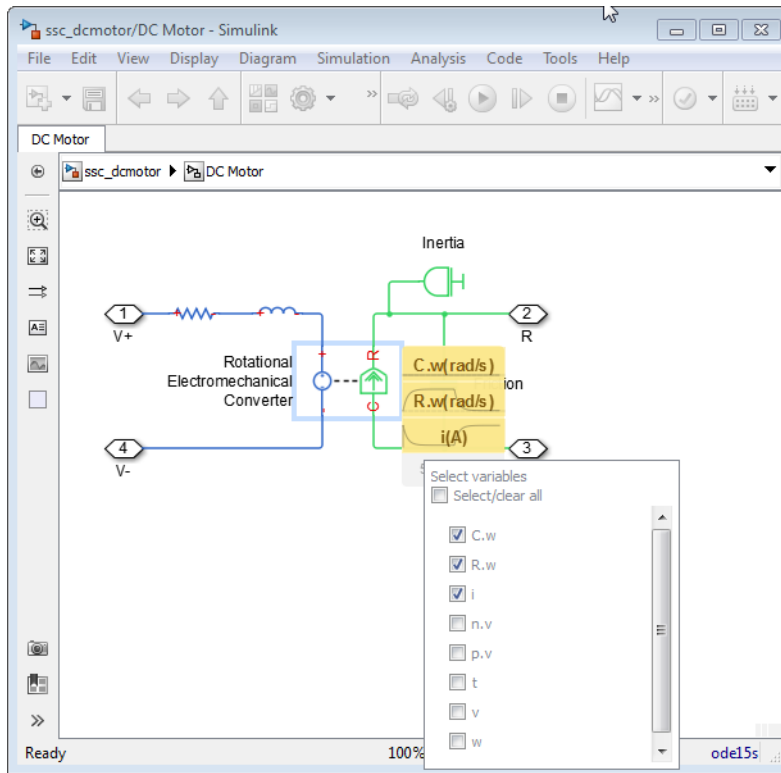
- 5 Select the Rotational Electromechanical Converter block. Sparkline plots of the first three variables available for this block are displayed on the canvas, and the field below the plots shows that 5 more variables are available.



- 6 To customize which plots are shown on the canvas, click the little wrench symbol in the field below the plots.

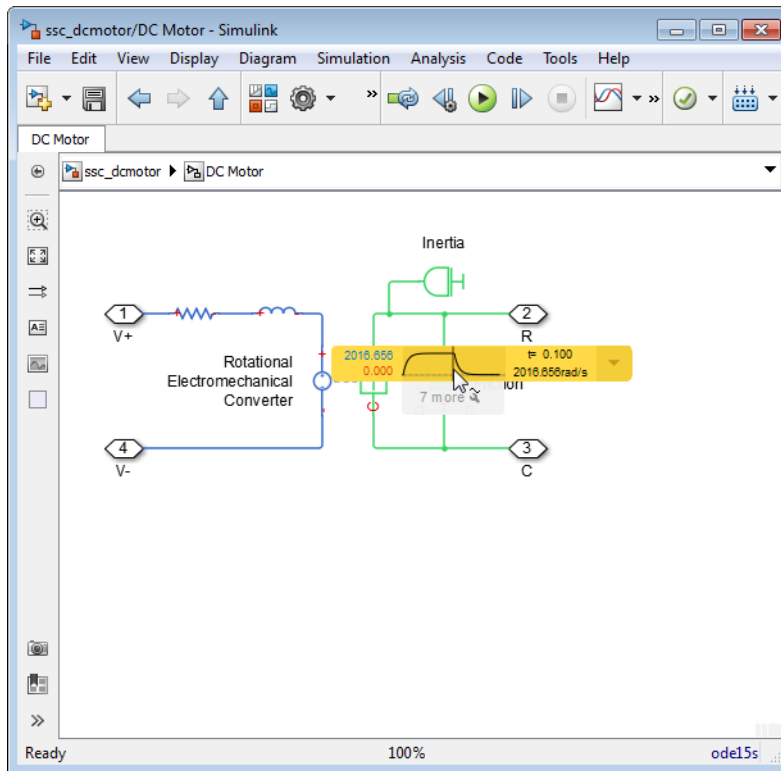


This action displays a list of all the block variables available, with check marks next to the one currently plotted.



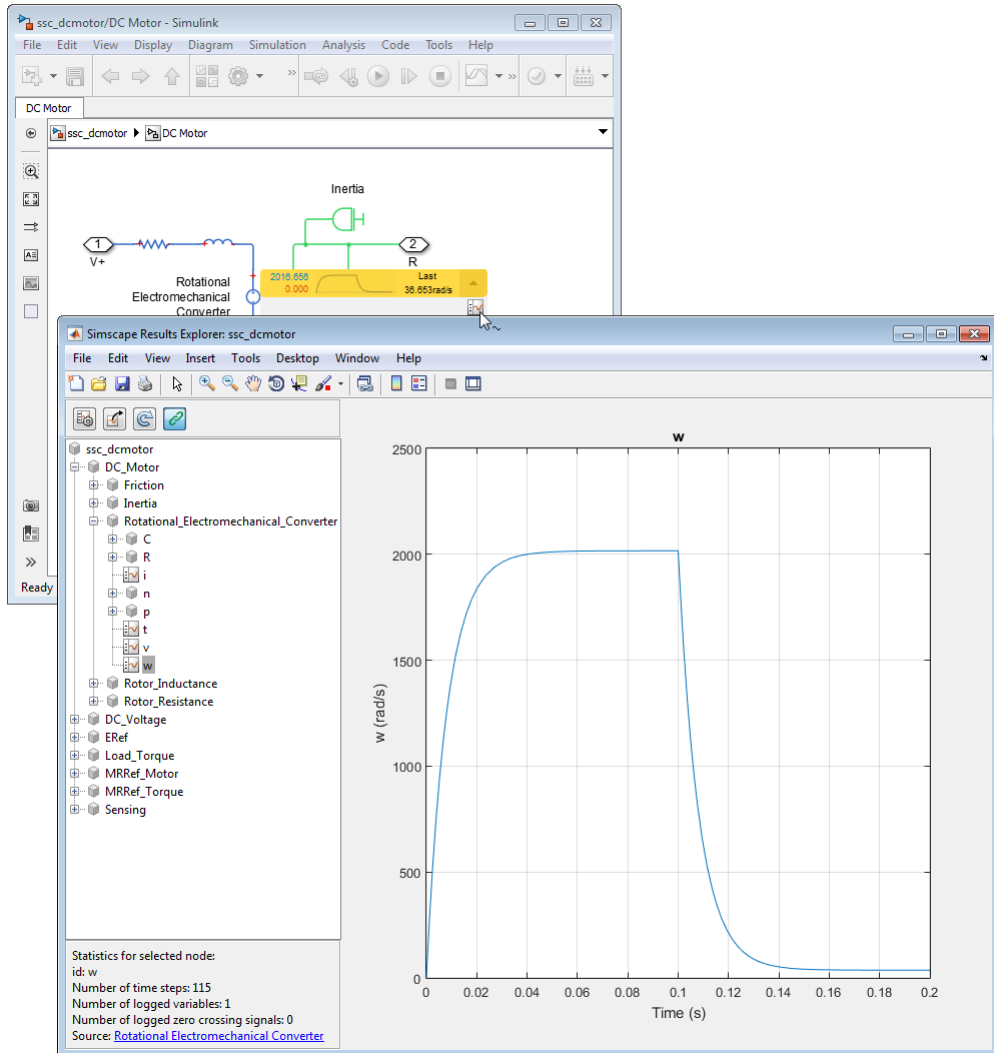
- 7 Clear all the check marks and select the last variable w instead. Then click anywhere on the model canvas to close the variable selection box.

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, select **Display > Simscape > Remove All Sparklines**. Then you can select more blocks and variables to display their sparkline plots.
 - Repeatedly selecting the **Toggle Sparklines When Clicked** menu 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 selecting a block toggles the display of its sparkline plots on and off.
-

Related Examples

- “Log, Navigate, and Plot Simulation Data” on page 9-21
- “Log and View Simulation Data for Selected Blocks” on page 9-17

More About

- “About Simulation Data Logging” on page 9-2
- “About the Simscape Results Explorer” on page 9-26

Model Statistics

- “Simscape Model Statistics” on page 10-2
- “1-D Physical System Statistics” on page 10-4
- “3-D Multibody System Statistics” on page 10-7
- “1-D/3-D Interface Statistics” on page 10-10
- “View Model Statistics” on page 10-11
- “Access Block Variables Using Statistics Viewer” on page 10-16

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 4-7.

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, SimDriveline™, SimHydraulics, SimElectronics®, and SimPowerSystems™ Simscape Components libraries.
- **3-D Multibody System** — This node represents aggregate statistics generated from all physical networks that are associated with blocks from SimMechanics™ Second Generation 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 SimMechanics Second Generation 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 **Variables** tab of the block dialog box.

Related Examples

- “View Model Statistics” on page 10-11
- “Access Block Variables Using Statistics Viewer” on page 10-16

More About

- “1-D Physical System Statistics” on page 10-4
- “3-D Multibody System Statistics” on page 10-7
- “1-D/3-D Interface Statistics” on page 10-10

1-D Physical System Statistics

This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape, SimDriveline, SimHydraulics, SimElectronics, and SimPowerSystems Simscape Components libraries.

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, 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 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 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.

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.
- The **Value** column contains the name of the variable, as it would appear in the **Variables** tab of the block dialog box.

Related Examples

- “View Model Statistics” on page 10-11
- “Access Block Variables Using Statistics Viewer” on page 10-16

3-D Multibody System Statistics

This node represents aggregate statistics generated from all physical networks that are associated with blocks from SimMechanics Second Generation 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 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 generally include blocks from the Body Elements library 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 block.

- **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 joints.

The kinematic graph provides a practical means to understand the topology of a model. This graph is a connected, undirected diagram in which each vertex corresponds to a rigid component and each edge corresponds to a joint. The total number of joints equals the total number of edges present in this graph.

The kinematic tree is a spanning tree of the kinematic graph in which each closed loop is opened by cutting one of its edges. If the kinematic graph contains no closed loops, it is identical to the kinematic tree.

- **Number of explicit tree joints** — This statistic provides the number of joints in the kinematic tree of a mechanical system that correspond to explicit joint blocks. Each tree joint corresponds to an edge in the kinematic tree. The number of explicit tree joints excludes joints cut from the kinematic graph to generate the kinematic tree.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **Number of implicit 6-DOF tree joints** — This statistic provides the number of 6-DOF joints in the kinematic tree of a mechanical system that do not correspond to explicit joint blocks. SimMechanics adds implicit 6-DOF joints when the kinematic graph of a model is not fully connected. These implicit joints connect previously disconnected portions of the graph to the ground body, adding the edges required to fully connect the graph. Implicit joints are always tree joints and do not create loops.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **Number of cut joints** — This statistic provides the number of joints that are cut from the kinematic graph of a mechanical system to generate the associated kinematic tree. The number of cut joints equals the number of closed loops present in the kinematic graph.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **Number of constraints** — This statistic provides the total number of constraint blocks in a mechanical system.
- **Number of tree degrees of freedom** — This statistic provides the total number of degrees of freedom in the kinematic tree of a mechanical system. This number equals the sum of all degrees of freedom that the tree joints provide. It excludes degrees of freedom associated with cut joints.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **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 two types of blocks: Constraints and Joints. Joint blocks contribute constraint equations only if the joints are cut in the kinematic tree. The number of position constraint equations that a cut joint contributes equals six minus the number of degrees of freedom that joint provides.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **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. SimMechanics 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 number of tree 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 SimMechanics 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 kinematic loop length** — This statistic provides the average number of edges—or, equivalently, vertices—in the closed loops of a kinematic graph. The average number is taken over all loops in the graph. If the graph has no kinematic loops, this number equals zero.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

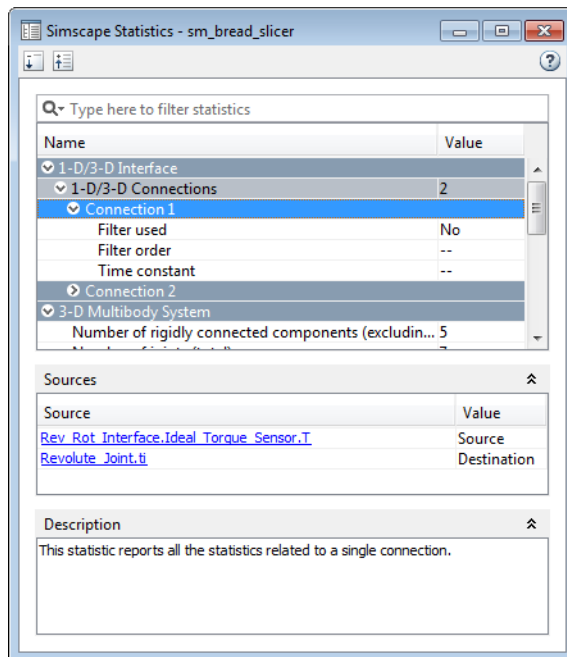
1-D/3-D Interface Statistics

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 SimMechanics Second Generation library to blocks from Simscape, SimDriveline, SimHydraulics, SimElectronics, and SimPowerSystems Simscape Components libraries.

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.

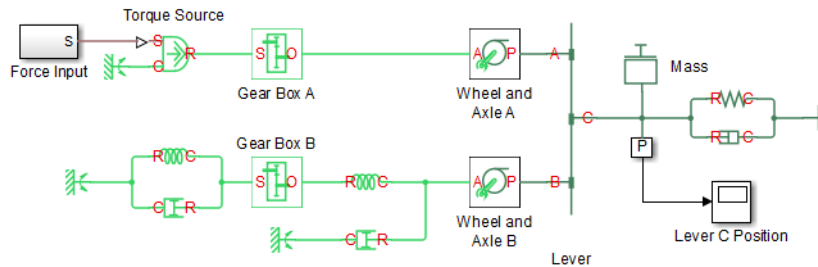
If you expand a connection node, the Statistics Viewer provides the filtering information: whether a filter is used, and, if yes, the filter order and time constant.



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.




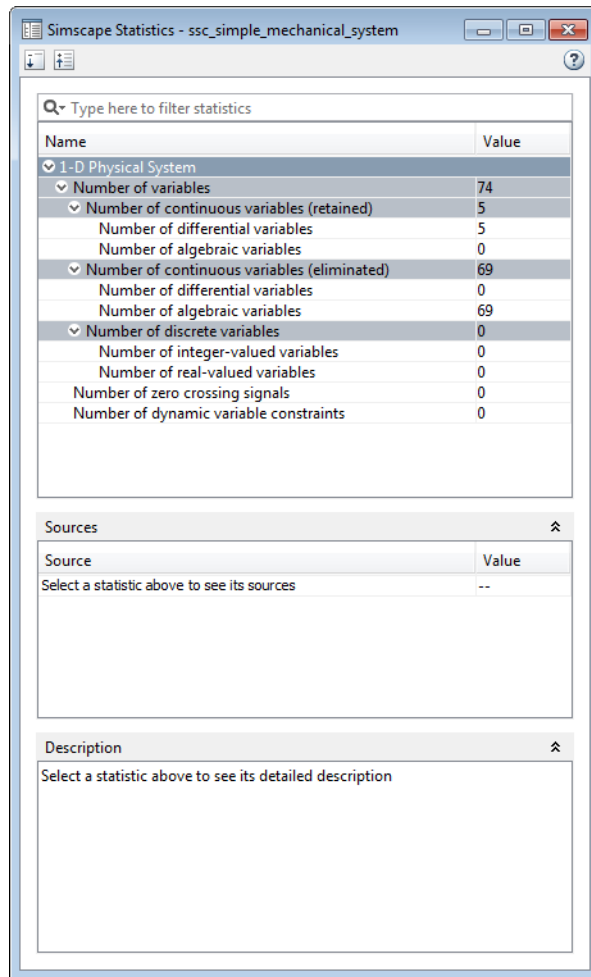
Simple Mechanical System

1. [Explore simulation results](#) using [sscxplore](#)
2. [Learn more](#) about this example

- 2 To view model statistics, in the top menu bar of the model window, select **Analysis > Simscape > Statistics Viewer**.

The Simscape Statistics window opens, displaying the name of the model and an overview of the models statistics in a collapsed state.

- 3 Click  to expand all nodes.



Simscape Statistics - ssc_simple_mechanical_system

Type here to filter statistics

Name	Value
1-D Physical System	
Number of variables	74
Number of continuous variables (retained)	5
Number of differential variables	5
Number of algebraic variables	0
Number of continuous variables (eliminated)	69
Number of differential variables	0
Number of algebraic variables	69
Number of discrete variables	0
Number of integer-valued variables	0
Number of real-valued variables	0
Number of zero crossing signals	0
Number of dynamic variable constraints	0

Sources

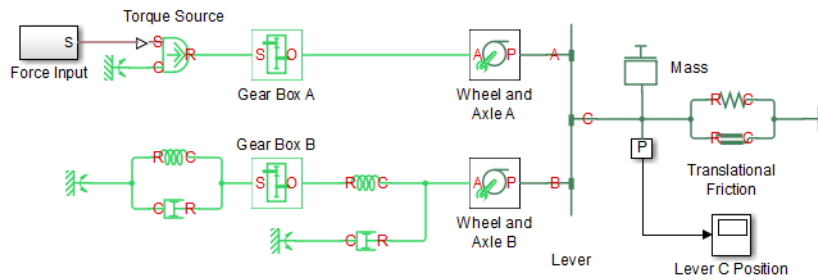
Source	Value
Select a statistic above to see its sources	--

Description

Select a statistic above to see its detailed description

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.

- 4 Replace the Translational Damper block in the model diagram with a Translational Friction block, as shown in the following figure.

**Simple Mechanical System**

1. [Explore simulation results](#) using [sscexplore](#)
2. [Learn more](#) about this example

5 Select **Analysis > Simscape > Statistics Viewer** to refresh the model statistics.

Name	Value
1-D Physical System	
Number of variables	74
Number of continuous variables (retained)	11
Number of differential variables	5
Number of algebraic variables	6
Number of continuous variables (eliminated)	63
Number of differential variables	0
Number of algebraic variables	63
Number of discrete variables	0
Number of integer-valued variables	0
Number of real-valued variables	0
Number of zero crossing signals	2
ssc_simple_mechanical_system/Translational Friction	2
Zero crossing signal 1	Source cod...
Zero crossing signal 2	Source cod...
Number of dynamic variable constraints	0
Sources	
Source	Value
Select a statistic above to see its sources	--
Description	
Select a statistic above to see its detailed description	

The revised model contains five differential variables, six algebraic variables, and two zero-crossing signals. This happened because you replaced a linear block (Translational Damper) with a nonlinear one (Translational Friction). Therefore the linear optimization that the solver initially performed on the model no longer applies.

Related Examples

- “Access Block Variables Using Statistics Viewer” on page 10-16

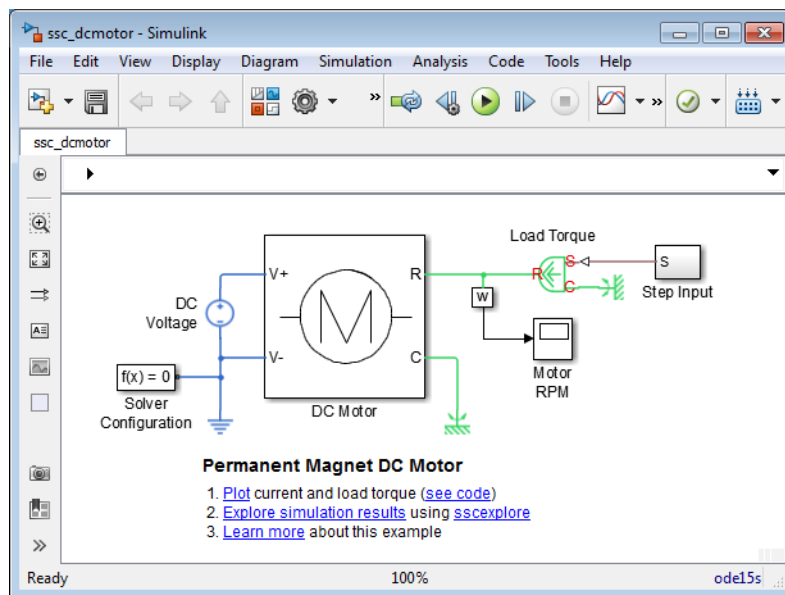
More About

- “1-D Physical System Statistics” on page 10-4

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.

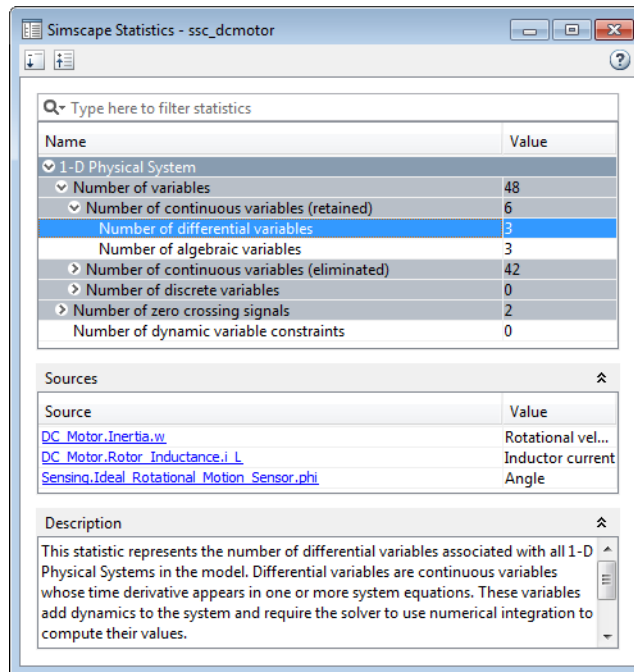
- 1 Open the Permanent Magnet DC Motor example model.



- 2 To view model statistics, in the top menu bar of the model window, select **Analysis > Simscape > Statistics Viewer**.

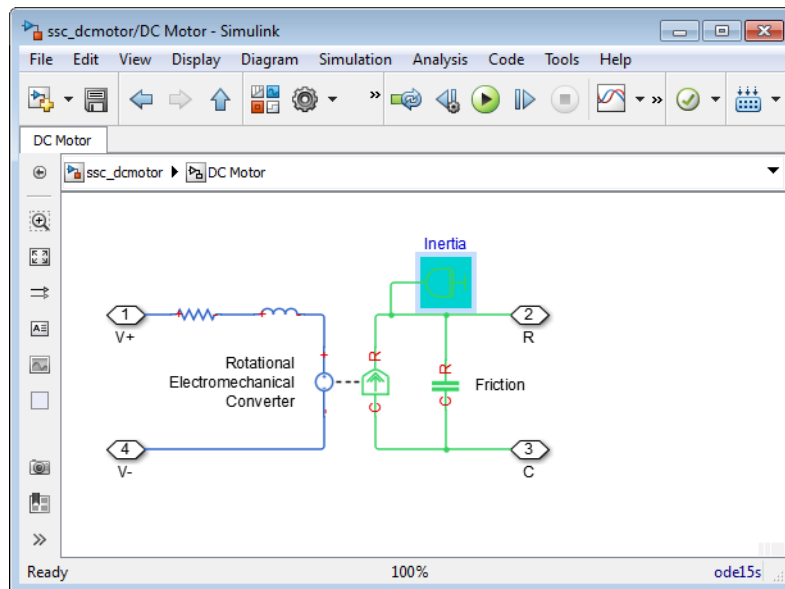
The Simscape Statistics window opens, displaying the name of the model and an overview of the models statistics in a collapsed state.

- 3 Expand the **Number of variables** node, then **Number of continuous variables (retained)**, and then click **Number of differential variables**.

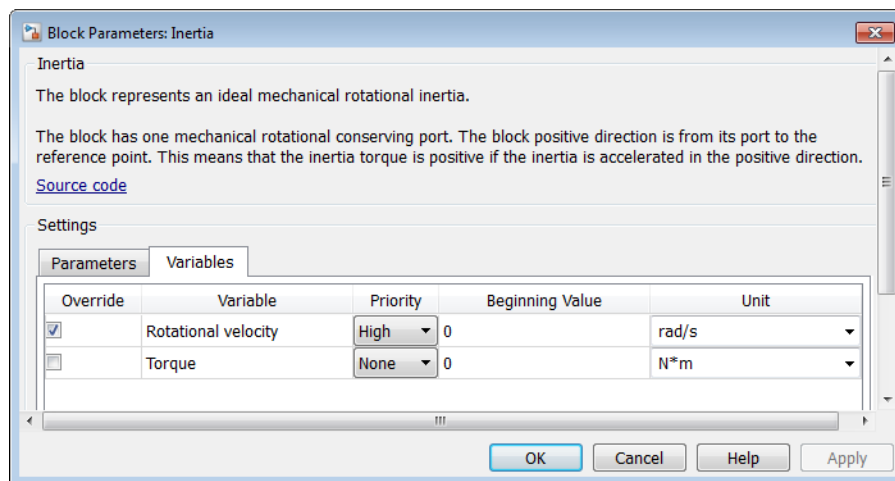


You can see that, after variable elimination, the model contains three continuous differential variables, and the **Sources** section of the Statistics Viewer lists these three variables. For each variable:

- 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 **Variables** tab of the block dialog box.
- 4 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.



- 5 Double-click the highlighted block.
- 6 In the block dialog box, click the **Variables** tab.



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.

Related Examples

- “Set Priority and Initial Target for Block Variables” on page 5-5

More About

- “1-D Physical System Statistics” on page 10-4
- “About Variable Initialization” on page 5-2
- “Variable Viewer” on page 5-23

Physical Units

- “How to Work with Physical Units” on page 11-2
- “Unit Definitions” on page 11-4
- “How to Specify Units in Block Dialogs” on page 11-9
- “Thermal Unit Conversions” on page 11-11
- “Angular Units” on page 11-15
- “Units for Angular Velocity and Frequency” on page 11-16

How to Work with Physical Units

Unlike Simulink signals, which are essentially unitless, physical signals can have units associated with them. You specify the units along with the parameter values in the block dialogs, 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 11-4). 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 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 11-9.

Note Currently, the blocks in the Physical Signals library (such as PS Add, PS Gain, and so on) ignore the physical unit of the input signal and just perform calculations on the value. The output signals of the Physical Signals library blocks are unitless.

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
Conductance	S	Siemens
	nS	Nanosiemens
	uS	Microsiemens
	mS	Millisiemens

Quantity	Abbreviation	Unit
Current	A	Ampere
	pA	Picoampere
	nA	Nanoampere
	uA	Microampere
	mA	Milliampere
	kA	Kiloampere
Energy	J	Joule
	Btu	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

Quantity	Abbreviation	Unit
Length	m	Meter
	cm	Centimeter
	mm	Millimeter
	km	Kilometer
	um	Micrometer
	in	Inch
	ft	Foot
	mi	Mile
	yd	Yard
Magnetic flux	Wb	Weber
Magnetic flux density	T	Tesla
	G	Gauss
Mass	kg	Kilogram
	g	Gram
	mg	Milligram
	lbm	Pound mass
	oz	Ounce
	slug	Slug
Pressure	Pa	Pascal
	kPa	Kilopascal
	MPa	Megapascal
	GPa	Gigapascal

Quantity	Abbreviation	Unit
	bar	Bar
	kbar	Kilobar
	atm	Atmosphere
	psi	Pound/inch ²
Power	W	Watt
	uW	Microwatt
	mW	Milliwatt
	kW	Kilowatt
	MW	Megawatt
	HP	Horsepower
Resistance	Ohm	Ohm
	kOhm	Kiloohm
	MOhm	Megaohm
	GOhm	Gigaohm
Temperature	K	Kelvin
	C	Celsius
	Fh	Fahrenheit
	R	Rankine
Time	s	Second
	min	Minute
	hr	Hour
	ms	Millisecond

Quantity	Abbreviation	Unit
	us	Microsecond
	ns	Nanosecond
Velocity	mph	Miles/hour
	fpm	Feet/minute
	fps	Feet/second
Viscosity absolute	Poise	Poise
	cP	Centipoise
	reyn	Reyn
Viscosity kinematic	St	Stokes
	cSt	Centistokes
	Newt	Newt
Volume	l	Liter
	gal	US liquid gallon
	igal	Imperial (UK) gallon
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 11-9.

How to Specify Units in Block Dialogs

Simscape block dialogs have drop-down combo boxes for units next to a parameter value. For example, in the Constant Volume Chamber block dialog box, the drop-down list for the **Chamber volume** parameter contains l, gal, in³, ft³, mm³, cm³, m³, and km³, and the drop-down list for the **Initial pressure** parameter contains Pa, bar, psi, and atm.

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 of the block dialog. 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 11-11.

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** or **OK** in the block dialog box, and issues an error if you type an incorrect unit.

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 4-7 for details.

Thermal Unit Conversions

In this section...

“About Affine Units” on page 11-11

“When to Apply Affine Conversion” on page 11-11

“How to Apply Affine Conversion” on page 11-12

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 (C) and Fahrenheit (Fh) affine units because they are both related to kelvin with an affine conversion. Rankine (R) 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('C', [1 273.15], 'K'); % defines Celsius in terms of kelvin
pm_addunit('Fh', [5/9 -32*5/9], 'C'); % defines Fahrenheit in terms of Celsius
pm_addunit('R', [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.

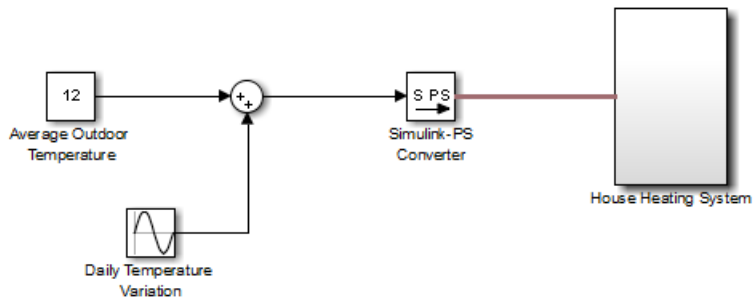
This is even better illustrated if you use degrees Celsius for the input units because the linear term for conversion between Celsius and kelvin is 1:

- 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).

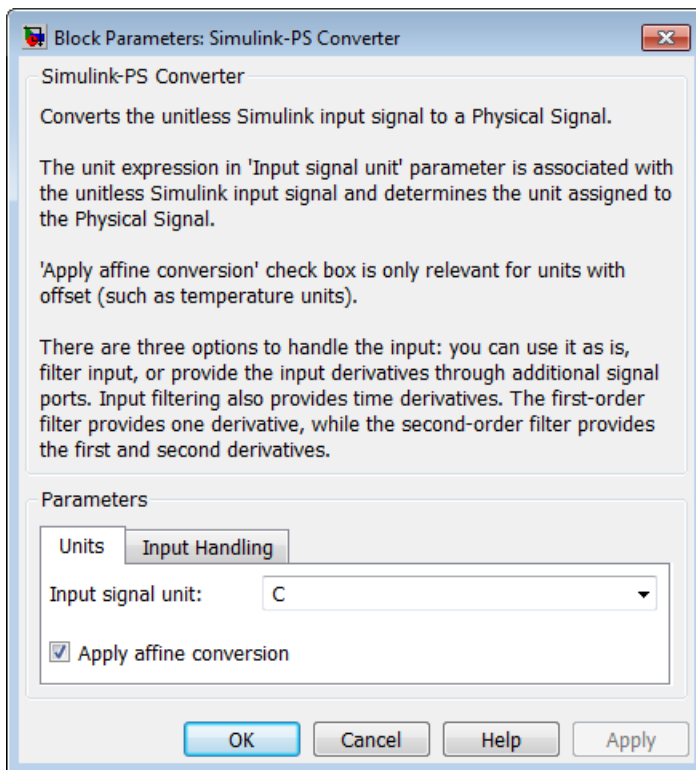
How to Apply Affine Conversion

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 11-11).

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 **C** 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.

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]. 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 $\text{N}\cdot\text{m}$) multiplied by angle (in rad) can be added directly to energy (in J, or $\text{N}\cdot\text{m}$). 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*, <http://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 2π conversion factor. In the SI unit system, however, the unit of frequency is hertz (Hz), defined as $1/\text{s}$.

Simscape software defines the unit hertz (Hz) as $1/\text{s}$, 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 2π 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 2π , to convert it to angular frequency before calculating the sine function.

Drop-down lists of suggested units in block dialogs 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 $1/\text{s}$, 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 11-15). 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.

Add-On Product License Management

- “About the Simscape Editing Mode” on page 12-2
- “Set the Model Loading Preference” on page 12-9
- “Save a Model in Restricted Mode” on page 12-11
- “Work with a Model in Restricted Mode” on page 12-14
- “Switch from Restricted to Full Mode” on page 12-28
- “Editing Mode Information” on page 12-30

About the Simscape Editing Mode

In this section...
“Suggested Workflows” on page 12-2
“What You Can Do in Restricted Mode” on page 12-3
“What You Can Do in Full Mode” on page 12-4
“Switching Between Modes” on page 12-4
“Working with Block Libraries” on page 12-7

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: SimDriveline, SimElectronics, SimHydraulics, SimMechanics, and SimPowerSystems. 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, SimHydraulics, and SimDriveline 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 MathWorks recommends 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, 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.

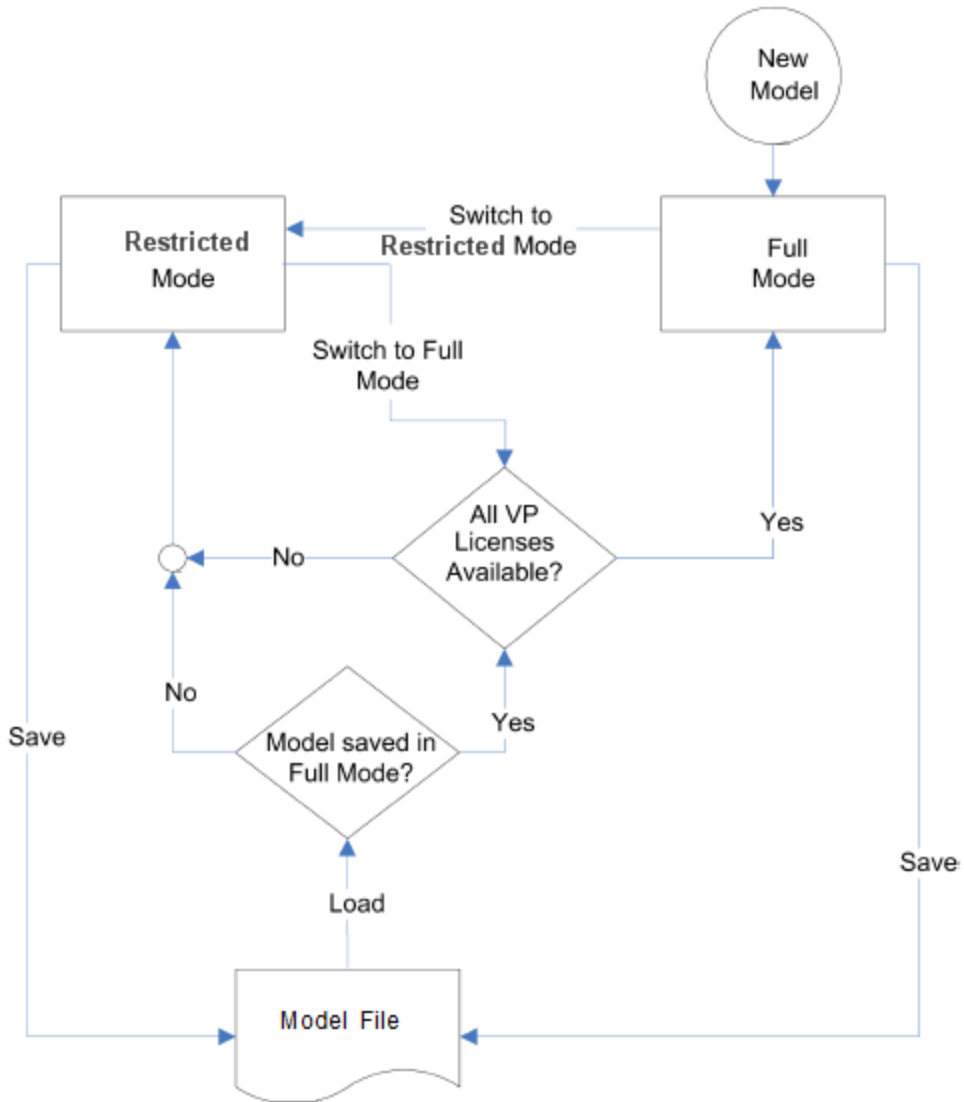
What You Can Do in Full Mode

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 (for more information, see “Protected Model”).

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 12-6.

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 12-6.
- 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 SimHydraulics and SimDriveline libraries, but the user who tries to open it has only the SimDriveline license available. It may happen that the license manager checks out a SimDriveline license first, and then tries to check out a SimHydraulics license, which is not available. The license manager then issues an error message and opens the model in Restricted mode, but the SimDriveline 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 physical modeling blocks to a library block, you need to work in Full mode.
 - If this library block had not previously contained physical modeling 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 physical modeling 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.

Related Examples

- “Set the Model Loading Preference” on page 12-9
- “Save a Model in Restricted Mode” on page 12-11
- “Work with a Model in Restricted Mode” on page 12-14
- “Switch from Restricted to Full Mode” on page 12-28

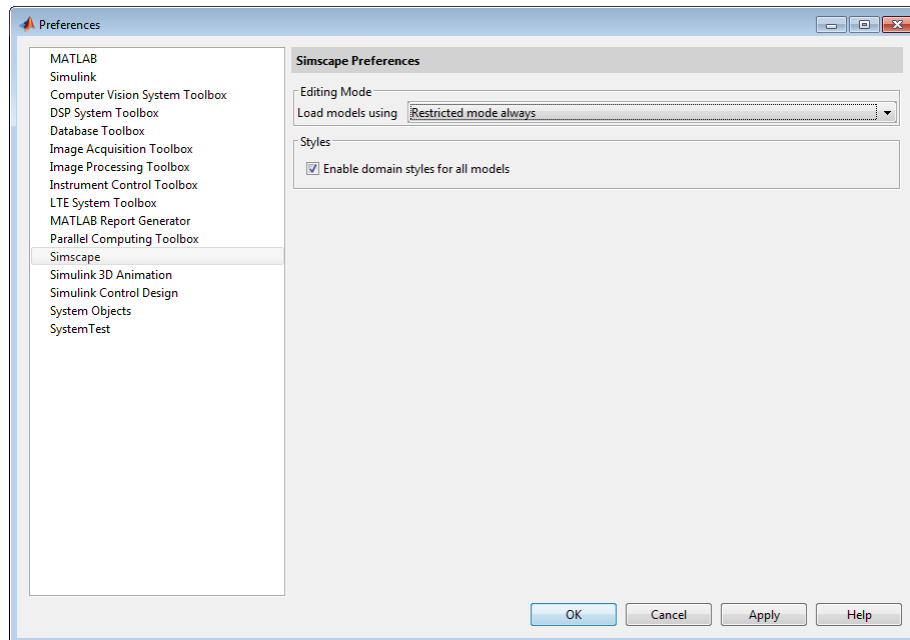
More About

- “Editing Mode Information” on page 12-30

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.

Related Examples

- “Save a Model in Restricted Mode” on page 12-11

- “Work with a Model in Restricted Mode” on page 12-14
- “Switch from Restricted to Full Mode” on page 12-28

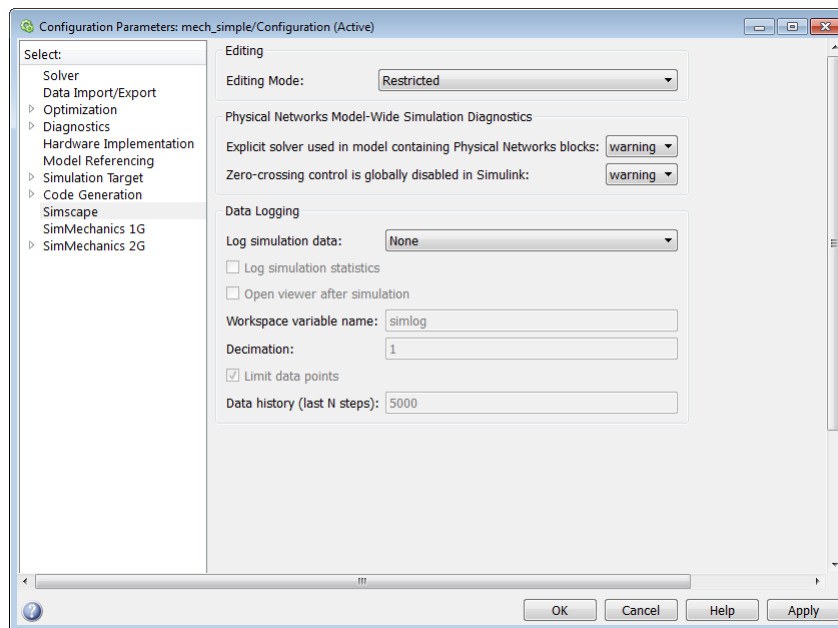
More About

- “About the Simscape Editing Mode” on page 12-2
- “Editing Mode Information” on page 12-30

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 From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. 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.

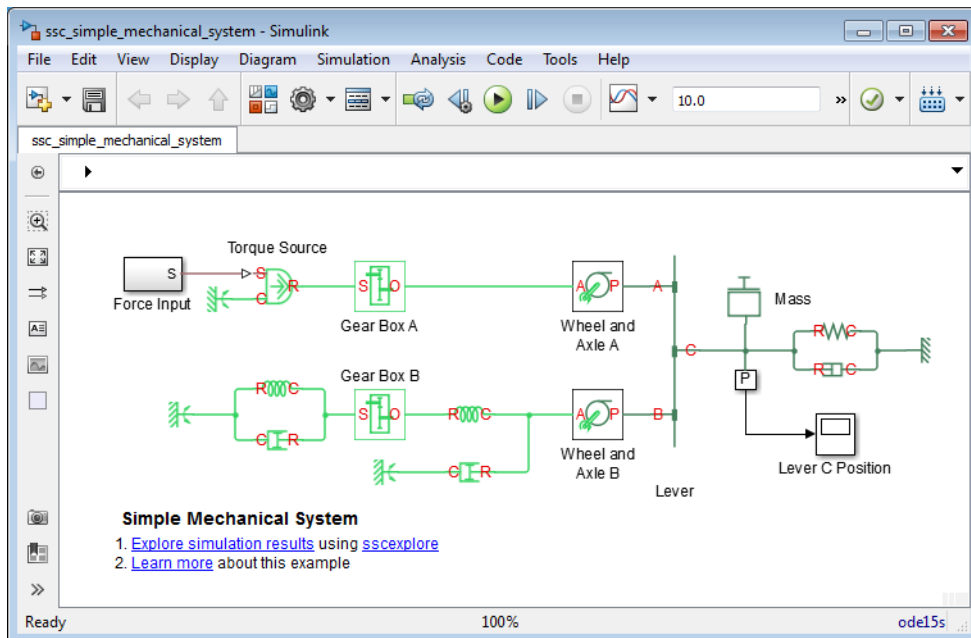
Once you have switched a model to Restricted mode, working with it follows the rules described in “Work with a Model in Restricted Mode” on page 12-14. Note, however, that the add-on product licenses for this model stay checked out until you quit the MATLAB session.

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 From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. 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`.

Related Examples

- “Set the Model Loading Preference” on page 12-9
- “Work with a Model in Restricted Mode” on page 12-14
- “Switch from Restricted to Full Mode” on page 12-28

More About

- “About the Simscape Editing Mode” on page 12-2
- “Editing Mode Information” on page 12-30

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 12-3.

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 Constant Volume Chamber block, the **Chamber specification** parameter is restricted. If, at the time the model entered Restricted mode, this parameter was set to **By volume**, then the nonrestricted parameters available for fine-tuning would be **Chamber volume**, **Specific heat ratio**, and **Initial pressure**. If, however, it was set to **By length and diameter**, you will 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.

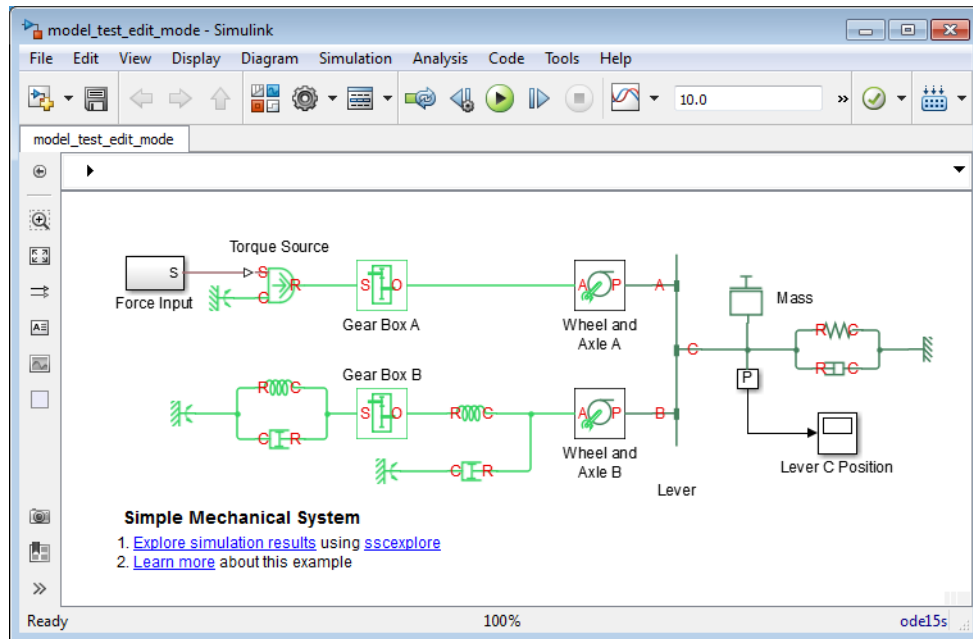
The following examples illustrate operations allowed and disallowed in Restricted mode:

In this section...
“How to Simulate and Fine-Tune a Model in Restricted Mode” on page 12-14
“How to Add and Delete Simulink Blocks in Restricted Mode” on page 12-19
“Performing an Operation Disallowed in Restricted Mode” on page 12-24

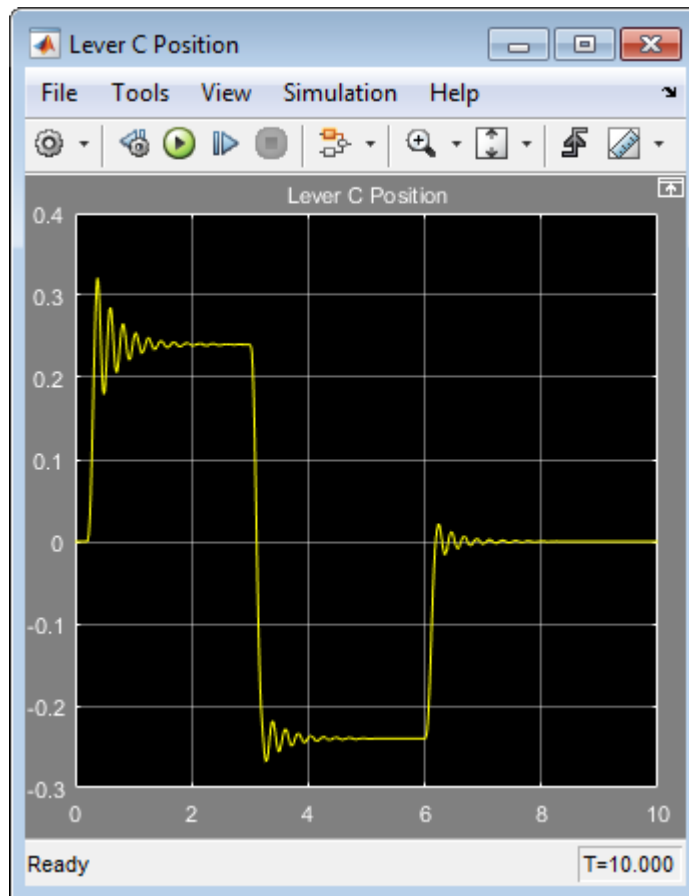
How to Simulate and Fine-Tune a Model in Restricted Mode

This example shows how you can work with a model in Restricted mode by changing certain parameter values and observing the simulation results.

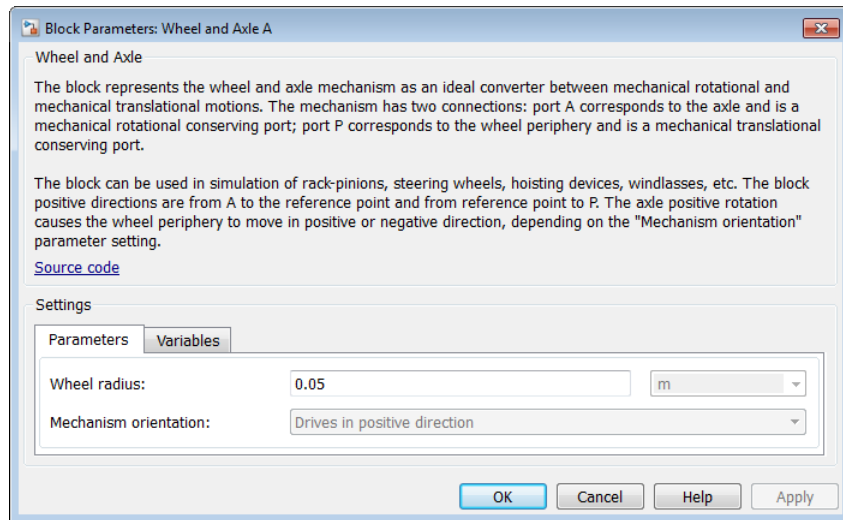
- 1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 12-12. The model opens in Restricted mode.



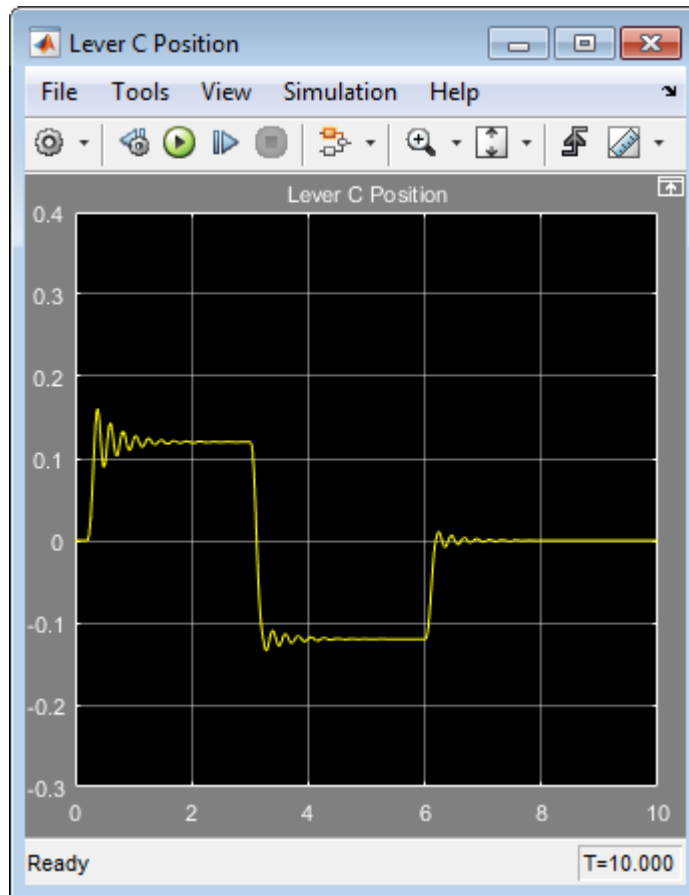
- 2 Open the Lever C Position scope and simulate the model. The models runs and simulates in Restricted mode.



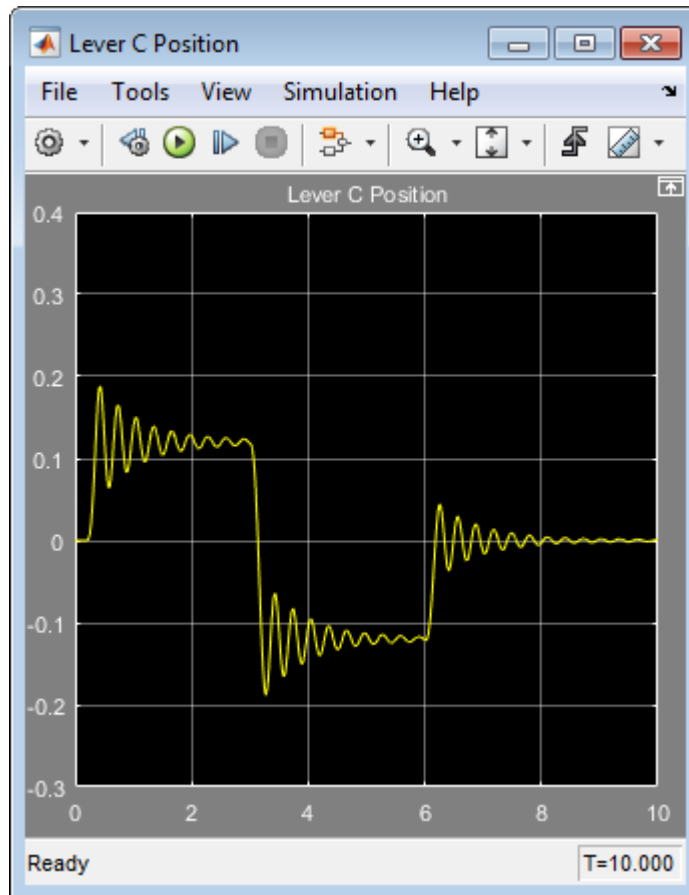
- 3 Double-click the Wheel and Axle A block to open its dialog box. Notice that the **Mechanism orientation** parameter is grayed out, because you cannot modify the block driving direction in Restricted mode.



- 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.



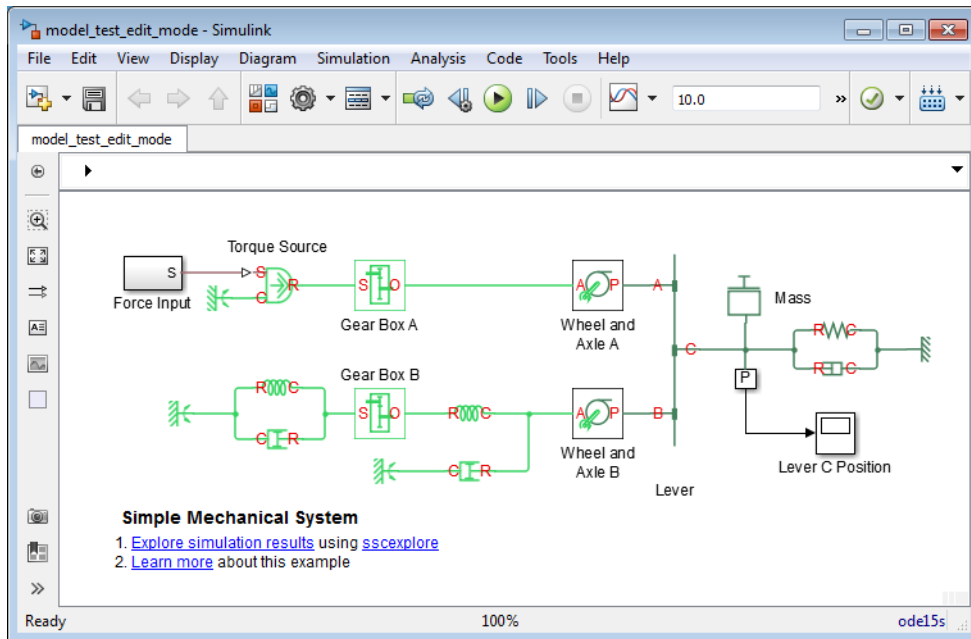
- 6 Double-click the Mass block and change the **Mass** parameter value to 24.
- 7 Simulate the model. Notice that doubling the mass resulted in increased vibrations.



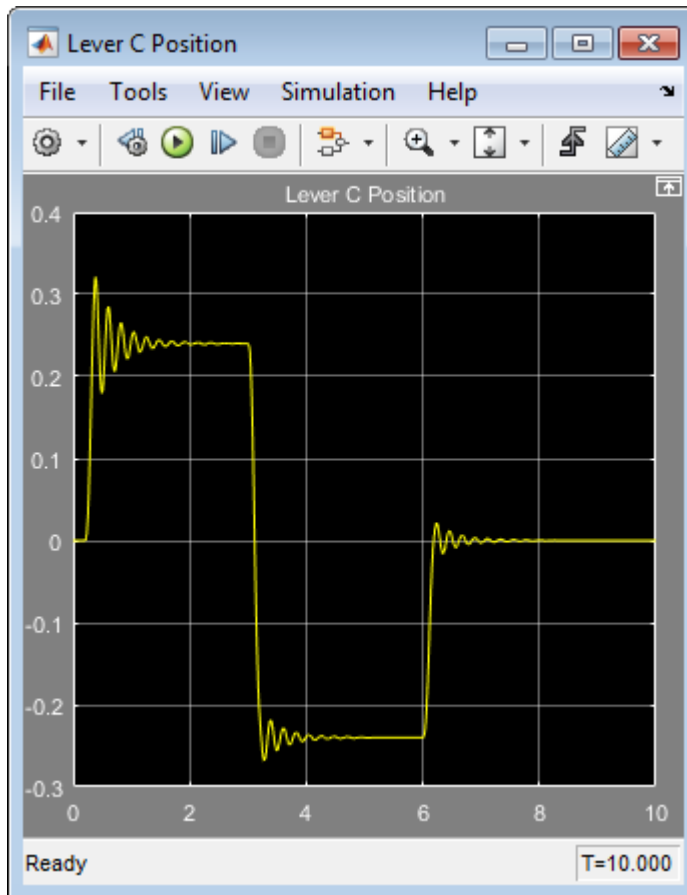
How to Add and Delete Simulink Blocks in Restricted Mode

This example shows how you can change the model input signal in Restricted mode by adding and deleting basic Simulink blocks.

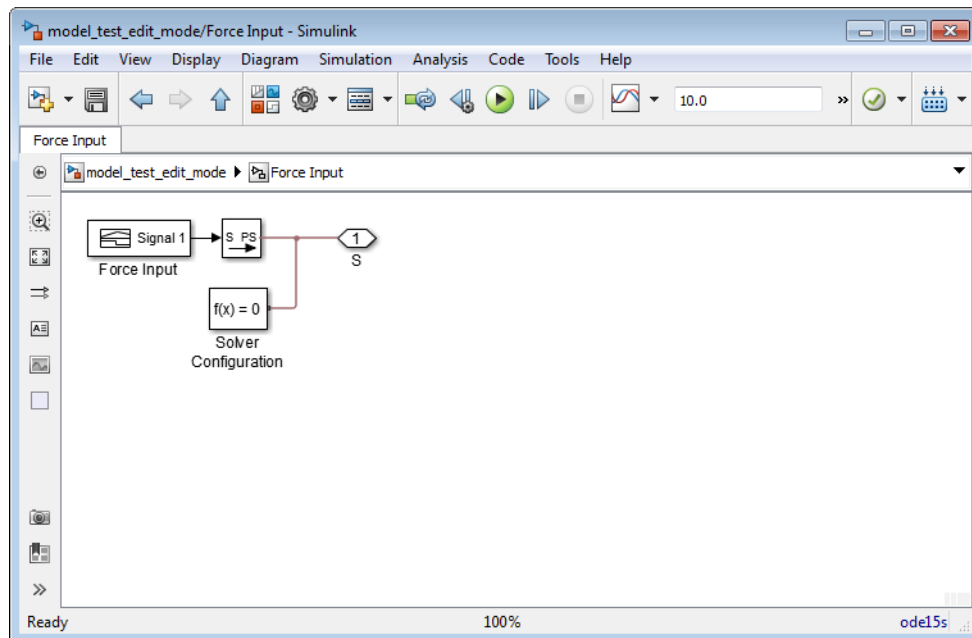
- 1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 12-12. The model opens in Restricted mode.



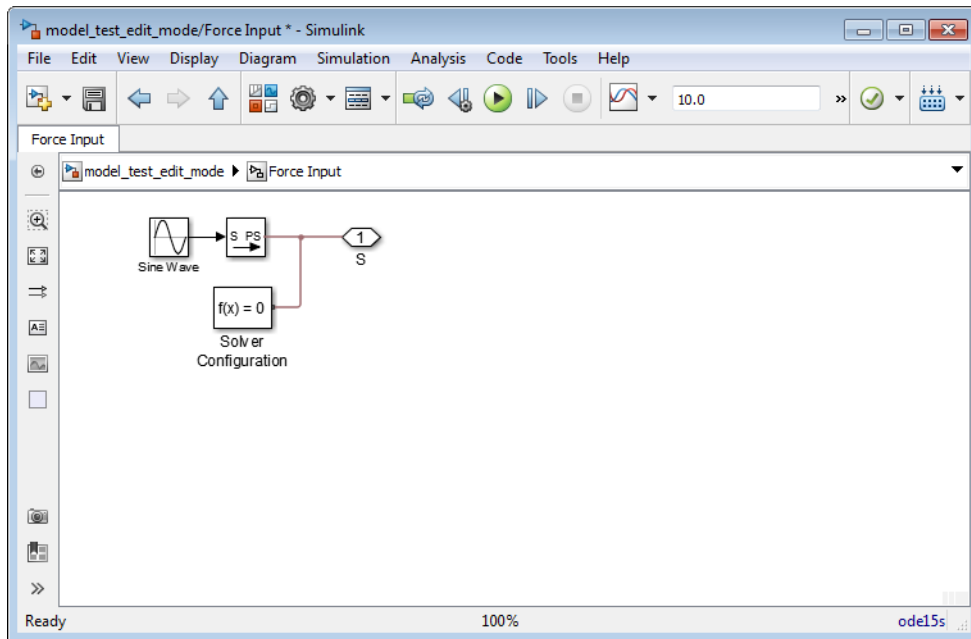
- 2 Open the Lever C Position scope and simulate the model.



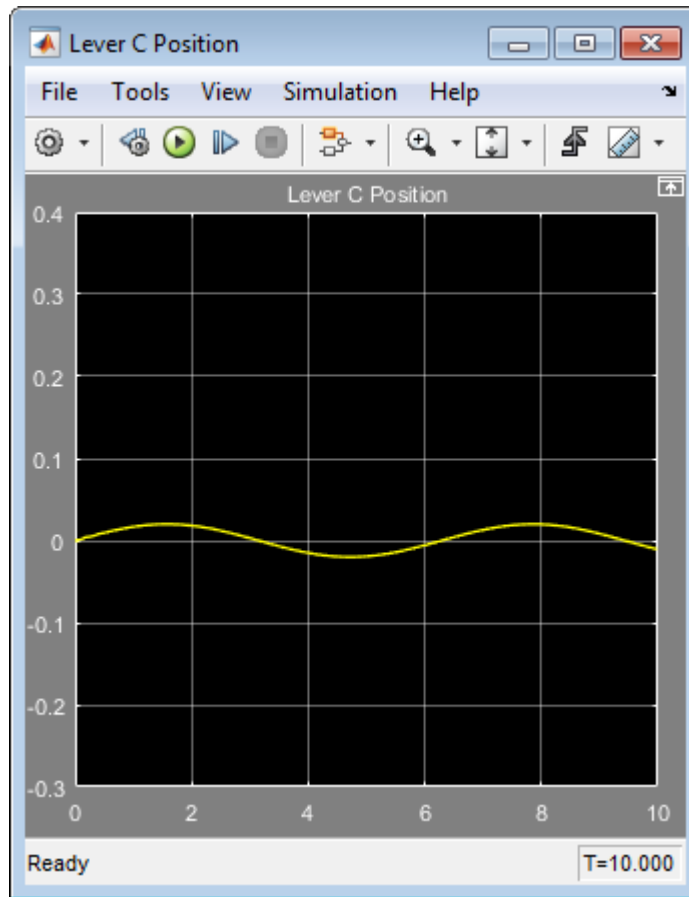
- 3 Double-click the Force Input subsystem to open it.



- 4 Inside the subsystem, delete the Signal Builder block named Force Input. Replace it with a Sine Wave block from the Simulink Sources library, as shown below.



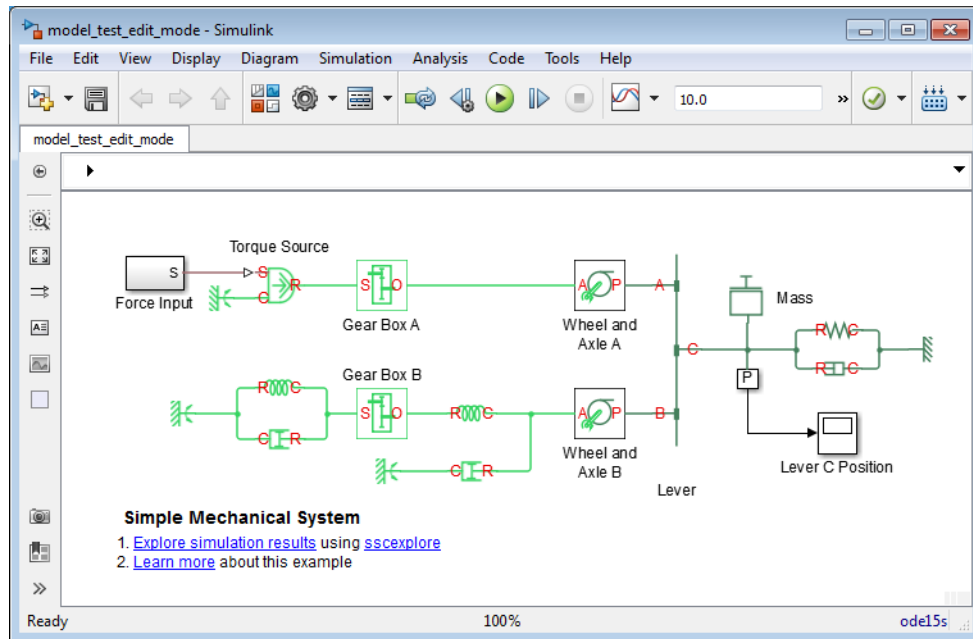
- 5 Simulate the model again. The model successfully compiles and simulates in Restricted mode.



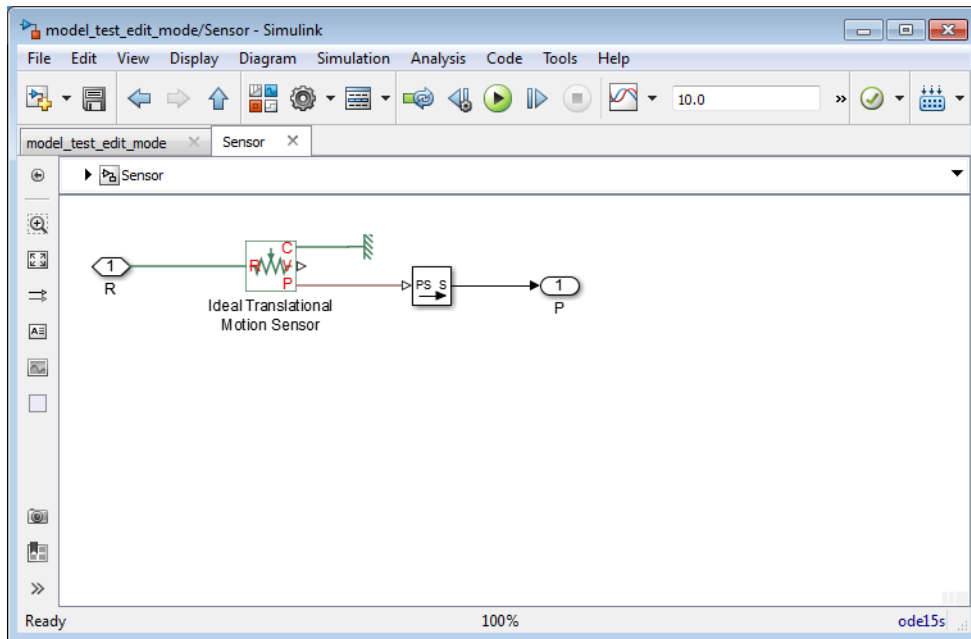
Performing an Operation Disallowed in Restricted Mode

This example shows what happens when you perform an operation that is disallowed in Restricted mode.

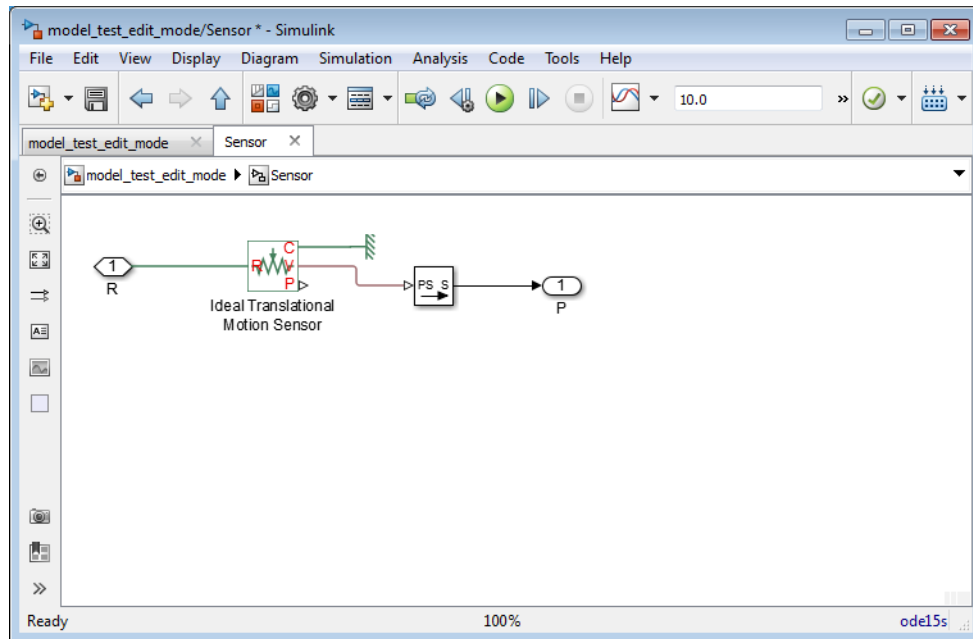
- 1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 12-12. The model opens in Restricted mode.



- 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 12-28.

Related Examples

- “Set the Model Loading Preference” on page 12-9
- “Save a Model in Restricted Mode” on page 12-11
- “Switch from Restricted to Full Mode” on page 12-28

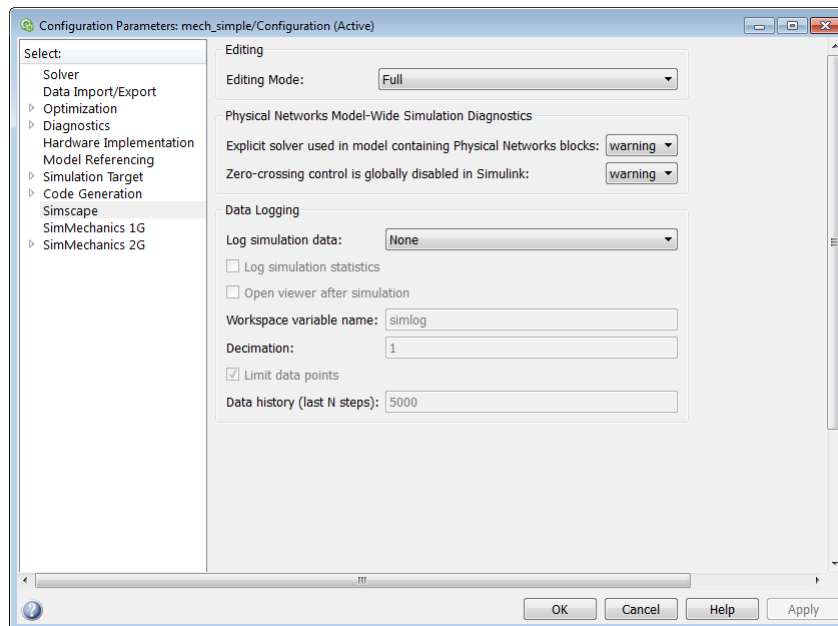
More About

- “About the Simscape Editing Mode” on page 12-2
- “Editing Mode Information” on page 12-30

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 From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. 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 12-6.

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.

Related Examples

- “Set the Model Loading Preference” on page 12-9
- “Save a Model in Restricted Mode” on page 12-11
- “Work with a Model in Restricted Mode” on page 12-14

More About

- “About the Simscape Editing Mode” on page 12-2
- “Editing Mode Information” on page 12-30

Editing Mode Information

In this section...
“What Is the Current Mode?” on page 12-30
“Which Licenses Are Checked Out?” on page 12-30

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 From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. 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.

Related Examples

- “Set the Model Loading Preference” on page 12-9
- “Save a Model in Restricted Mode” on page 12-11
- “Work with a Model in Restricted Mode” on page 12-14
- “Switch from Restricted to Full Mode” on page 12-28

More About

- “About the Simscape Editing Mode” on page 12-2